Example Language: Java *(Bad)*

```
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/CBC/PKCS5Padding");
    IvParameterSpec ips = new IvParameterSpec(iv);
    cipher.init(Cipher.ENCRYPT_MODE, key, ips);
    return cipher.doFinal(inpBytes);
  }
}
```

In both of these examples, the initialization vector (IV) is always a block of zeros. This makes the resulting cipher text much more predictable and susceptible to a dictionary attack.

Observed Examples

Reference	Description
CVE-2020-5408	encryption functionality in an authentication framework uses a fixed null IV with CBC mode, allowing attackers to decrypt traffic in applications that use this functionality *https://www.cve.org/CVERecord?id=CVE-2020-5408*
CVE-2017-17704	messages for a door-unlocking product use a fixed IV in CBC mode, which is the same after each restart *https://www.cve.org/CVERecord?id=CVE-2017-17704*
CVE-2017-11133	application uses AES in CBC mode, but the pseudo-random secret and IV are generated using math.random, which is not cryptographically strong. *https://www.cve.org/CVERecord?id=CVE-2017-11133*
CVE-2007-3528	Blowfish-CBC implementation constructs an IV where each byte is calculated modulo 8 instead of modulo 256, resulting in less than 12 bits for the effective IV length, and less than 4096 possible IV values. *https://www.cve.org/CVERecord?id=CVE-2007-3528*
CVE-2011-3389	BEAST attack in SSL 3.0 / TLS 1.0. In CBC mode, chained initialization vectors are non-random, allowing decryption of HTTPS traffic using a chosen plaintext attack. *https://www.cve.org/CVERecord?id=CVE-2011-3389*

Functional Areas

- Cryptography

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	2398
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1370	ICS Supply Chain: Common Mode Frailties	1358	2507
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Not using a random IV with CBC mode

References

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

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-1171]Matthew Green. "Why IND-CPA implies randomized encryption". 2018 August 4. < https://blog.cryptographyengineering.com/why-ind-cpa-implies-randomized-encryption/ >.

[REF-1172]NIST. "Recommendation for Block Cipher Modes of Operation". 2001 December. < https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf >.2023-04-07.

CWE-330: Use of Insufficiently Random Values

Weakness ID : 330
Structure : Simple
Abstraction : Class

Description

The product uses insufficiently random numbers or values in a security context that depends on unpredictable numbers.

Extended Description

When product generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520
ParentOf	B	331	Insufficient Entropy	821
ParentOf	B	334	Small Space of Random Values	827
ParentOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	829
ParentOf	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	837
ParentOf	C	340	Generation of Predictable Numbers or Identifiers	842

Nature	Type	ID	Name	Page
ParentOf	B	344	Use of Invariant Value in Dynamically Changing Context	849
ParentOf	B	1204	Generation of Weak Initialization Vector (IV)	1987
ParentOf	B	1241	Use of Predictable Algorithm in Random Number Generator	2030
CanPrecede	B	804	Guessable CAPTCHA	1701

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

Nature	Type	ID	Name	Page
ParentOf	B	331	Insufficient Entropy	821
ParentOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	829
ParentOf	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	837

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Weakness Ordinalities

Primary :

Applicable Platforms

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

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

Background Details

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated. There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Other	Other *When a protection mechanism relies on random values to restrict access to a sensitive resource, such as a session ID or a seed for generating a cryptographic key, then the resource being protected could be accessed by guessing the ID or key.*	
Access Control Other	Bypass Protection Mechanism Other *If product relies on unique, unguessable IDs to identify a resource, an attacker might be able to guess an ID for a resource that is owned by another user. The attacker could then read the resource, or pre-create a resource with the*	

Scope	Impact	Likelihood
	same ID to prevent the legitimate program from properly sending the resource to the intended user. For example, a product might maintain session information in a file whose name is based on a username. An attacker could pre-create this file for a victim user, then set the permissions so that the application cannot generate the session for the victim, preventing the victim from using the application.	
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *When an authorization or authentication mechanism relies on random values to restrict access to restricted functionality, such as a session ID or a seed for generating a cryptographic key, then an attacker may access the restricted functionality by guessing the ID or key.*	

Detection Methods

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and look for library functions that indicate when randomness is being used. Run the process multiple times to see if the seed changes. Look for accesses of devices or equivalent resources that are commonly used for strong (or weak) randomness, such as /dev/urandom on Linux. Look for library or system calls that access predictable information such as process IDs and system time.

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Man-in-the-middle attack tool

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use a well-vetted algorithm that is currently considered to be strong by experts in the field, and select well-tested implementations with adequate length seeds. In general, if a pseudo-random number generator is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

Phase: Implementation

Consider a PRNG that re-seeds itself as needed from high quality pseudo-random output sources, such as hardware devices.

Phase: Testing

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

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Demonstrative Examples

Example 1:

This code attempts to generate a unique random identifier for a user's session.

Example Language: PHP *(Bad)*

```php
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Example 2:

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

Example Language: Java *(Bad)*

```
String GenerateReceiptURL(String baseUrl) {
  Random ranGen = new Random();
  ranGen.setSeed((new Date()).getTime());
  return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Observed Examples

Reference	Description
CVE-2021-3692	PHP framework uses mt_rand() function (Marsenne Twister) when generating tokens *https://www.cve.org/CVERecord?id=CVE-2021-3692*
CVE-2020-7010	Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time. *https://www.cve.org/CVERecord?id=CVE-2020-7010*
CVE-2009-3278	Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks. *https://www.cve.org/CVERecord?id=CVE-2009-3278*
CVE-2009-3238	Random number generator can repeatedly generate the same value. *https://www.cve.org/CVERecord?id=CVE-2009-3238*
CVE-2009-2367	Web application generates predictable session IDs, allowing session hijacking. *https://www.cve.org/CVERecord?id=CVE-2009-2367*
CVE-2009-2158	Password recovery utility generates a relatively small number of random passwords, simplifying brute force attacks. *https://www.cve.org/CVERecord?id=CVE-2009-2158*
CVE-2009-0255	Cryptographic key created with a seed based on the system time. *https://www.cve.org/CVERecord?id=CVE-2009-0255*
CVE-2008-5162	Kernel function does not have a good entropy source just after boot. *https://www.cve.org/CVERecord?id=CVE-2008-5162*
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash. *https://www.cve.org/CVERecord?id=CVE-2008-4905*
CVE-2008-4929	Bulletin board application uses insufficiently random names for uploaded files, allowing other users to access private files. *https://www.cve.org/CVERecord?id=CVE-2008-4929*
CVE-2008-3612	Handheld device uses predictable TCP sequence numbers, allowing spoofing or hijacking of TCP connections. *https://www.cve.org/CVERecord?id=CVE-2008-3612*
CVE-2008-2433	Web management console generates session IDs based on the login time, making it easier to conduct session hijacking. *https://www.cve.org/CVERecord?id=CVE-2008-2433*
CVE-2008-0166	SSL library uses a weak random number generator that only generates 65,536 unique keys. *https://www.cve.org/CVERecord?id=CVE-2008-0166*

Reference	Description
CVE-2008-2108	Chain: insufficient precision causes extra zero bits to be assigned, reducing entropy for an API function that generates random numbers. *https://www.cve.org/CVERecord?id=CVE-2008-2108*
CVE-2008-2108	Chain: insufficient precision (CWE-1339) in random-number generator causes some zero bits to be reliably generated, reducing the amount of entropy (CWE-331) *https://www.cve.org/CVERecord?id=CVE-2008-2108*
CVE-2008-2020	CAPTCHA implementation does not produce enough different images, allowing bypass using a database of all possible checksums. *https://www.cve.org/CVERecord?id=CVE-2008-2020*
CVE-2008-0087	DNS client uses predictable DNS transaction IDs, allowing DNS spoofing. *https://www.cve.org/CVERecord?id=CVE-2008-0087*
CVE-2008-0141	Application generates passwords that are based on the time of day. *https://www.cve.org/CVERecord?id=CVE-2008-0141*

Functional Areas

- Cryptography
- Authentication
- Session Management

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	2314
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2350
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	2353
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2381
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2462
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2463
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Relationship

This can be primary to many other weaknesses such as cryptographic errors, authentication errors, symlink following, information leaks, and others.

Maintenance

As of CWE 4.3, CWE-330 and its descendants are being investigated by the CWE crypto team to identify gaps related to randomness and unpredictability, as well as the relationships between randomness and cryptographic primitives. This "subtree analysis" might result in the addition or deprecation of existing entries; the reorganization of relationships in some views, e.g. the research view (CWE-1000); more consistent use of terminology; and/or significant modifications to related entries.

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Randomness and Predictability
7 Pernicious Kingdoms			Insecure Randomness
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
CERT C Secure Coding	CON33-C	Imprecise	Avoid race conditions when using library functions
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers
CERT C Secure Coding	MSC32-C	CWE More Abstract	Properly seed pseudorandom number generators
WASC	11		Brute Force
WASC	18		Credential/Session Prediction
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
59	Session Credential Falsification through Prediction
112	Brute Force
485	Signature Spoofing by Key Recreation

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

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

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

CWE-331: Insufficient Entropy

Weakness ID : 331
Structure : Simple
Abstraction : Base

Description

The product uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814
ParentOf	V	332	Insufficient Entropy in PRNG	823
ParentOf	V	333	Improper Handling of Insufficient Entropy in TRNG	825

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477
MemberOf	C	310	Cryptographic Issues	2318

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *An attacker could guess the random numbers generated and could gain unauthorized access to a system if the random numbers are used for authentication and authorization.*	

Potential Mitigations

Phase: Implementation

Determine the necessary entropy to adequately provide for randomness and predictability. This can be achieved by increasing the number of bits of objects such as keys and seeds.

Demonstrative Examples

Example 1:

This code generates a unique random identifier for a user's session.

Example Language: PHP *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Example 2:

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

Example Language: Java *(Bad)*

```
String GenerateReceiptURL(String baseUrl) {
  Random ranGen = new Random();
  ranGen.setSeed((new Date()).getTime());
  return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Observed Examples

Reference	Description
CVE-2001-0950	Insufficiently random data used to generate session tokens using C rand(). Also, for certificate/key generation, uses a source that does not block when entropy is low. *https://www.cve.org/CVERecord?id=CVE-2001-0950*
CVE-2008-2108	Chain: insufficient precision (CWE-1339) in random-number generator causes some zero bits to be reliably generated, reducing the amount of entropy (CWE-331) *https://www.cve.org/CVERecord?id=CVE-2008-2108*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2463
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Entropy
WASC	11		Brute Force
CERT C Secure Coding	MSC32-C	Exact	Properly seed pseudorandom number generators

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
59	Session Credential Falsification through Prediction

References

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

CWE-332: Insufficient Entropy in PRNG

Weakness ID : 332
Structure : Simple
Abstraction : Variant

Description

The lack of entropy available for, or used by, a Pseudo-Random Number Generator (PRNG) can be a stability and security threat.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	331	Insufficient Entropy	821

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *If a pseudo-random number generator is using a limited entropy source which runs out (if the generator fails closed), the program may pause or crash.*	
Access Control Other	Bypass Protection Mechanism Other *If a PRNG is using a limited entropy source which runs out, and the generator fails open, the generator could produce predictable random numbers. Potentially a weak source of random numbers could weaken the encryption method used for authentication of users.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Consider a PRNG that re-seeds itself as needed from high-quality pseudo-random output, such as hardware devices.

Phase: Architecture and Design

When deciding which PRNG to use, look at its sources of entropy. Depending on what your security needs are, you may need to use a random number generator that always uses strong random data -- i.e., a random number generator that attempts to be strong but will fail in a weak way or will always provide some middle ground of protection through techniques like re-seeding. Generally, something that always provides a predictable amount of strength is preferable.

Observed Examples

Reference	Description
[REF-1374]	Chain: JavaScript-based cryptocurrency library can fall back to the insecure Math.random() function instead of reporting a failure (CWE-392), thus reducing the entropy (CWE-332) and leading to generation of non-unique cryptographic keys for Bitcoin wallets (CWE-1391) *https://www.unciphered.com/blog/randstorm-you-cant-patch-a-house-of-cards*
CVE-2019-1715	security product has insufficient entropy in the DRBG, allowing collisions and private key discovery *https://www.cve.org/CVERecord?id=CVE-2019-1715*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Insufficient entropy in PRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-1374]Unciphered. "Randstorm: You Can't Patch a House of Cards". 2023 November 4. < https://www.unciphered.com/blog/randstorm-you-cant-patch-a-house-of-cards >.2023-11-15.

CWE-333: Improper Handling of Insufficient Entropy in TRNG

Weakness ID : 333
Structure : Simple
Abstraction : Variant

Description

True random number generators (TRNG) generally have a limited source of entropy and therefore can fail or block.

Extended Description

The rate at which true random numbers can be generated is limited. It is important that one uses them only when they are needed for security.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1576
ChildOf	B	331	Insufficient Entropy	821

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *A program may crash or block if it runs out of random numbers.*	

Potential Mitigations

Phase: Implementation

Rather than failing on a lack of random numbers, it is often preferable to wait for more numbers to be created.

Demonstrative Examples

Example 1:

This code uses a TRNG to generate a unique session id for new connections to a server:

Example Language: C *(Bad)*

```c
while (1){
  if (haveNewConnection()){
    if (hwRandom()){
      int sessionID = hwRandom();
      createNewConnection(sessionID);
} } }
```

This code does not attempt to limit the number of new connections or make sure the TRNG can successfully generate a new random number. An attacker may be able to create many new connections and exhaust the entropy of the TRNG. The TRNG may then block and cause the program to crash or hang.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure of TRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-334: Small Space of Random Values

Weakness ID : 334
Structure : Simple
Abstraction : Base

Description

The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814
ParentOf	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	2

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477
MemberOf	C	310	Cryptographic Issues	2318

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *An attacker could easily guess the values used. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Demonstrative Examples

Example 1:

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

Example Language: XML *(Bad)*

```
<sun-web-app>
  ...
  <session-config>
    <session-properties>
      <property name="idLengthBytes" value="8">
        <description>The number of bytes in this web module's session ID.</description>
      </property>
    </session-properties>
  </session-config>
  ...
</sun-web-app>
```

This deployment descriptor has set the session ID length for this Java web application to 8 bytes (or 64 bits). The session ID length for Java web applications should be set to 16 bytes (128 bits) to prevent attackers from guessing and/or stealing a session ID and taking over a user's session.

Note for most application servers including the Sun Java Application Server the session ID length is by default set to 128 bits and should not be changed. And for many application servers the session ID length cannot be changed from this default setting. Check your application server documentation for the session ID length default setting and configuration options to ensure that the session ID length is set to 128 bits.

Observed Examples

Reference	Description
CVE-2002-0583	Product uses 5 alphanumeric characters for filenames of expense claim reports, stored under web root. *https://www.cve.org/CVERecord?id=CVE-2002-0583*
CVE-2002-0903	Product uses small number of random numbers for a code to approve an action, and also uses predictable new user IDs, allowing attackers to hijack new accounts. *https://www.cve.org/CVERecord?id=CVE-2002-0903*
CVE-2003-1230	SYN cookies implementation only uses 32-bit keys, making it easier to brute force ISN. *https://www.cve.org/CVERecord?id=CVE-2003-1230*
CVE-2004-0230	Complex predictability / randomness (reduced space). *https://www.cve.org/CVERecord?id=CVE-2004-0230*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Small Space of Random Values

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)

Weakness ID : 335
Structure : Simple
Abstraction : Base

Description

The product uses a Pseudo-Random Number Generator (PRNG) but does not correctly manage seeds.

Extended Description

PRNGs are deterministic and, while their output appears random, they cannot actually create entropy. They rely on cryptographically secure and unique seeds for entropy so proper seeding is critical to the secure operation of the PRNG.

Management of seeds could be broken down into two main areas:

- (1) protecting seeds as cryptographic material (such as a cryptographic key);
- (2) whenever possible, using a uniquely generated seed from a cryptographically secure source

PRNGs require a seed as input to generate a stream of numbers that are functionally indistinguishable from random numbers. While the output is, in many cases, sufficient for cryptographic uses, the output of any PRNG is directly determined by the seed provided as input. If the seed can be ascertained by a third party, the entire output of the PRNG can be made known to them. As such, the seed should be kept secret and should ideally not be able to be guessed.

For example, the current time may be a poor seed. Knowing the approximate time the PRNG was seeded greatly reduces the possible key space.

Seeds do not necessarily need to be unique, but reusing seeds may open up attacks if the seed is discovered.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814
ParentOf	V	336	Same Seed in Pseudo-Random Number Generator (PRNG)	832
ParentOf	V	337	Predictable Seed in Pseudo-Random Number Generator (PRNG)	834
ParentOf	V	339	Small Seed Space in PRNG	840

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477
MemberOf	C	310	Cryptographic Issues	2318

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *If a PRNG is used incorrectly, such as using the same seed for each initialization or using a predictable seed, then an attacker may be able to easily guess the seed and thus the random numbers. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.*	

Demonstrative Examples

Example 1:

The following code uses a statistical PRNG to generate account IDs.

Example Language: Java *(Bad)*

```
private static final long SEED = 1234567890;
```

```
public int generateAccountID() {
  Random random = new Random(SEED);
  return random.nextInt();
}
```

Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

Example 2:

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

Example Language: Java *(Bad)*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

Example Language: C *(Bad)*

```
srand(time());
int randNum = rand();
```

An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

Example 3:

This code grabs some random bytes and uses them for a seed in a PRNG, in order to generate a new cryptographic key.

Example Language: Python *(Bad)*

```
# getting 2 bytes of randomness for the seeding the PRNG
seed = os.urandom(2)
random.seed(a=seed)
key = random.getrandbits(128)
```

Since only 2 bytes are used as a seed, an attacker will only need to guess 2^16 (65,536) values before being able to replicate the state of the PRNG.

Observed Examples

Reference	Description
CVE-2020-7010	Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time. *https://www.cve.org/CVERecord?id=CVE-2020-7010*
CVE-2019-11495	server uses erlang:now() to seed the PRNG, which results in a small search space for potential random seeds *https://www.cve.org/CVERecord?id=CVE-2019-11495*
CVE-2018-12520	Product's PRNG is not seeded for the generation of session IDs *https://www.cve.org/CVERecord?id=CVE-2018-12520*
CVE-2016-10180	Router's PIN generation is based on rand(time(0)) seeding. *https://www.cve.org/CVERecord?id=CVE-2016-10180*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PRNG Seed Error

References

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

CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG)

Weakness ID : 336
Structure : Simple
Abstraction : Variant

Description

A Pseudo-Random Number Generator (PRNG) uses the same seed each time the product is initialized.

Extended Description

Given the deterministic nature of PRNGs, using the same seed for each initialization will lead to the same output in the same order. If an attacker can guess (or knows) the seed, then the attacker may be able to determine the random numbers that will be produced from the PRNG.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	829

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other Access Control	Other Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Do not reuse PRNG seeds. Consider a PRNG that periodically re-seeds itself as needed from a high quality pseudo-random output, such as hardware devices.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems, or use the more recent FIPS 140-3 [REF-1192] if possible.

Demonstrative Examples

Example 1:

The following code uses a statistical PRNG to generate account IDs.

Example Language: Java *(Bad)*

```
private static final long SEED = 1234567890;
public int generateAccountID() {
  Random random = new Random(SEED);
  return random.nextInt();
}
```

Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

Example 2:

This code attempts to generate a unique random identifier for a user's session.

Example Language: PHP *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

If the user IDs are generated sequentially, or otherwise restricted to a narrow range of values, then this example also exhibits a Small Seed Space (CWE-339).

Observed Examples

Reference	Description
CVE-2022-39218	SDK for JavaScript app builder for serverless code uses the same fixed seed for a PRNG, allowing cryptography bypass *https://www.cve.org/CVERecord?id=CVE-2022-39218*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Same Seed in PRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)

Weakness ID : 337
Structure : Simple
Abstraction : Variant

Description

A Pseudo-Random Number Generator (PRNG) is initialized from a predictable seed, such as the process ID or system time.

Extended Description

The use of predictable seeds significantly reduces the number of possible seeds that an attacker would need to test in order to predict which random numbers will be generated by the PRNG.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	829

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Use non-predictable inputs for seed generation.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems, or use the more recent FIPS 140-3 [REF-1192] if possible.

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Demonstrative Examples

Example 1:

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

Example Language: Java *(Bad)*

```
Random random = new Random(System.currentTimeMillis());
```

```
int accountID = random.nextInt();
```

Example Language: C *(Bad)*

```
srand(time());
int randNum = rand();
```

An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

Observed Examples

Reference	Description
CVE-2020-7010	Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time. *https://www.cve.org/CVERecord?id=CVE-2020-7010*
CVE-2019-11495	server uses erlang:now() to seed the PRNG, which results in a small search space for potential random seeds *https://www.cve.org/CVERecord?id=CVE-2019-11495*
CVE-2008-0166	The removal of a couple lines of code caused Debian's OpenSSL Package to only use the current process ID for seeding a PRNG *https://www.cve.org/CVERecord?id=CVE-2008-0166*
CVE-2016-10180	Router's PIN generation is based on rand(time(0)) seeding. *https://www.cve.org/CVERecord?id=CVE-2016-10180*
CVE-2018-9057	cloud provider product uses a non-cryptographically secure PRNG and seeds it with the current time *https://www.cve.org/CVERecord?id=CVE-2018-9057*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable Seed in PRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

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

CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)

Weakness ID : 338
Structure : Simple
Abstraction : Base

Description

The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG's algorithm is not cryptographically strong.

Extended Description

When a non-cryptographic PRNG is used in a cryptographic context, it can expose the cryptography to certain types of attacks.

Often a pseudo-random number generator (PRNG) is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms that use random numbers. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system. While such PRNGs might have very useful features, these same features could be used to break the cryptography.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	330	Use of Insufficiently Random Values	814

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	330	Use of Insufficiently Random Values	814

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477
MemberOf	C	310	Cryptographic Issues	2318

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If a PRNG is used for authentication and authorization, such as a session ID or a seed for generating a cryptographic key, then an attacker may be able to easily guess the ID or cryptographic key and gain access to restricted functionality.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CyptGenRandom on Windows, or hw_rand() on Linux.

Demonstrative Examples

Example 1:

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

Example Language: Java *(Bad)*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

Example Language: C *(Bad)*

```
srand(time());
int randNum = rand();
```

The random number functions used in these examples, rand() and Random.nextInt(), are not considered cryptographically strong. An attacker may be able to predict the random numbers generated by these functions. Note that these example also exhibit CWE-337 (Predictable Seed in PRNG).

Observed Examples

Reference	Description
CVE-2021-3692	PHP framework uses mt_rand() function (Marsenne Twister) when generating tokens *https://www.cve.org/CVERecord?id=CVE-2021-3692*
CVE-2009-3278	Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks. *https://www.cve.org/CVERecord?id=CVE-2009-3278*
CVE-2009-3238	Random number generator can repeatedly generate the same value. *https://www.cve.org/CVERecord?id=CVE-2009-3238*
CVE-2009-2367	Web application generates predictable session IDs, allowing session hijacking. *https://www.cve.org/CVERecord?id=CVE-2009-2367*
CVE-2008-0166	SSL library uses a weak random number generator that only generates 65,536 unique keys. *https://www.cve.org/CVERecord?id=CVE-2008-0166*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2463
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Non-cryptographic PRNG
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

CWE-339: Small Seed Space in PRNG

Weakness ID : 339
Structure : Simple
Abstraction : Variant

Description

A Pseudo-Random Number Generator (PRNG) uses a relatively small seed space, which makes it more susceptible to brute force attacks.

Extended Description

PRNGs are entirely deterministic once seeded, so it should be extremely difficult to guess the seed. If an attacker can collect the outputs of a PRNG and then brute force the seed by trying every possibility to see which seed matches the observed output, then the attacker will know the output of any subsequent calls to the PRNG. A small seed space implies that the attacker will have far fewer possible values to try to exhaust all possibilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	829
PeerOf	B	341	Predictable from Observable State	843

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Use well vetted pseudo-random number generating algorithms with adequate length seeds. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems, or use the more recent FIPS 140-3 [REF-1192] if possible.

Demonstrative Examples

Example 1:

This code grabs some random bytes and uses them for a seed in a PRNG, in order to generate a new cryptographic key.

Example Language: Python *(Bad)*

```
# getting 2 bytes of randomness for the seeding the PRNG
seed = os.urandom(2)
random.seed(a=seed)
key = random.getrandbits(128)
```

Since only 2 bytes are used as a seed, an attacker will only need to guess 2^16 (65,536) values before being able to replicate the state of the PRNG.

Observed Examples

Reference	Description
CVE-2019-10908	product generates passwords via org.apache.commons.lang.RandomStringUtils, which uses java.util.Random internally. This PRNG has only a 48-bit seed. *https://www.cve.org/CVERecord?id=CVE-2019-10908*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

This entry may have a chaining relationship with predictable from observable state (CWE-341).

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Small Seed Space in PRNG

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

CWE-340: Generation of Predictable Numbers or Identifiers

Weakness ID : 340
Structure : Simple
Abstraction : Class

Description

The product uses a scheme that generates numbers or identifiers that are more predictable than required.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814
ParentOf	B	341	Predictable from Observable State	843
ParentOf	B	342	Predictable Exact Value from Previous Values	845
ParentOf	B	343	Predictable Value Range from Previous Values	847

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Demonstrative Examples

Example 1:

This code generates a unique random identifier for a user's session.

Example Language: PHP *(Bad)*

```
function generateSessionID($userID){
   srand($userID);
   return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Observed Examples

Reference	Description
CVE-2022-29330	Product for administering PBX systems uses predictable identifiers and timestamps for filenames (CWE-340) which allows attackers to access files via direct request (CWE-425). *https://www.cve.org/CVERecord?id=CVE-2022-29330*

Reference	Description
CVE-2001-1141	PRNG allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers. *https://www.cve.org/CVERecord?id=CVE-2001-1141*
CVE-1999-0074	Listening TCP ports are sequentially allocated, allowing spoofing attacks. *https://www.cve.org/CVERecord?id=CVE-1999-0074*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictability problems
WASC	11		Brute Force

References

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

CWE-341: Predictable from Observable State

Weakness ID : 341
Structure : Simple
Abstraction : Base

Description

A number or object is predictable based on observations that the attacker can make about the state of the system or network, such as time, process ID, etc.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	340	Generation of Predictable Numbers or Identifiers	842
PeerOf	V	339	Small Seed Space in PRNG	840

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *This weakness could be exploited by an attacker in a number ways depending on the context. If a predictable number is used to generate IDs or keys that are used within protection mechanisms, then an attacker could gain unauthorized access to the system. If predictable filenames are used for storing sensitive information, then an attacker might gain access to the system and may be able to gain access to the information in the file.*	

Potential Mitigations

Phase: Implementation

Increase the entropy used to seed a PRNG.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Demonstrative Examples

Example 1:

This code generates a unique random identifier for a user's session.

Example Language: PHP *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Observed Examples

Reference	Description
CVE-2002-0389	Mail server stores private mail messages with predictable filenames in a world-executable directory, which allows local users to read private mailing list archives. *https://www.cve.org/CVERecord?id=CVE-2002-0389*
CVE-2001-1141	PRNG allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers. *https://www.cve.org/CVERecord?id=CVE-2001-1141*
CVE-2000-0335	DNS resolver library uses predictable IDs, which allows a local attacker to spoof DNS query results. *https://www.cve.org/CVERecord?id=CVE-2000-0335*
CVE-2005-1636	MFV. predictable filename and insecure permissions allows file modification to execute SQL queries. *https://www.cve.org/CVERecord?id=CVE-2005-1636*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable from Observable State

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

CWE-342: Predictable Exact Value from Previous Values

Weakness ID : 342

Structure : Simple
Abstraction : Base

Description

An exact value or random number can be precisely predicted by observing previous values.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	340	Generation of Predictable Numbers or Identifiers	842

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Increase the entropy used to seed a PRNG.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Observed Examples

Reference	Description
CVE-2002-1463	Firewall generates easily predictable initial sequence numbers (ISN), which allows remote attackers to spoof connections. *https://www.cve.org/CVERecord?id=CVE-2002-1463*
CVE-1999-0074	Listening TCP ports are sequentially allocated, allowing spoofing attacks. *https://www.cve.org/CVERecord?id=CVE-1999-0074*
CVE-1999-0077	Predictable TCP sequence numbers allow spoofing. *https://www.cve.org/CVERecord?id=CVE-1999-0077*
CVE-2000-0335	DNS resolver uses predictable IDs, allowing a local user to spoof DNS query results. *https://www.cve.org/CVERecord?id=CVE-2000-0335*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable Exact Value from Previous Values

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

CWE-343: Predictable Value Range from Previous Values

Weakness ID : 343
Structure : Simple
Abstraction : Base

Description

The product's random number generator produces a series of values which, when observed, can be used to infer a relatively small range of possibilities for the next value that could be generated.

Extended Description

The output of a random number generator should not be predictable based on observations of previous values. In some cases, an attacker cannot predict the exact value that will be produced next, but can narrow down the possibilities significantly. This reduces the amount of effort to perform a brute force attack. For example, suppose the product generates random numbers between 1 and 100, but it always produces a larger value until it reaches 100. If the generator produces an 80, then the attacker knows that the next value will be somewhere between 81 and 100. Instead of 100 possibilities, the attacker only needs to consider 20.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	340	Generation of Predictable Numbers or Identifiers	842

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Increase the entropy used to seed a PRNG.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable Value Range from Previous Values

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-320]Michal Zalewski. "Strange Attractors and TCP/IP Sequence Number Analysis". 2001. < https://lcamtuf.coredump.cx/oldtcp/tcpseq.html >.2023-04-07.

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

CWE-344: Use of Invariant Value in Dynamically Changing Context

Weakness ID : 344
Structure : Simple
Abstraction : Base

Description

The product uses a constant value, name, or reference, but this value can (or should) vary across different environments.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	814
ParentOf	B	323	Reusing a Nonce, Key Pair in Encryption	790
ParentOf	V	587	Assignment of a Fixed Address to a Pointer	1322
ParentOf	B	798	Use of Hard-coded Credentials	1690

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

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2477

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Demonstrative Examples

Example 1:

The following code is an example of an internal hard-coded password in the back-end:

Example Language: C *(Bad)*

```
int VerifyAdmin(char *password) {
  if (strcmp(password, "Mew!")) {
    printf("Incorrect Password!\n");
    return(0)
  }
  printf("Entering Diagnostic Mode...\n");
  return(1);
}
```

Example Language: Java *(Bad)*

```
int VerifyAdmin(String password) {
  if (!password.equals("Mew!")) {
    return(0)
  }
  //Diagnostic Mode
  return(1);
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

Example 2:

This code assumes a particular function will always be found at a particular address. It assigns a pointer to that address and calls the function.

Example Language: C *(Bad)*

```
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

The same function may not always be found at the same memory address. This could lead to a crash, or an attacker may alter the memory at the expected address, leading to arbitrary code execution.

Observed Examples

Reference	Description
CVE-2002-0980	Component for web browser writes an error message to a known location, which can then be referenced by attackers to process HTML/script in a less restrictive context *https://www.cve.org/CVERecord?id=CVE-2002-0980*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	2388
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Relationship

overlaps default configuration.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Static Value in Unpredictable Context

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

CWE-345: Insufficient Verification of Data Authenticity

Weakness ID : 345
Structure : Simple
Abstraction : Class

Description

The product does not sufficiently verify the origin or authenticity of data, in a way that causes it to accept invalid data.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520
ParentOf	C	346	Origin Validation Error	853
ParentOf	B	347	Improper Verification of Cryptographic Signature	857
ParentOf	B	348	Use of Less Trusted Source	859
ParentOf	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	861
ParentOf	B	351	Insufficient Type Distinction	866
ParentOf	♣	352	Cross-Site Request Forgery (CSRF)	868
ParentOf	B	353	Missing Support for Integrity Check	874
ParentOf	B	354	Improper Validation of Integrity Check Value	876
ParentOf	B	360	Trust of System Event Data	887
ParentOf	B	494	Download of Code Without Integrity Check	1185
ParentOf	V	616	Incomplete Identification of Uploaded File Variables (PHP)	1376
ParentOf	V	646	Reliance on File Name or Extension of Externally-Supplied File	1425
ParentOf	B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	1430
ParentOf	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	1830
ParentOf	B	1293	Missing Source Correlation of Multiple Independent Data	2149
PeerOf	C	20	Improper Input Validation	20
PeerOf	B	1304	Improperly Preserved Integrity of Hardware Configuration State During a Power Save/Restore Operation	2176

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

Nature	Type	ID	Name	Page
ParentOf	C	346	Origin Validation Error	853
ParentOf	B	347	Improper Verification of Cryptographic Signature	857
ParentOf	♣	352	Cross-Site Request Forgery (CSRF)	868

Nature	Type	ID	Name	Page
ParentOf	B	354	Improper Validation of Integrity Check Value	876
ParentOf	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	1830

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2429

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Demonstrative Examples

Example 1:

In 2022, the OT:ICEFALL study examined products by 10 different Operational Technology (OT) vendors. The researchers reported 56 vulnerabilities and said that the products were "insecure by design" [REF-1283]. If exploited, these vulnerabilities often allowed adversaries to change how the products operated, ranging from denial of service to changing the code that the products executed. Since these products were often used in industries such as power, electrical, water, and others, there could even be safety implications.

Multiple vendors did not sign firmware images.

Observed Examples

Reference	Description
CVE-2022-30260	Distributed Control System (DCS) does not sign firmware images and only relies on insecure checksums for integrity checks *https://www.cve.org/CVERecord?id=CVE-2022-30260*
CVE-2022-30267	Distributed Control System (DCS) does not sign firmware images and only relies on insecure checksums for integrity checks *https://www.cve.org/CVERecord?id=CVE-2022-30267*
CVE-2022-30272	Remote Terminal Unit (RTU) does not use signatures for firmware images and relies on insecure checksums *https://www.cve.org/CVERecord?id=CVE-2022-30272*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2335
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2395
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Notes

Relationship

"origin validation" could fall under this.

Maintenance

The specific ways in which the origin is not properly identified should be laid out as separate weaknesses. In some sense, this is more like a category.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Verification of Data
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	12		Content Spoofing

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
111	JSON Hijacking (aka JavaScript Hijacking)
141	Cache Poisoning
142	DNS Cache Poisoning
148	Content Spoofing
218	Spoofing of UDDI/ebXML Messages
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking
665	Exploitation of Thunderbolt Protection Flaws
701	Browser in the Middle (BiTM)

References

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

[REF-1283]Forescout Vedere Labs. "OT:ICEFALL: The legacy of "insecure by design" and its implications for certifications and risk management". 2022 June 0. < https://www.forescout.com/resources/ot-icefall-report/ >.

CWE-346: Origin Validation Error

Weakness ID : 346
Structure : Simple
Abstraction : Class

Description

The product does not properly verify that the source of data or communication is valid.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680
ChildOf	C	345	Insufficient Verification of Data Authenticity	851
ParentOf	B	940	Improper Verification of Source of a Communication Channel	1842
ParentOf	V	1385	Missing Origin Validation in WebSockets	2259
PeerOf	C	451	User Interface (UI) Misrepresentation of Critical Information	1079

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2429

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2477
MemberOf	C	417	Communication Channel Errors	2325

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Gain Privileges or Assume Identity Varies by Context *An attacker can access any functionality that is inadvertently accessible to the source.*	

Demonstrative Examples

Example 1:

This Android application will remove a user account when it receives an intent to do so:

Example Language: Java *(Bad)*

```
IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
public class DeleteReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    int userID = intent.getIntExtra("userID");
    destroyUserData(userID);
  }
}
```

This application does not check the origin of the intent, thus allowing any malicious application to remove a user. Always check the origin of an intent, or create an allowlist of trusted applications using the manifest.xml file.

Example 2:

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

Example Language: Java *(Bad)*

```
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
  if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
    if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
      writeDataToView(view, UserData);
      return false;
    }
    else{
      return true;
    }
  }
}
```

Example Language: Objective-C *(Bad)*

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:
(UIWebViewNavigationType)exNavigationType
{
  NSURL *URL = [exRequest URL];
  if ([[URL scheme] isEqualToString:@"exampleScheme"])
  {
    NSString *functionString = [URL resourceSpecifier];
    if ([functionString hasPrefix:@"specialFunction"])
    {
      // Make data available back in webview.
      UIWebView *webView = [self writeDataToView:[URL query]];
    }
    return NO;
  }
  return YES;
}
```

A call into native code can then be initiated by passing parameters within the URL:

Example Language: JavaScript *(Attack)*

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

Observed Examples

Reference	Description
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://www.cve.org/CVERecord?id=CVE-2000-1218*
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://www.cve.org/CVERecord?id=CVE-2005-0877*
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers *https://www.cve.org/CVERecord?id=CVE-2001-1452*
CVE-2005-2188	user ID obtained from untrusted source (URL) *https://www.cve.org/CVERecord?id=CVE-2005-2188*
CVE-2003-0174	LDAP service does not verify if a particular attribute was set by the LDAP server *https://www.cve.org/CVERecord?id=CVE-2003-0174*
CVE-1999-1549	product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements. *https://www.cve.org/CVERecord?id=CVE-1999-1549*
CVE-2003-0981	product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS. *https://www.cve.org/CVERecord?id=CVE-2003-0981*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2395
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494
MemberOf	C	1382	ICS Operations (& Maintenance): Emerging Energy Technologies	1358	2517
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Notes

Maintenance

This entry has some significant overlap with other CWE entries and may need some clarification. See terminology notes.

Terminology

The "Origin Validation Error" term was originally used in a 1995 thesis [REF-324]. Although not formally defined, an issue is considered to be an origin validation error if either (1) "an object [accepts] input from an unauthorized subject," or (2) "the system [fails] to properly or completely authenticate a subject." A later section says that an origin validation error can occur when the system (1) "does not properly authenticate a user or process" or (2) "does not properly authenticate the shared data or libraries." The only example provided in the thesis (covered by OSVDB:57615) involves a setuid program running command-line arguments without dropping privileges. So, this definition (and its examples in the thesis) effectively cover other weaknesses such as CWE-287 (Improper Authentication), CWE-285 (Improper Authorization), and CWE-250

(Execution with Unnecessary Privileges). There appears to be little usage of this term today, except in the SecurityFocus vulnerability database, where the term is used for a variety of issues, including web-browser problems that allow violation of the Same Origin Policy and improper validation of the source of an incoming message.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Origin Validation Error
ISA/IEC 62443	Part 3-3		Req SR 2.12 RE(1)
ISA/IEC 62443	Part 4-1		Req SD-1
ISA/IEC 62443	Part 4-1		Req SR-2
ISA/IEC 62443	Part 4-1		Req SVV-1
ISA/IEC 62443	Part 4-2		Req CR 2.12 RE(1)
ISA/IEC 62443	Part 4-2		Req CR 3.1 RE(1)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
75	Manipulating Writeable Configuration Files
76	Manipulating Web Input to File System Calls
89	Pharming
111	JSON Hijacking (aka JavaScript Hijacking)
141	Cache Poisoning
142	DNS Cache Poisoning
160	Exploit Script-Based APIs
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking
510	SaaS User Request Forgery

References

[REF-324]Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995 August 1. < http://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf >.

CWE-347: Improper Verification of Cryptographic Signature

Weakness ID : 347
Structure : Simple
Abstraction : Base

Description

The product does not verify, or incorrectly verifies, the cryptographic signature for data.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
ChildOf	●	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2477
MemberOf	C	310	Cryptographic Issues	2318

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity Confidentiality	Gain Privileges or Assume Identity Modify Application Data Execute Unauthorized Code or Commands *An attacker could gain access to sensitive data and possibly execute unauthorized code.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Demonstrative Examples

Example 1:

In the following code, a JarFile object is created from a downloaded file.

Example Language: Java *(Bad)*

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

The JAR file that was potentially downloaded from an untrusted source is created without verifying the signature (if present). An alternate constructor that accepts a boolean verify parameter should be used instead.

Observed Examples

Reference	Description
CVE-2002-1796	Does not properly verify signatures for "trusted" entities. *https://www.cve.org/CVERecord?id=CVE-2002-1796*

Reference	Description
CVE-2005-2181	Insufficient verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2005-2181*
CVE-2005-2182	Insufficient verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2005-2182*
CVE-2002-1706	Accepts a configuration file without a Message Integrity Check (MIC) signature. *https://www.cve.org/CVERecord?id=CVE-2002-1706*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2369
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	2398
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1402	Comprehensive Categorization: Encryption	1400	2527

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Verified Signature
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar
ISA/IEC 62443	Part 3-3		Req SR 1.9
ISA/IEC 62443	Part 4-1		Req SM-6
ISA/IEC 62443	Part 4-2		Req EDR 3.12
ISA/IEC 62443	Part 4-2		Req NDR 3.12
ISA/IEC 62443	Part 4-2		Req HDR 3.12

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack
475	Signature Spoofing by Improper Validation

CWE-348: Use of Less Trusted Source

Weakness ID : 348
Structure : Simple
Abstraction : Base

Description

The product has two different sources of the same data or information, but it uses the source that has less support for verification, is less trusted, or is less resistant to attack.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2477

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *An attacker could utilize the untrusted data source to bypass protection mechanisms and gain access to sensitive data.*	

Demonstrative Examples

Example 1:

This code attempts to limit the access of a page to certain IP Addresses. It checks the 'HTTP_X_FORWARDED_FOR' header in case an authorized user is sending the request through a proxy.

Example Language: PHP *(Bad)*

```
$requestingIP = '0.0.0.0';
if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)) {
  $requestingIP = $_SERVER['HTTP_X_FORWARDED_FOR'];
else{
  $requestingIP = $_SERVER['REMOTE_ADDR'];
}
if(in_array($requestingIP,$ipAllowlist)){
  generatePage();
  return;
}
else{
  echo "You are not authorized to view this page";
  return;
}
```

The 'HTTP_X_FORWARDED_FOR' header can be user controlled and so should never be trusted. An attacker can falsify the header to gain access to the page.

This fixed code only trusts the 'REMOTE_ADDR' header and so avoids the issue:

Example Language: PHP *(Good)*

```
$requestingIP = '0.0.0.0';
if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)) {
  echo "This application cannot be accessed through a proxy.";
  return;
else{
  $requestingIP = $_SERVER['REMOTE_ADDR'];
}
...
```

Be aware that 'REMOTE_ADDR' can still be spoofed. This may seem useless because the server will send the response to the fake address and not the attacker, but this may still be enough to conduct an attack. For example, if the generatePage() function in this code is resource intensive, an attacker could flood the server with fake requests using an authorized IP and consume significant resources. This could be a serious DoS attack even though the attacker would never see the page's sensitive content.

Observed Examples

Reference	Description
CVE-2001-0860	Product uses IP address provided by a client, instead of obtaining it from the packet headers, allowing easier spoofing. *https://www.cve.org/CVERecord?id=CVE-2001-0860*
CVE-2004-1950	Web product uses the IP address in the X-Forwarded-For HTTP header instead of a server variable that uses the connecting IP address, allowing filter bypass. *https://www.cve.org/CVERecord?id=CVE-2004-1950*
CVE-2001-0908	Product logs IP address specified by the client instead of obtaining it from the packet headers, allowing information hiding. *https://www.cve.org/CVERecord?id=CVE-2001-0908*
CVE-2006-1126	PHP application uses IP address from X-Forwarded-For HTTP header, instead of REMOTE_ADDR. *https://www.cve.org/CVERecord?id=CVE-2006-1126*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2406
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Use of Less Trusted Source

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
76	Manipulating Web Input to File System Calls
85	AJAX Footprinting
141	Cache Poisoning
142	DNS Cache Poisoning

CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data

Weakness ID : 349
Structure : Simple
Abstraction : Base

Description

The product, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were trusted.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2477

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity	Bypass Protection Mechanism Modify Application Data *An attacker could package untrusted data with trusted data to bypass protection mechanisms to gain access to and possibly modify sensitive data.*	

Observed Examples

Reference	Description
CVE-2002-0018	Does not verify that trusted entity is authoritative for all entities in its response. *https://www.cve.org/CVERecord?id=CVE-2002-0018*
CVE-2006-5462	use of extra data in a signature allows certificate signature forging *https://www.cve.org/CVERecord?id=CVE-2006-5462*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	860	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)	844	2370
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1150	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)	1133	2452
MemberOf	C	1365	ICS Communications: Unreliability	1358	2502
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1373	ICS Engineering (Construction/Deployment): Trust Model Problems	1358	2510
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Untrusted Data Appended with Trusted Data
The CERT Oracle Secure Coding Standard for Java (2011)	ENV01-J		Place all security-sensitive code in a single JAR and sign and seal it

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
75	Manipulating Writeable Configuration Files
141	Cache Poisoning
142	DNS Cache Poisoning

CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action

Weakness ID : 350
Structure : Simple
Abstraction : Variant

Description

The product performs reverse DNS resolution on an IP address to obtain the hostname and make a security decision, but it does not properly ensure that the IP address is truly associated with the hostname.

Extended Description

Since DNS names can be easily spoofed or misreported, and it may be difficult for the product to detect if a trusted DNS server has been compromised, DNS names do not constitute a valid authentication mechanism.

When the product performs a reverse DNS resolution for an IP address, if an attacker controls the DNS server for that IP address, then the attacker can cause the server to return an arbitrary hostname. As a result, the attacker may be able to bypass authentication, cause the wrong hostname to be recorded in log files to hide activities, or perform other attacks.

Attackers can spoof DNS names by either (1) compromising a DNS server and modifying its records (sometimes called DNS cache poisoning), or (2) having legitimate control over a DNS server associated with their IP address.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1714
ChildOf	B	290	Authentication Bypass by Spoofing	705
CanPrecede	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1827

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *Malicious users can fake authentication information by providing false DNS information.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

Phase: Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

Demonstrative Examples

Example 1:

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

Example Language: C *(Bad)*

```
struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
  trusted = true;
} else {
  trusted = false;
}
```

Example Language: Java *(Bad)*

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
  trusted = true;
}
```

Example Language: C# *(Bad)*

```
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
  trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Example 2:

In these examples, a connection is established if a request is made by a trusted host.

Example Language: C *(Bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  h=gethostbyname(inet_ntoa(cliAddr.sin_addr));
  if (h->h_name==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Example Language: Java *(Bad)*

```
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);
  }
}
```

These examples check if a request is from a trusted host before responding to a request, but the code only verifies the hostname as stored in the request packet. An attacker can spoof the hostname, thus impersonating a trusted client.

Observed Examples

Reference	Description
CVE-2001-1488	Does not do double-reverse lookup to prevent DNS spoofing. *https://www.cve.org/CVERecord?id=CVE-2001-1488*
CVE-2001-1500	Does not verify reverse-resolved hostnames in DNS. *https://www.cve.org/CVERecord?id=CVE-2001-1500*
CVE-2000-1221	Authentication bypass using spoofed reverse-resolved DNS hostnames. *https://www.cve.org/CVERecord?id=CVE-2000-1221*
CVE-2002-0804	Authentication bypass using spoofed reverse-resolved DNS hostnames. *https://www.cve.org/CVERecord?id=CVE-2002-0804*
CVE-2001-1155	Filter does not properly check the result of a reverse DNS lookup, which could allow remote attackers to bypass intended access restrictions via DNS spoofing. *https://www.cve.org/CVERecord?id=CVE-2001-1155*
CVE-2004-0892	Reverse DNS lookup used to spoof trusted content in intermediary.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2004-0892
CVE-2003-0981	Product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS. *https://www.cve.org/CVERecord?id=CVE-2003-0981*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2395
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

CWE-350, CWE-247, and CWE-292 were merged into CWE-350 in CWE 2.5. CWE-247 was originally derived from Seven Pernicious Kingdoms, CWE-350 from PLOVER, and CWE-292 from CLASP. All taxonomies focused closely on the use of reverse DNS for authentication of incoming requests.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Trusted Reverse DNS
CLASP			Trusting self-reported DNS name
Software Fault Patterns	SFP29		Faulty endpoint authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
89	Pharming
142	DNS Cache Poisoning
275	DNS Rebinding

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

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

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

CWE-351: Insufficient Type Distinction

Weakness ID : 351
Structure : Simple
Abstraction : Base

Description

The product does not properly distinguish between different types of elements in a way that leads to insecure behavior.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851
PeerOf	C	436	Interpretation Conflict	1057
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	1048

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2477

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Observed Examples

Reference	Description
CVE-2005-2260	Browser user interface does not distinguish between user-initiated and synthetic events. *https://www.cve.org/CVERecord?id=CVE-2005-2260*
CVE-2005-2801	Product does not compare all required data in two separate elements, causing it to think they are the same, leading to loss of ACLs. Similar to Same Name error. *https://www.cve.org/CVERecord?id=CVE-2005-2801*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	2417
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Notes

Relationship

Overlaps others, e.g. Multiple Interpretation Errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Type Distinction

CWE-352: Cross-Site Request Forgery (CSRF)

Weakness ID : 352
Structure : Composite
Abstraction : Compound

Description

The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

Composite Components

Nature	Type	ID	Name	Page
Requires	C	346	Origin Validation Error	853
Requires	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1064
Requires	C	642	External Control of Critical State Data	1414
Requires	B	613	Insufficient Session Expiration	1371

Extended Description

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851
PeerOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163
CanFollow	V	1275	Sensitive Cookie with Improper SameSite Attribute	2110

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Applicable Platforms

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

Technology : Web Server *(Prevalence = Undetermined)*

Alternate Terms

Session Riding :

Cross Site Reference Forgery :

XSRF :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism Read Application Data Modify Application Data DoS: Crash, Exit, or Restart *The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. An attacker could effectively perform any operations as the victim. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product, or using it to launch other attacks against all of the product's users. Because the attacker has the identity of the victim, the scope of CSRF is limited only by the victim's privileges.*	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual analysis can be useful for finding this weakness, and for minimizing false positives assuming an understanding of business logic. However, it might not achieve desired code coverage within limited time constraints. For black-box analysis, if credentials are not known for privileged accounts, then the most security-critical portions of the application may not receive sufficient attention. Consider using OWASP CSRFTester to identify potential issues and aid in manual analysis.

Effectiveness = High

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis

CSRF is currently difficult to detect reliably using automated techniques. This is because each application has its own implicit security policy that dictates which requests can be influenced by an outsider and automatically performed on behalf of a user, versus which requests require strong confidence that the user intends to make the request. For example, a keyword search of the public portion of a web site is typically expected to be encoded within a link that can be launched automatically when the user clicks on the link.

Effectiveness = Limited

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = High

Manual Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = SOAR Partial

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. [REF-330] Another example is the ESAPI Session Management control, which includes a component for CSRF. [REF-45]

Phase: Implementation

Ensure that the application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script.

Phase: Architecture and Design

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). [REF-332]

Phase: Architecture and Design

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Phase: Architecture and Design

Use the "double-submitted cookie" method as described by Felten and Zeller: When a user visits a site, the site should generate a pseudorandom value and set it as a cookie on the user's machine. The site should require every form submission to include this value as a form value and also as a cookie value. When a POST request is sent to the site, the request should only be considered valid if the form value and the cookie value are the same. Because of the same-origin policy, an attacker cannot read or modify the value stored in the cookie. To successfully submit a form on behalf of the user, the attacker would have to correctly guess the pseudorandom value. If the pseudorandom value is cryptographically strong, this will be prohibitively difficult. This technique requires Javascript, so it may not work for browsers that have Javascript disabled. [REF-331]

Phase: Architecture and Design

Do not use the GET method for any request that triggers a state change.

Phase: Implementation

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

Demonstrative Examples

Example 1:

This example PHP code attempts to secure the form submission process by validating that the user submitting the form has a valid session. A CSRF attack would not be prevented by this countermeasure because the attacker forges a request through the user's web browser in which a valid session already exists.

The following HTML is intended to allow a user to update a profile.

Example Language: HTML *(Bad)*

```
<form action="/url/profile.php" method="post">
<input type="text" name="firstname"/>
<input type="text" name="lastname"/>
<br/>
<input type="text" name="email"/>
<input type="submit" name="submit" value="Update"/>
</form>
```

profile.php contains the following code.

Example Language: PHP *(Bad)*

```
// initiate the session in order to validate sessions
session_start();
//if the session is registered to a valid user then allow update
if (! session_is_registered("username")) {
  echo "invalid session detected!";
  // Redirect user to login page
  [...]
  exit;
}
// The user session is valid, so process the request
// and update the information
update_profile();
function update_profile {
  // read in the data from $POST and send an update
  // to the database
  SendUpdateToDatabase($_SESSION['username'], $_POST['email']);
  [...]
  echo "Your profile has been successfully updated.";
}
```

This code may look protected since it checks for a valid session. However, CSRF attacks can be staged from virtually any tag or HTML construct, including image tags, links, embed or object tags, or other attributes that load background images.

The attacker can then host code that will silently change the username and email address of any user that visits the page while remaining logged in to the target web application. The code might be an innocent-looking web page such as:

Example Language: HTML *(Attack)*

```
<SCRIPT>
function SendAttack () {
  form.email = "attacker@example.com";
  // send to profile.php
  form.submit();
}
</SCRIPT>
<BODY onload="javascript:SendAttack();">
<form action="http://victim.example.com/profile.php" id="form" method="post">
<input type="hidden" name="firstname" value="Funny">
<input type="hidden" name="lastname" value="Joke">
<br/>
<input type="hidden" name="email">
</form>
```

Notice how the form contains hidden fields, so when it is loaded into the browser, the user will not notice it. Because SendAttack() is defined in the body's onload attribute, it will be automatically called when the victim loads the web page.

Assuming that the user is already logged in to victim.example.com, profile.php will see that a valid user session has been established, then update the email address to the attacker's own address. At this stage, the user's identity has been compromised, and messages sent through this profile could be sent to the attacker's address.

Observed Examples

Reference	Description
CVE-2004-1703	Add user accounts via a URL in an img tag *https://www.cve.org/CVERecord?id=CVE-2004-1703*
CVE-2004-1995	Add user accounts via a URL in an img tag *https://www.cve.org/CVERecord?id=CVE-2004-1995*
CVE-2004-1967	Arbitrary code execution by specifying the code in a crafted img tag or URL *https://www.cve.org/CVERecord?id=CVE-2004-1967*
CVE-2004-1842	Gain administrative privileges via a URL in an img tag *https://www.cve.org/CVERecord?id=CVE-2004-1842*
CVE-2005-1947	Delete a victim's information via a URL or an img tag *https://www.cve.org/CVERecord?id=CVE-2005-1947*
CVE-2005-2059	Change another user's settings via a URL or an img tag *https://www.cve.org/CVERecord?id=CVE-2005-2059*
CVE-2005-1674	Perform actions as administrator via a URL or an img tag *https://www.cve.org/CVERecord?id=CVE-2005-1674*
CVE-2009-3520	modify password for the administrator *https://www.cve.org/CVERecord?id=CVE-2009-3520*
CVE-2009-3022	CMS allows modification of configuration via CSRF attack against the administrator *https://www.cve.org/CVERecord?id=CVE-2009-3022*
CVE-2009-3759	web interface allows password changes or stopping a virtual machine via CSRF *https://www.cve.org/CVERecord?id=CVE-2009-3759*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	629	2331
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2352
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2354
MemberOf	C	814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)	809	2358
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2371
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	936	OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)	928	2392
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Relationship

There can be a close relationship between XSS and CSRF (CWE-352). An attacker might use CSRF in order to trick the victim into submitting requests to the server in which the requests contain an XSS payload. A well-known example of this was the Samy worm on MySpace [REF-956]. The worm used XSS to insert malicious HTML sequences into a user's profile and add the attacker as a MySpace friend. MySpace friends of that victim would then execute the payload to modify their own profiles, causing the worm to propagate exponentially. Since the victims did not intentionally insert the malicious script themselves, CSRF was a root cause.

Theoretical

The CSRF topology is multi-channel: Attacker (as outsider) to intermediary (as user). The interaction point is either an external or internal channel. Intermediary (as user) to server (as victim). The activation point is an internal channel.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-Site Request Forgery (CSRF)
OWASP Top Ten 2007	A5	Exact	Cross Site Request Forgery (CSRF)
WASC	9		Cross-site Request Forgery

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
62	Cross Site Request Forgery

CAPEC-ID	Attack Pattern Name
111	JSON Hijacking (aka JavaScript Hijacking)
462	Cross-Domain Search Timing
467	Cross Site Identification

References

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

[REF-329]Peter W. "Cross-Site Request Forgeries (Re: The Dangers of Allowing Users to Post Images)". Bugtraq. < http://marc.info/?l=bugtraq&m=99263135911884&w=2 >.

[REF-330]OWASP. "Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet". < http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet >.

[REF-331]Edward W. Felten and William Zeller. "Cross-Site Request Forgeries: Exploitation and Prevention". 2008 October 8. < https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.147.1445 >.2023-04-07.

[REF-332]Robert Auger. "CSRF - The Cross-Site Request Forgery (CSRF/XSRF) FAQ". < https://www.cgisecurity.com/csrf-faq.html >.2023-04-07.

[REF-333]"Cross-site request forgery". 2008 December 2. Wikipedia. < https://en.wikipedia.org/wiki/Cross-site_request_forgery >.2023-04-07.

[REF-334]Jason Lam. "Top 25 Series - Rank 4 - Cross Site Request Forgery". 2010 March 3. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/03/top-25-series-rank-4-cross-site-request-forgery >.

[REF-335]Jeff Atwood. "Preventing CSRF and XSRF Attacks". 2008 October 4. < https://blog.codinghorror.com/preventing-csrf-and-xsrf-attacks/ >.2023-04-07.

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

[REF-956]Wikipedia. "Samy (computer worm)". < https://en.wikipedia.org/wiki/Samy_(computer_worm) >.2018-01-16.

CWE-353: Missing Support for Integrity Check

Weakness ID : 353
Structure : Simple
Abstraction : Base

Description

The product uses a transmission protocol that does not include a mechanism for verifying the integrity of the data during transmission, such as a checksum.

Extended Description

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission. The lack of checksum functionality in a protocol removes the first application-level check of data that can be used. The end-to-end philosophy of checks states that integrity checks should be performed at the lowest level that they can be completely implemented. Excluding further sanity checks and input validation performed by applications, the protocol's checksum is the most important level of checksum, since it can be performed more completely than at any previous level and takes into account entire messages, as opposed to single packets.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851
PeerOf	B	354	Improper Validation of Integrity Check Value	876
PeerOf	B	354	Improper Validation of Integrity Check Value	876

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

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	2434

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2477

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Other *Data that is parsed and used may be corrupted.*	
Non-Repudiation Other	Hide Activities Other *Without a checksum it is impossible to determine if any changes have been made to the data after it was sent.*	

Potential Mitigations

Phase: Architecture and Design

Add an appropriately sized checksum to the protocol, ensuring that data received may be simply validated before it is parsed and used.

Phase: Implementation

Ensure that the checksums present in the protocol design are properly implemented and added to each message before it is sent.

Demonstrative Examples

Example 1:

In this example, a request packet is received, and privileged information is sent to the requester:

Example Language: Java *(Bad)*

```
while(true) {
  DatagramPacket rp = new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  InetAddress IPAddress = rp.getAddress();
  int port = rp.getPort();
  out = secret.getBytes();
  DatagramPacket sp =new DatagramPacket(out, out.length, IPAddress, port);
```

```
    outSock.send(sp);
}
```

The response containing secret data has no integrity check associated with it, allowing an attacker to alter the message without detection.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	2398
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to add integrity check value
ISA/IEC 62443	Part 2-4		Req SP.03.03 RE(1)
ISA/IEC 62443	Part 2-4		Req SP.04.02 RE(1)
ISA/IEC 62443	Part 2-4		Req SP.11.06 RE(2)
ISA/IEC 62443	Part 3-3		Req SR 3.1
ISA/IEC 62443	Part 4-1		Req SD-1
ISA/IEC 62443	Part 4-2		Req CR 3.1

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
39	Manipulating Opaque Client-based Data Tokens
74	Manipulating State
75	Manipulating Writeable Configuration Files
389	Content Spoofing Via Application API Manipulation
665	Exploitation of Thunderbolt Protection Flaws

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

CWE-354: Improper Validation of Integrity Check Value

Weakness ID : 354
Structure : Simple
Abstraction : Base

Description

The product does not validate or incorrectly validates the integrity check values or "checksums" of a message. This may prevent it from detecting if the data has been modified or corrupted in transmission.

Extended Description

Improper validation of checksums before use results in an unnecessary risk that can easily be mitigated. The protocol specification describes the algorithm used for calculating the checksum. It is then a simple matter of implementing the calculation and verifying that the calculated checksum and the received checksum match. Improper verification of the calculated checksum and the received checksum can lead to far greater consequences.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	754	Improper Check for Unusual or Exceptional Conditions	1568
ChildOf	Ⓒ	345	Insufficient Verification of Data Authenticity	851
PeerOf	Ⓑ	353	Missing Support for Integrity Check	874
PeerOf	Ⓑ	353	Missing Support for Integrity Check	874

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	🅲	1020	Verify Message Integrity	2434

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

Nature	Type	ID	Name	Page
MemberOf	🅲	1214	Data Integrity Issues	2477

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Modify Application Data Other *Integrity checks usually use a secret key that helps authenticate the data origin. Skipping integrity checking generally opens up the possibility that new data from an invalid source can be injected.*	
Integrity Other	Other *Data that is parsed and used may be corrupted.*	
Non-Repudiation Other	Hide Activities Other	

Scope	Impact	Likelihood
	Without a checksum check, it is impossible to determine if any changes have been made to the data after it was sent.	

Potential Mitigations

Phase: Implementation

Ensure that the checksums present in messages are properly checked in accordance with the protocol specification before they are parsed and used.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0); serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Example Language: Java *(Bad)*

```
while(true) {
  DatagramPacket packet = new DatagramPacket(data,data.length,IPAddress, port);
  socket.send(sendPacket);
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	2417
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 3-3		Req SR 3.1
CLASP			Failure to check integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
75	Manipulating Writeable Configuration Files
145	Checksum Spoofing
463	Padding Oracle Crypto Attack

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-356: Product UI does not Warn User of Unsafe Actions

Weakness ID : 356
Structure : Simple
Abstraction : Base

Description

The product's user interface does not warn the user before undertaking an unsafe action on behalf of that user. This makes it easier for attackers to trick users into inflicting damage to their system.

Extended Description

Product systems should warn users that a potentially dangerous action may occur if the user proceeds. For example, if the user downloads a file from an unknown source and attempts to execute the file on their machine, then the application's GUI can indicate that the file is unsafe.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	221	Information Loss or Omission	556

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

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	2320

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	

Observed Examples

Reference	Description
CVE-1999-1055	Product does not warn user when document contains certain dangerous functions or macros. *https://www.cve.org/CVERecord?id=CVE-1999-1055*
CVE-1999-0794	Product does not warn user when document contains certain dangerous functions or macros. *https://www.cve.org/CVERecord?id=CVE-1999-0794*
CVE-2000-0277	Product does not warn user when document contains certain dangerous functions or macros. *https://www.cve.org/CVERecord?id=CVE-2000-0277*
CVE-2000-0517	Product does not warn user about a certificate if it has already been accepted for a different site. Possibly resultant. *https://www.cve.org/CVERecord?id=CVE-2000-0517*
CVE-2005-0602	File extractor does not warn user if setuid/setgid files could be extracted. Overlaps privileges/permissions. *https://www.cve.org/CVERecord?id=CVE-2005-0602*
CVE-2000-0342	E-mail client allows bypass of warning for dangerous attachments via a Windows .LNK file that refers to the attachment. *https://www.cve.org/CVERecord?id=CVE-2000-0342*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	2418
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Notes

Relationship

Often resultant, e.g. in unhandled error conditions.

Relationship

Can overlap privilege errors, conceptually at least.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Product UI does not warn user of unsafe actions

CWE-357: Insufficient UI Warning of Dangerous Operations

Weakness ID : 357
Structure : Simple
Abstraction : Base

Description

The user interface provides a warning to a user regarding dangerous or sensitive operations, but the warning is not noticeable enough to warrant attention.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520
ParentOf	B	450	Multiple Interpretations of UI Input	1078

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

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	2320

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	

Observed Examples

Reference	Description
CVE-2007-1099	User not sufficiently warned if host key mismatch occurs *https://www.cve.org/CVERecord?id=CVE-2007-1099*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	2418
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient UI warning of dangerous operations

CWE-358: Improperly Implemented Security Check for Standard

Weakness ID : 358
Structure : Simple
Abstraction : Base

Description

The product does not implement or incorrectly implements one or more security-relevant checks as specified by the design of a standardized algorithm, protocol, or technique.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520
ChildOf	C	573	Improper Following of Specification by Caller	1298
PeerOf	B	325	Missing Cryptographic Step	794
CanAlsoBe	B	290	Authentication Bypass by Spoofing	705
CanAlsoBe	C	345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2002-0862	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates. *https://www.cve.org/CVERecord?id=CVE-2002-0862*
CVE-2002-0970	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates. *https://www.cve.org/CVERecord?id=CVE-2002-0970*
CVE-2002-1407	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates. *https://www.cve.org/CVERecord?id=CVE-2002-1407*
CVE-2005-0198	Logic error prevents some required conditions from being enforced during Challenge-Response Authentication Mechanism with MD5 (CRAM-MD5). *https://www.cve.org/CVERecord?id=CVE-2005-0198*
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies. *https://www.cve.org/CVERecord?id=CVE-2004-2163*
CVE-2005-2181	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages. *https://www.cve.org/CVERecord?id=CVE-2005-2181*
CVE-2005-2182	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages. *https://www.cve.org/CVERecord?id=CVE-2005-2182*
CVE-2005-2298	Security check not applied to all components, allowing bypass. *https://www.cve.org/CVERecord?id=CVE-2005-2298*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	978	SFP Secondary Cluster: Implementation	888	2408
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Relationship

This is a "missing step" error on the product side, which can overlap weaknesses such as insufficient verification and spoofing. It is frequently found in cryptographic and authentication errors. It is sometimes resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Implemented Security Check for Standard

CWE-359: Exposure of Private Personal Information to an Unauthorized Actor

Weakness ID : 359
Structure : Simple
Abstraction : Base

Description

The product does not properly prevent a person's private, personal information from being accessed by actors who either (1) are not explicitly authorized to access the information or (2) do not have the implicit consent of the person about whom the information is collected.

Extended Description

There are many types of sensitive information that products must protect from attackers, including system data, communications, configuration, business secrets, intellectual property, and an individual's personal (private) information. Private personal information may include a password, phone number, geographic location, personal messages, credit card number, etc. Private information is important to consider whether the person is a user of the product, or part of a data set that is processed by the product. An exposure of private information does not necessarily prevent the product from working properly, and in fact the exposure might be intended by the developer, e.g. as part of data sharing with other organizations. However, the exposure of personal private information can still be undesirable or explicitly prohibited by law or regulation.

Some types of private information include:

- Government identifiers, such as Social Security Numbers
- Contact information, such as home addresses and telephone numbers
- Geographic location - where the user is (or was)
- Employment history
- Financial data - such as credit card numbers, salary, bank accounts, and debts
- Pictures, video, or audio
- Behavioral patterns - such as web surfing history, when certain activities are performed, etc.
- Relationships (and types of relationships) with others - family, friends, contacts, etc.
- Communications - e-mail addresses, private messages, text messages, chat logs, etc.
- Health - medical conditions, insurance status, prescription records
- Account passwords and other credentials

Some of this information may be characterized as PII (Personally Identifiable Information), Protected Health Information (PHI), etc. Categories of private information may overlap or vary based on the intended usage or the policies and practices of a particular industry.

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.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	200	Exposure of Sensitive Information to an Unauthorized Actor	504

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	2312

Applicable Platforms

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

Technology : Mobile *(Prevalence = Undetermined)*

Alternate Terms

Privacy violation :

Privacy leak :

Privacy leakage :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Architecture or Design Review

Private personal 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 If the data is written to an external location - such as the console, file system, or network - a privacy violation may occur.

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Requirements

Identify and consult all relevant regulations for personal privacy. An organization may be required to comply with certain federal and state regulations, depending on its location, the type of business it conducts, and the nature of any private data it handles. Regulations may include Safe Harbor Privacy Framework [REF-340], Gramm-Leach Bliley Act (GLBA) [REF-341], Health Insurance Portability and Accountability Act (HIPAA) [REF-342], General Data Protection Regulation (GDPR) [REF-1047], California Consumer Privacy Act (CCPA) [REF-1048], and others.

Phase: Architecture and Design

Carefully evaluate how secure design may interfere with privacy, and vice versa. Security and privacy concerns often seem to compete with each other. From a security perspective, all important operations should be recorded so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk. Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted.

Demonstrative Examples

Example 1:

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.

Example Language: C# *(Bad)*

```
pass = GetPassword();
...
dbmsLog.WriteLine(id + ":" + pass + ":" + type + ":" + tstamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Example 2:

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

Example Language: XML *(Bad)*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Example Language: Java *(Bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Example 3:

In 2004, an employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [REF-338]. In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	2314
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2406

Nature	Type	ID	Name	V	Page
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	2436
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

Notes

Maintenance

This entry overlaps many other entries that are not organized around the kind of sensitive information that is exposed. However, because privacy is treated with such importance due to regulations and other factors, and it may be useful for weakness-finding tools to highlight capabilities that detect personal private information instead of system information, it is not clear whether - and how - this entry should be deprecated.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Privacy Violation
The CERT Oracle Secure Coding Standard for Java (2011)	FIO13-J		Do not log sensitive information outside a trust boundary

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
464	Evercookie
467	Cross Site Identification
498	Probe iOS Screenshots
508	Shoulder Surfing

References

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

[REF-338]J. Oates. "AOL man pleads guilty to selling 92m email addies". The Register. 2005. < https://www.theregister.com/2005/02/07/aol_email_theft/ >.2023-04-07.

[REF-339]NIST. "Guide to Protecting the Confidentiality of Personally Identifiable Information (SP 800-122)". 2010 April. < https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf >.2023-04-07.

[REF-340]U.S. Department of Commerce. "Safe Harbor Privacy Framework". < https://web.archive.org/web/20010223203241/http://www.export.gov/safeharbor/ >.2023-04-07.

[REF-341]Federal Trade Commission. "Financial Privacy: The Gramm-Leach Bliley Act (GLBA)". < https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act >.2023-04-07.

[REF-342]U.S. Department of Human Services. "Health Insurance Portability and Accountability Act (HIPAA)". < https://www.hhs.gov/hipaa/index.html >.2023-04-07.

[REF-343]Government of the State of California. "California SB-1386". 2002. < http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html >.

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < https://www.veracode.com/blog/2010/12/mobile-app-top-10-list >.2023-04-07.

[REF-1047]Wikipedia. "General Data Protection Regulation". < https://en.wikipedia.org/wiki/General_Data_Protection_Regulation >.

[REF-1048]State of California Department of Justice, Office of the Attorney General. "California Consumer Privacy Act (CCPA)". < https://oag.ca.gov/privacy/ccpa >.

CWE-360: Trust of System Event Data

Weakness ID : 360
Structure : Simple
Abstraction : Base

Description

Security based on event locations are insecure and can be spoofed.

Extended Description

Events are a messaging system which may provide control data to programs listening for events. Events often do not have any type of authentication framework to allow them to be verified from a trusted source. Any application, in Windows, on a given desktop can send a message to any window on the same desktop. There is no authentication framework for these messages. Therefore, any message can be used to manipulate any process on the desktop if the process does not check the validity and safeness of those messages.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	851
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	1022

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Gain Privileges or Assume Identity Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
	If one trusts the system-event information and executes commands based on it, one could potentially take actions based on a spoofed identity.	

Potential Mitigations

Phase: Architecture and Design

Never trust or rely any of the information in an Event for security.

Demonstrative Examples

Example 1:

This example code prints out secret information when an authorized user activates a button:

Example Language: Java *(Bad)*

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == button) {
    System.out.println("print out secret information");
  }
}
```

This code does not attempt to prevent unauthorized users from activating the button. Even if the button is rendered non-functional to unauthorized users in the application UI, an attacker can easily send a false button press event to the application window and expose the secret information.

Observed Examples

Reference	Description
CVE-2004-0213	Attacker uses Shatter attack to bypass GUI-enforced protection for CVE-2003-0908. *https://www.cve.org/CVERecord?id=CVE-2004-0213*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2395
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Trust of system event data
Software Fault Patterns	SFP29		Faulty endpoint authentication

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Weakness ID : 362
Structure : Simple
Abstraction : Class

Description

The product contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently.

Extended Description

This can have security implications when the expected synchronization is in security-critical code, such as recording whether a user is authenticated or modifying important state information that should not be influenced by an outsider.

A race condition occurs within concurrent environments, and is effectively a property of a code sequence. Depending on the context, a code sequence may be in the form of a function call, a small number of instructions, a series of program invocations, etc.

A race condition violates these properties, which are closely related:

- Exclusivity - the code sequence is given exclusive access to the shared resource, i.e., no other code sequence can modify properties of the shared resource before the original sequence has completed execution.
- Atomicity - the code sequence is behaviorally atomic, i.e., no other thread or process can concurrently execute the same sequence of instructions (or a subset) against the same resource.

A race condition exists when an "interfering code sequence" can still access the shared resource, violating exclusivity. Programmers may assume that certain code sequences execute too quickly to be affected by an interfering code sequence; when they are not, this violates atomicity. For example, the single "x++" statement may appear atomic at the code layer, but it is actually non-atomic at the instruction layer, since it involves a read (the original value of x), followed by a computation (x+1), followed by a write (save the result to x).

The interfering code sequence could be "trusted" or "untrusted." A trusted interfering code sequence occurs within the product; it cannot be modified by the attacker, and it can only be invoked indirectly. An untrusted interfering code sequence can be authored directly by the attacker, and typically it is external to the vulnerable product.

Relationships

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

<u>Relevant to the view "Research Concepts" (CWE-1000)</u>

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517
ParentOf	B	364	Signal Handler Race Condition	899
ParentOf	B	366	Race Condition within a Thread	904
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	906
ParentOf	B	368	Context Switching Race Condition	912
ParentOf	B	421	Race Condition During Access to Alternate Channel	1020
ParentOf	♣	689	Permission Race Condition During Resource Copy	1513
ParentOf	B	1223	Race Condition for Write-Once Attributes	2001
ParentOf	B	1298	Hardware Logic Contains Race Conditions	2158
CanFollow	C	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	906

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Java *(Prevalence = Sometimes)*

Technology : Mobile *(Prevalence = Undetermined)*

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *When a race condition makes it possible to bypass a resource cleanup routine or trigger multiple initialization routines, it may lead to resource exhaustion (CWE-400).*	
Availability	DoS: Crash, Exit, or Restart DoS: Instability *When a race condition allows multiple control flows to access a resource simultaneously, it might lead the product(s) into unexpected states, possibly resulting in a crash.*	
Confidentiality Integrity	Read Files or Directories Read Application Data *When a race condition is combined with predictable resource names and loose permissions, it may be possible for an attacker to overwrite or access confidential data (CWE-59).*	

Detection Methods

Black Box

Black box methods may be able to identify evidence of race conditions via methods such as multiple simultaneous connections, which may cause the software to become instable or crash. However, race conditions with very narrow timing windows would not be detectable.

White Box

Common idioms are detectable in white box analysis, such as time-of-check-time-of-use (TOCTOU) file operations (CWE-367), or double-checked locking (CWE-609).

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Race conditions may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. Insert breakpoints or delays in between relevant code statements to artificially expand the race window so that it will be easier to detect.

Effectiveness = Moderate

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = High

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

In languages that support it, use synchronization primitives. Only wrap these around critical code to minimize the impact on performance.

Phase: Architecture and Design

Use thread-safe capabilities such as the data access abstraction in Spring.

Phase: Architecture and Design

Minimize the usage of shared resources in order to remove as much complexity as possible from the control flow and to reduce the likelihood of unexpected conditions occurring. Additionally, this will minimize the amount of synchronization necessary and may even help to reduce the likelihood of a denial of service where an attacker may be able to repeatedly trigger a critical section (CWE-400).

Phase: Implementation

When using multithreading and operating on shared variables, only use thread-safe functions.

Phase: Implementation

Use atomic operations on shared variables. Be wary of innocent-looking constructs such as "x++". This may appear atomic at the code layer, but it is actually non-atomic at the instruction layer, since it involves a read, followed by a computation, followed by a write.

Phase: Implementation

Use a mutex if available, but be sure to avoid related weaknesses such as CWE-412.

Phase: Implementation

Avoid double-checked locking (CWE-609) and other implementation errors that arise when trying to avoid the overhead of synchronization.

Phase: Implementation

Disable interrupts or signals over critical parts of the code, but also make sure that the code does not go into a large or infinite loop.

Phase: Implementation

Use the volatile type modifier for critical variables to avoid unexpected compiler optimization or reordering. This does not necessarily solve the synchronization problem, but it can help.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Demonstrative Examples

Example 1:

This code could be used in an e-commerce application that supports transfers between accounts. It takes the total amount of the transfer, sends it to the new account, and deducts the amount from the original account.

Example Language: Perl *(Bad)*

```
$transfer_amount = GetTransferAmount();
$balance = GetBalanceFromDatabase();
if ($transfer_amount < 0) {
  FatalError("Bad Transfer Amount");
}
$newbalance = $balance - $transfer_amount;
if (($balance - $transfer_amount) < 0) {
  FatalError("Insufficient Funds");
}
SendNewBalanceToDatabase($newbalance);
NotifyUser("Transfer of $transfer_amount succeeded.");
NotifyUser("New balance: $newbalance");
```

A race condition could occur between the calls to GetBalanceFromDatabase() and SendNewBalanceToDatabase().

Suppose the balance is initially 100.00. An attack could be constructed as follows:

Example Language: Other *(Attack)*

```
In the following pseudocode, the attacker makes two simultaneous calls of the program, CALLER-1 and CALLER-2. Both callers are for the same user account.
CALLER-1 (the attacker) is associated with PROGRAM-1 (the instance that handles CALLER-1). CALLER-2 is associated with PROGRAM-2.
CALLER-1 makes a transfer request of 80.00.
PROGRAM-1 calls GetBalanceFromDatabase and sets $balance to 100.00
PROGRAM-1 calculates $newbalance as 20.00, then calls SendNewBalanceToDatabase().
Due to high server load, the PROGRAM-1 call to SendNewBalanceToDatabase() encounters a delay.
CALLER-2 makes a transfer request of 1.00.
PROGRAM-2 calls GetBalanceFromDatabase() and sets $balance to 100.00. This happens because the previous PROGRAM-1 request was not processed yet.
PROGRAM-2 determines the new balance as 99.00.
After the initial delay, PROGRAM-1 commits its balance to the database, setting it to 20.00.
PROGRAM-2 sends a request to update the database, setting the balance to 99.00
```

At this stage, the attacker should have a balance of 19.00 (due to 81.00 worth of transfers), but the balance is 99.00, as recorded in the database.

To prevent this weakness, the programmer has several options, including using a lock to prevent multiple simultaneous requests to the web application, or using a synchronization mechanism that includes all the code between GetBalanceFromDatabase() and SendNewBalanceToDatabase().

Example 2:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

Example Language: C *(Bad)*

```c
void f(pthread_mutex_t *mutex) {
  pthread_mutex_lock(mutex);
  /* access shared resource */
  pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason, the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races, correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting them to higher levels.

Example Language: C *(Good)*

```c
int f(pthread_mutex_t *mutex) {
  int result;
  result = pthread_mutex_lock(mutex);
  if (0 != result)
    return result;
  /* access shared resource */
  return pthread_mutex_unlock(mutex);
}
```

Example 3:

Suppose a processor's Memory Management Unit (MMU) has 5 other shadow MMUs to distribute its workload for its various cores. Each MMU has the start address and end address of "accessible" memory. Any time this accessible range changes (as per the processor's boot status), the main MMU sends an update message to all the shadow MMUs.

Suppose the interconnect fabric does not prioritize such "update" packets over other general traffic packets. This introduces a race condition. If an attacker can flood the target with enough messages so that some of those attack packets reach the target before the new access ranges gets updated, then the attacker can leverage this scenario.

Observed Examples

Reference	Description
CVE-2022-29527	Go application for cloud management creates a world-writable sudoers file that allows local attackers to inject sudo rules and escalate privileges to root by winning a race condition. *https://www.cve.org/CVERecord?id=CVE-2022-29527*
CVE-2021-1782	Chain: improper locking (CWE-667) leads to race condition (CWE-362), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-1782*
CVE-2021-0920	Chain: mobile platform race condition (CWE-362) leading to use-after-free (CWE-416), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-0920*
CVE-2020-6819	Chain: race condition (CWE-362) leads to use-after-free (CWE-416), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-6819*
CVE-2019-18827	chain: JTAG interface is not disabled (CWE-1191) during ROM code execution, introducing a race condition (CWE-362) to extract encryption keys *https://www.cve.org/CVERecord?id=CVE-2019-18827*
CVE-2019-1161	Chain: race condition (CWE-362) in anti-malware product allows deletion of files by creating a junction (CWE-1386) and using hard links during the time window in which a temporary file is created and deleted. *https://www.cve.org/CVERecord?id=CVE-2019-1161*
CVE-2015-1743	TOCTOU in sandbox process allows installation of untrusted browser add-ons by replacing a file after it has been verified, but before it is executed *https://www.cve.org/CVERecord?id=CVE-2015-1743*
CVE-2014-8273	Chain: chipset has a race condition (CWE-362) between when an interrupt handler detects an attempt to write-enable the BIOS (in violation of the lock bit), and when the handler resets the write-enable bit back to 0, allowing attackers to issue BIOS writes during the timing window [REF-1237]. *https://www.cve.org/CVERecord?id=CVE-2014-8273*
CVE-2008-5044	Race condition leading to a crash by calling a hook removal procedure while other activities are occurring at the same time. *https://www.cve.org/CVERecord?id=CVE-2008-5044*
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://www.cve.org/CVERecord?id=CVE-2008-2958*
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://www.cve.org/CVERecord?id=CVE-2008-1570*
CVE-2008-0058	Unsynchronized caching operation enables a race condition that causes messages to be sent to a deallocated object. *https://www.cve.org/CVERecord?id=CVE-2008-0058*
CVE-2008-0379	Race condition during initialization triggers a buffer overflow. *https://www.cve.org/CVERecord?id=CVE-2008-0379*
CVE-2007-6599	Daemon crash by quickly performing operations and undoing them, which eventually leads to an operation that does not acquire a lock. *https://www.cve.org/CVERecord?id=CVE-2007-6599*
CVE-2007-6180	chain: race condition triggers NULL pointer dereference *https://www.cve.org/CVERecord?id=CVE-2007-6180*

Reference	Description
CVE-2007-5794	Race condition in library function could cause data to be sent to the wrong process. *https://www.cve.org/CVERecord?id=CVE-2007-5794*
CVE-2007-3970	Race condition in file parser leads to heap corruption. *https://www.cve.org/CVERecord?id=CVE-2007-3970*
CVE-2008-5021	chain: race condition allows attacker to access an object while it is still being initialized, causing software to access uninitialized memory. *https://www.cve.org/CVERecord?id=CVE-2008-5021*
CVE-2009-4895	chain: race condition for an argument value, possibly resulting in NULL dereference *https://www.cve.org/CVERecord?id=CVE-2009-4895*
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference *https://www.cve.org/CVERecord?id=CVE-2009-3547*
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition (CWE-362) that leads to a double free (CWE-415). *https://www.cve.org/CVERecord?id=CVE-2006-5051*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2552
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2352
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2354
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2366
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2380
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	2412
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2448
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	C	1365	ICS Communications: Unreliability	1358	2502
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1376	ICS Engineering (Construction/Deployment): Security Gaps in Commissioning	1358	2512
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Maintenance

The relationship between race conditions and synchronization problems (CWE-662) needs to be further developed. They are not necessarily two perspectives of the same core concept, since synchronization is only one technique for avoiding race conditions, and synchronization can be used for other purposes besides race condition prevention.

Research Gap

Race conditions in web applications are under-studied and probably under-reported. However, in 2008 there has been growing interest in this area.

Research Gap

Much of the focus of race condition research has been in Time-of-check Time-of-use (TOCTOU) variants (CWE-367), but many race conditions are related to synchronization problems that do not necessarily require a time-of-check.

Research Gap

From a classification/taxonomy perspective, the relationships between concurrency and program state need closer investigation and may be useful in organizing related issues.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Race Conditions
The CERT Oracle Secure Coding Standard for Java (2011)	VNA03-J		Do not assume that a group of calls to independently atomic methods is atomic

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

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

[REF-349]Andrei Alexandrescu. "volatile - Multithreaded Programmer's Best Friend". Dr. Dobb's. 2008 February 1. < https://drdobbs.com/cpp/volatile-the-multithreaded-programmers-b/184403766 >.2023-04-07.

[REF-350]Steven Devijver. "Thread-safe webapps using Spring". < https://web.archive.org/web/20170609174845/http://www.javalobby.org/articles/thread-safe/index.jsp >.2023-04-07.

[REF-351]David Wheeler. "Prevent race conditions". 2007 October 4. < https://www.ida.liu.se/~TDDC90/literature/papers/SP-race-conditions.pdf >.2023-04-07.

[REF-352]Matt Bishop. "Race Conditions, Files, and Security Flaws; or the Tortoise and the Hare Redux". 1995 September. < https://seclab.cs.ucdavis.edu/projects/vulnerabilities/scriv/ucd-ecs-95-08.pdf >.2023-04-07.

[REF-353]David Wheeler. "Secure Programming for Linux and Unix HOWTO". 2003 March 3. < https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/avoid-race.html >.2023-04-07.

[REF-354]Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". 2002 April. < https://www.blakewatts.com/blog/discovering-and-exploiting-named-pipe-security-flaws-for-fun-and-profit >.2023-04-07.

[REF-355]Roberto Paleari, Davide Marrone, Danilo Bruschi and Mattia Monga. "On Race Vulnerabilities in Web Applications". < http://security.dico.unimi.it/~roberto/pubs/dimva08-web.pdf >.

[REF-356]"Avoiding Race Conditions and Insecure File Operations". Apple Developer Connection. < https://web.archive.org/web/20081010155022/http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html >.2023-04-07.

[REF-357]Johannes Ullrich. "Top 25 Series - Rank 25 - Race Conditions". 2010 March 6. SANS Software Security Institute. < https://web.archive.org/web/20100530231203/http://blogs.sans.org:80/appsecstreetfighter/2010/03/26/top-25-series-rank-25-race-conditions/ >.2023-04-07.

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

[REF-1237]CERT Coordination Center. "Intel BIOS locking mechanism contains race condition that enables write protection bypass". 2015 January 5. < https://www.kb.cert.org/vuls/id/766164/ >.

CWE-363: Race Condition Enabling Link Following

Weakness ID : 363
Structure : Simple
Abstraction : Base

Description

The product checks the status of a file or directory before accessing it, which produces a race condition in which the file can be replaced with a link before the access is performed, causing the product to access the wrong file.

Extended Description

While developers might expect that there is a very narrow time window between the time of check and time of use, there is still a race condition. An attacker could cause the product to slow down (e.g. with memory consumption), causing the time window to become larger. Alternately, in some situations, the attacker could win the race by performing a large number of attacks.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	906
CanPrecede	B	59	Improper Link Resolution Before File Access ('Link Following')	111

Applicable Platforms

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

Common Consequences

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

Demonstrative Examples

Example 1:

This code prints the contents of a file if a user has permission.

Example Language: PHP *(Bad)*

```
function readFile($filename){
  $user = getCurrentUser();
  //resolve file if its a symbolic link
  if(is_link($filename)){
    $filename = readlink($filename);
  }
  if(fileowner($filename) == $user){
    echo file_get_contents($realFile);
    return;
  }
  else{
    echo 'Access denied';
    return false;
  }
}
```

This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is_link() and file_get_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2351
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	2412
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2463
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Relationship

This is already covered by the "Link Following" weakness (CWE-59). It is included here because so many people associate race conditions with link problems; however, not all link following issues involve race conditions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Race condition enabling link following
CERT C Secure Coding	POS35-C	Exact	Avoid race conditions while checking for the existence of a symbolic link
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions

References

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

CWE-364: Signal Handler Race Condition

Weakness ID : 364
Structure : Simple
Abstraction : Base

Description

The product uses a signal handler that introduces a race condition.

Extended Description

Race conditions frequently occur in signal handlers, since signal handlers support asynchronous actions. These race conditions have a variety of root causes and symptoms. Attackers may be able to exploit a signal handler race condition to cause the product state to be corrupted, possibly leading to a denial of service or even code execution.

These issues occur when non-reentrant functions, or state-sensitive actions occur in the signal handler, where they may be called at any time. These behaviors can violate assumptions being made by the "regular" code that is interrupted, or by other signal handlers that may also be invoked. If these functions are called at an inopportune moment - such as while a non-reentrant function is already running - memory corruption could occur that may be exploitable for code execution. Another signal race condition commonly found occurs when free is called within a signal handler, resulting in a double free and therefore a write-what-where condition. Even if a given pointer is set to NULL after it has been freed, a race condition still exists between the time the memory was freed and the pointer was set to NULL. This is especially problematic if the same signal handler has been set for more than one signal -- since it means that the signal handler itself may be reentered.

There are several known behaviors related to signal handlers that have received the label of "signal handler race condition":

- Shared state (e.g. global data or static variables) that are accessible to both a signal handler and "regular" code
- Shared state between a signal handler and other signal handlers
- Use of non-reentrant functionality within a signal handler - which generally implies that shared state is being used. For example, malloc() and free() are non-reentrant because they may use global or static data structures for managing memory, and they are indirectly used by innocent-seeming functions such as syslog(); these functions could be exploited for memory corruption and, possibly, code execution.
- Association of the same signal handler function with multiple signals - which might imply shared state, since the same code and resources are accessed. For example, this can be a source of double-free and use-after-free weaknesses.
- Use of setjmp and longjmp, or other mechanisms that prevent a signal handler from returning control back to the original functionality
- While not technically a race condition, some signal handlers are designed to be called at most once, and being called more than once can introduce security problems, even when there are not any concurrent calls to the signal handler. This can be a source of double-free and use-after-free weaknesses.

Signal handler vulnerabilities are often classified based on the absence of a specific protection mechanism, although this style of classification is discouraged in CWE because programmers often have a choice of several different mechanisms for addressing the weakness. Such protection mechanisms may preserve exclusivity of access to the shared resource, and behavioral atomicity for the relevant code:

- Avoiding shared state
- Using synchronization in the signal handler
- Using synchronization in the regular code
- Disabling or masking other signals, which provides atomicity (which effectively ensures exclusivity)

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
ParentOf	B	432	Dangerous Signal Handler not Disabled During Sensitive Operations	1045
ParentOf	V	828	Signal Handler with Functionality that is not Asynchronous-Safe	1737
ParentOf	V	831	Signal Handler Function Associated with Multiple Signals	1749
CanPrecede	B	123	Write-what-where Condition	323
CanPrecede	V	415	Double Free	1008
CanPrecede	V	416	Use After Free	1012

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

Nature	Type	ID	Name	Page
MemberOf	C	387	Signal Errors	2321
MemberOf	C	557	Concurrency Issues	2329

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Application Data Modify Memory DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands *It may be possible to cause data corruption and possibly execute arbitrary code by modifying global variables or data structures at unexpected times, violating the assumptions of code that uses this global data.*	
Access Control	Gain Privileges or Assume Identity *If a signal handler interrupts code that is executing with privileges, it may be possible that the signal handler will*	

Scope	Impact	Likelihood
	also be executed with elevated privileges, possibly making subsequent exploits more severe.	

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design

Design signal handlers to only set flags, rather than perform complex functionality. These flags can then be checked and acted upon within the main program loop.

Phase: Implementation

Only use reentrant functions within signal handlers. Also, use validation to ensure that state is consistent while performing asynchronous actions that affect the state of execution.

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C *(Bad)*

```
char *logMessage;
void handler (int sigNum) {
  syslog(LOG_NOTICE, "%s\n", logMessage);
  free(logMessage);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
  exit(0);
}
int main (int argc, char* argv[]) {
  logMessage = strdup(argv[1]);
  /* Register signal handlers. */
  signal(SIGHUP, handler);
  signal(SIGTERM, handler);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.

- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

Example 2:

The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

Example Language: C *(Bad)*

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh (int dummy) {
  syslog(LOG_NOTICE,"%s\n",what);
  free(global2);
  free(global1);
  /* Sleep statements added to expand timing window for race condition */
  sleep(10);
  exit(0);
}
int main (int argc,char* argv[]) {
  what=argv[1];
  global1=strdup(argv[2]);
  global2=malloc(340);
  signal(SIGHUP,sh);
  signal(SIGTERM,sh);
  /* Sleep statements added to expand timing window for race condition */
  sleep(10);
  exit(0);
}
```

However, the following sequence of events may result in a double-free (CWE-415):

1. a SIGHUP is delivered to the process
2. sh() is invoked to process the SIGHUP
3. This first invocation of sh() reaches the point where global1 is freed
4. At this point, a SIGTERM is sent to the process
5. the second invocation of sh() might do another free of global1
6. this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the syslog call may use malloc calls which are not async-signal safe. This could cause corruption of the heap management structures. For more details, consult the example within "Delivering Signals for Fun and Profit" [REF-360].

Observed Examples

Reference	Description
CVE-1999-0035	Signal handler does not disable other signal handlers, allowing it to be interrupted, causing other functionality to access files/etc. with raised privileges *https://www.cve.org/CVERecord?id=CVE-1999-0035*
CVE-2001-0905	Attacker can send a signal while another signal handler is already running, leading to crash or execution with root privileges *https://www.cve.org/CVERecord?id=CVE-2001-0905*
CVE-2001-1349	unsafe calls to library functions from signal handler *https://www.cve.org/CVERecord?id=CVE-2001-1349*
CVE-2004-0794	SIGURG can be used to remotely interrupt signal handler; other variants exist *https://www.cve.org/CVERecord?id=CVE-2004-0794*
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. *https://www.cve.org/CVERecord?id=CVE-2004-2259*

Functional Areas

- Signals
- Interprocess Communication

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Signal handler race condition
7 Pernicious Kingdoms			Signal Handling Race Conditions
CLASP			Race condition in signal handler
Software Fault Patterns	SFP19		Missing Lock

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < https://lcamtuf.coredump.cx/signals.txt >.2023-04-07.

[REF-361]"Race Condition: Signal Handling". < https://vulncat.fortify.com/en/detail?id=desc.structural.cpp.race_condition_signal_handling#:~:text=Signal%20handling%20race%20conditions%20can,installed%20to%20handle%20multiple%20signals.s >.2023-04-07.

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

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

CWE-366: Race Condition within a Thread

Weakness ID : 366
Structure : Simple
Abstraction : Base

Description

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.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888

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

Nature	Type	ID	Name	Page
ChildOf	●	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf	●	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Alter Execution Logic Unexpected State *The main problem is that -- if a lock is overcome -- data could be altered in a bad state.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use locking functionality. This is the recommended solution. Implement some form of locking mechanism around code which alters or reads persistent data in a multithreaded environment.

Phase: Architecture and Design

Create resource-locking validation 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.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
int foo = 0;
int storenum(int num) {
  static int counter = 0;
  counter++;
  if (num > foo) foo = num;
  return foo;
}
```

Example Language: Java *(Bad)*

```
public classRace {
  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);
    }
  }
}
```

Observed Examples

Reference	Description
CVE-2022-2621	Chain: two threads in a web browser use the same resource (CWE-366), but one of those threads can destroy the resource before the other has completed (CWE-416). *https://www.cve.org/CVERecord?id=CVE-2022-2621*

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2351
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2366
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2380
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2448
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2462
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Race condition within a thread
CERT C Secure Coding	CON32-C	CWE More Abstract	Prevent data races when accessing bit-fields from multiple threads
CERT C Secure Coding	CON40-C	CWE More Abstract	Do not refer to an atomic variable twice in an expression
CERT C Secure Coding	CON43-C	Exact	Do not allow data races in multithreaded code
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
The CERT Oracle Secure Coding Standard for Java (2011)	VNA03-J		Do not assume that a group of calls to independently atomic methods is atomic
Software Fault Patterns	SFP19		Missing Lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

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

CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

Weakness ID : 367
Structure : Simple
Abstraction : Base

Description

The product checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the product to perform invalid actions when the resource is in an unexpected state.

Extended Description

This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
ParentOf	B	363	Race Condition Enabling Link Following	897
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	942
CanFollow	B	609	Double-Checked Locking	1362

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

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Applicable Platforms

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

Alternate Terms

TOCTTOU : The TOCTTOU acronym expands to "Time Of Check To Time Of Use".

TOCCTOU : The TOCCTOU acronym is most likely a typo of TOCTTOU, but it has been used in some influential documents, so the typo is repeated fairly frequently.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Alter Execution Logic Unexpected State *The attacker can gain access to otherwise unauthorized resources.*	
Integrity Other	Modify Application Data Modify Files or Directories Modify Memory Other	

Scope	Impact	Likelihood
	Race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question.	
Integrity Other	Other *The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.*	
Non-Repudiation	Hide Activities *If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur.*	
Non-Repudiation Other	Other *In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.

Phase: Implementation

When the file being altered is owned by the current user and group, set the effective gid and uid to that of the current user and group when executing this statement.

Phase: Architecture and Design

Limit the interleaving of operations on files from multiple processes.

Phase: Implementation

Phase: Architecture and Design

If you cannot perform operations atomically and you must share access to the resource between multiple processes or threads, then try to limit the amount of time (CPU cycles) between the check and use of the resource. This will not fix the problem, but it could make it more difficult for an attack to succeed.

Phase: Implementation

Recheck the resource after the use call to verify that the action was taken appropriately.

Phase: Architecture and Design

Ensure that some environmental locking mechanism can be used to protect resources effectively.

Phase: 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.

Demonstrative Examples

Example 1:

The following code checks a file, then updates its contents.

Example Language: C *(Bad)*

```
struct stat *sb;
...
lstat("...",sb); // it has not been updated since the last time it was read
printf("stated file\n");
if (sb->st_mtimespec==...){
  print("Now updating things\n");
  updateThings();
}
```

Potentially the file could have been updated between the time of the check and the lstat, especially since the printf has latency.

Example 2:

The following code is from a program installed setuid root. The program performs certain file operations on behalf of non-privileged users, and uses access checks to ensure that it does not use its root privileges to perform operations that should otherwise be unavailable the current user. The program uses the access() system call to check if the person running the program has permission to access the specified file before it opens the file and performs the necessary operations.

Example Language: C *(Bad)*

```
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.

Example 3:

This code prints the contents of a file if a user has permission.

Example Language: PHP *(Bad)*

```
function readFile($filename){
  $user = getCurrentUser();
  //resolve file if its a symbolic link
  if(is_link($filename)){
    $filename = readlink($filename);
  }
  if(fileowner($filename) == $user){
    echo file_get_contents($realFile);
    return;
  }
  else{
    echo 'Access denied';
    return false;
  }
}
```

This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is_link() and file_get_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

Example 4:

This example is adapted from [REF-18]. Assume that this code block is invoked from multiple threads. The switch statement will execute different code depending on the time when MYFILE.txt was last changed.

Example Language: C *(Bad)*

```
#include <sys/types.h>
#include <sys/stat.h>
...
struct stat sb;
stat("MYFILE.txt",&sb);
printf("file change time: %d\n",sb->st_ctime);
switch(sb->st_ctime % 2){
  case 0: printf("Option 1\n"); break;
  case 1: printf("Option 2\n"); break;
  default: printf("this should be unreachable?\n"); break;
}
```

If this code block were executed within multiple threads, and MYFILE.txt changed between the operation of one thread and another, then the switch could produce different, possibly unexpected results.

Observed Examples

Reference	Description
CVE-2015-1743	TOCTOU in sandbox process allows installation of untrusted browser add-ons by replacing a file after it has been verified, but before it is executed *https://www.cve.org/CVERecord?id=CVE-2015-1743*
CVE-2003-0813	A multi-threaded race condition allows remote attackers to cause a denial of service (crash or reboot) by causing two threads to process the same RPC request, which causes one thread to use memory after it has been freed. *https://www.cve.org/CVERecord?id=CVE-2003-0813*
CVE-2004-0594	PHP flaw allows remote attackers to execute arbitrary code by aborting execution before the initialization of key data structures is complete. *https://www.cve.org/CVERecord?id=CVE-2004-0594*
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://www.cve.org/CVERecord?id=CVE-2008-2958*
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://www.cve.org/CVERecord?id=CVE-2008-1570*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	2412
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Relationship

TOCTOU issues do not always involve symlinks, and not every symlink issue is a TOCTOU problem.

Research Gap

Non-symlink TOCTOU issues are not reported frequently, but they are likely to occur in code that attempts to be secure.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Time-of-check Time-of-use race condition
7 Pernicious Kingdoms			File Access Race Conditions: TOCTOU
CLASP			Time of check, time of use race condition
CLASP			Race condition in switch
CERT C Secure Coding	FIO01-C		Be careful using functions that use file names for identification
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-367]Dan Tsafrir, Tomer Hertz, David Wagner and Dilma Da Silva. "Portably Solving File TOCTTOU Races with Hardness Amplification". 2008 February 8. < https://www.usenix.org/legacy/events/fast08/tech/tsafrir.html >.2023-04-07.

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

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

CWE-368: Context Switching Race Condition

Weakness ID : 368
Structure : Simple
Abstraction : Base

Description

A product performs a series of non-atomic actions to switch between contexts that cross privilege or other security boundaries, but a race condition allows an attacker to modify or misrepresent the product's behavior during the switch.

Extended Description

This is commonly seen in web browser vulnerabilities in which the attacker can perform certain actions while the browser is transitioning from a trusted to an untrusted domain, or vice versa, and the browser performs the actions on one domain using the trust level and resources of the other domain.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
CanAlsoBe	Ⓑ	364	Signal Handler Race Condition	899

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Weakness Ordinalities

Primary : This weakness can be primary to almost anything, depending on the context of the race condition.

Resultant : This weakness can be resultant from insufficient compartmentalization (CWE-653), incorrect locking, improper initialization or shutdown, or a number of other weaknesses.

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Modify Application Data Read Application Data	

Observed Examples

Reference	Description
CVE-2009-1837	Chain: race condition (CWE-362) from improper handling of a page transition in web client while an applet is loading (CWE-368) leads to use after free (CWE-416) *https://www.cve.org/CVERecord?id=CVE-2009-1837*
CVE-2004-2260	Browser updates address bar as soon as user clicks on a link instead of when the page has loaded, allowing spoofing by redirecting to another page using

Reference	Description
	onUnload method. ** this is one example of the role of "hooks" and context switches, and should be captured somehow - also a race condition of sorts ** *https://www.cve.org/CVERecord?id=CVE-2004-2260*
CVE-2004-0191	XSS when web browser executes Javascript events in the context of a new page while it's being loaded, allowing interaction with previous page in different domain. *https://www.cve.org/CVERecord?id=CVE-2004-0191*
CVE-2004-2491	Web browser fills in address bar of clicked-on link before page has been loaded, and doesn't update afterward. *https://www.cve.org/CVERecord?id=CVE-2004-2491*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Relationship

Can overlap signal handler race conditions.

Research Gap

Under-studied as a concept. Frequency unknown; few vulnerability reports give enough detail to know when a context switching race condition is a factor.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Context Switching Race Condition

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

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

CWE-369: Divide By Zero

Weakness ID : 369
Structure : Simple
Abstraction : Base

Description

The product divides a value by zero.

Extended Description

This weakness typically occurs when an unexpected value is provided to the product, or if an error occurs that is not properly detected. It frequently occurs in calculations involving physical dimensions such as size, length, width, and height.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1499

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1499

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1499

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1499

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

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	2312

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *A Divide by Zero results in a crash.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Demonstrative Examples

Example 1:

The following Java example contains a function to compute an average but does not validate that the input value used as the denominator is not zero. This will create an exception for attempting to divide by zero. If this error is not handled by Java exception handling, unexpected results can occur.

Example Language: Java *(Bad)*

```
public int computeAverageResponseTime (int totalTime, int numRequests) {
  return totalTime / numRequests;
}
```

By validating the input value used as the denominator the following code will ensure that a divide by zero error will not cause unexpected results. The following Java code example will validate the input value, output an error message, and throw an exception.

Example Language: *(Good)*

```
public int computeAverageResponseTime (int totalTime, int numRequests) throws ArithmeticException {
  if (numRequests == 0) {
    System.out.println("Division by zero attempted!");
    throw ArithmeticException;
  }
  return totalTime / numRequests;
}
```

Example 2:

The following C/C++ example contains a function that divides two numeric values without verifying that the input value used as the denominator is not zero. This will create an error for attempting to divide by zero, if this error is not caught by the error handling capabilities of the language, unexpected results can occur.

Example Language: C *(Bad)*

```
double divide(double x, double y){
  return x/y;
}
```

By validating the input value used as the denominator the following code will ensure that a divide by zero error will not cause unexpected results. If the method is called and a zero is passed as the second argument a DivideByZero error will be thrown and should be caught by the calling block with an output message indicating the error.

Example Language: *(Good)*

```
const int DivideByZero = 10;
double divide(double x, double y){
  if ( 0 == y ){
    throw DivideByZero;
  }
  return x/y;
}
...
try{
  divide(10, 0);
}
catch( int i ){
  if(i==DivideByZero) {
    cerr<<"Divide by zero error";
  }
}
```

Example 3:

The following C# example contains a function that divides two numeric values without verifying that the input value used as the denominator is not zero. This will create an error for attempting to divide by zero, if this error is not caught by the error handling capabilities of the language, unexpected results can occur.

Example Language: C# *(Bad)*

```
int Division(int x, int y){
  return (x / y);
}
```

The method can be modified to raise, catch and handle the DivideByZeroException if the input value used as the denominator is zero.

Example Language: *(Good)*

```
int SafeDivision(int x, int y){
  try{
    return (x / y);
  }
  catch (System.DivideByZeroException dbz){
    System.Console.WriteLine("Division by zero attempted!");
    return 0;
  }
}
```

Observed Examples

Reference	Description
CVE-2007-3268	Invalid size value leads to divide by zero. *https://www.cve.org/CVERecord?id=CVE-2007-3268*
CVE-2007-2723	"Empty" content triggers divide by zero. *https://www.cve.org/CVERecord?id=CVE-2007-2723*
CVE-2007-2237	Height value of 0 triggers divide by zero. *https://www.cve.org/CVERecord?id=CVE-2007-2237*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2342
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2343
MemberOf	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	2363
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2374
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2375
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	2445

Nature	Type	ID	Name	V	Page
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2456
MemberOf	C	1408	Comprehensive Categorization: Incorrect Calculation	1400	2534

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FLP03-C		Detect and handle floating point errors
CERT C Secure Coding	INT33-C	Exact	Ensure that division and remainder operations do not result in divide-by-zero errors
The CERT Oracle Secure Coding Standard for Java (2011)	NUM02-J		Ensure that division and modulo operations do not result in divide-by-zero errors
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-371]Alex Allain. "Handling Errors Exceptionally Well in C++". < https://www.cprogramming.com/tutorial/exceptions.html >.2023-04-07.

[REF-372]Microsoft. "Exceptions and Exception Handling (C# Programming Guide)". < https://msdn.microsoft.com/pl-pl/library/ms173160(v=vs.100).aspx >.

CWE-370: Missing Check for Certificate Revocation after Initial Check

Weakness ID : 370
Structure : Simple
Abstraction : Variant

Description

The product does not check the revocation status of a certificate after its initial revocation check, which can cause the product to perform privileged actions even after the certificate is revoked at a later time.

Extended Description

If the revocation status of a certificate is not checked before each action that requires privileges, the system may be subject to a race condition. If a certificate is revoked after the initial check, all subsequent actions taken with the owner of the revoked certificate will lose all benefits guaranteed by the certificate. In fact, it is almost certain that the use of a revoked certificate indicates malicious activity.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	299	Improper Check for Certificate Revocation	727
PeerOf	B	296	Improper Following of a Certificate's Chain of Trust	719
PeerOf	V	297	Improper Validation of Certificate with Host Mismatch	722
PeerOf	V	298	Improper Validation of Certificate Expiration	726

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2429

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *Trust may be assigned to an entity who is not who it claims to be.*	
Integrity	Modify Application Data *Data from an untrusted (and possibly malicious) source may be integrated.*	
Confidentiality	Read Application Data *Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.*	

Potential Mitigations

Phase: Architecture and Design

Ensure that certificates are checked for revoked status before each use of a protected resource. If the certificate is checked before each access of a protected resource, the delay subject to a possible race condition becomes almost negligible and significantly reduces the risk associated with this issue.

Demonstrative Examples

Example 1:

The following code checks a certificate before performing an action.

Example Language: C *(Bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
  foo=SSL_get_verify_result(ssl);
  if (X509_V_OK==foo)
    //do stuff
    foo=SSL_get_verify_result(ssl);
    //do more stuff without the check.
```

While the code performs the certificate verification before each action, it does not check the result of the verification after the initial attempt. The certificate may have been revoked in the time between the privileged actions.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	2412
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Race condition in checking for certificate revocation
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

CWE-372: Incomplete Internal State Distinction

Weakness ID : 372
Structure : Simple
Abstraction : Base

Description

The product does not properly determine which state it is in, causing it to assume it is in state X when in fact it is in state Y, causing it to perform incorrect operations in a security-relevant manner.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454

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

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	2321

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2400

Nature	Type	ID	Name	V	Page
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

This conceptually overlaps other categories such as insufficient verification, but this entry refers to the product's incorrect perception of its own state.

Relationship

This is probably resultant from other weaknesses such as unhandled error conditions, inability to handle out-of-order steps, multiple interpretation errors, etc.

Maintenance

This entry is being considered for deprecation. It was poorly-defined in PLOVER and is not easily described using the behavior/resource/property model of vulnerability theory.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Internal State Distinction

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State
140	Bypassing of Intermediate Forms in Multiple-Form Sets

CWE-374: Passing Mutable Objects to an Untrusted Method

Weakness ID : 374
Structure : Simple
Abstraction : Base

Description

The product sends non-cloned mutable data as an argument to a method or function.

Extended Description

The function or method that has been called can alter or delete the mutable data. This could violate assumptions that the calling function has made about its state. In situations where unknown code is called with references to mutable data, this external code could make changes to the data sent. If this data was not previously cloned, the modified data might not be valid in the context of execution.

Relationships

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

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

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

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

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	2321

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *Potentially data could be tampered with by another function which should not have been tampered with.*	

Potential Mitigations

Phase: Implementation

Pass in data which should not be altered as constant or immutable.

Phase: Implementation

Clone all mutable data before passing it into an external function . This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
private:
  int foo;
  complexType bar;
  String baz;
  otherClass externalClass;
public:
  void doStuff() {
    externalClass.doOtherStuff(foo, bar, baz)
  }
```

In this example, bar and baz will be passed by reference to doOtherStuff() which may change them.

Example 2:

In the following Java example, the BookStore class manages the sale of books in a bookstore, this class includes the member objects for the bookstore inventory and sales database manager classes. The BookStore class includes a method for updating the sales database and inventory when a book is sold. This method retrieves a Book object from the bookstore inventory object using the supplied ISBN number for the book class, then calls a method for the sales object to update the sales information and then calls a method for the inventory object to update inventory for the BookStore.

Example Language: Java *(Bad)*

```
public class BookStore {
  private BookStoreInventory inventory;
  private SalesDBManager sales;
  ...
  // constructor for BookStore
  public BookStore() {
    this.inventory = new BookStoreInventory();
    this.sales = new SalesDBManager();
    ...
  }
  public void updateSalesAndInventoryForBookSold(String bookISBN) {
    // Get book object from inventory using ISBN
    Book book = inventory.getBookWithISBN(bookISBN);
    // update sales information for book sold
    sales.updateSalesInformation(book);
    // update inventory
    inventory.updateInventory(book);
  }
  // other BookStore methods
  ...
}
public class Book {
  private String title;
  private String author;
  private String isbn;
  // Book object constructors and get/set methods
  ...
}
```

However, in this example the Book object that is retrieved and passed to the method of the sales object could have its contents modified by the method. This could cause unexpected results when the book object is sent to the method for the inventory object to update the inventory.

In the Java programming language arguments to methods are passed by value, however in the case of objects a reference to the object is passed by value to the method. When an object reference is passed as a method argument a copy of the object reference is made within the method and therefore both references point to the same object. This allows the contents of the object to be modified by the method that holds the copy of the object reference. [REF-374]

In this case the contents of the Book object could be modified by the method of the sales object prior to the call to update the inventory.

To prevent the contents of the Book object from being modified, a copy of the Book object should be made before the method call to the sales object. In the following example a copy of the Book object is made using the clone() method and the copy of the Book object is passed to the method of the sales object. This will prevent any changes being made to the original Book object.

Example Language: Java *(Good)*

```
...
public void updateSalesAndInventoryForBookSold(String bookISBN) {
  // Get book object from inventory using ISBN
  Book book = inventory.getBookWithISBN(bookISBN);
  // Create copy of book object to make sure contents are not changed
  Book bookSold = (Book) book.clone();
  // update sales information for book sold
  sales.updateSalesInformation(bookSold);
  // update inventory
  inventory.updateInventory(book);
}
...
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2364
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2446
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Passing mutable objects to an untrusted method
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ04-J		Provide mutable classes with copy functionality to safely allow passing instances to untrusted code
Software Fault Patterns	SFP23		Exposed Data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-374]Tony Sintes. "Does Java pass by reference or pass by value?". JavaWorld.com. 2000 May 6. < https://web.archive.org/web/20000619025001/https://www.javaworld.com/javaworld/javaqa/2000-05/03-qa-0526-pass.html >.2023-04-07.

[REF-375]Herbert Schildt. "Java: The Complete Reference, J2SE 5th Edition".

CWE-375: Returning a Mutable Object to an Untrusted Caller

Weakness ID : 375
Structure : Simple
Abstraction : Base

Description

Sending non-cloned mutable data as a return value may result in that data being altered or deleted by the calling function.

Extended Description

In situations where functions return references to mutable data, it is possible that the external code which called the function may make changes to the data sent. If this data was not previously cloned, the class will then be using modified data which may violate assumptions about its internal state.

Relationships

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

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

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

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

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	2321

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity	Modify Memory *Potentially data could be tampered with by another function which should not have been tampered with.*	

Potential Mitigations

Phase: Implementation

Declare returned data which should not be altered as constant or immutable.

Phase: 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.

Demonstrative Examples

Example 1:

This class has a private list of patients, but provides a way to see the list :

Example Language: Java *(Bad)*

```
public class ClinicalTrial {
  private PatientClass[] patientList = new PatientClass[50];
  public getPatients(...){
    return patientList;
  }
}
```

While this code only means to allow reading of the patient list, the getPatients() method returns a reference to the class's original patient list instead of a reference to a copy of the list. Any caller of this method can arbitrarily modify the contents of the patient list even though it is a private member of the class.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2364
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2446
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Mutable object returned
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ04-J		Provide mutable classes with copy functionality to safely allow passing instances to untrusted code
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ05-J		Defensively copy private mutable class members before returning their references
SEI CERT Perl Coding Standard	EXP34-PL	Imprecise	Do not modify $_ in list or sorting functions
Software Fault Patterns	SFP23		Exposed Data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-377: Insecure Temporary File

Weakness ID : 377
Structure : Simple
Abstraction : Class

Description

Creating and using insecure temporary files can leave application and system data vulnerable to attack.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1469
ParentOf	B	378	Creation of Temporary File With Insecure Permissions	928
ParentOf	B	379	Creation of Temporary File in Directory with Insecure Permissions	930

Applicable Platforms

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

Common Consequences

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

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control

flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Demonstrative Examples

Example 1:

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

Example Language: C *(Bad)*

```
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.

Observed Examples

Reference	Description
CVE-2022-41954	A library uses the Java File.createTempFile() method which creates a file with "-rw-r--r--" default permissions on Unix-like operating systems *https://www.cve.org/CVERecord?id=CVE-2022-41954*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf	C	964	SFP Secondary Cluster: Exposure Temporary File	888	2402
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2462
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Other

Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows(R) API. Most of these functions are vulnerable to various forms of attacks. The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file. - Group 1: "Unique" Filenames: The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like tmpnam(), tempnam(), mktemp() and their C++ equivalents prefaced with an _ (underscore) as well as the GetTempFileName() function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a malicious collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess. If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions. Finally, in the best case the file will be opened with the a call to open() using the O_CREAT and O_EXCL flags or to CreateFile() using the CREATE_NEW attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions. - Group 2: "Unique" Files: The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like tmpfile() and its C++ equivalents prefaced with an _ (underscore), as well as the slightly better-behaved C Library function mkstemp(). The tmpfile() style functions construct a unique filename and open it in the same way that fopen() would if passed the flags "wb+", that is, as a binary file in read/write mode. If the file already exists, tmpfile() will truncate it to size zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by tmpfile(). Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if tmpfile() does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance. Finally, mkstemp() is a reasonably safe way create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, mkstemp() still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes mkstemp() to fail by predicting and pre-creating the filenames to be used.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Temporary File
CERT C Secure Coding	CON33-C	Imprecise	Avoid race conditions when using library functions
The CERT Oracle Secure Coding Standard for Java (2011)	FIO00-J		Do not operate on files in shared directories

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
149	Explore for Predictable Temporary File Names
155	Screen Temporary Files for Sensitive Information

References

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

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

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

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

CWE-378: Creation of Temporary File With Insecure Permissions

Weakness ID : 378
Structure : Simple
Abstraction : Base

Description

Opening temporary files without appropriate measures or controls can leave the file, its contents and any function that it impacts vulnerable to attack.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	377	Insecure Temporary File	925

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

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2480

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *If the temporary file can be read by the attacker, sensitive information may be in that file which could be revealed.*	
Authorization Other	Other *If that file can be written to by the attacker, the file might be moved into a place to which the attacker does not have access. This will allow the attacker to gain selective resource access-control privileges.*	
Integrity Other	Other *Depending on the data stored in the temporary file, there is the potential for an attacker to gain an additional input vector which is trusted as non-malicious. It may be possible to make arbitrary changes to data structures, user information, or even process ownership.*	

Potential Mitigations

Phase: Requirements

Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

Phase: Implementation

Ensure that you use proper file permissions. This can be achieved by using a safe temp file function. Temporary files should be writable and readable only by the process that owns the file.

Phase: 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.

Demonstrative Examples

Example 1:

In the following code examples a temporary file is created and written to. After using the temporary file, the file is closed and deleted from the file system.

Example Language: C *(Bad)*

```
FILE *stream;
if( (stream = tmpfile()) == NULL ) {
  perror("Could not open new temporary file\n");
  return (-1);
}
// write data to tmp file
...
// remove tmp file
rmtmp();
```

However, within this C/C++ code the method tmpfile() is used to create and open the temp file. The tmpfile() method works the same way as the fopen() method would with read/write permission, allowing attackers to read potentially sensitive information contained in the temp file or modify the contents of the file.

Example Language: Java *(Bad)*

```
try {
  File temp = File.createTempFile("pattern", ".suffix");
  temp.deleteOnExit();
  BufferedWriter out = new BufferedWriter(new FileWriter(temp));
  out.write("aString");
  out.close();
}
catch (IOException e) {
}
```

Similarly, the createTempFile() method used in the Java code creates a temp file that may be readable and writable to all users.

Additionally both methods used above place the file into a default directory. On UNIX systems the default directory is usually "/tmp" or "/var/tmp" and on Windows systems the default directory is usually "C:\\Windows\\Temp", which may be easily accessible to attackers, possibly enabling them to read and modify the contents of the temp file.

Observed Examples

Reference	Description
CVE-2022-24823	A network application framework uses the Java function createTempFile(), which will create a file that is readable by other local users of the system *https://www.cve.org/CVERecord?id=CVE-2022-24823*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	964	SFP Secondary Cluster: Exposure Temporary File	888	2402
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper temp file opening

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-379: Creation of Temporary File in Directory with Insecure Permissions

Weakness ID : 379
Structure : Simple
Abstraction : Base

Description

The product creates a temporary file in a directory whose permissions allow unintended actors to determine the file's existence or otherwise access that file.

Extended Description

On some operating systems, the fact that the temporary file exists may be apparent to any user with sufficient privileges to access that directory. Since the file is visible, the application that is using the temporary file could be known. If one has access to list the processes on the system, the attacker has gained information about what the user is doing at that time. By correlating this with the applications the user is running, an attacker could potentially discover what a user's actions are. From this, higher levels of security could be breached.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	377	Insecure Temporary File	925

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

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2480

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Since the file is visible and the application which is using the temp file could be known, the attacker has gained information about what the user is doing at that time.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Requirements

Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

Phase: Implementation

Try to store sensitive tempfiles in a directory which is not world readable -- i.e., per-user directories.

Phase: Implementation

Avoid using vulnerable temp file functions.

Demonstrative Examples

Example 1:

In the following code examples a temporary file is created and written to. After using the temporary file, the file is closed and deleted from the file system.

Example Language: C *(Bad)*

```
FILE *stream;
if( (stream = tmpfile()) == NULL ) {
  perror("Could not open new temporary file\n");
  return (-1);
}
// write data to tmp file
...
// remove tmp file
rmtmp();
```

However, within this C/C++ code the method tmpfile() is used to create and open the temp file. The tmpfile() method works the same way as the fopen() method would with read/write permission, allowing attackers to read potentially sensitive information contained in the temp file or modify the contents of the file.

Example Language: Java *(Bad)*

```
try {
  File temp = File.createTempFile("pattern", ".suffix");
  temp.deleteOnExit();
  BufferedWriter out = new BufferedWriter(new FileWriter(temp));
  out.write("aString");
  out.close();
}
catch (IOException e) {
}
```

Similarly, the createTempFile() method used in the Java code creates a temp file that may be readable and writable to all users.

Additionally both methods used above place the file into a default directory. On UNIX systems the default directory is usually "/tmp" or "/var/tmp" and on Windows systems the default directory is usually "C:\\Windows\\Temp", which may be easily accessible to attackers, possibly enabling them to read and modify the contents of the temp file.

Observed Examples

Reference	Description
CVE-2022-27818	A hotkey daemon written in Rust creates a domain socket file underneath /tmp, which is accessible by any user. *https://www.cve.org/CVERecord?id=CVE-2022-27818*
CVE-2021-21290	A Java-based application for a rapid-development framework uses File.createTempFile() to create a random temporary file with insecure default permissions. *https://www.cve.org/CVERecord?id=CVE-2021-21290*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377

Nature	Type	ID	Name	V	Page
MemberOf	C	964	SFP Secondary Cluster: Exposure Temporary File	888	2402
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Guessed or visible temporary file
CERT C Secure Coding	FIO15-C		Ensure that file operations are performed in a secure directory

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

CWE-382: J2EE Bad Practices: Use of System.exit()

Weakness ID : 382
Structure : Simple
Abstraction : Variant

Description

A J2EE application uses System.exit(), which also shuts down its container.

Extended Description

It is never a good idea for a web application to attempt to shut down the application container. Access to a function that can shut down the application is an avenue for Denial of Service (DoS) attacks.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	705	Incorrect Control Flow Scoping	1542

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

The shutdown function should be a privileged function available only to a properly authorized administrative user

Phase: Implementation

Web applications should not call methods that cause the virtual machine to exit, such as System.exit()

Phase: Implementation

Web applications should also not throw any Throwables to the application server as this may adversely affect the container.

Phase: Implementation

Non-web applications may have a main() method that contains a System.exit(), but generally should not call System.exit() from other locations in the code

Demonstrative Examples

Example 1:

Included in the doPost() method defined below is a call to System.exit() in the event of a specific exception.

Example Language: Java *(Bad)*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  try {
    ...
  } catch (ApplicationSpecificException ase) {
    logger.error("Caught: " + ase.toString());
    System.exit(1);
  }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2365
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2448
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: System.exit()
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
The CERT Oracle Secure Coding Standard for Java (2011)	ERR09-J		Do not allow untrusted code to terminate the JVM
Software Fault Patterns	SFP3		Use of an improper API

References

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

CWE-383: J2EE Bad Practices: Direct Use of Threads

Weakness ID : 383
Structure : Simple
Abstraction : Variant

Description

Thread management in a Web application is forbidden in some circumstances and is always highly error prone.

Extended Description

Thread management in a web application is forbidden by the J2EE standard in some circumstances and is always highly error prone. Managing threads is difficult and is likely to interfere in unpredictable ways with the behavior of the application container. Even without interfering with the container, thread management usually leads to bugs that are hard to detect and diagnose like deadlock, race conditions, and other synchronization errors.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1524

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control

flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

For EJB, use framework approaches for parallel execution, instead of using threads.

Demonstrative Examples

Example 1:

In the following example, a new Thread object is created and invoked directly from within the body of a doGet() method in a Java servlet.

Example Language: Java *(Bad)*

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  // Perform servlet tasks.
  ...
  // Create a new thread to handle background processing.
  Runnable r = new Runnable() {
    public void run() {
      // Process and store request statistics.
      ...
    }
  };
  new Thread(r).start();
}
```

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: Threads
Software Fault Patterns	SFP3		Use of an improper API

References

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

CWE-384: Session Fixation

Weakness ID : 384
Structure : Composite
Abstraction : Compound

Description

Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.

Composite Components

Nature	Type	ID	Name	Page
Requires	C	346	Origin Validation Error	853
Requires	B	472	External Control of Assumed-Immutable Web Parameter	1123
Requires	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1064

Extended Description

Such a scenario is commonly observed when:

- A web application authenticates a user without first invalidating the existing session, thereby continuing to use the session already associated with the user.
- An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session.
- The application or container uses predictable session identifiers. In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to associate, and possibly authenticate, against the server using that session identifier, giving the attacker access to the user's account through the active session.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1364

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

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1364

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

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	2432

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Invalidate any existing session identifiers prior to authorizing a new user session.

Phase: Architecture and Design

For platforms such as ASP that do not generate new values for sessionid cookies, utilize a secondary cookie. In this approach, set a secondary cookie on the user's browser to a random value and set a session variable to the same value. If the session variable and the cookie value ever don't match, invalidate the session, and force the user to log on again.

Demonstrative Examples

Example 1:

The following example shows a snippet of code from a J2EE web application where the application authenticates users with LoginContext.login() without first calling HttpSession.invalidate().

Example Language: Java *(Bad)*

```
private void auth(LoginContext lc, HttpSession session) throws LoginException {
  ...
  lc.login();
  ...
}
```

In order to exploit the code above, an attacker could first create a session (perhaps by logging into the application) from a public terminal, record the session identifier assigned by the application, and reset the browser to the login page. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session. Even given a vulnerable application, the success of the specific attack described here is dependent on several factors working in the favor of the attacker: access to an unmonitored public terminal, the ability to keep the compromised session active and a victim interested in logging into the vulnerable application on the public terminal.

In most circumstances, the first two challenges are surmountable given a sufficient investment of time. Finding a victim who is both using a public terminal and interested in logging into the vulnerable application is possible as well, so long as the site is reasonably popular. The less well known the site is, the lower the odds of an interested victim using the public terminal and the lower the chance of success for the attack vector described above. The biggest challenge an attacker faces in exploiting session fixation vulnerabilities is inducing victims to authenticate against the vulnerable application using a session identifier known to the attacker.

In the example above, the attacker did this through a direct method that is not subtle and does not scale suitably for attacks involving less well-known web sites. However, do not be lulled into complacency; attackers have many tools in their belts that help bypass the limitations of this attack vector. The most common technique employed by attackers involves taking advantage of cross-site scripting or HTTP response splitting vulnerabilities in the target site [12]. By tricking the victim into submitting a malicious request to a vulnerable application that reflects JavaScript or other code back to the victim's browser, an attacker can create a cookie that will cause the victim to reuse a session identifier controlled by the attacker. It is worth noting that cookies are often tied to the top level domain associated with a given URL. If multiple applications reside on the same top level domain, such as bank.example.com and recipes.example.com, a vulnerability in one application can allow an attacker to set a cookie with a fixed session identifier that will be used in all interactions with any application on the domain example.com [29].

Example 2:

The following example shows a snippet of code from a J2EE web application where the application authenticates users with a direct post to the <code>j_security_check</code>, which typically does not invalidate the existing session before processing the login request.

Example Language: HTML *(Bad)*

```
<form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="text" name="j_password">
</form>
```

Observed Examples

Reference	Description
CVE-2022-2820	Website software for game servers does not proprerly terminate user sessions, allowing for possible session fixation *https://www.cve.org/CVERecord?id=CVE-2022-2820*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2335
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2389
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2436
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Other

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Session Fixation
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	37		Session Fixation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
196	Session Credential Falsification through Forging

References

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

CWE-385: Covert Timing Channel

Weakness ID : 385
Structure : Simple
Abstraction : Base

Description

Covert timing channels convey information by modulating some aspect of system behavior over time, so that the program receiving the information can observe system behavior and infer protected information.

Extended Description

In some instances, knowing when data is transmitted between parties can provide a malicious user with privileged information. Also, externally monitoring the timing of operations can potentially reveal sensitive data. For example, a cryptographic operation can expose its internal state if the time it takes to perform the operation varies, based on the state.

Covert channels are frequently classified as either storage or timing channels. Some examples of covert timing channels are the system's paging rate, the time a certain transaction requires to execute, and the time it takes to gain access to a shared bus.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	514	Covert Channel	1218
CanFollow	B	208	Observable Timing Discrepancy	529

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2325

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Other	Read Application Data Other *Information exposure.*	

Potential Mitigations

Phase: Architecture and Design

Whenever possible, specify implementation strategies that do not introduce time variances in operations.

Phase: Implementation

Often one can artificially manipulate the time which operations take or -- when operations occur -- can remove information from the attacker.

Phase: Implementation

It is reasonable to add artificial or random delays so that the amount of CPU time consumed is independent of the action being taken by the application.

Demonstrative Examples

Example 1:

In this example, the attacker observes how long an authentication takes when the user types in the correct password.

When the attacker tries their own values, they can first try strings of various length. When they find 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 they guess the first character right, the computation will take longer than a wrong guesses. Such an attack can break even the most sophisticated password with a few hundred guesses.

Example Language: Python *(Bad)*

```
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
```

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.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	968	SFP Secondary Cluster: Covert Channel	888	2404
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2544

Notes

Maintenance

As of CWE 4.9, members of the CWE Hardware SIG are working to improve CWE's coverage of transient execution weaknesses, which include issues related to Spectre, Meltdown, and other attacks that create or exploit covert channels. As a result of that work, this entry might change in CWE 4.10.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Timing
CLASP			Covert Timing Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
462	Cross-Domain Search Timing

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-386: Symbolic Name not Mapping to Correct Object

Weakness ID : 386
Structure : Simple
Abstraction : Base

Description

A constant symbolic reference to an object is used, even though the reference can resolve to a different object over time.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1544
PeerOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	906
PeerOf	V	486	Comparison of Classes by Name	1164
PeerOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1364

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
	The attacker can gain access to otherwise unauthorized resources.	
Integrity Confidentiality Other	Modify Application Data Modify Files or Directories Read Application Data Read Files or Directories Other *Race conditions such as this kind may be employed to gain read or write access to resources not normally readable or writable by the user in question.*	
Integrity Other	Modify Application Data Other *The resource in question, or other resources (through the corrupted one) may be changed in undesirable ways by a malicious user.*	
Non-Repudiation	Hide Activities *If a file or other resource is written in this method, as opposed to a valid way, logging of the activity may not occur.*	
Non-Repudiation Integrity	Modify Files or Directories *In some cases it may be possible to delete files that a malicious user might not otherwise have access to -- such as log files.*	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2409
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Symbolic name not mapping to correct object

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-390: Detection of Error Condition Without Action

Weakness ID : 390
Structure : Simple
Abstraction : Base

Description

The product detects a specific error, but takes no actions to handle the error.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	755	Improper Handling of Exceptional Conditions	1576
PeerOf	V	600	Uncaught Exception in Servlet	1343
CanPrecede	V	401	Missing Release of Memory after Effective Lifetime	973

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

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	2434

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State Alter Execution Logic *An attacker could utilize an ignored error condition to place the system in an unexpected state that could lead to the execution of unintended logic and could cause other unintended behavior.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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.

Phase: Implementation

If a function returns an error, it is important to either fix the problem and try again, alert the user that an error has happened and let the program continue, or alert the user and close and cleanup the program.

Phase: Testing

Subject the product to extensive testing to discover some of the possible instances of where/how errors or return values are not handled. Consider testing techniques such as ad hoc, equivalence partitioning, robustness and fault tolerance, mutation, and fuzzing.

Demonstrative Examples

Example 1:

The following example attempts to allocate memory for a character. After the call to malloc, an if statement is used to check whether the malloc function failed.

Example Language: C *(Bad)*

```
foo=malloc(sizeof(char)); //the next line checks to see if malloc failed
if (foo==NULL) {
  //We do nothing so we just ignore the error.
}
```

The conditional successfully detects a NULL return value from malloc indicating a failure, however it does not do anything to handle the problem. Unhandled errors may have unexpected results and may cause the program to crash or terminate.

Instead, the if block should contain statements that either attempt to fix the problem or notify the user that an error has occurred and continue processing or perform some cleanup and gracefully terminate the program. The following example notifies the user that the malloc function did not allocate the required memory resources and returns an error code.

Example Language: C *(Good)*

```
foo=malloc(sizeof(char)); //the next line checks to see if malloc failed
if (foo==NULL) {
  printf("Malloc failed to allocate memory resources");
  return -1;
}
```

Example 2:

In the following C++ example the method readFile() will read the file whose name is provided in the input parameter and will return the contents of the file in char string. The method calls open() and read() may result in errors if the file does not exist or does not contain any data to read. These errors will be thrown when the is_open() method and good() method indicate errors opening or reading the file. However, these errors are not handled within the catch statement. Catch statements that do not perform any processing will have unexpected results. In this case an empty char string will be returned, and the file will not be properly closed.

Example Language: C++ *(Bad)*

```
char* readfile (char *filename) {
  try {
    // open input file
    ifstream infile;
    infile.open(filename);
    if (!infile.is_open()) {
      throw "Unable to open file " + filename;
    }
    // get length of file
    infile.seekg (0, ios::end);
    int length = infile.tellg();
    infile.seekg (0, ios::beg);
```

```
        // allocate memory
        char *buffer = new char [length];
        // read data from file
        infile.read (buffer,length);
        if (!infile.good()) {
            throw "Unable to read from file " + filename;
        }
        infile.close();
        return buffer;
    }
    catch (...) {
        /* bug: insert code to handle this later */
    }
}
```

The catch statement should contain statements that either attempt to fix the problem or notify the user that an error has occurred and continue processing or perform some cleanup and gracefully terminate the program. The following C++ example contains two catch statements. The first of these will catch a specific error thrown within the try block, and the second catch statement will catch all other errors from within the catch block. Both catch statements will notify the user that an error has occurred, close the file, and rethrow to the block that called the readFile() method for further handling or possible termination of the program.

Example Language: C++ *(Good)*

```
char* readFile (char *filename) {
    try {
        // open input file
        ifstream infile;
        infile.open(filename);
        if (!infile.is_open()) {
            throw "Unable to open file " + filename;
        }
        // get length of file
        infile.seekg (0, ios::end);
        int length = infile.tellg();
        infile.seekg (0, ios::beg);
        // allocate memory
        char *buffer = new char [length];
        // read data from file
        infile.read (buffer,length);
        if (!infile.good()) {
            throw "Unable to read from file " + filename;
        }
        infile.close();
        return buffer;
    }
    catch (char *str) {
        printf("Error: %s \n", str);
        infile.close();
        throw str;
    }
    catch (...) {
        printf("Error occurred trying to read from file \n");
        infile.close();
        throw;
    }
}
```

Example 3:

In the following Java example the method readFile will read the file whose name is provided in the input parameter and will return the contents of the file in a String object. The constructor of the FileReader object and the read method call may throw exceptions and therefore must be within a try/catch block. While the catch statement in this example will catch thrown exceptions in order for the method to compile, no processing is performed to handle the thrown exceptions. Catch statements that do not perform any processing will have unexpected results. In this case, this will result in the return of a null String.

Example Language: Java *(Bad)*

```
public String readFile(String filename) {
  String retString = null;
  try {
    // initialize File and FileReader objects
    File file = new File(filename);
    FileReader fr = new FileReader(file);
    // initialize character buffer
    long fLen = file.length();
    char[] cBuf = new char[(int) fLen];
    // read data from file
    int iRead = fr.read(cBuf, 0, (int) fLen);
    // close file
    fr.close();
    retString = new String(cBuf);
  } catch (Exception ex) {
    /* do nothing, but catch so it'll compile... */
  }
  return retString;
}
```

The catch statement should contain statements that either attempt to fix the problem, notify the user that an exception has been raised and continue processing, or perform some cleanup and gracefully terminate the program. The following Java example contains three catch statements. The first of these will catch the FileNotFoundException that may be thrown by the FileReader constructor called within the try/catch block. The second catch statement will catch the IOException that may be thrown by the read method called within the try/catch block. The third catch statement will catch all other exceptions thrown within the try block. For all catch statements the user is notified that the exception has been thrown and the exception is rethrown to the block that called the readFile() method for further processing or possible termination of the program. Note that with Java it is usually good practice to use the getMessage() method of the exception class to provide more information to the user about the exception raised.

Example Language: Java *(Good)*

```
public String readFile(String filename) throws FileNotFoundException, IOException, Exception {
  String retString = null;
  try {
    // initialize File and FileReader objects
    File file = new File(filename);
    FileReader fr = new FileReader(file);
    // initialize character buffer
    long fLen = file.length();
    char [] cBuf = new char[(int) fLen];
    // read data from file
    int iRead = fr.read(cBuf, 0, (int) fLen);
    // close file
    fr.close();
    retString = new String(cBuf);
  } catch (FileNotFoundException ex) {
    System.err.println ("Error: FileNotFoundException opening the input file: " + filename );
    System.err.println ("" + ex.getMessage() );
    throw new FileNotFoundException(ex.getMessage());
  } catch (IOException ex) {
    System.err.println("Error: IOException reading the input file.\n" + ex.getMessage() );
    throw new IOException(ex);
  } catch (Exception ex) {
    System.err.println("Error: Exception reading the input file.\n" + ex.getMessage() );
    throw new Exception(ex);
  }
```

```
    return retString;
}
```

Observed Examples

Reference	Description
CVE-2022-21820	A GPU data center manager detects an error due to a malformed request but does not act on it, leading to memory corruption. *https://www.cve.org/CVERecord?id=CVE-2022-21820*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2337
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2365
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2379
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper error handling
The CERT Oracle Secure Coding Standard for Java (2011)	ERR00-J		Do not suppress or ignore checked exceptions
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

CWE-391: Unchecked Error Condition

Weakness ID : 391
Structure : Simple
Abstraction : Base

Description

[PLANNED FOR DEPRECATION. SEE MAINTENANCE NOTES AND CONSIDER CWE-252, CWE-248, OR CWE-1069.] Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1568

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

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	2434

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

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1535

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

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1535

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State Alter Execution Logic	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Requirements

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.

Phase: Requirements

A language can be used which requires, at compile time, to catch all serious exceptions. However, one must make sure to use the most current version of the API as new exceptions could be added.

Phase: 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.

Demonstrative Examples

Example 1:

The following code excerpt ignores a rarely-thrown exception from doExchange().

Example Language: Java *(Bad)*

```
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.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	2322
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2337
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2350
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2379
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	2457
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2461
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2463
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Notes

Maintenance

This entry is slated for deprecation; it has multiple widespread interpretations by CWE analysts. It currently combines information from three different taxonomies, but each taxonomy is talking about a slightly different issue. CWE analysts might map to this entry based on any of these issues. 7PK has "Empty Catch Block" which has an association with empty exception block (CWE-1069); in this case, the exception has performed the check, but does not handle. In PLOVER there is "Unchecked Return Value" which is CWE-252, but unlike "Empty Catch Block" there isn't even a check of the issue - and "Unchecked Error Condition" implies lack of a check. For CLASP, "Uncaught Exception" (CWE-248) is associated with incorrect error propagation - uncovered in CWE 3.2 and earlier, at least. There are other issues related to error handling and checks.

Other

When a programmer ignores an exception, they implicitly state that they are operating under one of two assumptions: This method call can never fail. It doesn't matter if this call fails.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unchecked Return Value
7 Pernicious Kingdoms			Empty Catch Block
CLASP			Uncaught exception
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy
CERT C Secure Coding	ERR33-C	CWE More Abstract	Detect and handle standard library errors
CERT C Secure Coding	ERR34-C	CWE More Abstract	Detect errors when converting a string to a number
CERT C Secure Coding	FLP32-C	Imprecise	Prevent or detect domain and range errors in math functions
CERT C Secure Coding	POS54-C	CWE More Abstract	Detect and handle POSIX library errors
SEI CERT Perl Coding Standard	EXP31-PL	Imprecise	Do not suppress or ignore exceptions
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-392: Missing Report of Error Condition

Weakness ID : 392
Structure : Simple
Abstraction : Base

Description

The product encounters an error but does not provide a status code or return value to indicate that an error has occurred.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	684	Incorrect Provision of Specified Functionality	1505
ChildOf	C	755	Improper Handling of Exceptional Conditions	1576

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2427

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

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1535

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

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1535

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State *Errors that are not properly reported could place the system in an unexpected state that could lead to unintended behaviors.*	

Demonstrative Examples

Example 1:

In the following snippet from a doPost() servlet method, the server returns "200 OK" (default) even if an error occurs.

Example Language: Java *(Bad)*

```
try {
  // Something that may throw an exception.
  ...
} catch (Throwable t) {
  logger.error("Caught: " + t.toString());
  return;
}
```

Observed Examples

Reference	Description
[REF-1374]	Chain: JavaScript-based cryptocurrency library can fall back to the insecure Math.random() function instead of reporting a failure (CWE-392), thus reducing the entropy (CWE-332) and leading to generation of non-unique cryptographic keys for Bitcoin wallets (CWE-1391) *https://www.unciphered.com/blog/randstorm-you-cant-patch-a-house-of-cards*
CVE-2004-0063	Function returns "OK" even if another function returns a different status code than expected, leading to accepting an invalid PIN number. *https://www.cve.org/CVERecord?id=CVE-2004-0063*
CVE-2002-1446	Error checking routine in PKCS#11 library returns "OK" status even when invalid signature is detected, allowing spoofed messages. *https://www.cve.org/CVERecord?id=CVE-2002-1446*
CVE-2002-0499	Kernel function truncates long pathnames without generating an error, leading to operation on wrong directory. *https://www.cve.org/CVERecord?id=CVE-2002-0499*
CVE-2005-2459	Function returns non-error value when a particular erroneous condition is encountered, leading to resultant NULL dereference. *https://www.cve.org/CVERecord?id=CVE-2005-2459*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	2367
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2399
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	2450
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Error Status Code
The CERT Oracle Secure Coding Standard for Java (2011)	TPS03-J		Ensure that tasks executing in a thread pool do not fail silently
Software Fault Patterns	SFP6		Incorrect Exception Behavior

References

[REF-1374]Unciphered. "Randstorm: You Can't Patch a House of Cards". 2023 November 4. < https://www.unciphered.com/blog/randstorm-you-cant-patch-a-house-of-cards >.2023-11-15.

CWE-393: Return of Wrong Status Code

Weakness ID : 393
Structure : Simple
Abstraction : Base

Description

A function or operation returns an incorrect return value or status code that does not indicate an error, but causes the product to modify its behavior based on the incorrect result.

Extended Description

This can lead to unpredictable behavior. If the function is used to make security-critical decisions or provide security-critical information, then the wrong status code can cause the product to assume that an action is safe, even when it is not.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1535
ChildOf	C	684	Incorrect Provision of Specified Functionality	1505

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Alter Execution Logic *This weakness could place the system in a state that could lead unexpected logic to be executed or other unintended behaviors.*	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Demonstrative Examples

Example 1:

In the following example, an HTTP 404 status code is returned in the event of an IOException encountered in a Java servlet. A 404 code is typically meant to indicate a non-existent resource and would be somewhat misleading in this case.

Example Language: Java *(Bad)*

```
try {
  // something that might throw IOException
  ...
} catch (IOException ioe) {
  response.sendError(SC_NOT_FOUND);
}
```

Observed Examples

Reference	Description
CVE-2003-1132	DNS server returns wrong response code for non-existent AAAA record, which effectively says that the domain is inaccessible. *https://www.cve.org/CVERecord?id=CVE-2003-1132*
CVE-2001-1509	Hardware-specific implementation of system call causes incorrect results from geteuid. *https://www.cve.org/CVERecord?id=CVE-2001-1509*
CVE-2001-1559	Chain: System call returns wrong value (CWE-393), leading to a resultant NULL dereference (CWE-476). *https://www.cve.org/CVERecord?id=CVE-2001-1559*
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing Adversary-in-the-Middle (AITM) attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://www.cve.org/CVERecord?id=CVE-2014-1266*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2399
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Relationship

This can be primary or resultant, but it is probably most often primary to other issues.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Wrong Status Code
Software Fault Patterns	SFP6		Incorrect Exception Behavior

CWE-394: Unexpected Status Code or Return Value

Weakness ID : 394
Structure : Simple
Abstraction : Base

Description

The product does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the product.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	754	Improper Check for Unusual or Exceptional Conditions	1568

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Alter Execution Logic	

Observed Examples

Reference	Description
CVE-2004-1395	Certain packets (zero byte and other lengths) cause a recvfrom call to produce an unexpected return code that causes a server's listening loop to exit. *https://www.cve.org/CVERecord?id=CVE-2004-1395*
CVE-2002-2124	Unchecked return code from recv() leads to infinite loop. *https://www.cve.org/CVERecord?id=CVE-2002-2124*
CVE-2005-2553	Kernel function does not properly handle when a null is returned by a function call, causing it to call another function that it shouldn't. *https://www.cve.org/CVERecord?id=CVE-2005-2553*
CVE-2005-1858	Memory not properly cleared when read() function call returns fewer bytes than expected. *https://www.cve.org/CVERecord?id=CVE-2005-1858*
CVE-2000-0536	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname. *https://www.cve.org/CVERecord?id=CVE-2000-0536*
CVE-2001-0910	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname. *https://www.cve.org/CVERecord?id=CVE-2001-0910*
CVE-2004-2371	Game server doesn't check return values for functions that handle text strings and associated size values. *https://www.cve.org/CVERecord?id=CVE-2004-2371*
CVE-2005-1267	Resultant infinite loop when function call returns -1 value. *https://www.cve.org/CVERecord?id=CVE-2005-1267*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2337
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Notes

Relationship

Usually primary, but can be resultant from issues such as behavioral change or API abuse. This can produce resultant vulnerabilities.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unexpected Status Code or Return Value
Software Fault Patterns	SFP4		Unchecked Status Condition
SEI CERT Perl Coding Standard	EXP00-PL	Imprecise	Do not return undef

CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

Weakness ID : 395
Structure : Simple
Abstraction : Base

Description

Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.

Extended Description

Programmers typically catch NullPointerException under three circumstances:

- The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.
- The program explicitly throws a NullPointerException to signal an error condition.
- The code is part of a test harness that supplies unexpected input to the classes under test.

Of these three circumstances, only the last is acceptable.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1576
ChildOf	C	705	Incorrect Control Flow Scoping	1542

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application.

Demonstrative Examples

Example 1:

The following code mistakenly catches a NullPointerException.

Example Language: Java *(Bad)*

```
try {
  mysteryMethod();
} catch (NullPointerException npe) {
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	2322
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2365
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Catching NullPointerException
The CERT Oracle Secure Coding Standard for Java (2011)	ERR08-J		Do not catch NullPointerException or any of its ancestors

References

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

CWE-396: Declaration of Catch for Generic Exception

Weakness ID : 396
Structure : Simple
Abstraction : Base

Description

Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Extended Description

Multiple catch blocks can get ugly and repetitive, but "condensing" catch blocks by catching a high-level class like Exception can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of a language'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.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	221	Information Loss or Omission	556
ChildOf	●	755	Improper Handling of Exceptional Conditions	1576
ChildOf	●	705	Incorrect Control Flow Scoping	1542

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Language : Python *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation Other	Hide Activities Alter Execution Logic	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Demonstrative Examples

Example 1:

The following code excerpt handles three types of exceptions in an identical fashion.

Example Language: Java *(Good)*

```
try {
  doExchange();
}
catch (IOException e) {
  logger.error("doExchange failed", e);
}
catch (InvocationTargetException e) {
  logger.error("doExchange failed", e);
}
catch (SQLException e) {
  logger.error("doExchange failed", e);
}
```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

Example Language: *(Bad)*

```
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.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	2322
MemberOf	C	960	SFP Secondary Cluster: Ambiguous Exception Type	888	2399
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Overly-Broad Catch Block
Software Fault Patterns	SFP5		Ambiguous Exception Type
OMG ASCSM	ASCSM-CWE-396		
OMG ASCRM	ASCRM-CWE-396		

References

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

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

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

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

CWE-397: Declaration of Throws for Generic Exception

Weakness ID : 397
Structure : Simple
Abstraction : Base

Description

Throwing overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Extended Description

Declaring a method to throw Exception or Throwable makes it difficult for callers to perform proper error handling and error recovery. Java's exception mechanism, for example, is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	221	Information Loss or Omission	556
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1535
ChildOf	C	705	Incorrect Control Flow Scoping	1542

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

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	2322

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation Other	Hide Activities Alter Execution Logic	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Demonstrative Examples

Example 1:

The following method throws three types of exceptions.

Example Language: Java *(Good)*

```
public void doExchange() throws IOException, InvocationTargetException, SQLException {
  ...
}
```

While it might seem tidier to write

Example Language: *(Bad)*

```
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.

Example 2:

Early versions of C++ (C++98, C++03, C++11) included a feature known as Dynamic Exception Specification. This allowed functions to declare what type of exceptions it may throw. It is possible to declare a general class of exception to cover any derived exceptions that may be throw.

Example Language: *(Bad)*

```
int myfunction() throw(std::exception) {
  if (0) throw out_of_range();
  throw length_error();
}
```

In the example above, the code declares that myfunction() can throw an exception of type "std::exception" thus hiding details about the possible derived exceptions that could potentially be thrown.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	2322
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2365
MemberOf	C	960	SFP Secondary Cluster: Ambiguous Exception Type	888	2399
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2448
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Notes

Applicable Platform

For C++, this weakness only applies to C++98, C++03, and C++11. It relies on a feature known as Dynamic Exception Specification, which was part of early versions of C++ but was deprecated in C++11. It has been removed for C++17 and later.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Overly-Broad Throws Declaration
The CERT Oracle Secure Coding Standard for Java (2011)	ERR07-J		Do not throw RuntimeException, Exception, or Throwable
Software Fault Patterns	SFP5		Ambiguous Exception Type
OMG ASCSM	ASCSM-CWE-397		
OMG ASCRM	ASCRM-CWE-397		

References

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

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

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

CWE-400: Uncontrolled Resource Consumption

Weakness ID : 400
Structure : Simple
Abstraction : Class

Description

The product does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources.

Extended Description

Limited resources include memory, file system storage, database connection pool entries, and CPU. If an attacker can trigger the allocation of these limited resources, but the number or size of the resources is not controlled, then the attacker could cause a denial of service that consumes all available resources. This would prevent valid users from accessing the product, and it could potentially have an impact on the surrounding environment. For example, a memory exhaustion attack against an application could slow down the application as well as its host operating system.

There are at least three distinct scenarios which can commonly lead to resource exhaustion:

- Lack of throttling for the number of allocated resources
- Losing all references to a resource before reaching the shutdown stage
- Not closing/returning a resource after processing

Resource exhaustion problems are often result due to an incorrect implementation of the following situations:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for releasing the resource.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	C	405	Asymmetric Resource Consumption (Amplification)	986
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1613
ParentOf	B	771	Missing Reference to Active Allocated Resource	1622

Nature	Type	ID	Name	Page
ParentOf	B	779	Logging of Excessive Data	1642
ParentOf	B	920	Improper Restriction of Power Consumption	1823
ParentOf	B	1235	Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations	2017
ParentOf	B	1246	Improper Write Handling in Limited-write Non-Volatile Memories	2043
CanFollow	B	410	Insufficient Resource Pool	998

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

Nature	Type	ID	Name	Page
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1613
ParentOf	B	920	Improper Restriction of Power Consumption	1823

Applicable Platforms

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

Alternate Terms

Resource Exhaustion :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *The most common result of resource exhaustion is denial of service. The product may slow down, crash due to unhandled errors, or lock out legitimate users.*	
Access Control Other	Bypass Protection Mechanism Other *In some cases it may be possible to force the product to "fail open" in the event of resource exhaustion. The state of the product -- and possibly the security functionality - may then be compromised.*	

Detection Methods

Automated Static Analysis

Automated static analysis typically has limited utility in recognizing resource exhaustion problems, except for program-independent system resources such as files, sockets, and processes. For system resources, automated static analysis may be able to detect circumstances in which resources are not released after they have expired. Automated analysis of configuration files may be able to detect settings that do not specify a maximum value. Automated static analysis tools will not be appropriate for detecting exhaustion of custom resources, such as an intended security policy in which a bulletin board user is only allowed to make a limited number of posts per day.

Effectiveness = Limited

Automated Dynamic Analysis

Certain automated dynamic analysis techniques may be effective in spotting resource exhaustion problems, especially with resources such as processes, memory, and connections. The technique may involve generating a large number of requests to the product within a short time frame.

Effectiveness = Moderate

Fuzzing

While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently find resource exhaustion problems. This can occur when the fuzzer generates a large number of test cases but does not restart the targeted product in between test cases. If an individual test case produces a crash, but it does not do so reliably, then an inability to handle resource exhaustion may be the cause.

Effectiveness = Opportunistic

Potential Mitigations

Phase: Architecture and Design

Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.

Phase: Architecture and Design

Mitigation of resource exhaustion attacks requires that the target system either: recognizes the attack and denies that user further access for a given amount of time, or uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed. The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, they 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.

Phase: Architecture and Design

Ensure that protocols have specific limits of scale placed on them.

Phase: Implementation

Ensure that all failures in resource allocation place the system into a safe posture.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(Bad)*

```
class Worker implements Executor {
  ...
  public void execute(Runnable r) {
    try {
      ...
    }
    catch (InterruptedException ie) {
      // postpone response
      Thread.currentThread().interrupt();
    }
  }
  public Worker(Channel ch, int nworkers) {
    ...
  }
  protected void activate() {
    Runnable loop = new Runnable() {
      public void run() {
        try {
          for (;;) {
            Runnable r = ...;
            r.run();
          }
        }
        catch (InterruptedException ie) {
          ...
        }
      }
    };
    new Thread(loop).start();
  }
}
```

There are no limits to runnables. Potentially an attacker could cause resource problems very quickly.

Example 2:

This code allocates a socket and forks each time it receives a new connection.

Example Language: C *(Bad)*

```
sock=socket(AF_INET, SOCK_STREAM, 0);
while (1) {
  newsock=accept(sock, ...);
  printf("A connection has been accepted\n");
  pid = fork();
}
```

The program does not track how many connections have been made, and it does not limit the number of connections. Because forking is a relatively expensive operation, an attacker would be able to cause the system to run out of CPU, processes, or memory by making a large number of connections. Alternatively, an attacker could consume all available connections, preventing others from accessing the system remotely.

Example 3:

In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method openSocketConnection establishes a server socket to accept requests from a client. When a client establishes a connection to this service the getNextMessage method is first used to retrieve from the socket the name of the file to store the data, the openFileToWrite method will validate the filename and open a file to write to on the local file system. The getNextMessage is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

Example Language: C *(Bad)*

```
int writeDataFromSocketToFile(char *host, int port)
{
  char filename[FILENAME_SIZE];
  char buffer[BUFFER_SIZE];
  int socket = openSocketConnection(host, port);
  if (socket < 0) {
    printf("Unable to open socket connection");
    return(FAIL);
  }
```

```
    if (getNextMessage(socket, filename, FILENAME_SIZE) > 0) {
        if (openFileToWrite(filename) > 0) {
            while (getNextMessage(socket, buffer, BUFFER_SIZE) > 0){
                if (!(writeToFile(buffer) > 0))
                    break;
            }
        }
        closeFile();
    }
    closeSocket(socket);
}
```

This example creates a situation where data can be dumped to a file on the local file system without any limits on the size of the file. This could potentially exhaust file or disk resources and/or limit other clients' ability to access the service.

Example 4:

In the following example, the processMessage method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The getMessageLength method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

Example Language: C *(Bad)*

```
/* process message accepts a two-dimensional character array of the form [length][body] containing the message to be
processed */
int processMessage(char **message)
{
    char *body;
    int length = getMessageLength(message[0]);
    if (length > 0) {
        body = &message[1][0];
        processMessageBody(body);
        return(SUCCESS);
    }
    else {
        printf("Unable to process message; invalid message length");
        return(FAIL);
    }
}
```

This example creates a situation where the length of the body character array can be very large and will consume excessive memory, exhausting system resources. This can be avoided by restricting the length of the second character array with a maximum length check

Also, consider changing the type from 'int' to 'unsigned int', so that you are always guaranteed that the number is positive. This might not be possible if the protocol specifically requires allowing negative values, or if you cannot control the return value from getMessageLength(), but it could simplify the check to ensure the input is positive, and eliminate other errors such as signed-to-unsigned conversion errors (CWE-195) that may occur elsewhere in the code.

Example Language: C *(Good)*

```
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

Example 5:

In the following example, a server object creates a server socket and accepts client connections to the socket. For every client connection to the socket a separate thread object is generated using the ClientSocketThread class that handles request made by the client through the socket.

Example Language: Java *(Bad)*

```
public void acceptConnections() {
  try {
    ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
    int counter = 0;
    boolean hasConnections = true;
    while (hasConnections) {
      Socket client = serverSocket.accept();
      Thread t = new Thread(new ClientSocketThread(client));
      t.setName(client.getInetAddress().getHostName() + ":" + counter++);
      t.start();
    }
    serverSocket.close();
  } catch (IOException ex) {...}
}
```

In this example there is no limit to the number of client connections and client threads that are created. Allowing an unlimited number of client connections and threads could potentially overwhelm the system and system resources.

The server should limit the number of client connections and the client threads that are created. This can be easily done by creating a thread pool object that limits the number of threads that are generated.

Example Language: Java *(Good)*

```
public static final int SERVER_PORT = 4444;
public static final int MAX_CONNECTIONS = 10;
...
public void acceptConnections() {
  try {
    ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
    int counter = 0;
    boolean hasConnections = true;
    while (hasConnections) {
      hasConnections = checkForMoreConnections();
      Socket client = serverSocket.accept();
      Thread t = new Thread(new ClientSocketThread(client));
      t.setName(client.getInetAddress().getHostName() + ":" + counter++);
      ExecutorService pool = Executors.newFixedThreadPool(MAX_CONNECTIONS);
      pool.execute(t);
    }
    serverSocket.close();
  } catch (IOException ex) {...}
}
```

Example 6:

In the following example, the serve function receives an http request and an http response writer. It reads the entire request body.

Example Language: Go *(Bad)*

```
func serve(w http.ResponseWriter, r *http.Request) {
  var body []byte
  if r.Body != nil {
    if data, err := io.ReadAll(r.Body); err == nil {
      body = data
    }
  }
}
```

Because ReadAll is defined to read from src until EOF, it does not treat an EOF from Read as an error to be reported. This example creates a situation where the length of the body supplied can be very large and will consume excessive memory, exhausting system resources. This can be avoided by ensuring the body does not exceed a predetermined length of bytes.

MaxBytesReader prevents clients from accidentally or maliciously sending a large request and wasting server resources. If possible, the code could be changed to tell ResponseWriter to close the connection after the limit has been reached.

Example Language: Go *(Good)*

```go
func serve(w http.ResponseWriter, r *http.Request) {
   var body []byte
   const MaxRespBodyLength = 1e6
   if r.Body != nil {
      r.Body = http.MaxBytesReader(w, r.Body, MaxRespBodyLength)
      if data, err := io.ReadAll(r.Body); err == nil {
         body = data
      }
   }
}
```

Observed Examples

Reference	Description
CVE-2022-21668	Chain: Python library does not limit the resources used to process images that specify a very large number of bands (CWE-1284), leading to excessive memory consumption (CWE-789) or an integer overflow (CWE-190). *https://www.cve.org/CVERecord?id=CVE-2022-21668*
CVE-2020-7218	Go-based workload orchestrator does not limit resource usage with unauthenticated connections, allowing a DoS by flooding the service *https://www.cve.org/CVERecord?id=CVE-2020-7218*
CVE-2020-3566	Resource exhaustion in distributed OS because of "insufficient" IGMP queue management, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-3566*
CVE-2009-2874	Product allows attackers to cause a crash via a large number of connections. *https://www.cve.org/CVERecord?id=CVE-2009-2874*
CVE-2009-1928	Malformed request triggers uncontrolled recursion, leading to stack exhaustion. *https://www.cve.org/CVERecord?id=CVE-2009-1928*
CVE-2009-2858	Chain: memory leak (CWE-404) leads to resource exhaustion. *https://www.cve.org/CVERecord?id=CVE-2009-2858*
CVE-2009-2726	Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption. *https://www.cve.org/CVERecord?id=CVE-2009-2726*
CVE-2009-2540	Large integer value for a length property in an object causes a large amount of memory allocation. *https://www.cve.org/CVERecord?id=CVE-2009-2540*
CVE-2009-2299	Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data. *https://www.cve.org/CVERecord?id=CVE-2009-2299*
CVE-2009-2054	Product allows exhaustion of file descriptors when processing a large number of TCP packets. *https://www.cve.org/CVERecord?id=CVE-2009-2054*
CVE-2008-5180	Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created. *https://www.cve.org/CVERecord?id=CVE-2008-5180*

Reference	Description
CVE-2008-2121	TCP implementation allows attackers to consume CPU and prevent new connections using a TCP SYN flood attack. *https://www.cve.org/CVERecord?id=CVE-2008-2121*
CVE-2008-2122	Port scan triggers CPU consumption with processes that attempt to read data from closed sockets. *https://www.cve.org/CVERecord?id=CVE-2008-2122*
CVE-2008-1700	Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window. *https://www.cve.org/CVERecord?id=CVE-2008-1700*
CVE-2007-4103	Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake. *https://www.cve.org/CVERecord?id=CVE-2007-4103*
CVE-2006-1173	Mail server does not properly handle deeply nested multipart MIME messages, leading to stack exhaustion. *https://www.cve.org/CVERecord?id=CVE-2006-1173*
CVE-2007-0897	Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans. *https://www.cve.org/CVERecord?id=CVE-2007-0897*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2368
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	2411
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	2451
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

"Resource consumption" could be interpreted as a consequence instead of an insecure behavior, so this entry is being considered for modification. It appears to be referenced too frequently when more precise mappings are available. Some of its children, such as CWE-771, might be better considered as a chain.

Theoretical

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect the underlying weaknesses that enable these attacks (or consequences) to take place.

Other

Database queries that take a long time to process are good DoS targets. An attacker would have to write a few lines of Perl code to generate enough traffic to exceed the site's ability to keep up. This would effectively prevent authorized users from using the site at all. Resources can be exploited simply by ensuring that the target machine must do much more work and consume more resources in order to service a request than the attacker must do to initiate a request. A prime example of this can be found in old switches that were vulnerable to "macof" attacks (so named for a tool developed by Dugsong). These attacks flooded a switch with random IP and MAC address combinations, therefore exhausting the switch's cache, which held the information of which port corresponded to which MAC addresses. Once this cache was exhausted, the switch would fail in an insecure way and would begin to act simply as a hub, broadcasting all traffic on all ports and allowing for basic sniffing attacks.

Maintenance

The Taxonomy_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Resource exhaustion (file descriptor, disk space, sockets, ...)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	10		Denial of Service
WASC	41		XML Attribute Blowup
The CERT Oracle Secure Coding Standard for Java (2011)	SER12-J		Avoid memory and resource leaks during serialization
The CERT Oracle Secure Coding Standard for Java (2011)	MSC05-J		Do not exhaust heap space
Software Fault Patterns	SFP13		Unrestricted Consumption
ISA/IEC 62443	Part 3-3		Req SR 7.1
ISA/IEC 62443	Part 3-3		Req SR 7.2
ISA/IEC 62443	Part 4-1		Req SI-1
ISA/IEC 62443	Part 4-1		Req SVV-3
ISA/IEC 62443	Part 4-2		Req CR 7.1
ISA/IEC 62443	Part 4-2		Req CR 7.2

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
147	XML Ping of the Death
227	Sustained Client Engagement

CAPEC-ID	Attack Pattern Name
492	Regular Expression Exponential Blowup

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-386]Joao Antunes, Nuno Ferreira Neves and Paulo Verissimo. "Detection and Prediction of Resource-Exhaustion Vulnerabilities". Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE). 2008 November. < http://homepages.di.fc.ul.pt/~nuno/PAPERS/ISSRE08.pdf >.

[REF-387]D.J. Bernstein. "Resource exhaustion". < http://cr.yp.to/docs/resources.html >.

[REF-388]Pascal Meunier. "Resource exhaustion". Secure Programming Educational Material. 2004. < http://homes.cerias.purdue.edu/~pmeunier/secprog/sanitized/class1/6.resource%20exhaustion.ppt >.

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

CWE-401: Missing Release of Memory after Effective Lifetime

Weakness ID : 401
Structure : Simple
Abstraction : Variant

Description

The product does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.

Extended Description

This is often triggered by improper handling of malformed data or unexpectedly interrupted sessions. In some languages, developers are responsible for tracking memory allocation and releasing the memory. If there are no more pointers or references to the memory, then it can no longer be tracked and identified for release.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	772	Missing Release of Resource after Effective Lifetime	1624
CanFollow	B	390	Detection of Error Condition Without Action	943

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

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	980

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

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	980

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Memory Leak :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Instability DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *Most memory leaks result in general product reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing or hanging the program) or take advantage of other unexpected program behavior resulting from a low memory condition.*	
Other	Reduce Performance	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Libraries or Frameworks

Choose a language or tool that provides automatic memory management, or makes manual memory management less error-prone. For example, glibc in Linux provides protection against free of invalid pointers. When using Xcode to target OS X or iOS, enable automatic reference counting (ARC) [REF-391]. To help correctly and consistently manage memory when programming in C++, consider using a smart pointer class such as std::auto_ptr (defined by ISO/IEC ISO/IEC 14882:2003), std::shared_ptr and std::unique_ptr (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Phase: Architecture and Design

Phase: Build and Compilation

The Boehm-Demers-Weiser Garbage Collector or valgrind can be used to detect leaks in code.

Demonstrative Examples

Example 1:

The following C function leaks a block of allocated memory if the call to read() does not return the expected number of bytes:

Example Language: C *(Bad)*

```
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;
}
```

Observed Examples

Reference	Description
CVE-2005-3119	Memory leak because function does not free() an element of a data structure. *https://www.cve.org/CVERecord?id=CVE-2005-3119*
CVE-2004-0427	Memory leak when counter variable is not decremented. *https://www.cve.org/CVERecord?id=CVE-2004-0427*
CVE-2002-0574	chain: reference count is not decremented, leading to memory leak in OS by sending ICMP packets. *https://www.cve.org/CVERecord?id=CVE-2002-0574*
CVE-2005-3181	Kernel uses wrong function to release a data structure, preventing data from being properly tracked by other code. *https://www.cve.org/CVERecord?id=CVE-2005-3181*
CVE-2004-0222	Memory leak via unknown manipulations as part of protocol test suite. *https://www.cve.org/CVERecord?id=CVE-2004-0222*
CVE-2001-0136	Memory leak via a series of the same command. *https://www.cve.org/CVERecord?id=CVE-2001-0136*

Functional Areas

- Memory Management

Affected Resources

- Memory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2323
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1238	SFP Primary Cluster: Failure to Release Memory	888	2482
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Relationship

This is often a resultant weakness due to improper handling of malformed data or early termination of sessions.

Terminology

"memory leak" has sometimes been used to describe other kinds of issues, e.g. for information leaks in which the contents of memory are inadvertently leaked (CVE-2003-0400 is one such example of this terminology conflict).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Memory leak
7 Pernicious Kingdoms			Memory Leak
CLASP			Failure to deallocate data
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	MEM31-C	Exact	Free dynamically allocated memory when no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	MSC04-J		Do not leak memory
Software Fault Patterns	SFP14		Failure to Release Resource
OMG ASCPEM	ASCPEM-PRF-14		

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-390]J. Whittaker and H. Thompson. "How to Break Software Security". 2003. Addison Wesley.

[REF-391]iOS Developer Library. "Transitioning to ARC Release Notes". 2013 August 8. < https://developer.apple.com/library/archive/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html >.2023-04-07.

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < https://www.omg.org/spec/ASCPEM/ >.2023-04-07.

CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak')

Weakness ID : 402

Structure : Simple
Abstraction : Class

Description

The product makes resources available to untrusted parties when those resources are only intended to be accessed by the product.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1469
ParentOf	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	978
ParentOf	B	619	Dangling Database Cursor ('Cursor Injection')	1382

Alternate Terms

Resource Leak :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Observed Examples

Reference	Description
CVE-2003-0740	Server leaks a privileged file descriptor, allowing the server to be hijacked. *https://www.cve.org/CVERecord?id=CVE-2003-0740*
CVE-2004-1033	File descriptor leak allows read of restricted files. *https://www.cve.org/CVERecord?id=CVE-2004-1033*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Resource leaks

CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')

Weakness ID : 403
Structure : Simple
Abstraction : Base

Description

A process does not close sensitive file descriptors before invoking a child process, which allows the child to perform unauthorized I/O operations using those descriptors.

Extended Description

When a new process is forked or executed, the child process inherits any open file descriptors. When the child process has fewer privileges than the parent process, this might introduce a vulnerability if the child process can access the file descriptor but does not have the privileges to access the associated file.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	976

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

Applicable Platforms

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

Operating_System : Unix *(Prevalence = Undetermined)*

Alternate Terms

File descriptor leak : While this issue is frequently called a file descriptor leak, the "leak" term is often used in two different ways - exposure of a resource, or consumption of a resource. Use of this term could cause confusion.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Observed Examples

Reference	Description
CVE-2003-0740	Server leaks a privileged file descriptor, allowing the server to be hijacked. *https://www.cve.org/CVERecord?id=CVE-2003-0740*
CVE-2004-1033	File descriptor leak allows read of restricted files. *https://www.cve.org/CVERecord?id=CVE-2004-1033*
CVE-2000-0094	Access to restricted resource using modified file descriptor for stderr. *https://www.cve.org/CVERecord?id=CVE-2000-0094*
CVE-2002-0638	Open file descriptor used as alternate channel in complex race condition. *https://www.cve.org/CVERecord?id=CVE-2002-0638*
CVE-2003-0489	Program does not fully drop privileges after creating a file descriptor, which allows access to the descriptor via a separate vulnerability. *https://www.cve.org/CVERecord?id=CVE-2003-0489*
CVE-2003-0937	User bypasses restrictions by obtaining a file descriptor then calling setuid program, which does not close the descriptor. *https://www.cve.org/CVERecord?id=CVE-2003-0937*
CVE-2004-2215	Terminal manager does not properly close file descriptors, allowing attackers to access terminals of other users. *https://www.cve.org/CVERecord?id=CVE-2004-2215*
CVE-2006-5397	Module opens a file for reading twice, allowing attackers to read files. *https://www.cve.org/CVERecord?id=CVE-2006-5397*

Affected Resources

- System Process
- File or Directory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNIX file descriptor leak
CERT C Secure Coding	FIO42-C		Ensure files are properly closed when they are no longer needed
Software Fault Patterns	SFP23		Exposed Data

References

[REF-392]Paul Roberts. "File descriptors and setuid applications". 2007 February 5. < https://blogs.oracle.com/paulr/entry/file_descriptors_and_setuid_applications >.

[REF-393]Apple. "Introduction to Secure Coding Guide". < https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Articles/AccessControl.html >.2023-04-07.

CWE-404: Improper Resource Shutdown or Release

Weakness ID : 404
Structure : Simple
Abstraction : Class

Description

The product does not release or incorrectly releases a resource before it is made available for re-use.

Extended Description

When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation, such as a set period of time or revocation.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	B	299	Improper Check for Certificate Revocation	727
ParentOf	B	459	Incomplete Cleanup	1099
ParentOf	B	763	Release of Invalid Pointer or Reference	1599
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	1624
ParentOf	B	1266	Improper Scrubbing of Sensitive Data from Decommissioned Device	2091
PeerOf	C	405	Asymmetric Resource Consumption (Amplification)	986
PeerOf	V	239	Failure to Handle Incomplete Element	582
CanPrecede	B	619	Dangling Database Cursor ('Cursor Injection')	1382

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

Nature	Type	ID	Name	Page
ParentOf	V	401	Missing Release of Memory after Effective Lifetime	973
ParentOf	B	459	Incomplete Cleanup	1099
ParentOf	B	763	Release of Invalid Pointer or Reference	1599
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	1624

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

Nature	Type	ID	Name	Page
ParentOf	V	401	Missing Release of Memory after Effective Lifetime	973
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	1624
ParentOf	V	775	Missing Release of File Descriptor or Handle after Effective Lifetime	1631

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

Nature	Type	ID	Name	Page
ParentOf	V	761	Free of Pointer not at Start of Buffer	1592
ParentOf	V	762	Mismatched Memory Management Routines	1596
ParentOf	B	763	Release of Invalid Pointer or Reference	1599
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	1624

Nature	Type	ID	Name	Page
ParentOf	V	775	Missing Release of File Descriptor or Handle after Effective Lifetime	1631

Weakness Ordinalities

Primary : Improper release or shutdown of resources can be primary to resource exhaustion, performance, and information confidentiality problems to name a few.

Resultant : Improper release or shutdown of resources can be resultant from improper error handling or insufficient resource tracking.

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability Other	DoS: Resource Consumption (Other) Varies by Context *Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.*	
Confidentiality	Read Application Data *When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.*	

Detection Methods

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Resource clean up errors might be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the product under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control

flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

Phase: Implementation

It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free memory in a function. If you allocate memory that you intend to free upon completion of the function, you must be sure to free the memory at all exit points for that function including error conditions.

Phase: Implementation

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

Phase: Implementation

When releasing a complex object or structure, ensure that you properly dispose of all of its member components, not just the object itself.

Demonstrative Examples

Example 1:

The following method never closes the new file handle. Given enough time, the Finalize() method for BufferReader should eventually call Close(), but there is no guarantee as to how long this action will take. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, the Operating System could use up all of the available file handles before the Close() function is called.

Example Language: Java *(Bad)*

```
private void processFile(string fName)
{
  BufferReader fil = new BufferReader(new FileReader(fName));
  String line;
  while ((line = fil.ReadLine()) != null)
  {
    processLine(line);
  }
}
```

The good code example simply adds an explicit call to the Close() function when the system is done using the file. Within a simple example such as this the problem is easy to see and fix. In a real system, the problem may be considerably more obscure.

Example Language: Java *(Good)*

```
private void processFile(string fName)
{
  BufferReader fil = new BufferReader(new FileReader(fName));
  String line;
  while ((line = fil.ReadLine()) != null)
  {
        processLine(line);
    }
    fil.Close();
}
```

Example 2:

This code attempts to open a connection to a database and catches any exceptions that may occur.

Example Language: Java *(Bad)*

```
try {
    Connection con = DriverManager.getConnection(some_connection_string);
}
catch ( Exception e ) {
    log( e );
}
```

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application.

Example 3:

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

Example Language: C# *(Bad)*

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

Example 4:

The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

Example Language: C *(Bad)*

```
int decodeFile(char* fName) {
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    }
    else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            }
            else {
                decodeBlock(buf);
```

```
      }
    }
  }
  fclose(f);
  return DECODE_SUCCESS;
}
```

Example 5:

In this example, the program does not use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

Example Language: C++ *(Bad)*

```
class A {
  void foo();
};
void A::foo(){
  int *ptr;
  ptr = (int*)malloc(sizeof(int));
  delete ptr;
}
```

Example 6:

In this example, the program calls the delete[] function on non-heap memory.

Example Language: C++ *(Bad)*

```
class A{
  void foo(bool);
};
void A::foo(bool heap) {
  int localArray[2] = {
    11,22
  };
  int *p = localArray;
  if (heap){
    p = new int[2];
  }
  delete[] p;
}
```

Observed Examples

Reference	Description
CVE-1999-1127	Does not shut down named pipe connections if malformed data is sent. *https://www.cve.org/CVERecord?id=CVE-1999-1127*
CVE-2001-0830	Sockets not properly closed when attacker repeatedly connects and disconnects from server. *https://www.cve.org/CVERecord?id=CVE-2001-0830*
CVE-2002-1372	Chain: Return values of file/socket operations are not checked (CWE-252), allowing resultant consumption of file descriptors (CWE-772). *https://www.cve.org/CVERecord?id=CVE-2002-1372*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2323
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2380
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	2410
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

Overlaps memory leaks, asymmetric resource consumption, malformed input errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper resource shutdown or release
7 Pernicious Kingdoms			Unreleased Resource
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
CERT C Secure Coding	MEM31-C	CWE More Abstract	Free dynamically allocated memory when no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Release resources when they are no longer needed
Software Fault Patterns	SFP14		Failure to release resource

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
125	Flooding
130	Excessive Allocation
131	Resource Leak Exposure
494	TCP Fragmentation
495	UDP Fragmentation

CAPEC-ID	Attack Pattern Name
496	ICMP Fragmentation
666	BlueSmacking

References

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

CWE-405: Asymmetric Resource Consumption (Amplification)

Weakness ID : 405
Structure : Simple
Abstraction : Class

Description

The product does not properly control situations in which an adversary can cause the product to consume or produce excessive resources without requiring the adversary to invest equivalent work or otherwise prove authorization, i.e., the adversary's influence is "asymmetric."

Extended Description

This can lead to poor performance due to "amplification" of resource consumption, typically in a non-linear fashion. This situation is worsened if the product allows malicious users or attackers to consume more resources than their access level permits.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	400	Uncontrolled Resource Consumption	964
ParentOf	Ⓒ	406	Insufficient Control of Network Message Volume (Network Amplification)	990
ParentOf	Ⓒ	407	Inefficient Algorithmic Complexity	992
ParentOf	Ⓑ	408	Incorrect Behavior Order: Early Amplification	995
ParentOf	Ⓑ	409	Improper Handling of Highly Compressed Data (Data Amplification)	996
ParentOf	Ⓑ	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1633
ParentOf	Ⓑ	1050	Excessive Platform Resource Consumption within a Loop	1885
ParentOf	Ⓑ	1072	Data Resource Access without Use of Connection Pooling	1912
ParentOf	Ⓑ	1073	Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses	1913
ParentOf	Ⓑ	1084	Invokable Control Element with Excessive File or Data Access Operations	1924
ParentOf	Ⓑ	1089	Large Data Table with Excessive Number of Indices	1929
ParentOf	Ⓑ	1094	Excessive Index Range Scan for a Data Resource	1934
ParentOf	Ⓒ	1176	Inefficient CPU Computation	1971
PeerOf	Ⓒ	404	Improper Resource Shutdown or Release	980

Applicable Platforms

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

Operating_System : Not OS-Specific *(Prevalence = Undetermined)*

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

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

Technology : Client Server *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *Sometimes this is a factor in "flood" attacks, but other types of amplification exist.*	High

Potential Mitigations

Phase: Architecture and Design

An application must make resources available to a client commensurate with the client's access level.

Phase: Architecture and Design

An application must, at all times, keep track of allocated resources and meter their usage appropriately.

Phase: System Configuration

Consider disabling resource-intensive algorithms on the server side, such as Diffie-Hellman key exchange.

Effectiveness = High

Business requirements may prevent disabling resource-intensive algorithms.

Demonstrative Examples

Example 1:

This code listens on a port for DNS requests and sends the result to the requesting address.

Example Language: Python *(Bad)*

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (UDP_IP,UDP_PORT) )
while true:
  data = sock.recvfrom(1024)
  if not data:
    break
  (requestIP, nameToResolve) = parseUDPpacket(data)
  record = resolveName(nameToResolve)
  sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Example 2:

This function prints the contents of a specified file requested by a user.

Example Language: PHP *(Bad)*

```
function printFile($username,$filename){
  //read file into string
  $file = file_get_contents($filename);
  if ($file && isOwnerOf($username,$filename)){
    echo $file;
    return true;
  }
  else{
    echo 'You are not authorized to view this file';
  }
  return false;
}
```

This code first reads a specified file into memory, then prints the file if the user is authorized to see its contents. The read of the file into memory may be resource intensive and is unnecessary if the user is not allowed to see the file anyway.

Example 3:

The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is 2^0. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or 2^1. Ultimately, we reach entity THIRTYTWO, which will expand to 2^32 characters in length, or 4 GB, probably consuming far more data than expected.

Example Language: XML *(Attack)*

```
<?xml version="1.0"?>
<!DOCTYPE MaliciousDTD [
<!ENTITY ZERO "A">
<!ENTITY ONE "&ZERO;&ZERO;">
<!ENTITY TWO "&ONE;&ONE;">
...
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;">
]>
<data>&THIRTYTWO;</data>
```

Example 4:

This example attempts to check if an input string is a "sentence" [REF-1164].

Example Language: JavaScript *(Bad)*

```
var test_string = "Bad characters: $@#";
var bad_pattern = /^(\w+\s?)*$/i;
var result = test_string.search(bad_pattern);
```

The regular expression has a vulnerable backtracking clause inside (\w+\s?)*$ which can be triggered to cause a Denial of Service by processing particular phrases.

To fix the backtracking problem, backtracking is removed with the ?= portion of the expression which changes it to a lookahead and the \2 which prevents the backtracking. The modified example is:

Example Language: JavaScript *(Good)*

```
var test_string = "Bad characters: $@#";
var good_pattern = /^((?=(\w+))\2\s?)*$/i;
var result = test_string.search(good_pattern);
```

Note that [REF-1164] has a more thorough (and lengthy) explanation of everything going on within the RegEx.

Example 5:

An adversary can cause significant resource consumption on a server by filtering the cryptographic algorithms offered by the client to the ones that are the most resource-intensive on the server side. After discovering which cryptographic algorithms are supported by the server, a malicious client can send the initial cryptographic handshake messages that contains only the resource-intensive algorithms. For some cryptographic protocols, these messages can be completely prefabricated, as the resource-intensive part of the handshake happens on the server-side first (such as TLS), rather than on the client side. In the case of cryptographic protocols where the resource-intensive part should happen on the client-side first (such as SSH), a malicious client can send a forged/precalculated computation result, which seems correct to the server, so the resource-intensive part of the handshake is going to happen on the server side. A malicious client is required to send only the initial messages of a cryptographic handshake to initiate the resource-consuming part of the cryptographic handshake. These messages are usually small, and generating them requires minimal computational effort, enabling a denial-of-service attack. An additional risk is the fact that higher key size increases the effectiveness of the attack. Cryptographic protocols where the clients have influence over the size of the used key (such as TLS 1.3 or SSH) are most at risk, as the client can enforce the highest key size supported by the server.

Observed Examples

Reference	Description
CVE-1999-0513	Classic "Smurf" attack, using spoofed ICMP packets to broadcast addresses. *https://www.cve.org/CVERecord?id=CVE-1999-0513*
CVE-2003-1564	Parsing library allows XML bomb *https://www.cve.org/CVERecord?id=CVE-2003-1564*
CVE-2004-2458	Tool creates directories before authenticating user. *https://www.cve.org/CVERecord?id=CVE-2004-2458*
CVE-2020-10735	Python has "quadratic complexity" issue when converting string to int with many digits in unexpected bases *https://www.cve.org/CVERecord?id=CVE-2020-10735*
CVE-2020-5243	server allows ReDOS with crafted User-Agent strings, due to overlapping capture groups that cause excessive backtracking. *https://www.cve.org/CVERecord?id=CVE-2020-5243*
CVE-2013-5211	composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses. *https://www.cve.org/CVERecord?id=CVE-2013-5211*
CVE-2002-20001	Diffie-Hellman (DHE) Key Agreement Protocol allows attackers to send arbitrary numbers that are not public keys, which causes the server to perform expensive, unnecessary computation of modular exponentiation. *https://www.cve.org/CVERecord?id=CVE-2002-20001*
CVE-2022-40735	The Diffie-Hellman Key Agreement Protocol allows use of long exponents, which are more computationally expensive than using certain "short exponents" with particular properties. *https://www.cve.org/CVERecord?id=CVE-2022-40735*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	2367
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407

Nature	Type	ID	Name	V	Page
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	2450
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Asymmetric resource consumption (amplification)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	41		XML Attribute Blowup
The CERT Oracle Secure Coding Standard for Java (2011)	TPS00-J		Use thread pools to enable graceful degradation of service during traffic bursts
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Release resources when they are no longer needed

References

[REF-1164]Ilya Kantor. "Catastrophic backtracking". 2020 December 3. < https://javascript.info/regexp-catastrophic-backtracking >.

CWE-406: Insufficient Control of Network Message Volume (Network Amplification)

Weakness ID : 406
Structure : Simple
Abstraction : Class

Description

The product does not sufficiently monitor or control transmitted network traffic volume, so that an actor can cause the product to transmit more traffic than should be allowed for that actor.

Extended Description

In the absence of a policy to restrict asymmetric resource consumption, the application or system cannot distinguish between legitimate transmissions and traffic intended to serve as an amplifying attack on target systems. Systems can often be configured to restrict the amount of traffic sent out on behalf of a client, based on the client's origin or access level. This is usually defined in a resource allocation policy. In the absence of a mechanism to keep track of transmissions, the system or application can be easily abused to transmit asymmetrically greater traffic than the request or client should be permitted to.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	405	Asymmetric Resource Consumption (Amplification)	986

Nature	Type	ID	Name	Page
CanFollow	B	941	Incorrectly Specified Destination in a Communication Channel	1845

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *System resources can be quickly consumed leading to poor application performance or system crash. This may affect network performance and could be used to attack other systems and applications relying on network performance.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

An application must make network resources available to a client commensurate with the client's access level.

Phase: Policy

Define a clear policy for network resource allocation and consumption.

Phase: Implementation

An application must, at all times, keep track of network resources and meter their usage appropriately.

Demonstrative Examples

Example 1:

This code listens on a port for DNS requests and sends the result to the requesting address.

Example Language: Python *(Bad)*

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (UDP_IP,UDP_PORT) )
while true:
  data = sock.recvfrom(1024)
  if not data:
    break
  (requestIP, nameToResolve) = parseUDPpacket(data)
  record = resolveName(nameToResolve)
  sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Observed Examples

Reference	Description
CVE-1999-0513	Classic "Smurf" attack, using spoofed ICMP packets to broadcast addresses.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-1999-0513
CVE-1999-1379	DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker. *https://www.cve.org/CVERecord?id=CVE-1999-1379*
CVE-2000-0041	Large datagrams are sent in response to malformed datagrams. *https://www.cve.org/CVERecord?id=CVE-2000-0041*
CVE-1999-1066	Game server sends a large amount. *https://www.cve.org/CVERecord?id=CVE-1999-1066*
CVE-2013-5211	composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses. *https://www.cve.org/CVERecord?id=CVE-2013-5211*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1382	ICS Operations (& Maintenance): Emerging Energy Technologies	1358	2517
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

This can be resultant from weaknesses that simplify spoofing attacks.

Theoretical

Network amplification, when performed with spoofing, is normally a multi-channel attack from attacker (acting as user) to amplifier, and amplifier to victim.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Network Amplification

CWE-407: Inefficient Algorithmic Complexity

Weakness ID : 407
Structure : Simple
Abstraction : Class

Description

An algorithm in a product has an inefficient worst-case computational complexity that may be detrimental to system performance and can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	405	Asymmetric Resource Consumption (Amplification)	986
ParentOf	Ⓑ	1333	Inefficient Regular Expression Complexity	2230

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

Nature	Type	ID	Name	Page
ParentOf	Ⓑ	1333	Inefficient Regular Expression Complexity	2230

Applicable Platforms

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

Alternate Terms

Quadratic Complexity : Used when the algorithmic complexity is related to the square of the number of inputs (N^2)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *The typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur.*	

Demonstrative Examples

Example 1:

This example attempts to check if an input string is a "sentence" [REF-1164].

Example Language: JavaScript *(Bad)*

```
var test_string = "Bad characters: $@#";
var bad_pattern = /^(\w+\s?)*$/i;
var result = test_string.search(bad_pattern);
```

The regular expression has a vulnerable backtracking clause inside (\w+\s?)*$ which can be triggered to cause a Denial of Service by processing particular phrases.

To fix the backtracking problem, backtracking is removed with the ?= portion of the expression which changes it to a lookahead and the \2 which prevents the backtracking. The modified example is:

Example Language: JavaScript *(Good)*

```
var test_string = "Bad characters: $@#";
var good_pattern = /^((?=(\w+))\2\s?)*$/i;
var result = test_string.search(good_pattern);
```

Note that [REF-1164] has a more thorough (and lengthy) explanation of everything going on within the RegEx.

Observed Examples

Reference	Description
CVE-2021-32617	C++ library for image metadata has "quadratic complexity" issue with unnecessarily repetitive parsing each time an invalid character is encountered *https://www.cve.org/CVERecord?id=CVE-2021-32617*
CVE-2020-10735	Python has "quadratic complexity" issue when converting string to int with many digits in unexpected bases *https://www.cve.org/CVERecord?id=CVE-2020-10735*
CVE-2020-5243	server allows ReDOS with crafted User-Agent strings, due to overlapping capture groups that cause excessive backtracking. *https://www.cve.org/CVERecord?id=CVE-2020-5243*
CVE-2014-1474	Perl-based email address parser has "quadratic complexity" issue via a string that does not contain a valid address *https://www.cve.org/CVERecord?id=CVE-2014-1474*
CVE-2003-0244	CPU consumption via inputs that cause many hash table collisions. *https://www.cve.org/CVERecord?id=CVE-2003-0244*
CVE-2003-0364	CPU consumption via inputs that cause many hash table collisions. *https://www.cve.org/CVERecord?id=CVE-2003-0364*
CVE-2002-1203	Product performs unnecessary processing before dropping an invalid packet. *https://www.cve.org/CVERecord?id=CVE-2002-1203*
CVE-2001-1501	CPU and memory consumption using many wildcards. *https://www.cve.org/CVERecord?id=CVE-2001-1501*
CVE-2004-2527	Product allows attackers to cause multiple copies of a program to be loaded more quickly than the program can detect that other copies are running, then exit. This type of error should probably have its own category, where teardown takes more time than initialization. *https://www.cve.org/CVERecord?id=CVE-2004-2527*
CVE-2006-6931	Network monitoring system allows remote attackers to cause a denial of service (CPU consumption and detection outage) via crafted network traffic, aka a "backtracking attack." *https://www.cve.org/CVERecord?id=CVE-2006-6931*
CVE-2006-3380	Wiki allows remote attackers to cause a denial of service (CPU consumption) by performing a diff between large, crafted pages that trigger the worst case algorithmic complexity. *https://www.cve.org/CVERecord?id=CVE-2006-3380*
CVE-2006-3379	Wiki allows remote attackers to cause a denial of service (CPU consumption) by performing a diff between large, crafted pages that trigger the worst case algorithmic complexity. *https://www.cve.org/CVERecord?id=CVE-2006-3379*
CVE-2005-2506	OS allows attackers to cause a denial of service (CPU consumption) via crafted Gregorian dates. *https://www.cve.org/CVERecord?id=CVE-2005-2506*
CVE-2005-1792	Memory leak by performing actions faster than the software can clear them. *https://www.cve.org/CVERecord?id=CVE-2005-1792*

Functional Areas

- Cryptography

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Algorithmic Complexity

References

[REF-395]Scott A. Crosby and Dan S. Wallach. "Algorithmic Complexity Attacks". Proceedings of the 12th USENIX Security Symposium. 2003 August. < https://www.usenix.org/legacy/events/sec03/tech/full_papers/crosby/crosby.pdf >.

[REF-1164]Ilya Kantor. "Catastrophic backtracking". 2020 December 3. < https://javascript.info/regexp-catastrophic-backtracking >.

CWE-408: Incorrect Behavior Order: Early Amplification

Weakness ID : 408
Structure : Simple
Abstraction : Base

Description

The product allows an entity to perform a legitimate but expensive operation before authentication or authorization has taken place.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	405	Asymmetric Resource Consumption (Amplification)	986
ChildOf	G	696	Incorrect Behavior Order	1527

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory)	

Scope	Impact	Likelihood
	System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.	

Demonstrative Examples

Example 1:

This function prints the contents of a specified file requested by a user.

Example Language: PHP *(Bad)*

```
function printFile($username,$filename){
  //read file into string
  $file = file_get_contents($filename);
  if ($file && isOwnerOf($username,$filename)){
    echo $file;
    return true;
  }
  else{
    echo 'You are not authorized to view this file';
  }
  return false;
}
```

This code first reads a specified file into memory, then prints the file if the user is authorized to see its contents. The read of the file into memory may be resource intensive and is unnecessary if the user is not allowed to see the file anyway.

Observed Examples

Reference	Description
CVE-2004-2458	Tool creates directories before authenticating user. *https://www.cve.org/CVERecord?id=CVE-2004-2458*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Notes

Relationship

Overlaps authentication errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Early Amplification

CWE-409: Improper Handling of Highly Compressed Data (Data Amplification)

Weakness ID : 409
Structure : Simple
Abstraction : Base

Description

The product does not handle or incorrectly handles a compressed input with a very high compression ratio that produces a large output.

Extended Description

An example of data amplification is a "decompression bomb," a small ZIP file that can produce a large amount of data when it is decompressed.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	405	Asymmetric Resource Consumption (Amplification)	986

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

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	2309

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.*	

Demonstrative Examples

Example 1:

The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is 2^0. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or 2^1. Ultimately, we reach entity THIRTYTWO, which will expand to 2^32 characters in length, or 4 GB, probably consuming far more data than expected.

Example Language: XML *(Attack)*

```
<?xml version="1.0"?>
<!DOCTYPE MaliciousDTD [
<!ENTITY ZERO "A">
<!ENTITY ONE "&ZERO;&ZERO;">
<!ENTITY TWO "&ONE;&ONE;">
...
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;">
]>
<data>&THIRTYTWO;</data>
```

Observed Examples

Reference	Description
CVE-2009-1955	XML bomb in web server module *https://www.cve.org/CVERecord?id=CVE-2009-1955*
CVE-2003-1564	Parsing library allows XML bomb *https://www.cve.org/CVERecord?id=CVE-2003-1564*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	2362
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	2444
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Data Amplification
The CERT Oracle Secure Coding Standard for Java (2011)	IDS04-J		Limit the size of files passed to ZipInputStream

CWE-410: Insufficient Resource Pool

Weakness ID : 410
Structure : Simple
Abstraction : Base

Description

The product's resource pool is not large enough to handle peak demand, which allows an attacker to prevent others from accessing the resource by using a (relatively) large number of requests for resources.

Extended Description

Frequently the consequence is a "flood" of connection or sessions.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
CanPrecede	●	400	Uncontrolled Resource Consumption	964

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Other	DoS: Crash, Exit, or Restart Other *Floods often cause a crash or other problem besides denial of the resource itself; these are likely examples of *other* vulnerabilities, not an insufficient resource pool.*	

Potential Mitigations

Phase: Architecture and Design

Do not perform resource-intensive transactions for unauthenticated users and/or invalid requests.

Phase: Architecture and Design

Consider implementing a velocity check mechanism which would detect abusive behavior.

Phase: Operation

Consider load balancing as an option to handle heavy loads.

Phase: Implementation

Make sure that resource handles are properly closed when no longer needed.

Phase: Architecture and Design

Identify the system's resource intensive operations and consider protecting them from abuse (e.g. malicious automated script which runs the resources out).

Demonstrative Examples

Example 1:

In the following snippet from a Tomcat configuration file, a JDBC connection pool is defined with a maximum of 5 simultaneous connections (with a 60 second timeout). In this case, it may be trivial for an attacker to instigate a denial of service (DoS) by using up all of the available connections in the pool.

Example Language: XML *(Bad)*

```
<Resource name="jdbc/exampledb"
auth="Container"
type="javax.sql.DataSource"
removeAbandoned="true"
removeAbandonedTimeout="30"
maxActive="5"
maxIdle="5"
maxWait="60000"
username="testuser"
password="testpass"
driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/exampledb"/>
```

Observed Examples

Reference	Description
CVE-1999-1363	Large number of locks on file exhausts the pool and causes crash.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-1999-1363
CVE-2001-1340	Product supports only one connection and does not disconnect a user who does not provide credentials. *https://www.cve.org/CVERecord?id=CVE-2001-1340*
CVE-2002-0406	Large number of connections without providing credentials allows connection exhaustion. *https://www.cve.org/CVERecord?id=CVE-2002-0406*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	2367
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	2450
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Pool
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
The CERT Oracle Secure Coding Standard for Java (2011)	TPS00-J		Use thread pools to enable graceful degradation of service during traffic bursts

References

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

CWE-412: Unrestricted Externally Accessible Lock

Weakness ID : 412
Structure : Simple
Abstraction : Base

Description

The product properly checks for the existence of a lock, but the lock can be externally controlled or influenced by an actor that is outside of the intended sphere of control.

Extended Description

This prevents the product from acting on associated resources or performing other behaviors that are controlled by the presence of the lock. Relevant locks might include an exclusive lock or mutex, or modifying a shared resource that is treated as a lock. If the lock can be held for an indefinite period of time, then the denial of service could be permanent.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	667	Improper Locking	1464
CanAlsoBe	Ⓑ	410	Insufficient Resource Pool	998

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

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	2325

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *When an attacker can control a lock, the program may wait indefinitely until the attacker releases the lock, causing a denial of service to other users of the program. This is especially problematic if there is a blocking operation on the lock.*	

Detection Methods

White Box

Automated code analysis techniques might not be able to reliably detect this weakness, since the application's behavior and general security model dictate which resource locks are critical. Interpretation of the weakness might require knowledge of the environment, e.g. if the existence of a file is used as a lock, but the file is created in a world-writable directory.

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Use any access control that is offered by the functionality that is offering the lock.

Phase: Architecture and Design

Phase: Implementation

Use unpredictable names or identifiers for the locks. This might not always be possible or feasible.

Phase: Architecture and Design

Consider modifying your code to use non-blocking synchronization methods.

Demonstrative Examples

Example 1:

This code tries to obtain a lock for a file, then writes to it.

Example Language: PHP *(Bad)*

```
function writeToLog($message){
  $logfile = fopen("logFile.log", "a");
  //attempt to get logfile lock
```

```
    if (flock($logfile, LOCK_EX)) {
        fwrite($logfile,$message);
        // unlock logfile
        flock($logfile, LOCK_UN);
    }
    else {
        print "Could not obtain lock on logFile.log, message not recorded\n";
    }
}
fclose($logFile);
```

PHP by default will wait indefinitely until a file lock is released. If an attacker is able to obtain the file lock, this code will pause execution, possibly leading to denial of service for other users. Note that in this case, if an attacker can perform an flock() on the file, they may already have privileges to destroy the log file. However, this still impacts the execution of other programs that depend on flock().

Observed Examples

Reference	Description
CVE-2001-0682	Program can not execute when attacker obtains a mutex. *https://www.cve.org/CVERecord?id=CVE-2001-0682*
CVE-2002-1914	Program can not execute when attacker obtains a lock on a critical output file. *https://www.cve.org/CVERecord?id=CVE-2002-1914*
CVE-2002-1915	Program can not execute when attacker obtains a lock on a critical output file. *https://www.cve.org/CVERecord?id=CVE-2002-1915*
CVE-2002-0051	Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition. *https://www.cve.org/CVERecord?id=CVE-2002-0051*
CVE-2000-0338	Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness. *https://www.cve.org/CVERecord?id=CVE-2000-0338*
CVE-2000-1198	Chain: Lock files with predictable names. Resultant from randomness. *https://www.cve.org/CVERecord?id=CVE-2000-1198*
CVE-2002-1869	Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close. *https://www.cve.org/CVERecord?id=CVE-2002-1869*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2320
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2339
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2366
MemberOf	C	989	SFP Secondary Cluster: Unrestricted Lock	888	2413
MemberOf	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	2449
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Relationship

This overlaps Insufficient Resource Pool when the "pool" is of size 1. It can also be resultant from race conditions, although the timing window could be quite large in some cases.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted Critical Resource Lock
7 Pernicious Kingdoms			Deadlock
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
The CERT Oracle Secure Coding Standard for Java (2011)	LCK00-J		Use private final lock objects to synchronize classes that may interact with untrusted code
The CERT Oracle Secure Coding Standard for Java (2011)	LCK07-J		Avoid deadlock by requesting and releasing locks in the same order
Software Fault Patterns	SFP22		Unrestricted lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

CWE-413: Improper Resource Locking

Weakness ID : 413
Structure : Simple
Abstraction : Base

Description

The product does not lock or does not correctly lock a resource when the product must have exclusive access to the resource.

Extended Description

When a resource is not properly locked, an attacker could modify the resource while it is being operated on by the product. This might violate the product's assumption that the resource will not change, potentially leading to unexpected behaviors.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1464
ParentOf	V	591	Sensitive Data Storage in Improperly Locked Memory	1329

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

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	2325

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	Modify Application Data DoS: Instability DoS: Crash, Exit, or Restart	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use a non-conflicting privilege scheme.

Phase: Architecture and Design

Phase: Implementation

Use synchronization when locking a resource.

Demonstrative Examples

Example 1:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

Example Language: C *(Bad)*

```
void f(pthread_mutex_t *mutex) {
  pthread_mutex_lock(mutex);
  /* access shared resource */
  pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason, the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races, correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting them to higher levels.

Example Language: C *(Good)*

```
int f(pthread_mutex_t *mutex) {
  int result;
  result = pthread_mutex_lock(mutex);
  if (0 != result)
    return result;
  /* access shared resource */
  return pthread_mutex_unlock(mutex);
}
```

Example 2:

This Java example shows a simple BankAccount class with deposit and withdraw methods.

Example Language: Java *(Bad)*

```
public class BankAccount {
   // variable for bank account balance
   private double accountBalance;
   // constructor for BankAccount
   public BankAccount() {
      accountBalance = 0;
   }
   // method to deposit amount into BankAccount
   public void deposit(double depositAmount) {
      double newBalance = accountBalance + depositAmount;
      accountBalance = newBalance;
   }
   // method to withdraw amount from BankAccount
   public void withdraw(double withdrawAmount) {
      double newBalance = accountBalance - withdrawAmount;
      accountBalance = newBalance;
   }
   // other methods for accessing the BankAccount object
   ...
}
```

However, the deposit and withdraw methods have shared access to the account balance private class variable. This can result in a race condition if multiple threads attempt to call the deposit and withdraw methods simultaneously where the account balance is modified by one thread before another thread has completed modifying the account balance. For example, if a thread attempts to withdraw funds using the withdraw method before another thread that is depositing funds using the deposit method completes the deposit then there may not be sufficient funds for the withdraw transaction.

To prevent multiple threads from having simultaneous access to the account balance variable the deposit and withdraw methods should be synchronized using the synchronized modifier.

Example Language: Java *(Good)*

```
public class BankAccount {
   ...
   // synchronized method to deposit amount into BankAccount
   public synchronized void deposit(double depositAmount) {
      ...
   }
   // synchronized method to withdraw amount from BankAccount
   public synchronized void withdraw(double withdrawAmount) {
      ...
   }
   ...
}
```

An alternative solution is to use a lock object to ensure exclusive access to the bank account balance variable. As shown below, the deposit and withdraw methods use the lock object to set a lock to block access to the BankAccount object from other threads until the method has completed updating the bank account balance variable.

Example Language: Java *(Good)*

```
public class BankAccount {
   ...
   // lock object for thread access to methods
   private ReentrantLock balanceChangeLock;
   // condition object to temporarily release lock to other threads
   private Condition sufficientFundsCondition;
   // method to deposit amount into BankAccount
   public void deposit(double amount) {
      // set lock to block access to BankAccount from other threads
      balanceChangeLock.lock();
      try {
        double newBalance = balance + amount;
        balance = newBalance;
        // inform other threads that funds are available
        sufficientFundsCondition.signalAll();
      } catch (Exception e) {...}
      finally {
        // unlock lock object
        balanceChangeLock.unlock();
      }
    }
    // method to withdraw amount from bank account
    public void withdraw(double amount) {
      // set lock to block access to BankAccount from other threads
      balanceChangeLock.lock();
      try {
        while (balance < amount) {
          // temporarily unblock access
          // until sufficient funds are available
          sufficientFundsCondition.await();
        }
        double newBalance = balance - amount;
        balance = newBalance;
      } catch (Exception e) {...}
      finally {
        // unlock lock object
        balanceChangeLock.unlock();
      }
    }
    ...
  }
```

Observed Examples

Reference	Description
CVE-2022-20141	Chain: an operating system kernel has insufficent resource locking (CWE-413) leading to a use after free (CWE-416). *https://www.cve.org/CVERecord?id=CVE-2022-20141*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2366
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2366
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2448
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Locking
The CERT Oracle Secure Coding Standard for Java (2011)	VNA00-J		Ensure visibility when accessing shared primitive variables

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
The CERT Oracle Secure Coding Standard for Java (2011)	LCK00-J		Use private final lock objects to synchronize classes that may interact with untrusted code
Software Fault Patterns	SFP19		Missing Lock

CWE-414: Missing Lock Check

Weakness ID : 414
Structure : Simple
Abstraction : Base

Description

A product does not check to see if a lock is present before performing sensitive operations on a resource.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	667	Improper Locking	1464

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

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	2325

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	Modify Application Data DoS: Instability DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Implement a reliable lock mechanism.

Observed Examples

Reference	Description
CVE-2004-1056	Product does not properly check if a lock is present, allowing other attackers to access functionality. *https://www.cve.org/CVERecord?id=CVE-2004-1056*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Lock Check
Software Fault Patterns	SFP19		Missing Lock

CWE-415: Double Free

Weakness ID : 415
Structure : Simple
Abstraction : Variant

Description

The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.

Extended Description

When a program calls free() twice with the same argument, the program's memory management data structures become corrupted. This corruption can cause the program to crash or, in some circumstances, cause two later calls to malloc() to return the same pointer. If malloc() returns the same value twice and the program later gives the attacker control over the data that is written into this doubly-allocated memory, the program becomes vulnerable to a buffer overflow attack.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1462
ChildOf	B	1341	Multiple Releases of Same Resource or Handle	2246
ChildOf	B	825	Expired Pointer Dereference	1732
PeerOf	B	123	Write-what-where Condition	323
PeerOf	V	416	Use After Free	1012
CanFollow	B	364	Signal Handler Race Condition	899

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

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1479

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

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1479

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	672	Operation on a Resource after Expiration or Release	1479

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Double-free :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Memory Execute Unauthorized Code or Commands *Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.*	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: 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.

Phase: Implementation

Use a static analysis tool to find double free instances.

Demonstrative Examples

Example 1:

The following code shows a simple example of a double free vulnerability.

Example Language: C *(Bad)*

```
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 this example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Example 2:

While contrived, this code should be exploitable on Linux distributions that do not ship with heap-chunk check summing turned on.

Example Language: C *(Bad)*

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
  char *buf1R1;
  char *buf2R1;
  char *buf1R2;
  buf1R1 = (char *) malloc(BUFSIZE2);
  buf2R1 = (char *) malloc(BUFSIZE2);
  free(buf1R1);
  free(buf2R1);
  buf1R2 = (char *) malloc(BUFSIZE1);
  strncpy(buf1R2, argv[1], BUFSIZE1-1);
  free(buf2R1);
  free(buf1R2);
}
```

Observed Examples

Reference	Description
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition (CWE-362) that leads to a double free (CWE-415). *https://www.cve.org/CVERecord?id=CVE-2006-5051*
CVE-2004-0642	Double free resultant from certain error conditions. *https://www.cve.org/CVERecord?id=CVE-2004-0642*
CVE-2004-0772	Double free resultant from certain error conditions. *https://www.cve.org/CVERecord?id=CVE-2004-0772*
CVE-2005-1689	Double free resultant from certain error conditions. *https://www.cve.org/CVERecord?id=CVE-2005-1689*
CVE-2003-0545	Double free from invalid ASN.1 encoding. *https://www.cve.org/CVERecord?id=CVE-2003-0545*
CVE-2003-1048	Double free from malformed GIF. *https://www.cve.org/CVERecord?id=CVE-2003-1048*
CVE-2005-0891	Double free from malformed GIF. *https://www.cve.org/CVERecord?id=CVE-2005-0891*

Reference	Description
CVE-2002-0059	Double free from malformed compressed data. *https://www.cve.org/CVERecord?id=CVE-2002-0059*

Affected Resources

- Memory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2323
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	2404
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2482
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Relationship

This is usually resultant from another weakness, such as an unhandled error or race condition between threads. It could also be primary to weaknesses such as buffer overflows.

Theoretical

It could be argued that Double Free would be most appropriately located as a child of "Use after Free", but "Use" and "Release" are considered to be distinct operations within vulnerability theory, therefore this is more accurately "Release of a Resource after Expiration or Release", which doesn't exist yet.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			DFREE - Double-Free Vulnerability
7 Pernicious Kingdoms			Double Free
CLASP			Doubly freeing memory
CERT C Secure Coding	MEM00-C		Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C		Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM30-C	CWE More Specific	Do not access freed memory
CERT C Secure Coding	MEM31-C		Free dynamically allocated memory exactly once
Software Fault Patterns	SFP12		Faulty Memory Release

References

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

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

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-416: Use After Free

Weakness ID : 416
Structure : Simple
Abstraction : Variant

Description

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

Extended Description

The use of previously-freed memory can have any number of adverse consequences, ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Use-after-free errors have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for freeing the memory.

In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process.

If the newly allocated data happens to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	825	Expired Pointer Dereference	1732
PeerOf	V	415	Double Free	1008
CanFollow	B	364	Signal Handler Race Condition	899
CanFollow	B	1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	2088
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	304
CanPrecede	B	123	Write-what-where Condition	323

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

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1479

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

Nature	Type	ID	Name	Page
ChildOf	🟢	672	Operation on a Resource after Expiration or Release	1479

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

Nature	Type	ID	Name	Page
ChildOf	🟢	672	Operation on a Resource after Expiration or Release	1479

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Dangling pointer :

Use-After-Free :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.*	
Availability	DoS: Crash, Exit, or Restart *If chunk consolidation occurs after the use of previously freed data, the process may crash when invalid data is used as chunk information.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *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.*	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: Implementation

When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZER1 512
#define BUFSIZER2 ((BUFSIZER1/2) - 8)
int main(int argc, char **argv) {
  char *buf1R1;
  char *buf2R1;
  char *buf2R2;
  char *buf3R2;
  buf1R1 = (char *) malloc(BUFSIZER1);
  buf2R1 = (char *) malloc(BUFSIZER1);
  free(buf2R1);
  buf2R2 = (char *) malloc(BUFSIZER2);
  buf3R2 = (char *) malloc(BUFSIZER2);
  strncpy(buf2R1, argv[1], BUFSIZER1-1);
  free(buf1R1);
  free(buf2R2);
  free(buf3R2);
}
```

Example 2:

The following code illustrates a use after free error:

Example Language: C *(Bad)*

```
char* ptr = (char*)malloc (SIZE);
if (err) {
  abrt = 1;
  free(ptr);
}
...
if (abrt) {
  logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Observed Examples

Reference	Description
CVE-2022-20141	Chain: an operating system kernel has insufficent resource locking (CWE-413) leading to a use after free (CWE-416). *https://www.cve.org/CVERecord?id=CVE-2022-20141*
CVE-2022-2621	Chain: two threads in a web browser use the same resource (CWE-366), but one of those threads can destroy the resource before the other has completed (CWE-416). *https://www.cve.org/CVERecord?id=CVE-2022-2621*
CVE-2021-0920	Chain: mobile platform race condition (CWE-362) leading to use-after-free (CWE-416), as exploited in the wild per CISA KEV.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2021-0920
CVE-2020-6819	Chain: race condition (CWE-362) leads to use-after-free (CWE-416), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-6819*
CVE-2010-4168	Use-after-free triggered by closing a connection while data is still being transmitted. *https://www.cve.org/CVERecord?id=CVE-2010-4168*
CVE-2010-2941	Improper allocation for invalid data leads to use-after-free. *https://www.cve.org/CVERecord?id=CVE-2010-2941*
CVE-2010-2547	certificate with a large number of Subject Alternate Names not properly handled in realloc, leading to use-after-free *https://www.cve.org/CVERecord?id=CVE-2010-2547*
CVE-2010-1772	Timers are not disabled when a related object is deleted *https://www.cve.org/CVERecord?id=CVE-2010-1772*
CVE-2010-1437	Access to a "dead" object that is being cleaned up *https://www.cve.org/CVERecord?id=CVE-2010-1437*
CVE-2010-1208	object is deleted even with a non-zero reference count, and later accessed *https://www.cve.org/CVERecord?id=CVE-2010-1208*
CVE-2010-0629	use-after-free involving request containing an invalid version number *https://www.cve.org/CVERecord?id=CVE-2010-0629*
CVE-2010-0378	unload of an object that is currently being accessed by other functionality *https://www.cve.org/CVERecord?id=CVE-2010-0378*
CVE-2010-0302	incorrectly tracking a reference count leads to use-after-free *https://www.cve.org/CVERecord?id=CVE-2010-0302*
CVE-2010-0249	use-after-free related to use of uninitialized memory *https://www.cve.org/CVERecord?id=CVE-2010-0249*
CVE-2010-0050	HTML document with incorrectly-nested tags *https://www.cve.org/CVERecord?id=CVE-2010-0050*
CVE-2009-3658	Use after free in ActiveX object by providing a malformed argument to a method *https://www.cve.org/CVERecord?id=CVE-2009-3658*
CVE-2009-3616	use-after-free by disconnecting during data transfer, or a message containing incorrect data types *https://www.cve.org/CVERecord?id=CVE-2009-3616*
CVE-2009-3553	disconnect during a large data transfer causes incorrect reference count, leading to use-after-free *https://www.cve.org/CVERecord?id=CVE-2009-3553*
CVE-2009-2416	use-after-free found by fuzzing *https://www.cve.org/CVERecord?id=CVE-2009-2416*
CVE-2009-1837	Chain: race condition (CWE-362) from improper handling of a page transition in web client while an applet is loading (CWE-368) leads to use after free (CWE-416) *https://www.cve.org/CVERecord?id=CVE-2009-1837*
CVE-2009-0749	realloc generates new buffer and pointer, but previous pointer is still retained, leading to use after free *https://www.cve.org/CVERecord?id=CVE-2009-0749*
CVE-2010-3328	Use-after-free in web browser, probably resultant from not initializing memory. *https://www.cve.org/CVERecord?id=CVE-2010-3328*
CVE-2008-5038	use-after-free when one thread accessed memory that was freed by another thread *https://www.cve.org/CVERecord?id=CVE-2008-5038*
CVE-2008-0077	assignment of malformed values to certain properties triggers use after free *https://www.cve.org/CVERecord?id=CVE-2008-0077*

Reference	Description
CVE-2006-4434	mail server does not properly handle a long header. *https://www.cve.org/CVERecord?id=CVE-2006-4434*
CVE-2010-2753	chain: integer overflow leads to use-after-free *https://www.cve.org/CVERecord?id=CVE-2010-2753*
CVE-2006-4997	freed pointer dereference *https://www.cve.org/CVERecord?id=CVE-2006-4997*

Affected Resources

- Memory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2323
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	983	SFP Secondary Cluster: Faulty Resource Use	888	2410
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SI-1
7 Pernicious Kingdoms			Use After Free
CLASP			Using freed memory
CERT C Secure Coding	MEM00-C		Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C		Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM30-C	Exact	Do not access freed memory
Software Fault Patterns	SFP15		Faulty Resource Use

References

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

papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security %20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https:// cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

CWE-419: Unprotected Primary Channel

Weakness ID : 419
Structure : Simple
Abstraction : Base

Description

The product uses a primary channel for administration or restricted functionality, but it does not properly protect the channel.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	923	Improper Restriction of Communication Channel to Intended Endpoints	1827

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2325

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Do not expose administrative functionnality on the user UI.

Phase: Architecture and Design

Protect the administrative/restricted functionality with a strong authentication mechanism.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	2397
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unprotected Primary Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
383	Harvesting Information via API Event Monitoring

CWE-420: Unprotected Alternate Channel

Weakness ID : 420
Structure : Simple
Abstraction : Base

Description

The product protects a primary channel, but it does not use the same level of protection for an alternate channel.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	923	Improper Restriction of Communication Channel to Intended Endpoints	1827
ParentOf	B	421	Race Condition During Access to Alternate Channel	1020
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	1022
ParentOf	B	1299	Missing Protection Mechanism for Alternate Hardware Interface	2162
PeerOf	B	288	Authentication Bypass Using an Alternate Path or Channel	700

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2325

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Identify all alternate channels and use the same protection mechanisms that are used for the primary channels.

Demonstrative Examples

Example 1:

Register SECURE_ME is located at address 0xF00. A mirror of this register called COPY_OF_SECURE_ME is at location 0x800F00. The register SECURE_ME is protected from malicious agents and only allows access to select, while COPY_OF_SECURE_ME is not.

Access control is implemented using an allowlist (as indicated by acl_oh_allowlist). The identity of the initiator of the transaction is indicated by the one hot input, incoming_id. This is checked against the acl_oh_allowlist (which contains a list of initiators that are allowed to access the asset).

Though this example is shown in Verilog, it will apply to VHDL as well.

Example Language: Verilog *(Informative)*

```
module foo_bar(data_out, data_in, incoming_id, address, clk, rst_n);
output [31:0] data_out;
input [31:0] data_in, incoming_id, address;
input clk, rst_n;
wire write_auth, addr_auth;
reg [31:0] data_out, acl_oh_allowlist, q;
assign write_auth = | (incoming_id & acl_oh_allowlist) ? 1 : 0;
always @*
  acl_oh_allowlist <= 32'h8312;
assign addr_auth = (address == 32'hF00) ? 1: 0;
always @ (posedge clk or negedge rst_n)
  if (!rst_n)
    begin
      q <= 32'h0;
      data_out <= 32'h0;
    end
  else
    begin
      q <= (addr_auth & write_auth) ? data_in: q;
      data_out <= q;
    end
  end
endmodule
```

Example Language: Verilog *(Bad)*

```
assign addr_auth = (address == 32'hF00) ? 1: 0;
```

The bugged line of code is repeated in the Bad example above. The weakness arises from the fact that the SECURE_ME register can be modified by writing to the shadow register COPY_OF_SECURE_ME. The address of COPY_OF_SECURE_ME should also be included in the check. That buggy line of code should instead be replaced as shown in the Good Code Snippet below.

Example Language: Verilog *(Good)*

```
assign addr_auth = (address == 32'hF00 || address == 32'h800F00) ? 1: 0;
```

Observed Examples

Reference	Description
CVE-2020-8004	When the internal flash is protected by blocking access on the Data Bus (DBUS), it can still be indirectly accessed through the Instruction Bus (IBUS). *https://www.cve.org/CVERecord?id=CVE-2020-8004*
CVE-2002-0567	DB server assumes that local clients have performed authentication, allowing attacker to directly connect to a process to load libraries and execute commands; a socket interface also exists (another alternate channel), so attack can be remote. *https://www.cve.org/CVERecord?id=CVE-2002-0567*
CVE-2002-1578	Product does not restrict access to underlying database, so attacker can bypass restrictions by directly querying the database. *https://www.cve.org/CVERecord?id=CVE-2002-1578*
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing. *https://www.cve.org/CVERecord?id=CVE-2003-1035*
CVE-2002-1863	FTP service can not be disabled even when other access controls would require it. *https://www.cve.org/CVERecord?id=CVE-2002-1863*
CVE-2002-0066	Windows named pipe created without authentication/access control, allowing configuration modification. *https://www.cve.org/CVERecord?id=CVE-2002-0066*
CVE-2004-1461	Router management interface spawns a separate TCP connection after authentication, allowing hijacking by attacker coming from the same IP address. *https://www.cve.org/CVERecord?id=CVE-2004-1461*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	2397
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Relationship

This can be primary to authentication errors, and resultant from unhandled error conditions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unprotected Alternate Channel

CWE-421: Race Condition During Access to Alternate Channel

Weakness ID : 421
Structure : Simple
Abstraction : Base

Description

The product opens an alternate channel to communicate with an authorized user, but the channel is accessible to other actors.

Extended Description

This creates a race condition that allows an attacker to access the channel before the authorized user does.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
ChildOf	B	420	Unprotected Alternate Channel	1018

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-1999-0351	FTP "Pizza Thief" vulnerability. Attacker can connect to a port that was intended for use by another client. *https://www.cve.org/CVERecord?id=CVE-1999-0351*
CVE-2003-0230	Product creates Windows named pipe during authentication that another attacker can hijack by connecting to it. *https://www.cve.org/CVERecord?id=CVE-2003-0230*

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	2397
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate Channel Race Condition

References

[REF-354]Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". 2002 April. < https://www.blakewatts.com/blog/discovering-and-exploiting-named-pipe-security-flaws-for-fun-and-profit >.2023-04-07.

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

CWE-422: Unprotected Windows Messaging Channel ('Shatter')

Weakness ID : 422
Structure : Simple
Abstraction : Variant

Description

The product does not properly verify the source of a message in the Windows Messaging System while running at elevated privileges, creating an alternate channel through which an attacker can directly send a message to the product.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	360	Trust of System Event Data	887
ChildOf	B	420	Unprotected Alternate Channel	1018

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Always verify and authenticate the source of the message.

Observed Examples

Reference	Description
CVE-2002-0971	Bypass GUI and access restricted dialog box. *https://www.cve.org/CVERecord?id=CVE-2002-0971*
CVE-2002-1230	Gain privileges via Windows message. *https://www.cve.org/CVERecord?id=CVE-2002-1230*
CVE-2003-0350	A control allows a change to a pointer for a callback function using Windows message. *https://www.cve.org/CVERecord?id=CVE-2003-0350*
CVE-2003-0908	Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access "open file" dialog.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2003-0908
CVE-2004-0213	Attacker uses Shatter attack to bypass GUI-enforced protection for CVE-2003-0908. *https://www.cve.org/CVERecord?id=CVE-2004-0213*
CVE-2004-0207	User can call certain API functions to modify certain properties of privileged programs. *https://www.cve.org/CVERecord?id=CVE-2004-0207*

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	953	SFP Secondary Cluster: Missing Endpoint Authentication	888	2397
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Relationship

Overlaps privilege errors and UI errors.

Research Gap

Possibly under-reported, probably under-studied. It is suspected that a number of publicized vulnerabilities that involve local privilege escalation on Windows systems may be related to Shatter attacks, but they are not labeled as such. Alternate channel attacks likely exist in other operating systems and messaging models, e.g. in privileged X Windows applications, but examples are not readily available.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unprotected Windows Messaging Channel ('Shatter')
Software Fault Patterns	SFP30		Missing endpoint authentication

References

[REF-402]Paget. "Exploiting design flaws in the Win32 API for privilege escalation. Or... Shatter Attacks - How to break Windows". 2002 August. < http://web.archive.org/web/20060115174629/http://security.tombom.co.uk/shatter.html >.

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

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

CWE-424: Improper Protection of Alternate Path

Weakness ID : 424
Structure : Simple
Abstraction : Class

Description

The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520
ChildOf	C	638	Not Using Complete Mediation	1404
ParentOf	B	425	Direct Request ('Forced Browsing')	1025

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Deploy different layers of protection to implement security in depth.

Observed Examples

Reference	Description
CVE-2022-29238	Access-control setting in web-based document collaboration tool is not properly implemented by the code, which prevents listing hidden directories but does not prevent direct requests to files in those directories. *https://www.cve.org/CVERecord?id=CVE-2022-29238*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	945	SFP Secondary Cluster: Insecure Resource Access	888	2394
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate Path Errors
Software Fault Patterns	SFP35		Insecure resource access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
127	Directory Indexing
554	Functionality Bypass

CWE-425: Direct Request ('Forced Browsing')

Weakness ID : 425
Structure : Simple
Abstraction : Base

Description

The web application does not adequately enforce appropriate authorization on all restricted URLs, scripts, or files.

Extended Description

Web applications susceptible to direct request attacks often make the false assumption that such resources can only be reached through a given navigation path and so only apply authorization at certain points in the path.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	288	Authentication Bypass Using an Alternate Path or Channel	700
ChildOf	C	424	Improper Protection of Alternate Path	1023
ChildOf	C	862	Missing Authorization	1780
CanPrecede	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	236
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	1121

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

Nature	Type	ID	Name	Page
ChildOf	C	862	Missing Authorization	1780

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2476
MemberOf	C	417	Communication Channel Errors	2325

Applicable Platforms

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

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

forced browsing : The "forced browsing" term could be misinterpreted to include weaknesses such as CSRF or XSS, so its use is discouraged.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Read Application Data Modify Application Data Execute Unauthorized Code or Commands Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Apply appropriate access control authorizations for each access to all restricted URLs, scripts or files.

Phase: Architecture and Design

Consider using MVC based frameworks such as Struts.

Demonstrative Examples

Example 1:

If forced browsing is possible, an attacker may be able to directly access a sensitive page by entering a URL similar to the following.

Example Language: JSP *(Attack)*

```
http://somesite.com/someapplication/admin.jsp
```

Observed Examples

Reference	Description
CVE-2022-29238	Access-control setting in web-based document collaboration tool is not properly implemented by the code, which prevents listing hidden directories but does not prevent direct requests to files in those directories. *https://www.cve.org/CVERecord?id=CVE-2022-29238*
CVE-2022-23607	Python-based HTTP library did not scope cookies to a particular domain such that "supercookies" could be sent to any domain on redirect. *https://www.cve.org/CVERecord?id=CVE-2022-23607*
CVE-2004-2144	Bypass authentication via direct request. *https://www.cve.org/CVERecord?id=CVE-2004-2144*
CVE-2005-1892	Infinite loop or infoleak triggered by direct requests. *https://www.cve.org/CVERecord?id=CVE-2005-1892*
CVE-2004-2257	Bypass auth/auth via direct request. *https://www.cve.org/CVERecord?id=CVE-2004-2257*
CVE-2005-1688	Direct request leads to infoleak by error. *https://www.cve.org/CVERecord?id=CVE-2005-1688*
CVE-2005-1697	Direct request leads to infoleak by error. *https://www.cve.org/CVERecord?id=CVE-2005-1697*
CVE-2005-1698	Direct request leads to infoleak by error. *https://www.cve.org/CVERecord?id=CVE-2005-1698*
CVE-2005-1685	Authentication bypass via direct request. *https://www.cve.org/CVERecord?id=CVE-2005-1685*
CVE-2005-1827	Authentication bypass via direct request. *https://www.cve.org/CVERecord?id=CVE-2005-1827*
CVE-2005-1654	Authorization bypass using direct request.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2005-1654
CVE-2005-1668	Access privileged functionality using direct request. *https://www.cve.org/CVERecord?id=CVE-2005-1668*
CVE-2002-1798	Upload arbitrary files via direct request. *https://www.cve.org/CVERecord?id=CVE-2002-1798*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	2333
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2334
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf	C	953	SFP Secondary Cluster: Missing Endpoint Authentication	888	2397
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	2437
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Relationship

Overlaps Modification of Assumed-Immutable Data (MAID), authorization errors, container errors; often primary to other weaknesses such as XSS and SQL injection.

Theoretical

"Forced browsing" is a step-based manipulation involving the omission of one or more steps, whose order is assumed to be immutable. The application does not verify that the first step was performed successfully before the second step. The consequence is typically "authentication bypass" or "path disclosure," although it can be primary to all kinds of weaknesses, especially in languages such as PHP, which allow external modification of assumed-immutable variables.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Request aka 'Forced Browsing'
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
WASC	34		Predictable Resource Location
Software Fault Patterns	SFP30		Missing endpoint authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
87	Forceful Browsing
127	Directory Indexing
143	Detect Unpublicized Web Pages
144	Detect Unpublicized Web Services
668	Key Negotiation of Bluetooth Attack (KNOB)

CWE-426: Untrusted Search Path

Weakness ID : 426
Structure : Simple
Abstraction : Base

Description

The product searches for critical resources using an externally-supplied search path that can point to resources that are not under the product's direct control.

Extended Description

This might allow attackers to execute their own programs, access unauthorized data files, or modify configuration in unexpected ways. If the product uses a search path to locate critical resources such as programs, then an attacker could modify that search path to point to a malicious program, which the targeted product would then execute. The problem extends to any type of critical resource that the product trusts.

Some of the most common variants of untrusted search path are:

- In various UNIX and Linux-based systems, the PATH environment variable may be consulted to locate executable programs, and LD_PRELOAD may be used to locate a separate library.
- In various Microsoft-based systems, the PATH environment variable is consulted to locate a DLL, if the DLL is not found in other paths that appear earlier in the search order.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	673	External Influence of Sphere Definition	1483
ChildOf	C	642	External Control of Critical State Data	1414
PeerOf	B	427	Uncontrolled Search Path Element	1033
PeerOf	B	428	Unquoted Search Path or Element	1039

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

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

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2480

Applicable Platforms

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

Operating_System : Not OS-Specific *(Prevalence = Undetermined)*

Alternate Terms

Untrusted Path :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Gain Privileges or Assume Identity Execute Unauthorized Code or Commands *There is the potential for arbitrary code execution with privileges of the vulnerable program.*	
Availability	DoS: Crash, Exit, or Restart *The program could be redirected to the wrong files, potentially triggering a crash or hang when the targeted file is too large or does not have the expected format.*	
Confidentiality	Read Files or Directories *The program could send the output of unauthorized files to the attacker.*	

Detection Methods

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and look for library functions and system calls that suggest when a search path is being used. One pattern is when the program performs multiple accesses of the same file but in different directories, with repeated failures until the proper filename is found. Library calls such as getenv() or their equivalent can be checked to see if any path-related variables are being accessed.

Automated Static Analysis

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

Effectiveness = High

Manual Analysis

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428.

Phase: Implementation

When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths.

Phase: Implementation

Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH, and other settings that identify the location of code libraries, and any application-specific search paths.

Phase: Implementation

Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory.

Phase: Implementation

Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of it, while execl() and execv() require a full path.

Demonstrative Examples

Example 1:

This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

Example Language: C *(Bad)*

```
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

Example Language: *(Attack)*

- The user sets the PATH to reference a directory under the attacker's control, such as "/my/dir/".
- The attacker creates a malicious program called "ls", and puts that program in /my/dir
- The user executes the program.
- When system() is executed, the shell consults the PATH to find the ls program

- The program finds the attacker's malicious program, "/my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".
- The program executes the attacker's malicious program with the raised privileges.

Example 2:

This code prints all of the running processes belonging to the current user.

Example Language: PHP *(Bad)*

```
//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (avoiding CWE-78)
$userName = getCurrentUser();
$command = 'ps aux | grep ' . $userName;
system($command);
```

If invoked by an unauthorized web user, it is providing a web page of potentially sensitive information on the underlying system, such as command-line arguments (CWE-497). This program is also potentially vulnerable to a PATH based attack (CWE-426), as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

Example 3:

The following code is from a web application that allows users access to an interface through which they can update their password on the system. In this environment, user passwords can be managed using the Network Information System (NIS), which is commonly used on UNIX systems. When performing NIS updates, part of the process for updating passwords is to run a make command in the /var/yp directory. Performing NIS updates requires extra privileges.

Example Language: Java *(Bad)*

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

The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

Observed Examples

Reference	Description
CVE-1999-1120	Application relies on its PATH environment variable to find and execute program. *https://www.cve.org/CVERecord?id=CVE-1999-1120*
CVE-2008-1810	Database application relies on its PATH environment variable to find and execute program. *https://www.cve.org/CVERecord?id=CVE-2008-1810*
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages. *https://www.cve.org/CVERecord?id=CVE-2007-2027*
CVE-2008-3485	Untrusted search path using malicious .EXE in Windows environment. *https://www.cve.org/CVERecord?id=CVE-2008-3485*

Reference	Description
CVE-2008-2613	setuid program allows compromise using path that finds and loads a malicious library. *https://www.cve.org/CVERecord?id=CVE-2008-2613*
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded. *https://www.cve.org/CVERecord?id=CVE-2008-1319*

Functional Areas

- Program Invocation
- Code Libraries

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	2348
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2378
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Untrusted Search Path
CLASP			Relative path library search
CERT C Secure Coding	ENV03-C		Sanitize the environment when invoking external programs

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
38	Leveraging/Manipulating Configuration File Search Paths

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

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

[REF-176]Michael Howard and David LeBlanc. "Writing Secure Code". 1st Edition. 2001 November 3. Microsoft Press.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

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

CWE-427: Uncontrolled Search Path Element

Weakness ID : 427
Structure : Simple
Abstraction : Base

Description

The product uses a fixed or controlled search path to find resources, but one or more locations in that path can be under the control of unintended actors.

Extended Description

Although this weakness can occur with any type of resource, it is frequently introduced when a product uses a directory search path to find executables or code libraries, but the path contains a directory that can be modified by an attacker, such as "/tmp" or the current working directory.

In Windows-based systems, when the LoadLibrary or LoadLibraryEx function is called with a DLL name that does not contain a fully qualified path, the function follows a search order that includes two path elements that might be uncontrolled:

- the directory from which the program has been loaded
- the current working directory

In some cases, the attack can be conducted remotely, such as when SMB or WebDAV network shares are used.

One or more locations in that path could include the Windows drive root or its subdirectories. This often exists in Linux-based code assuming the controlled nature of the root directory (/) or its subdirectories (/etc, etc), or a code that recursively accesses the parent directory. In Windows, the drive root and some of its subdirectories have weak permissions by default, which makes them uncontrolled.

In some Unix-based systems, a PATH might be created that contains an empty element, e.g. by splicing an empty variable into the PATH. This empty element can be interpreted as equivalent to the current working directory, which might be an untrusted search element.

In software package management frameworks (e.g., npm, RubyGems, or PyPi), the framework may identify dependencies on third-party libraries or other packages, then consult a repository that contains the desired package. The framework may search a public repository before a private repository. This could be exploited by attackers by placing a malicious package in the public repository that has the same name as a package from the private repository. The search path might not be directly under control of the developer relying on the framework, but this search order effectively contains an untrusted element.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1469
PeerOf	B	426	Untrusted Search Path	1028

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

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

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

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2480

Applicable Platforms

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

Operating_System : Not OS-Specific *(Prevalence = Undetermined)*

Alternate Terms

DLL preloading : This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

Binary planting : This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

Insecure library loading : This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

Dependency confusion : As of February 2021, this term is used to describe CWE-427 in the context of managing installation of software package dependencies, in which attackers release packages on public sites where the names are the same as package names used by private repositories, and the search for the dependent package tries the public site first, downloading untrusted code. It may also be referred to as a "substitution attack."

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428.

Phase: Implementation

Strategy = Attack Surface Reduction

When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths.

Phase: Implementation

Strategy = Attack Surface Reduction

Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH, and other settings that identify the location of code libraries, and any application-specific search paths.

Phase: Implementation

Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory. Since this is a denylist approach, it might not be a complete solution.

Phase: Implementation

Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of finding the program using the PATH environment variable, while execl() and execv() require a full path.

Demonstrative Examples

Example 1:

The following code is from a web application that allows users access to an interface through which they can update their password on the system. In this environment, user passwords can be managed using the Network Information System (NIS), which is commonly used on UNIX systems. When performing NIS updates, part of the process for updating passwords is to run a make command in the /var/yp directory. Performing NIS updates requires extra privileges.

Example Language: Java *(Bad)*

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

The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

Example 2:

In versions of Go prior to v1.19, the LookPath function would follow the conventions of the runtime OS and look for a program in the directiories listed in the current path [REF-1325].

Therefore, Go would prioritize searching the current directory when the provided command name does not contain a directory separator and continued to search for programs even when the specified program name is empty.

Consider the following where an application executes a git command to run on the system.

Example Language: Go *(Bad)*

```
func ExecuteGitCommand(name string, arg []string) error {
  c := exec.Command(name, arg...)
  var err error
  c.Path, err = exec.LookPath(name)
  if err != nil {
    return err
  }
}
```

An attacker could create a malicious repository with a file named ..exe and another file named git.exe. If git.exe is not found in the system PATH, then ..exe would execute [REF-1326].

Example 3:

In February 2021 [REF-1169], a researcher was able to demonstrate the ability to breach major technology companies by using "dependency confusion" where the companies would download and execute untrusted packages.

The researcher discovered the names of some internal, private packages by looking at dependency lists in public source code, such as package.json. The researcher then created new, untrusted packages with the same name as the internal packages, then uploaded them to package hosting services. These services included the npm registry for Node, PyPi for Python, and RubyGems. In affected companies, their dependency resolution would search the public hosting services first before consulting their internal service, causing the untrusted packages to be automatically downloaded and executed.

Observed Examples

Reference	Description
CVE-2023-25815	chain: a change in an underlying package causes the gettext function to use implicit initialization with a hard-coded path (CWE-1419) under the user-writable C:\ drive, introducing an untrusted search path element (CWE-427) that enables spoofing of messages. *https://www.cve.org/CVERecord?id=CVE-2023-25815*
CVE-2022-4826	Go-based git extension on Windows can search for and execute a malicious "..exe" in a repository because Go searches the current working directory if git.exe is not found in the PATH *https://www.cve.org/CVERecord?id=CVE-2022-4826*
CVE-2020-26284	A Static Site Generator built in Go, when running on Windows, searches the current working directory for a command, possibly allowing code execution using a malicious .exe or .bat file with the name being searched *https://www.cve.org/CVERecord?id=CVE-2020-26284*
CVE-2022-24765	Windows-based fork of git creates a ".git" folder in the C: drive, allowing local attackers to create a .git folder with a malicious config file *https://www.cve.org/CVERecord?id=CVE-2022-24765*
CVE-2019-1552	SSL package searches under "C:/usr/local" for configuration files and other critical data, but C:/usr/local might be world-writable. *https://www.cve.org/CVERecord?id=CVE-2019-1552*
CVE-2010-3402	"DLL hijacking" issue in document editor. *https://www.cve.org/CVERecord?id=CVE-2010-3402*

Reference	Description
CVE-2010-3397	"DLL hijacking" issue in encryption software. *https://www.cve.org/CVERecord?id=CVE-2010-3397*
CVE-2010-3138	"DLL hijacking" issue in library used by multiple media players. *https://www.cve.org/CVERecord?id=CVE-2010-3138*
CVE-2010-3152	"DLL hijacking" issue in illustration program. *https://www.cve.org/CVERecord?id=CVE-2010-3152*
CVE-2010-3147	"DLL hijacking" issue in address book. *https://www.cve.org/CVERecord?id=CVE-2010-3147*
CVE-2010-3135	"DLL hijacking" issue in network monitoring software. *https://www.cve.org/CVERecord?id=CVE-2010-3135*
CVE-2010-3131	"DLL hijacking" issue in web browser. *https://www.cve.org/CVERecord?id=CVE-2010-3131*
CVE-2010-1795	"DLL hijacking" issue in music player/organizer. *https://www.cve.org/CVERecord?id=CVE-2010-1795*
CVE-2002-1576	Product uses the current working directory to find and execute a program, which allows local users to gain privileges by creating a symlink that points to a malicious version of the program. *https://www.cve.org/CVERecord?id=CVE-2002-1576*
CVE-1999-1461	Product trusts the PATH environmental variable to find and execute a program, which allows local users to obtain root access by modifying the PATH to point to a malicous version of that program. *https://www.cve.org/CVERecord?id=CVE-1999-1461*
CVE-1999-1318	Software uses a search path that includes the current working directory (.), which allows local users to gain privileges via malicious programs. *https://www.cve.org/CVERecord?id=CVE-1999-1318*
CVE-2003-0579	Admin software trusts the user-supplied -uv.install command line option to find and execute the uv.install program, which allows local users to gain privileges by providing a pathname that is under control of the user. *https://www.cve.org/CVERecord?id=CVE-2003-0579*
CVE-2000-0854	When a document is opened, the directory of that document is first used to locate DLLs , which could allow an attacker to execute arbitrary commands by inserting malicious DLLs into the same directory as the document. *https://www.cve.org/CVERecord?id=CVE-2000-0854*
CVE-2001-0943	Database trusts the PATH environment variable to find and execute programs, which allows local users to modify the PATH to point to malicious programs. *https://www.cve.org/CVERecord?id=CVE-2001-0943*
CVE-2001-0942	Database uses an environment variable to find and execute a program, which allows local users to execute arbitrary programs by changing the environment variable. *https://www.cve.org/CVERecord?id=CVE-2001-0942*
CVE-2001-0507	Server uses relative paths to find system files that will run in-process, which allows local users to gain privileges via a malicious file. *https://www.cve.org/CVERecord?id=CVE-2001-0507*
CVE-2002-2017	Product allows local users to execute arbitrary code by setting an environment variable to reference a malicious program. *https://www.cve.org/CVERecord?id=CVE-2002-2017*
CVE-1999-0690	Product includes the current directory in root's PATH variable. *https://www.cve.org/CVERecord?id=CVE-1999-0690*
CVE-2001-0912	Error during packaging causes product to include a hard-coded, non-standard directory in search path. *https://www.cve.org/CVERecord?id=CVE-2001-0912*
CVE-2001-0289	Product searches current working directory for configuration file. *https://www.cve.org/CVERecord?id=CVE-2001-0289*

Reference	Description
CVE-2005-1705	Product searches current working directory for configuration file. *https://www.cve.org/CVERecord?id=CVE-2005-1705*
CVE-2005-1307	Product executable other program from current working directory. *https://www.cve.org/CVERecord?id=CVE-2005-1307*
CVE-2002-2040	Untrusted path. *https://www.cve.org/CVERecord?id=CVE-2002-2040*
CVE-2005-2072	Modification of trusted environment variable leads to untrusted path vulnerability. *https://www.cve.org/CVERecord?id=CVE-2005-2072*
CVE-2005-1632	Product searches /tmp for modules before other paths. *https://www.cve.org/CVERecord?id=CVE-2005-1632*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	2416
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Relationship

Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere (i.e., modification of a search path), this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control (i.e., the search path cannot be modified by an attacker, but one element of the path can be under attacker control).

Theoretical

This weakness is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model might need enhancement or clarification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Uncontrolled Search Path Element

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
38	Leveraging/Manipulating Configuration File Search Paths
471	Search Order Hijacking

References

[REF-409]Georgi Guninski. "Double clicking on MS Office documents from Windows Explorer may execute arbitrary programs in some cases". Bugtraq. 2000 September 8. < https://seclists.org/bugtraq/2000/Sep/331 >.2023-01-30.

[REF-410]Mitja Kolsek. "ACROS Security: Remote Binary Planting in Apple iTunes for Windows (ASPR #2010-08-18-1)". Bugtraq. 2010 August 8. < https://lists.openwall.net/bugtraq/2010/08/18/4 >.2023-01-30.

[REF-411]Taeho Kwon and Zhendong Su. "Automatic Detection of Vulnerable Dynamic Component Loadings". < https://dl.acm.org/doi/10.1145/1831708.1831722 >.2023-04-07.

[REF-412]"Dynamic-Link Library Search Order". 2010 September 2. Microsoft. < https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order?redirectedfrom=MSDN >.2023-04-07.

[REF-413]"Dynamic-Link Library Security". 2010 September 2. Microsoft. < https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security >.2023-04-07.

[REF-414]"An update on the DLL-preloading remote attack vector". 2010 August 1. Microsoft. < https://msrc.microsoft.com/blog/2010/08/an-update-on-the-dll-preloading-remote-attack-vector/ >.2023-04-07.

[REF-415]"Insecure Library Loading Could Allow Remote Code Execution". 2010 August 3. Microsoft. < https://learn.microsoft.com/en-us/security-updates/securityadvisories/2010/2269637#insecure-library-loading-could-allow-remote-code-execution >.2023-04-07.

[REF-416]HD Moore. "Application DLL Load Hijacking". 2010 August 3. < https://www.rapid7.com/blog/?p=5325 >.2023-04-07.

[REF-417]Oliver Lavery. "DLL Hijacking: Facts and Fiction". 2010 August 6. < https://threatpost.com/dll-hijacking-facts-and-fiction-082610/74384/ >.2023-04-07.

[REF-1168]Catalin Cimpanu. "Microsoft warns enterprises of new 'dependency confusion' attack technique". ZDNet. 2021 February 0. < https://www.zdnet.com/article/microsoft-warns-enterprises-of-new-dependency-confusion-attack-technique/ >.

[REF-1169]Alex Birsan. "Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies". 2021 February 9. < https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610 >.

[REF-1170]Microsoft. "3 Ways to Mitigate Risk When Using Private Package Feeds". 2021 February 9. < https://azure.microsoft.com/mediahandler/files/resourcefiles/3-ways-to-mitigate-risk-using-private-package-feeds/3%20Ways%20to%20Mitigate%20Risk%20When%20Using%20Private%20Package%20Feeds%20-%20v1.0.pdf >.

[REF-1325]"exec package - os/exec - Go Packages". 2023 April 4. < https://pkg.go.dev/os/exec >.2023-04-21.

[REF-1326]Brian M. Carlson. "Git LFS Changelog". 2022 April 9. < https://github.com/git-lfs/git-lfs/commit/032dca8ee69c193208cd050024c27e82e11aef81 >.2023-04-21.

CWE-428: Unquoted Search Path or Element

Weakness ID : 428
Structure : Simple
Abstraction : Base

Description

The product uses a search path that contains an unquoted element, in which the element contains whitespace or other separators. This can cause the product to access resources in a parent path.

Extended Description

If a malicious individual has access to the file system, it is possible to elevate privileges by inserting such a file as "C:\Program.exe" to be run by a privileged program making use of WinExec.

Relationships

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

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

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

Nature	Type	ID	Name	Page
PeerOf	B	426	Untrusted Search Path	1028

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

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

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

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2480

Applicable Platforms

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

Operating_System : Windows NT *(Prevalence = Sometimes)*

Operating_System : macOS *(Prevalence = Rarely)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Properly quote the full search path before executing a program on the system.

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation

Strategy = Input Validation

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

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
UINT errCode = WinExec( "C:\\Program Files\\Foo\\Bar", SW_SHOW );
```

Observed Examples

Reference	Description
CVE-2005-1185	Small handful of others. Program doesn't quote the "C:\Program Files\" path when calling a program to be executed - or any other path with a directory or file whose name contains a space - so attacker can put a malicious program.exe into C:. *https://www.cve.org/CVERecord?id=CVE-2005-1185*
CVE-2005-2938	CreateProcess() and CreateProcessAsUser() can be misused by applications to allow "program.exe" style attacks in C: *https://www.cve.org/CVERecord?id=CVE-2005-2938*
CVE-2000-1128	Applies to "Common Files" folder, with a malicious common.exe, instead of "Program Files"/program.exe. *https://www.cve.org/CVERecord?id=CVE-2000-1128*

Functional Areas

- Program Invocation

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	2409
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Applicable Platform

This weakness could apply to any OS that supports spaces in filenames, especially any OS that make it easy for a user to insert spaces into filenames or folders, such as Windows. While spaces are technically supported in Unix, the practice is generally avoided. .

Maintenance

This weakness primarily involves the lack of quoting, which is not explicitly stated as a part of CWE-116. CWE-116 also describes output in light of structured messages, but the generation of a filename or search path (as in this weakness) might not be considered a structured message. An additional complication is the relationship to control spheres. Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere, this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control. This is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model needs enhancement or clarification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unquoted Search Path or Element

References

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

CWE-430: Deployment of Wrong Handler

Weakness ID : 430
Structure : Simple
Abstraction : Base

Description

The wrong "handler" is assigned to process an object.

Extended Description

An example of deploying the wrong handler would be calling a servlet to reveal source code of a .JSP file, or automatically "determining" type of the object even if it is contradictory to an explicitly specified type.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	1048
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	1048
CanPrecede	V	433	Unparsed Raw Web Content Delivery	1046

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

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	2326

Weakness Ordinalities

Resultant : This weakness is usually resultant from other weaknesses.

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Perform a type check before interpreting an object.

Phase: Architecture and Design

Reject any inconsistent types, such as a file with a .GIF extension that appears to consist of PHP code.

Observed Examples

Reference	Description
CVE-2001-0004	Source code disclosure via manipulated file extension that causes parsing by wrong DLL. *https://www.cve.org/CVERecord?id=CVE-2001-0004*

Reference	Description
CVE-2002-0025	Web browser does not properly handle the Content-Type header field, causing a different application to process the document. *https://www.cve.org/CVERecord?id=CVE-2002-0025*
CVE-2000-1052	Source code disclosure by directly invoking a servlet. *https://www.cve.org/CVERecord?id=CVE-2000-1052*
CVE-2002-1742	Arbitrary Perl functions can be loaded by calling a non-existent function that activates a handler. *https://www.cve.org/CVERecord?id=CVE-2002-1742*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper Handler Deployment

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
11	Cause Web Server Misclassification

References

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

CWE-431: Missing Handler

Weakness ID : 431
Structure : Simple
Abstraction : Base

Description

A handler is not available or implemented.

Extended Description

When an exception is thrown and not caught, the process has given up an opportunity to decide if a given failure or event is worth a change in execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517
CanPrecede	V	433	Unparsed Raw Web Content Delivery	1046

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

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	2326

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Handle all possible situations (e.g. error condition).

Phase: Implementation

If an operation can throw an Exception, implement a handler for that specific exception.

Demonstrative Examples

Example 1:

If a Servlet does not catch all exceptions, it may reveal debugging information that will help an adversary form a plan of attack. In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Example Language: Java *(Bad)*

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
  String ip = req.getRemoteAddr();
  InetAddress addr = InetAddress.getByName(ip);
  ...
  out.println("hello " + addr.getHostName());
}
```

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker.

Observed Examples

Reference	Description
CVE-2022-25302	SDK for OPC Unified Architecture (OPC UA) is missing a handler for when a cast fails, allowing for a crash *https://www.cve.org/CVERecord?id=CVE-2022-25302*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Handler
Software Fault Patterns	SFP4		Unchecked Status Condition

References

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

CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations

Weakness ID : 432
Structure : Simple
Abstraction : Base

Description

The product uses a signal handler that shares state with other signal handlers, but it does not properly mask or prevent those signal handlers from being invoked while the original signal handler is still running.

Extended Description

During the execution of a signal handler, it can be interrupted by another handler when a different signal is sent. If the two handlers share state - such as global variables - then an attacker can corrupt the state by sending another signal before the first handler has completed execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	364	Signal Handler Race Condition	899

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

Turn off dangerous handlers when performing sensitive operations.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG00-C		Mask signals handled by noninterruptible signal handlers
PLOVER			Dangerous handler not cleared/ disabled during sensitive operations

CWE-433: Unparsed Raw Web Content Delivery

Weakness ID : 433
Structure : Simple
Abstraction : Variant

Description

The product stores raw content or supporting code under the web document root with an extension that is not specifically handled by the server.

Extended Description

If code is stored in a file with an extension such as ".inc" or ".pl", and the web server does not have a handler for that extension, then the server will likely send the contents of the file directly to the requester without the pre-processing that was expected. When that file contains sensitive information such as database credentials, this may allow the attacker to compromise the application or associated components.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	219	Storage of File with Sensitive Data Under Web Root	553
CanFollow	B	178	Improper Handling of Case Sensitivity	445
CanFollow	B	430	Deployment of Wrong Handler	1042
CanFollow	B	431	Missing Handler	1043

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Perform a type check before interpreting files.

Phase: Architecture and Design

Do not store sensitive information in files which may be misinterpreted.

Demonstrative Examples

Example 1:

The following code uses an include file to store database credentials:

database.inc

Example Language: PHP *(Bad)*

```
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdh#67nkjd3$3$';
?>
```

login.php

Example Language: PHP *(Bad)*

```
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db.authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password.

Observed Examples

Reference	Description
CVE-2002-1886	".inc" file stored under web document root and returned unparsed by the server *https://www.cve.org/CVERecord?id=CVE-2002-1886*
CVE-2002-2065	".inc" file stored under web document root and returned unparsed by the server *https://www.cve.org/CVERecord?id=CVE-2002-2065*
CVE-2005-2029	".inc" file stored under web document root and returned unparsed by the server *https://www.cve.org/CVERecord?id=CVE-2005-2029*
CVE-2001-0330	direct request to .pl file leaves it unparsed *https://www.cve.org/CVERecord?id=CVE-2001-0330*
CVE-2002-0614	.inc file *https://www.cve.org/CVERecord?id=CVE-2002-0614*
CVE-2004-2353	unparsed config.conf file *https://www.cve.org/CVERecord?id=CVE-2004-2353*
CVE-2007-3365	Chain: uppercase file extensions causes web server to return script source code instead of executing the script. *https://www.cve.org/CVERecord?id=CVE-2007-3365*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Relationship

This overlaps direct requests (CWE-425), alternate path (CWE-424), permissions (CWE-275), and sensitive file under web root (CWE-219).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unparsed Raw Web Content Delivery

References

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

CWE-434: Unrestricted Upload of File with Dangerous Type

Weakness ID : 434
Structure : Simple
Abstraction : Base

Description

The product allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1471
PeerOf	B	351	Insufficient Type Distinction	866
PeerOf	B	430	Deployment of Wrong Handler	1042
PeerOf	C	436	Interpretation Conflict	1057
PeerOf	B	430	Deployment of Wrong Handler	1042
CanFollow	B	73	External Control of File Name or Path	132
CanFollow	B	183	Permissive List of Allowed Inputs	458
CanFollow	B	184	Incomplete List of Disallowed Inputs	459

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1471

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	2326

Weakness Ordinalities

Primary : This can be primary when there is no check at all.

Resultant : This is frequently resultant when use of double extensions (e.g. ".php.gif") bypasses a sanity check.

Resultant : This can be resultant from client-side enforcement (CWE-602); some products will include web script in web clients to check the filename, without verifying on the server side.

Applicable Platforms

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

Language : PHP *(Prevalence = Often)*

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

Technology : Web Server *(Prevalence = Sometimes)*

Alternate Terms

Unrestricted File Upload : Used in vulnerability databases and elsewhere, but it is insufficiently precise. The phrase could be interpreted as the lack of restrictions on the size or number of uploaded files, which is a resource consumption issue.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *Arbitrary code execution is possible if an uploaded file is interpreted and executed as code by the recipient. This is especially true for .asp and .php extensions uploaded to web servers because these file types are often treated as automatically executable, even when file system permissions do not specify execution. For example, in Unix environments, programs typically cannot run unless the execute bit is set, but PHP programs may be executed by the web server without directly invoking them on the operating system.*	

Detection Methods

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Generate a new, unique filename for an uploaded file instead of using the user-supplied filename, so that no external input is used at all.[REF-422] [REF-423]

Phase: Architecture and Design

Strategy = Enforcement by Conversion

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

Phase: Architecture and Design

Consider storing the uploaded files outside of the web document root entirely. Then, use other mechanisms to deliver the files dynamically. [REF-423]

Phase: Implementation

Strategy = Input Validation

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

Phase: Architecture and Design

Define a very limited set of allowable extensions and only generate filenames that end in these extensions. Consider the possibility of XSS (CWE-79) before allowing .html or .htm file types.

Phase: Implementation

Strategy = Input Validation

Ensure that only one extension is used in the filename. Some web servers, including some versions of Apache, may process files based on inner extensions so that "filename.php.gif" is fed to the PHP interpreter.[REF-422] [REF-423]

Phase: Implementation

When running on a web server that supports case-insensitive filenames, perform case-insensitive evaluations of the extensions that are provided.

Phase: Architecture and Design

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

Phase: Implementation

Do not rely exclusively on sanity checks of file contents to ensure that the file is of the expected type and size. It may be possible for an attacker to hide code in some file segments that will still be executed by the server. For example, GIF images may contain a free-form comments field.

Phase: Implementation

Do not rely exclusively on the MIME content type or filename attribute when determining how to render a file. Validating the MIME content type and ensuring that it matches the extension is only a partial solution.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

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

Demonstrative Examples

Example 1:

The following code intends to allow a user to upload a picture to the web server. The HTML code that drives the form on the user end has an input field of type "file".

Example Language: HTML *(Good)*

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

Once submitted, the form above sends the file to upload_picture.php on the web server. PHP stores the file in a temporary location until it is retrieved (or discarded) by the server side code. In this example, the file is moved to a more permanent pictures/ directory.

Example Language: PHP *(Bad)*

```
// Define the target location where the picture being
// uploaded is going to be saved.
$target = "pictures/" . basename($_FILES['uploadedfile']['name']);
// Move the uploaded file to the new location.
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target))
{
  echo "The picture has been successfully uploaded.";
}
else
{
  echo "There was an error uploading the picture, please try again.";
}
```

The problem with the above code is that there is no check regarding type of file being uploaded. Assuming that pictures/ is available in the web document root, an attacker could upload a file with the name:

Example Language: *(Attack)*

```
malicious.php
```

Since this filename ends in ".php" it can be executed by the web server. In the contents of this uploaded file, the attacker could use:

Example Language: PHP *(Attack)*

```
<?php
  system($_GET['cmd']);
?>
```

Once this file has been installed, the attacker can enter arbitrary commands to execute using a URL such as:

Example Language: *(Attack)*

```
http://server.example.com/upload_dir/malicious.php?cmd=ls%20-l
```

which runs the "ls -l" command - or any other type of command that the attacker wants to specify.

Example 2:

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

Example Language: HTML *(Good)*

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

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

Example Language: Java *(Bad)*

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

This code does not perform a check on the type of the file being uploaded (CWE-434). This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-23). Since the code does not check the filename that is provided in the header, an attacker can use "../" sequences to write to files outside of the intended directory. Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Observed Examples

Reference	Description
CVE-2023-5227	PHP-based FAQ management app does not check the MIME type for uploaded images *https://www.cve.org/CVERecord?id=CVE-2023-5227*
CVE-2001-0901	Web-based mail product stores ".shtml" attachments that could contain SSI *https://www.cve.org/CVERecord?id=CVE-2001-0901*
CVE-2002-1841	PHP upload does not restrict file types *https://www.cve.org/CVERecord?id=CVE-2002-1841*
CVE-2005-1868	upload and execution of .php file *https://www.cve.org/CVERecord?id=CVE-2005-1868*
CVE-2005-1881	upload file with dangerous extension *https://www.cve.org/CVERecord?id=CVE-2005-1881*
CVE-2005-0254	program does not restrict file types *https://www.cve.org/CVERecord?id=CVE-2005-0254*

Reference	Description
CVE-2004-2262	improper type checking of uploaded files *https://www.cve.org/CVERecord?id=CVE-2004-2262*
CVE-2006-4558	Double "php" extension leaves an active php extension in the generated filename. *https://www.cve.org/CVERecord?id=CVE-2006-4558*
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks *https://www.cve.org/CVERecord?id=CVE-2006-6994*
CVE-2005-3288	ASP file upload *https://www.cve.org/CVERecord?id=CVE-2005-3288*
CVE-2006-2428	ASP file upload *https://www.cve.org/CVERecord?id=CVE-2006-2428*

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	2331
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2354
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2371
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Relationship

This can have a chaining relationship with incomplete denylist / permissive allowlist errors when the product tries, but fails, to properly limit which types of files are allowed (CWE-183, CWE-184). This can also overlap multiple interpretation errors for intermediaries, e.g. anti-virus products that do not remove or quarantine attachments with certain file extensions that can be processed by client systems.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted File Upload
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OMG ASCSM	ASCSM-CWE-434		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs

References

[REF-422]Richard Stanway (r1CH). "Dynamic File Uploads, Security and You". < https://web.archive.org/web/20090208005456/http://shsc.info/FileUploadSecurity >.2023-04-07.

[REF-423]Johannes Ullrich. "8 Basic Rules to Implement Secure File Uploads". 2009 December 8. < https://www.sans.org/blog/8-basic-rules-to-implement-secure-file-uploads/ >.2023-04-07.

[REF-424]Johannes Ullrich. "Top 25 Series - Rank 8 - Unrestricted Upload of Dangerous File Type". 2010 February 5. SANS Software Security Institute. < https://www.sans.org/blog/top-25-series-rank-8-unrestricted-upload-of-dangerous-file-type/ >.2023-04-07.

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

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

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

CWE-435: Improper Interaction Between Multiple Correctly-Behaving Entities

Weakness ID : 435
Structure : Simple
Abstraction : Pillar

Description

An interaction error occurs when two entities have correct behavior when running independently of each other, but when they are integrated as components in a larger system or process, they introduce incorrect behaviors that may cause resultant weaknesses.

Extended Description

When a system or process combines multiple independent components, this often produces new, emergent behaviors at the system level. However, if the interactions between these components are not fully accounted for, some of the emergent behaviors can be incorrect or even insecure.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2575
ParentOf	B	188	Reliance on Data/Memory Layout	470
ParentOf	C	436	Interpretation Conflict	1057
ParentOf	B	439	Behavioral Change in New Version or Environment	1061
ParentOf	C	1038	Insecure Automated Optimizations	1872

Applicable Platforms

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

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

Alternate Terms

Interaction Error :

Emergent Fault :

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State Varies by Context	

Demonstrative Examples

Example 1:

The paper "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" [REF-428] shows that OSes varied widely in how they manage unusual packets, which made it difficult or impossible for intrusion detection systems to properly detect certain attacker manipulations that took advantage of these OS differences.

Observed Examples

Reference	Description
CVE-2002-0485	Anti-virus product allows bypass via Content-Type and Content-Disposition headers that are mixed case, which are still processed by some clients. *https://www.cve.org/CVERecord?id=CVE-2002-0485*
CVE-2003-0411	chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text". *https://www.cve.org/CVERecord?id=CVE-2003-0411*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	2398
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2524

Notes

Research Gap

Weaknesses related to this Pillar appear to be under-studied, especially with respect to classification schemes. Input from academic and other communities could help identify and resolve gaps or organizational difficulties within CWE.

Relationship

The "Interaction Error" term, in CWE and elsewhere, is only intended to describe products that behave according to specification. When one or more of the products do not comply with specifications, then it is more likely to be API Abuse (CWE-227) or an interpretation conflict (CWE-436). This distinction can be blurred in real world scenarios, especially when "de facto" standards do not comply with specifications, or when there are no standards but there is widespread adoption. As a result, it can be difficult to distinguish these weaknesses during mapping and classification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Interaction Errors

References

[REF-428]Thomas H. Ptacek and Timothy N. Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". 1998 January. < https://insecure.org/stf/secnet_ids/secnet_ids.pdf >.2023-04-07.

[REF-568]Taimur Aslam, Ivan Krsul and Eugene H. Spafford. "Use of A Taxonomy of Security Faults". 1995 August 1. < https://csrc.nist.gov/csrc/media/publications/conference-paper/1996/10/22/proceedings-of-the-19th-nissc-1996/documents/paper057/paper.pdf >.2023-04-07.

CWE-436: Interpretation Conflict

Weakness ID : 436
Structure : Simple
Abstraction : Class

Description

Product A handles inputs or steps differently than Product B, which causes A to perform incorrect actions based on its perception of B's state.

Extended Description

This is generally found in proxies, firewalls, anti-virus software, and other intermediary devices that monitor, allow, deny, or modify traffic based on how the client or server is expected to behave.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	1055
ParentOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	190
ParentOf	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Request/Response Splitting')	271
ParentOf	B	115	Misinterpretation of Input	280

Nature	Type	ID	Name	Page
ParentOf	Ⓑ	437	Incomplete Model of Endpoint Features	1059
ParentOf	Ⓑ	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling')	1068
ParentOf	Ⓥ	626	Null Byte Interaction Error (Poison Null Byte)	1394
ParentOf	Ⓥ	650	Trusting HTTP Permission Methods on the Server Side	1432
PeerOf	Ⓑ	351	Insufficient Type Distinction	866
PeerOf	Ⓑ	434	Unrestricted Upload of File with Dangerous Type	1048

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

Nature	Type	ID	Name	Page
ParentOf	Ⓑ	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling')	1068

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Varies by Context	

Demonstrative Examples

Example 1:

The paper "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" [REF-428] shows that OSes varied widely in how they manage unusual packets, which made it difficult or impossible for intrusion detection systems to properly detect certain attacker manipulations that took advantage of these OS differences.

Example 2:

Null characters have different interpretations in Perl and C, which have security consequences when Perl invokes C functions. Similar problems have been reported in ASP [REF-429] and PHP.

Observed Examples

Reference	Description
CVE-2005-1215	Bypass filters or poison web cache using requests with multiple Content-Length headers, a non-standard behavior. *https://www.cve.org/CVERecord?id=CVE-2005-1215*
CVE-2002-0485	Anti-virus product allows bypass via Content-Type and Content-Disposition headers that are mixed case, which are still processed by some clients. *https://www.cve.org/CVERecord?id=CVE-2002-0485*
CVE-2002-1978	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass. *https://www.cve.org/CVERecord?id=CVE-2002-1978*
CVE-2002-1979	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass. *https://www.cve.org/CVERecord?id=CVE-2002-1979*
CVE-2002-0637	Virus product bypass with spaces between MIME header fields and the ":" separator, a non-standard message that is accepted by some clients. *https://www.cve.org/CVERecord?id=CVE-2002-0637*

Reference	Description
CVE-2002-1777	AV product detection bypass using inconsistency manipulation (file extension in MIME Content-Type vs. Content-Disposition field). *https://www.cve.org/CVERecord?id=CVE-2002-1777*
CVE-2005-3310	CMS system allows uploads of files with GIF/JPG extensions, but if they contain HTML, Internet Explorer renders them as HTML instead of images. *https://www.cve.org/CVERecord?id=CVE-2005-3310*
CVE-2005-4260	Interpretation conflict allows XSS via invalid "<" when a ">" is expected, which is treated as ">" by many web browsers. *https://www.cve.org/CVERecord?id=CVE-2005-4260*
CVE-2005-4080	Interpretation conflict (non-standard behavior) enables XSS because browser ignores invalid characters in the middle of tags. *https://www.cve.org/CVERecord?id=CVE-2005-4080*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	2398
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2524

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Interpretation Error (MIE)
WASC	27		HTTP Response Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
34	HTTP Response Splitting
105	HTTP Request Splitting
273	HTTP Response Smuggling

References

[REF-427]Steve Christey. "On Interpretation Conflict Vulnerabilities". Bugtraq. 2005 November 3. < https://seclists.org/bugtraq/2005/Nov/30 >.2023-04-07.

[REF-428]Thomas H. Ptacek and Timothy N. Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". 1998 January. < https://insecure.org/stf/secnet_ids/secnet_ids.pdf >.2023-04-07.

[REF-429]Brett Moore. "0x00 vs ASP file upload scripts". 2004 July 3. < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

[REF-430]Rain Forest Puppy. "Poison NULL byte". Phrack.

[REF-431]David F. Skoll. "Re: Corsaire Security Advisory - Multiple vendor MIME RFC2047 encoding". Bugtraq. 2004 September 5. < http://marc.info/?l=bugtraq&m=109525864717484&w=2 >.

CWE-437: Incomplete Model of Endpoint Features

Weakness ID : 437
Structure : Simple

Abstraction : Base

Description

A product acts as an intermediary or monitor between two or more endpoints, but it does not have a complete model of an endpoint's features, behaviors, or state, potentially causing the product to perform incorrect actions based on this incomplete model.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	436	Interpretation Conflict	1057

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Varies by Context	

Demonstrative Examples

Example 1:

HTTP request smuggling is an attack against an intermediary such as a proxy. This attack works because the proxy expects the client to parse HTTP headers one way, but the client parses them differently.

Example 2:

Anti-virus products that reside on mail servers can suffer from this issue if they do not know how a mail client will handle a particular attachment. The product might treat an attachment type as safe, not knowing that the client's configuration treats it as executable.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	2398
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2524

Notes

Relationship

This can be related to interaction errors, although in some cases, one of the endpoints is not performing correctly according to specification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Extra Unhandled Features

CWE-439: Behavioral Change in New Version or Environment

Weakness ID : 439
Structure : Simple
Abstraction : Base

Description

A's behavior or functionality changes with a new version of A, or a new environment, which is not known (or manageable) by B.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	1055

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

Applicable Platforms

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

Alternate Terms

Functional change :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Observed Examples

Reference	Description
CVE-2002-1976	Linux kernel 2.2 and above allow promiscuous mode using a different method than previous versions, and ifconfig is not aware of the new method (alternate path property). *https://www.cve.org/CVERecord?id=CVE-2002-1976*
CVE-2005-1711	Product uses defunct method from another product that does not return an error code and allows detection avoidance. *https://www.cve.org/CVERecord?id=CVE-2005-1711*
CVE-2003-0411	chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text". *https://www.cve.org/CVERecord?id=CVE-2003-0411*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2524

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			CHANGE Behavioral Change

CWE-440: Expected Behavior Violation

Weakness ID : 440
Structure : Simple
Abstraction : Base

Description

A feature, API, or function does not perform according to its specification.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	684	Incorrect Provision of Specified Functionality	1505

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Demonstrative Examples

Example 1:

The provided code is extracted from the Control and Status Register (CSR), csr_regfile, module within the Hack@DAC'21 OpenPiton System-on-Chip (SoC). This module is designed to implement CSR registers in accordance with the RISC-V specification. The mie (machine interrupt enable) register is a 64-bit register [REF-1384], where bits correspond to different interrupt sources. As the name suggests, mie is a machine-level register that determines which interrupts are enabled. Note that in the example below the mie_q and mie_d registers represent the conceptual mie reigster in the RISC-V specification. The mie_d register is the value to be stored in the mie register while the mie_q register holds the current value of the mie register [REF-1385].

The mideleg (machine interrupt delegation) register, also 64-bit wide, enables the delegation of specific interrupt sources from machine privilege mode to lower privilege levels. By setting specific bits in the mideleg register, the handling of certain interrupts can be delegated to lower privilege levels without engaging the machine-level privilege mode. For example, in supervisor mode, the mie register is limited to a specific register called the sie (supervisor interrupt enable) register. If delegated, an interrupt becomes visible in the sip (supervisor interrupt pending) register and can be enabled or blocked using the sie register. If no delegation occurs, the related bits in sip and sie are set to zero.

The sie register value is computed based on the current value of mie register, i.e., mie_q, and the mideleg register.

Example Language: Verilog *(Bad)*

```
module csr_regfile #(...)(...);
...
// --------------------------
// CSR Write and update logic
// --------------------------
...
  if (csr_we) begin
    unique case (csr_addr.address)
    ...
      riscv::CSR_SIE: begin
        // the mideleg makes sure only delegate-able register
        //(and therefore also only implemented registers) are written
        mie_d = (mie_q & ~mideleg_q) | (csr_wdata & mideleg_q) | utval_q;
      end
      ...
    endcase
  end
endmodule
```

The above code snippet illustrates an instance of a vulnerable implementation of the sie register update logic, where users can tamper with the mie_d register value through the utval (user trap value) register. This behavior violates the RISC-V specification.

The code shows that the value of utval, among other signals, is used in updating the mie_d value within the sie update logic. While utval is a register accessible to users, it should not influence or compromise the integrity of sie. Through manipulation of the utval register, it becomes feasible to manipulate the sie register's value. This opens the door for potential attacks, as an adversary can gain control over or corrupt the sie value. Consequently, such manipulation empowers an attacker to enable or disable critical supervisor-level interrupts, resulting in various security risks such as privilege escalation or denial-of-service attacks.

A fix to this issue is to remove the utval from the right-hand side of the assignment. That is the value of the mie_d should be updated as shown in the good code example [REF-1386].

Example Language: Verilog *(Good)*

```
module csr_regfile #(...)(...);
...
// --------------------------
// CSR Write and update logic
// --------------------------
...
  if (csr_we) begin
    unique case (csr_addr.address)
    ...
      riscv::CSR_SIE: begin
        // the mideleg makes sure only delegate-able register
        //(and therefore also only implemented registers) are written
        mie_d = (mie_q & ~mideleg_q) | (csr_wdata & mideleg_q);
      end
```

```
        ...
      endcase
    end
endmodule
```

Observed Examples

Reference	Description
CVE-2003-0187	Program uses large timeouts on unconfirmed connections resulting from inconsistency in linked lists implementations. *https://www.cve.org/CVERecord?id=CVE-2003-0187*
CVE-2003-0465	"strncpy" in Linux kernel acts different than libc on x86, leading to expected behavior difference - sort of a multiple interpretation error? *https://www.cve.org/CVERecord?id=CVE-2003-0465*
CVE-2005-3265	Buffer overflow in product stems the use of a third party library function that is expected to have internal protection against overflows, but doesn't. *https://www.cve.org/CVERecord?id=CVE-2005-3265*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf	C	1208	Cross-Cutting Problems	1194	2474
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2505
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Theoretical

The behavior of an application that is not consistent with the expectations of the developer may lead to incorrect use of the software.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Expected behavior violation

References

[REF-1384]"The RISC-V Instruction Set Manual Volume II: Privileged Architecture page 28". 2021. < https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf >.2024-01-16.

[REF-1385]"csr_regfile.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/csr_regfile.sv >.2024-01-16.

[REF-1386]"Fix for csr_regfile.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/2341c625a28d2fb87d370e32c45b68bd711cc43b/piton/design/chip/tile/ariane/src/csr_regfile.sv#L519C4-L522C20 >.2024-01-16.

CWE-441: Unintended Proxy or Intermediary ('Confused Deputy')

Weakness ID : 441
Structure : Simple
Abstraction : Class

Description

The product receives a request, message, or directive from an upstream component, but the product does not sufficiently preserve the original source of the request before forwarding the request to an external actor that is outside of the product's control sphere. This causes the product to appear to be the source of the request, leading it to act as a proxy or other intermediary between the upstream component and the external actor.

Extended Description

If an attacker cannot directly contact a target, but the product has access to the target, then the attacker can send a request to the product and have it be forwarded to the target. The request would appear to be coming from the product's system, not the attacker's system. As a result, the attacker can bypass access controls (such as firewalls) or hide the source of malicious requests, since the requests would not be coming directly from the attacker.

Since proxy functionality and message-forwarding often serve a legitimate purpose, this issue only becomes a vulnerability when:

- The product runs with different privileges or on a different system, or otherwise has different levels of access than the upstream component;
- The attacker is prevented from making the request directly to the target; and
- The attacker can create a request that the proxy does not explicitly intend to be forwarded on the behalf of the requester. Such a request might point to an unexpected hostname, port number, hardware IP, or service. Or, the request might be sent to an allowed service, but the request could contain disallowed directives, commands, or resources.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1364
ParentOf	B	918	Server-Side Request Forgery (SSRF)	1820
ParentOf	B	1021	Improper Restriction of Rendered UI Layers or Frames	1860
PeerOf	B	611	Improper Restriction of XML External Entity Reference	1367
CanPrecede	C	668	Exposure of Resource to Wrong Sphere	1469

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2429

Applicable Platforms

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

Operating_System : Not OS-Specific *(Prevalence = Undetermined)*

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

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

Alternate Terms

Confused Deputy : This weakness is sometimes referred to as the "Confused deputy" problem, in which an attacker misused the authority of one victim (the "confused deputy") when targeting another victim.

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation Access Control	Gain Privileges or Assume Identity Hide Activities Execute Unauthorized Code or Commands	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Enforce the use of strong mutual authentication mechanism between the two parties.

Phase: Architecture and Design

Whenever a product is an intermediary or proxy for transactions between two other components, the proxy core should not drop the identity of the initiator of the transaction. The immutability of the identity of the initiator must be maintained and should be forwarded all the way to the target.

Demonstrative Examples

Example 1:

A SoC contains a microcontroller (running ring-3 (least trusted ring) code), a Memory Mapped Input Output (MMIO) mapped IP core (containing design-house secrets), and a Direct Memory Access (DMA) controller, among several other compute elements and peripherals. The SoC implements access control to protect the registers in the IP core (which registers store the design-house secrets) from malicious, ring-3 (least trusted ring) code executing on the microcontroller. The DMA controller, however, is not blocked off from accessing the IP core for functional reasons.

Example Language: Other *(Bad)*

```
The code in ring-3 (least trusted ring) of the microcontroller attempts to directly read the protected registers in IP
core through MMIO transactions. However, this attempt is blocked due to the implemented access control. Now, the
microcontroller configures the DMA core to transfer data from the protected registers to a memory region that it has access
to. The DMA core, which is acting as an intermediary in this transaction, does not preserve the identity of the microcontroller
and, instead, initiates a new transaction with its own identity. Since the DMA core has access, the transaction (and hence,
the attack) is successful.
```

The weakness here is that the intermediary or the proxy agent did not ensure the immutability of the identity of the microcontroller initiating the transaction.

Example Language: Other *(Good)*

```
The DMA core forwards this transaction with the identity of the code executing on the microcontroller, which is the original
initiator of the end-to-end transaction. Now the transaction is blocked, as a result of forwarding the identity of the true
initiator which lacks the permission to access the confidential MMIO mapped IP core.
```

Observed Examples

Reference	Description
CVE-1999-0017	FTP bounce attack. The design of the protocol allows an attacker to modify the PORT command to cause the FTP server to connect to other machines besides the attacker's. *https://www.cve.org/CVERecord?id=CVE-1999-0017*
CVE-1999-0168	RPC portmapper could redirect service requests from an attacker to another entity, which thinks the requests came from the portmapper. *https://www.cve.org/CVERecord?id=CVE-1999-0168*
CVE-2005-0315	FTP server does not ensure that the IP address in a PORT command is the same as the FTP user's session, allowing port scanning by proxy. *https://www.cve.org/CVERecord?id=CVE-2005-0315*
CVE-2002-1484	Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning. *https://www.cve.org/CVERecord?id=CVE-2002-1484*
CVE-2004-2061	CGI script accepts and retrieves incoming URLs. *https://www.cve.org/CVERecord?id=CVE-2004-2061*
CVE-2001-1484	Bounce attack allows access to TFTP from trusted side. *https://www.cve.org/CVERecord?id=CVE-2001-1484*
CVE-2010-1637	Web-based mail program allows internal network scanning using a modified POP3 port number. *https://www.cve.org/CVERecord?id=CVE-2010-1637*
CVE-2009-0037	URL-downloading library automatically follows redirects to file:// and scp:// URLs *https://www.cve.org/CVERecord?id=CVE-2009-0037*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	2397
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Relationship

This weakness has a chaining relationship with CWE-668 (Exposure of Resource to Wrong Sphere) because the proxy effectively provides the attacker with access to the target's resources that the attacker cannot directly obtain.

Maintenance

This could possibly be considered as an emergent resource.

Theoretical

It could be argued that the "confused deputy" is a fundamental aspect of most vulnerabilities that require an active attacker. Even for common implementation issues such as buffer overflows, SQL injection, OS command injection, and path traversal, the vulnerable program already has the authorization to run code or access files. The vulnerability arises when the attacker causes the program to run unexpected code or access unexpected files.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unintended proxy/intermediary
PLOVER			Proxied Trusted Channel
WASC	32		Routing Detour

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
219	XML Routing Detour Attacks
465	Transparent Proxy Abuse

References

[REF-432]Norm Hardy. "The Confused Deputy (or why capabilities might have been invented)". 1988. < http://www.cap-lore.com/CapTheory/ConfusedDeputy.html >.

[REF-1125]moparisthebest. "Validation Vulnerabilities". 2015 June 5. < https://mailarchive.ietf.org/arch/msg/acme/s6Q5PdJP48LEUwgzrVuw_XPKCsM/ >.

CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling')

Weakness ID : 444
Structure : Simple
Abstraction : Base

Description

The product acts as an intermediary HTTP agent (such as a proxy or firewall) in the data flow between two entities such as a client and server, but it does not interpret malformed HTTP requests or responses in ways that are consistent with how the messages will be processed by those entities that are at the ultimate destination.

Extended Description

HTTP requests or responses ("messages") can be malformed or unexpected in ways that cause web servers or clients to interpret the messages in different ways than intermediary HTTP agents such as load balancers, reverse proxies, web caching proxies, application firewalls, etc. For example, an adversary may be able to add duplicate or different header fields that a client or server might interpret as one set of messages, whereas the intermediary might interpret the same sequence of bytes as a different set of messages. For example, discrepancies can arise in how to handle duplicate headers like two Transfer-encoding (TE) or two Content-length (CL), or the malicious HTTP message will have different headers for TE and CL.

The inconsistent parsing and interpretation of messages can allow the adversary to "smuggle" a message to the client/server without the intermediary being aware of it.

This weakness is usually the result of the usage of outdated or incompatible HTTP protocol versions in the HTTP agents.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	436	Interpretation Conflict	1057

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

Nature	Type	ID	Name	Page
ChildOf	●	436	Interpretation Conflict	1057

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

Applicable Platforms

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

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

HTTP Request Smuggling :

HTTP Response Smuggling :

HTTP Smuggling :

Common Consequences

Scope	Impact	Likelihood
Integrity Non-Repudiation Access Control	Unexpected State Hide Activities Bypass Protection Mechanism *An attacker could create HTTP messages to exploit a number of weaknesses including 1) the message can trick the web server to associate a URL with another URL's webpage and caching the contents of the webpage (web cache poisoning attack), 2) the message can be structured to bypass the firewall protection mechanisms and gain unauthorized access to a web application, and 3) the message can invoke a script or a page that returns client credentials (similar to a Cross Site Scripting attack).*	

Potential Mitigations

Phase: Implementation

Use a web server that employs a strict HTTP parsing procedure, such as Apache [REF-433].

Phase: Implementation

Use only SSL communication.

Phase: Implementation

Terminate the client session after each request.

Phase: System Configuration

Turn all pages to non-cacheable.

Demonstrative Examples

Example 1:

In the following example, a malformed HTTP request is sent to a website that includes a proxy server and a web server with the intent of poisoning the cache to associate one webpage with another malicious webpage.

Example Language: *(Attack)*

```
POST http://www.website.com/foobar.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Content-Length: 54
GET /poison.html HTTP/1.1
Host: www.website.com
Bla: GET http://www.website.com/page_to_poison.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
```

When this request is sent to the proxy server, the proxy server parses the first four lines of the POST request and encounters the two "Content-Length" headers. The proxy server ignores the first header, so it assumes the request has a body of length 54 bytes. Therefore, it treats the data in the next three lines that contain exactly 54 bytes as the first request's body:

Example Language: *(Result)*

```
GET /poison.html HTTP/1.1
Host: www.website.com
Bla:
```

The proxy then parses the remaining bytes, which it treats as the client's second request:

Example Language: *(Attack)*

```
GET http://www.website.com/page_to_poison.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
```

The original request is forwarded by the proxy server to the web server. Unlike the proxy, the web server uses the first "Content-Length" header and considers that the first POST request has no body.

Example Language: *(Attack)*

```
POST http://www.website.com/foobar.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Content-Length: 54 (ignored by server)
```

Because the web server has assumed the original POST request was length 0, it parses the second request that follows, i.e. for GET /poison.html:

Example Language: *(Attack)*

```
GET /poison.html HTTP/1.1
Host: www.website.com
Bla: GET http://www.website.com/page_to_poison.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
```

Note that the "Bla:" header is treated as a regular header, so it is not parsed as a separate GET request.

The requests the web server sees are "POST /foobar.html" and "GET /poison.html", so it sends back two responses with the contents of the "foobar.html" page and the "poison.html" page, respectively. The proxy matches these responses to the two requests it thinks were sent by the client - "POST /foobar.html" and "GET /page_to_poison.html". If the response is cacheable, the proxy caches the contents of "poison.html" under the URL "page_to_poison.html", and the cache is poisoned! Any client requesting "page_to_poison.html" from the proxy would receive the "poison.html" page.

When a website includes both a proxy server and a web server, some protection against this type of attack can be achieved by installing a web application firewall, or using a web server that includes a stricter HTTP parsing procedure or make all webpages non-cacheable.

Additionally, if a web application includes a Java servlet for processing requests, the servlet can check for multiple "Content-Length" headers and if they are found the servlet can return an error response thereby preventing the poison page to be cached, as shown below.

Example Language: Java *(Good)*

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
  // Set up response writer object
  ...
  try {
    // check for multiple content length headers
    Enumeration contentLengthHeaders = request.getHeaders("Content-Length");
    int count = 0;
    while (contentLengthHeaders.hasMoreElements()) {
      count++;
    }
    if (count > 1) {
      // output error response
    }
    else {
      // process request
    }
  } catch (Exception ex) {...}
}
```

Example 2:

In the following example, a malformed HTTP request is sent to a website that includes a web server with a firewall with the intent of bypassing the web server firewall to smuggle malicious code into the system.

Example Language: *(Attack)*

```
POST /page.asp HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
Content-Length: 49223
zzz...zzz ["z" x 49152]
POST /page.asp HTTP/1.0
Connection: Keep-Alive
Content-Length: 30
POST /page.asp HTTP/1.0
Bla: POST /page.asp?cmd.exe HTTP/1.0
Connection: Keep-Alive
```

When this request is sent to the web server, the first POST request has a content-length of 49,223 bytes, and the firewall treats the line with 49,152 copies of "z" and the lines with an additional lines with 71 bytes as its body (49,152+71=49,223). The firewall then continues to parse what it thinks is the second request starting with the line with the third POST request.

Note that there is no CRLF after the "Bla: " header so the POST in the line is parsed as the value of the "Bla:" header. Although the line contains the pattern identified with a worm ("cmd.exe"), it is not blocked, since it is considered part of a header value. Therefore, "cmd.exe" is smuggled through the firewall.

When the request is passed through the firewall the web server the first request is ignored because the web server does not find an expected "Content-Type: application/x-www-form-urlencoded" header, and starts parsing the second request.

This second request has a content-length of 30 bytes, which is exactly the length of the next two lines up to the space after the "Bla:" header. And unlike the firewall, the web server processes the final POST as a separate third request and the "cmd.exe" worm is smuggled through the firewall to the web server.

To avoid this attack a Web server firewall product must be used that is designed to prevent this type of attack.

Example 3:

The interpretation of HTTP responses can be manipulated if response headers include a space between the header name and colon, or if HTTP 1.1 headers are sent through a proxy configured for HTTP 1.0, allowing for HTTP response smuggling. This can be exploited in web browsers and other applications when used in combination with various proxy servers. For instance, the HTTP response interpreted by the front-end/client HTTP agent/entity - in this case the web browser - can interpret a single response from an adversary-compromised web server as being two responses from two different web sites. In the Example below, notice the extra space after the Content-Length and Set-Cookie headers.

Example Language: *(Attack)*

```
HTTP/1.1 200 OK
Date: Fri, 08 Aug 2016 08:12:31 GMT
Server: Apache (Unix)
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html
Content-Length : 2345
Transfer-Encoding: chunked
Set-Cookie : token="Malicious Code"
<HTML> ... "Malicious Code"
```

Observed Examples

Reference	Description
CVE-2022-24766	SSL/TLS-capable proxy allows HTTP smuggling when used in tandem with HTTP/1.0 services, due to inconsistent interpretation and input sanitization of HTTP messages within the body of another message *https://www.cve.org/CVERecord?id=CVE-2022-24766*
CVE-2021-37147	Chain: caching proxy server has improper input validation (CWE-20) of headers, allowing HTTP response smuggling (CWE-444) using an "LF line ending" *https://www.cve.org/CVERecord?id=CVE-2021-37147*
CVE-2020-8287	Node.js platform allows request smuggling via two Transfer-Encoding headers *https://www.cve.org/CVERecord?id=CVE-2020-8287*
CVE-2006-6276	Web servers allow request smuggling via inconsistent HTTP headers. *https://www.cve.org/CVERecord?id=CVE-2006-6276*
CVE-2005-2088	HTTP server allows request smuggling with both a "Transfer-Encoding: chunked" header and a Content-Length header *https://www.cve.org/CVERecord?id=CVE-2005-2088*
CVE-2005-2089	HTTP server allows request smuggling with both a "Transfer-Encoding: chunked" header and a Content-Length header *https://www.cve.org/CVERecord?id=CVE-2005-2089*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2524

Notes

Theoretical

Request smuggling can be performed due to a multiple interpretation error, where the target is an intermediary or monitor, via a consistency manipulation (Transfer-Encoding and Content-Length headers).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			HTTP Request Smuggling
WASC	26		HTTP Request Smuggling
WASC	27		HTTP Response Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
33	HTTP Request Smuggling
273	HTTP Response Smuggling

References

[REF-433]Chaim Linhart, Amit Klein, Ronen Heled and Steve Orrin. "HTTP Request Smuggling". < https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf >.2023-04-07.

[REF-1273]Robert Auger. "HTTP Response Smuggling". 2011 February 1. < http://projects.webappsec.org/w/page/13246930/HTTP%20Response%20Smuggling >.

[REF-1274]Dzevad Alibegovic. "HTTP Request Smuggling: Complete Guide to Attack Types and Prevention". 2021 August 3. < https://brightsec.com/blog/http-request-smuggling-hrs/ >.

[REF-1275]Busra Demir. "A Pentester's Guide to HTTP Request Smuggling". 2020 October 5. < https://www.cobalt.io/blog/a-pentesters-guide-to-http-request-smuggling >.

[REF-1276]Edi Kogan and Daniel Kerman. "HTTP Desync Attacks in the Wild and How to Defend Against Them". 2019 October 9. < https://www.imperva.com/blog/http-desync-attacks-and-defence-methods/ >.

[REF-1277]James Kettle. "HTTP Desync Attacks: Request Smuggling Reborn". 2019 August 7. < https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn >.2023-04-07.

[REF-1278]PortSwigger. "HTTP request smuggling". < https://portswigger.net/web-security/request-smuggling >.2023-04-07.

CWE-446: UI Discrepancy for Security Feature

Weakness ID : 446
Structure : Simple
Abstraction : Class

Description

The user interface does not correctly enable or configure a security feature, but the interface provides feedback that causes the user to believe that the feature is in a secure state.

Extended Description

When the user interface does not properly reflect what the user asks of it, then it can lead the user into a false sense of security. For example, the user might check a box to enable a security option to enable encrypted communications, but the product does not actually enable the encryption. Alternately, the user might provide a "restrict ALL" access control rule, but the product only implements "restrict SOME".

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	684	Incorrect Provision of Specified Functionality	1505
ParentOf	B	447	Unimplemented or Unsupported Feature in UI	1075
ParentOf	B	448	Obsolete Feature in UI	1076
ParentOf	B	449	The UI Performs the Wrong Action	1077

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Observed Examples

Reference	Description
CVE-1999-1446	UI inconsistency; visited URLs list not cleared when "Clear History" option is selected. *https://www.cve.org/CVERecord?id=CVE-1999-1446*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	2418
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Maintenance

This entry is likely a loose composite that could be broken down into the different types of errors that cause the user interface to have incorrect interactions with the underlying security feature.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			User interface inconsistency

CWE-447: Unimplemented or Unsupported Feature in UI

Weakness ID : 447
Structure : Simple
Abstraction : Base

Description

A UI function for a security feature appears to be supported and gives feedback to the user that suggests that it is supported, but the underlying functionality is not implemented.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	446	UI Discrepancy for Security Feature	1073
ChildOf	C	671	Lack of Administrator Control over Security	1478

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

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	2320

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Testing

Perform functionality testing before deploying the application.

Observed Examples

Reference	Description
CVE-2000-0127	GUI configuration tool does not enable a security option when a checkbox is selected, although that option is honored when manually set in the configuration file. *https://www.cve.org/CVERecord?id=CVE-2000-0127*
CVE-2001-0863	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass. *https://www.cve.org/CVERecord?id=CVE-2001-0863*
CVE-2001-0865	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass. *https://www.cve.org/CVERecord?id=CVE-2001-0865*
CVE-2004-0979	Web browser does not properly modify security setting when the user sets it. *https://www.cve.org/CVERecord?id=CVE-2004-0979*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2418
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

Research Gap

This issue needs more study, as there are not many examples. It is not clear whether it is primary or resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unimplemented or unsupported feature in UI

CWE-448: Obsolete Feature in UI

Weakness ID : 448
Structure : Simple
Abstraction : Base

Description

A UI function is obsolete and the product does not warn the user.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	446	UI Discrepancy for Security Feature	1073

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

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	2320

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Remove the obsolete feature from the UI. Warn the user that the feature is no longer supported.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2418
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Obsolete feature in UI

CWE-449: The UI Performs the Wrong Action

Weakness ID : 449
Structure : Simple
Abstraction : Base

Description

The UI performs the wrong action with respect to the user's request.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	●	446	UI Discrepancy for Security Feature	1073

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

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	2320

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Testing

Perform extensive functionality testing of the UI. The UI should behave as specified.

Observed Examples

Reference	Description
CVE-2001-1387	Network firewall accidentally implements one command line option as if it were another, possibly leading to behavioral infoleak. *https://www.cve.org/CVERecord?id=CVE-2001-1387*

Reference	Description
CVE-2001-0081	Command line option correctly suppresses a user prompt but does not properly disable a feature, although when the product prompts the user, the feature is properly disabled. *https://www.cve.org/CVERecord?id=CVE-2001-0081*
CVE-2002-1977	Product does not "time out" according to user specification, leaving sensitive data available after it has expired. *https://www.cve.org/CVERecord?id=CVE-2002-1977*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2418
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			The UI performs the wrong action

CWE-450: Multiple Interpretations of UI Input

Weakness ID : 450
Structure : Simple
Abstraction : Base

Description

The UI has multiple interpretations of user input but does not prompt the user when it selects the less secure interpretation.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	357	Insufficient UI Warning of Dangerous Operations	880

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

Phase: Implementation

Strategy = Input Validation

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2418
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Interpretations of UI Input

CWE-451: User Interface (UI) Misrepresentation of Critical Information

Weakness ID : 451
Structure : Simple
Abstraction : Class

Description

The user interface (UI) does not properly represent critical information to the user, allowing the information - or its source - to be obscured or spoofed. This is often a component in phishing attacks.

Extended Description

If an attacker can cause the UI to display erroneous data, or to otherwise convince the user to display information that appears to come from a trusted source, then the attacker could trick the user into performing the wrong action. This is often a component in phishing attacks, but other kinds of problems exist. For example, if the UI is used to monitor the security state of a system or network, then omitting or obscuring an important indicator could prevent the user from detecting and reacting to a security-critical event.

UI misrepresentation can take many forms:

- Incorrect indicator: incorrect information is displayed, which prevents the user from understanding the true state of the product or the environment the product is monitoring, especially of potentially-dangerous conditions or operations. This can be broken down into several different subtypes.

- Overlay: an area of the display is intended to give critical information, but another process can modify the display by overlaying another element on top of it. The user is not interacting with the expected portion of the user interface. This is the problem that enables clickjacking attacks, although many other types of attacks exist that involve overlay.
- Icon manipulation: the wrong icon, or the wrong color indicator, can be influenced (such as making a dangerous .EXE executable look like a harmless .GIF)
- Timing: the product is performing a state transition or context switch that is presented to the user with an indicator, but a race condition can cause the wrong indicator to be used before the product has fully switched context. The race window could be extended indefinitely if the attacker can trigger an error.
- Visual truncation: important information could be truncated from the display, such as a long filename with a dangerous extension that is not displayed in the GUI because the malicious portion is truncated. The use of excessive whitespace can also cause truncation, or place the potentially-dangerous indicator outside of the user's field of view (e.g. "filename.txt .exe").
  A different type of truncation can occur when a portion of the information is removed due to reasons other than length, such as the accidental insertion of an end-of-input marker in the middle of an input, such as a NUL byte in a C-style string.
- Visual distinction: visual information might be presented in a way that makes it difficult for the user to quickly and correctly distinguish between critical and unimportant segments of the display.
- Homographs: letters from different character sets, fonts, or languages can appear very similar (i.e. may be visually equivalent) in a way that causes the human user to misread the text (for example, to conduct phishing attacks to trick a user into visiting a malicious web site with a visually-similar name as a trusted site). This can be regarded as a type of visual distinction issue.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	221	Information Loss or Omission	556
ChildOf	Ⓒ	684	Incorrect Provision of Specified Functionality	1505
ParentOf	Ⓑ	1007	Insufficient Visual Distinction of Homoglyphs Presented to User	1857
ParentOf	Ⓑ	1021	Improper Restriction of Rendered UI Layers or Frames	1860
PeerOf	Ⓒ	346	Origin Validation Error	853

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation Access Control	Hide Activities Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Perform data validation (e.g. syntax, length, etc.) before interpreting the data.

Phase: Architecture and Design

Strategy = Output Encoding

Create a strategy for presenting information, and plan for how to display unusual characters.

Observed Examples

Reference	Description
CVE-2004-2227	Web browser's filename selection dialog only shows the beginning portion of long filenames, which can trick users into launching executables with dangerous extensions. *https://www.cve.org/CVERecord?id=CVE-2004-2227*
CVE-2001-0398	Attachment with many spaces in filename bypasses "dangerous content" warning and uses different icon. Likely resultant. *https://www.cve.org/CVERecord?id=CVE-2001-0398*
CVE-2001-0643	Misrepresentation and equivalence issue. *https://www.cve.org/CVERecord?id=CVE-2001-0643*
CVE-2005-0593	Lock spoofing from several different weaknesses. *https://www.cve.org/CVERecord?id=CVE-2005-0593*
CVE-2004-1104	Incorrect indicator: web browser can be tricked into presenting the wrong URL *https://www.cve.org/CVERecord?id=CVE-2004-1104*
CVE-2005-0143	Incorrect indicator: Lock icon displayed when an insecure page loads a binary file loaded from a trusted site. *https://www.cve.org/CVERecord?id=CVE-2005-0143*
CVE-2005-0144	Incorrect indicator: Secure "lock" icon is presented for one channel, while an insecure page is being simultaneously loaded in another channel. *https://www.cve.org/CVERecord?id=CVE-2005-0144*
CVE-2004-0761	Incorrect indicator: Certain redirect sequences cause security lock icon to appear in web browser, even when page is not encrypted. *https://www.cve.org/CVERecord?id=CVE-2004-0761*
CVE-2004-2219	Incorrect indicator: Spoofing via multi-step attack that causes incorrect information to be displayed in browser address bar. *https://www.cve.org/CVERecord?id=CVE-2004-2219*
CVE-2004-0537	Overlay: Wide "favorites" icon can overlay and obscure address bar *https://www.cve.org/CVERecord?id=CVE-2004-0537*
CVE-2005-2271	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://www.cve.org/CVERecord?id=CVE-2005-2271*
CVE-2005-2272	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://www.cve.org/CVERecord?id=CVE-2005-2272*
CVE-2005-2273	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://www.cve.org/CVERecord?id=CVE-2005-2273*
CVE-2005-2274	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://www.cve.org/CVERecord?id=CVE-2005-2274*
CVE-2001-1410	Visual distinction: Browser allows attackers to create chromeless windows and spoof victim's display using unprotected Javascript method. *https://www.cve.org/CVERecord?id=CVE-2001-1410*

Reference	Description
CVE-2002-0197	Visual distinction: Chat client allows remote attackers to spoof encrypted, trusted messages with lines that begin with a special sequence, which makes the message appear legitimate. *https://www.cve.org/CVERecord?id=CVE-2002-0197*
CVE-2005-0831	Visual distinction: Product allows spoofing names of other users by registering with a username containing hex-encoded characters. *https://www.cve.org/CVERecord?id=CVE-2005-0831*
CVE-2003-1025	Visual truncation: Special character in URL causes web browser to truncate the user portion of the "user@domain" URL, hiding real domain in the address bar. *https://www.cve.org/CVERecord?id=CVE-2003-1025*
CVE-2005-0243	Visual truncation: Chat client does not display long filenames in file dialog boxes, allowing dangerous extensions via manipulations including (1) many spaces and (2) multiple file extensions. *https://www.cve.org/CVERecord?id=CVE-2005-0243*
CVE-2005-1575	Visual truncation: Web browser file download type can be hidden using whitespace. *https://www.cve.org/CVERecord?id=CVE-2005-1575*
CVE-2004-2530	Visual truncation: Visual truncation in chat client using whitespace to hide dangerous file extension. *https://www.cve.org/CVERecord?id=CVE-2004-2530*
CVE-2005-0590	Visual truncation: Dialog box in web browser allows user to spoof the hostname via a long "user:pass" sequence in the URL, which appears before the real hostname. *https://www.cve.org/CVERecord?id=CVE-2005-0590*
CVE-2004-1451	Visual truncation: Null character in URL prevents entire URL from being displayed in web browser. *https://www.cve.org/CVERecord?id=CVE-2004-1451*
CVE-2004-2258	Miscellaneous -- [step-based attack, GUI] -- Password-protected tab can be bypassed by switching to another tab, then back to original tab. *https://www.cve.org/CVERecord?id=CVE-2004-2258*
CVE-2005-1678	Miscellaneous -- Dangerous file extensions not displayed. *https://www.cve.org/CVERecord?id=CVE-2005-1678*
CVE-2002-0722	Miscellaneous -- Web browser allows remote attackers to misrepresent the source of a file in the File Download dialog box. *https://www.cve.org/CVERecord?id=CVE-2002-0722*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2418
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1379	ICS Operations (& Maintenance): Human factors in ICS environments	1358	2514
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Maintenance

This entry should be broken down into more precise entries. See extended description.

Research Gap

Misrepresentation problems are frequently studied in web browsers, but there are no known efforts for classifying these kinds of problems in terms of the shortcomings of the interface. In addition, many misrepresentation issues are resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UI Misrepresentation of Critical Information

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
98	Phishing
154	Resource Location Spoofing
163	Spear Phishing
164	Mobile Phishing
173	Action Spoofing

References

[REF-434]David Wheeler. "Secure Programming for Linux and Unix HOWTO". 2003 March 3. < http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/semantic-attacks.html >.2023-04-07.

CWE-453: Insecure Default Variable Initialization

Weakness ID : 453
Structure : Simple
Abstraction : Variant

Description

The product, by default, initializes an internal variable with an insecure or less secure value than is possible.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	1188	Initialization of a Resource with an Insecure Default	1974

Applicable Platforms

Language : PHP *(Prevalence = Sometimes)*

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could gain access to and modify sensitive data or system information.*	

Potential Mitigations

Phase: System Configuration

Disable or change default settings when they can be used to abuse the system. Since those default settings are shipped with the product they are likely to be known by a potential attacker who is familiar with the product. For instance, default credentials should be changed or the associated accounts should be disabled.

Demonstrative Examples

Example 1:

This code attempts to login a user using credentials from a POST request:

Example Language: PHP *(Bad)*

```
// $user and $pass automatically set from POST request
if (login_user($user,$pass)) {
  $authorized = true;
}
...
if ($authorized) {
  generatePage();
}
```

Because the $authorized variable is never initialized, PHP will automatically set $authorized to any value included in the POST request if register_globals is enabled. An attacker can send a POST request with an unexpected third value 'authorized' set to 'true' and gain authorized status without supplying valid credentials.

Here is a fixed version:

Example Language: PHP *(Good)*

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$authorized = false;
if (login_user($user,$pass)) {
  $authorized = true;
}
...
```

This code avoids the issue by initializing the $authorized variable to false and explicitly retrieving the login credentials from the $_POST variable. Regardless, register_globals should never be enabled and is disabled by default in current versions of PHP.

Observed Examples

Reference	Description
CVE-2022-36349	insecure default variable initialization in BIOS firmware for a hardware board allows DoS *https://www.cve.org/CVERecord?id=CVE-2022-36349*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2403
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

This overlaps other categories, probably should be split into separate items.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insecure default variable initialization

CWE-454: External Initialization of Trusted Variables or Data Stores

Weakness ID : 454
Structure : Simple
Abstraction : Base

Description

The product initializes critical internal variables or data stores using inputs that can be modified by untrusted actors.

Extended Description

A product system should be reluctant to trust variables that have been initialized outside of its trust boundary, especially if they are initialized by users. The variables may have been initialized incorrectly. If an attacker can initialize the variable, then they can influence what the vulnerable system will do.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	1419	Incorrect Initialization of Resource	2280
CanAlsoBe	Ⓥ	456	Missing Initialization of a Variable	1089

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

Nature	Type	ID	Name	Page
MemberOf	🅲	452	Initialization and Cleanup Errors	2327

Applicable Platforms

Language : PHP *(Prevalence = Sometimes)*

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could gain access to and modify sensitive data or system information.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

A product system should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking (e.g. input validation) is performed when relying on input from outside a trust boundary.

Phase: Architecture and Design

Avoid any external control of variables. If necessary, restrict the variables that can be modified using an allowlist, and use a different namespace or naming convention if possible.

Demonstrative Examples

Example 1:

In the Java example below, a system property controls the debug level of the application.

Example Language: Java *(Bad)*

```java
int debugLevel = Integer.getInteger("com.domain.application.debugLevel").intValue();
```

If an attacker is able to modify the system property, then it may be possible to coax the application into divulging sensitive information by virtue of the fact that additional debug information is printed/exposed as the debug level increases.

Example 2:

This code checks the HTTP POST request for a debug switch, and enables a debug mode if the switch is set.

Example Language: PHP *(Bad)*

```php
$debugEnabled = false;
if ($_POST["debug"] == "true"){
  $debugEnabled = true;
}
/.../
function login($username, $password){
  if($debugEnabled){
    echo 'Debug Activated';
    phpinfo();
    $isAdmin = True;
    return True;
  }
}
```

Any user can activate the debug mode, gaining administrator privileges. An attacker may also use the information printed by the phpinfo() function to further exploit the system. .

This example also exhibits Information Exposure Through Debug Information (CWE-215)

Observed Examples

Reference	Description
CVE-2022-43468	WordPress module sets internal variables based on external inputs, allowing false reporting of the number of views *https://www.cve.org/CVERecord?id=CVE-2022-43468*
CVE-2000-0959	Does not clear dangerous environment variables, enabling symlink attack. *https://www.cve.org/CVERecord?id=CVE-2000-0959*
CVE-2001-0033	Specify alternate configuration directory in environment variable, enabling untrusted path. *https://www.cve.org/CVERecord?id=CVE-2001-0033*
CVE-2001-0872	Dangerous environment variable not cleansed. *https://www.cve.org/CVERecord?id=CVE-2001-0872*
CVE-2001-0084	Specify arbitrary modules using environment variable. *https://www.cve.org/CVERecord?id=CVE-2001-0084*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2417
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

Overlaps Missing variable initialization, especially in PHP.

Applicable Platform

This is often found in PHP due to register_globals and the common practice of storing library/ include files under the web document root so that they are available using a direct request.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			External initialization of trusted variables or values
Software Fault Patterns	SFP25		Tainted input to variable

CWE-455: Non-exit on Failed Initialization

Weakness ID : 455
Structure : Simple
Abstraction : Base

Description

The product does not exit or otherwise modify its operation when security-relevant errors occur during initialization, such as when a configuration file has a format error or a hardware security module (HSM) cannot be activated, which can cause the product to execute in a less secure fashion than intended by the administrator.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	636	Not Failing Securely ('Failing Open')	1401
ChildOf	C	665	Improper Initialization	1456
ChildOf	C	705	Incorrect Control Flow Scoping	1542

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

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	2327

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Modify Application Data Alter Execution Logic *The application could be placed in an insecure state that may allow an attacker to modify sensitive data or allow unintended logic to be executed.*	

Potential Mitigations

Phase: Implementation

Follow the principle of failing securely when an error occurs. The system should enter a state where it is not vulnerable and will not display sensitive error messages to a potential attacker.

Demonstrative Examples

Example 1:

The following code intends to limit certain operations to the administrator only.

Example Language: Perl *(Bad)*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
  $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
  DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Observed Examples

Reference	Description
CVE-2005-1345	Product does not trigger a fatal error if missing or invalid ACLs are in a configuration file. *https://www.cve.org/CVERecord?id=CVE-2005-1345*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2399
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Notes

Research Gap

Under-studied. These issues are not frequently reported, and it is difficult to find published examples.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Non-exit on Failed Initialization

CWE-456: Missing Initialization of a Variable

Weakness ID : 456
Structure : Simple
Abstraction : Variant

Description

The product does not initialize critical variables, which causes the execution environment to use unexpected values.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	909	Missing Initialization of Resource	1797
CanPrecede	Ⓑ	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	201
CanPrecede	Ⓥ	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	236
CanPrecede	Ⓑ	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	304
CanPrecede	Ⓥ	457	Use of Uninitialized Variable	1094

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	665	Improper Initialization	1456

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	665	Improper Initialization	1456

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Quality Degradation Varies by Context *The uninitialized data may be invalid, causing logic errors within the program. In some cases, this could result in a security problem.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Check that critical variables are initialized.

Phase: Testing

Use a static analysis tool to spot non-initialized variables.

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Example Language: *(Attack)*

```
123:
```

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

Example 2:

Here, an uninitialized field in a Java class is used in a seldom-called method, which would cause a NullPointerException to be thrown.

Example Language: Java *(Bad)*

```
private User user;
public void someMethod() {
  // Do something interesting.
  ...
  // Throws NPE if user hasn't been properly initialized.
  String username = user.getName();
}
```

Example 3:

This code first authenticates a user, then allows a delete command if the user is an administrator.

Example Language: PHP *(Bad)*

```
if (authenticate($username,$password) && setAdmin($username)){
   $isAdmin = true;
}
/.../
if ($isAdmin){
   deleteUser($userToDelete);
}
```

The $isAdmin variable is set to true if the user is an admin, but is uninitialized otherwise. If PHP's register_globals feature is enabled, an attacker can set uninitialized variables like $isAdmin to arbitrary values, in this case gaining administrator privileges by setting $isAdmin to true.

Example 4:

In the following Java code the BankManager class uses the user variable of the class User to allow authorized users to perform bank manager tasks. The user variable is initialized within the method setUser that retrieves the User from the User database. The user is then authenticated as unauthorized user through the method authenticateUser.

Example Language: Java *(Bad)*

```
public class BankManager {
   // user allowed to perform bank manager tasks
   private User user = null;
   private boolean isUserAuthentic = false;
   // constructor for BankManager class
   public BankManager() {
      ...
   }
   // retrieve user from database of users
   public User getUserFromUserDatabase(String username){
      ...
   }
   // set user variable using username
   public void setUser(String username) {
      this.user = getUserFromUserDatabase(username);
   }
   // authenticate user
   public boolean authenticateUser(String username, String password) {
      if (username.equals(user.getUsername()) && password.equals(user.getPassword())) {
         isUserAuthentic = true;
      }
      return isUserAuthentic;
   }
   // methods for performing bank manager tasks
   ...
}
```

However, if the method setUser is not called before authenticateUser then the user variable will not have been initialized and will result in a NullPointerException. The code should verify that the user variable has been initialized before it is used, as in the following code.

Example Language: Java *(Good)*

```
public class BankManager {
   // user allowed to perform bank manager tasks
   private User user = null;
   private boolean isUserAuthentic = false;
   // constructor for BankManager class
   public BankManager(String username) {
      user = getUserFromUserDatabase(username);
   }
   // retrieve user from database of users
   public User getUserFromUserDatabase(String username) {...}
```

```
  // authenticate user
  public boolean authenticateUser(String username, String password) {
    if (user == null) {
      System.out.println("Cannot find user " + username);
    }
    else {
      if (password.equals(user.getPassword())) {
        isUserAuthentic = true;
      }
    }
    return isUserAuthentic;
  }
    // methods for performing bank manager tasks
    ...
}
```

Example 5:

This example will leave test_string in an unknown condition when i is the same value as err_val, because test_string is not initialized (CWE-456). Depending on where this code segment appears (e.g. within a function body), test_string might be random if it is stored on the heap or stack. If the variable is declared in static memory, it might be zero or NULL. Compiler optimization might contribute to the unpredictability of this address.

Example Language: C *(Bad)*

```
char *test_string;
if (i != err_val)
{
  test_string = "Hello World!";
}
printf("%s", test_string);
```

When the printf() is reached, test_string might be an unexpected address, so the printf might print junk strings (CWE-457).

To fix this code, there are a couple approaches to making sure that test_string has been properly set once it reaches the printf().

One solution would be to set test_string to an acceptable default before the conditional:

Example Language: C *(Good)*

```
char *test_string = "Done at the beginning";
if (i != err_val)
{
  test_string = "Hello World!";
}
printf("%s", test_string);
```

Another solution is to ensure that each branch of the conditional - including the default/else branch - could ensure that test_string is set:

Example Language: C *(Good)*

```
char *test_string;
if (i != err_val)
{
  test_string = "Hello World!";
}
else {
  test_string = "Done on the other side!";
}
printf("%s", test_string);
```