


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

Nature	Type	ID	Name	V	Page
MemberOf		949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2416
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

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> >.2024-11-17.

[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, 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		345	Insufficient Verification of Data Authenticity	858
PeerOf		436	Interpretation Conflict	1065
PeerOf		434	Unrestricted Upload of File with Dangerous Type	1055

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

Nature	Type	ID	Name	Page
MemberOf		1214	Data Integrity Issues	2498

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

Nature	Type	ID	Name		Page
MemberOf		993	SFP Secondary Cluster: Incorrect Input Handling	888	2438
MemberOf		1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

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		346	Origin Validation Error	860
Requires		441	Unintended Proxy or Intermediary ('Confused Deputy')	1072
Requires		642	External Control of Critical State Data	1422
Requires		613	Insufficient Session Expiration	1380




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, 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		345	Insufficient Verification of Data Authenticity	858
PeerOf		79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	168
CanFollow		1275	Sensitive Cookie with Improper SameSite Attribute	2123

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	858

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

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	Gain Privileges or Assume Identity	
Integrity	Bypass Protection Mechanism	
Availability	Read Application Data	
Non-Repudiation	Modify Application Data	
Access Control	DoS: Crash, Exit, or Restart	
<p><i>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.</i></p>		

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

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2573
MemberOf	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	629	2353
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2373
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2375
MemberOf	C	814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)	809	2379
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2392
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	936	OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)	928	2413
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

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
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](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_\(computer_worm\)](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, 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		345	Insufficient Verification of Data Authenticity	858
PeerOf		354	Improper Validation of Integrity Check Value	883
PeerOf		354	Improper Validation of Integrity Check Value	883

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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

Nature	Type	ID	Name	Page
MemberOf		1214	Data Integrity Issues	2498

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

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

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	2419
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

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> >.2024-11-17.

[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, 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		754	Improper Check for Unusual or Exceptional Conditions	1577
ChildOf		345	Insufficient Verification of Data Authenticity	858
PeerOf		353	Missing Support for Integrity Check	881
PeerOf		353	Missing Support for Integrity Check	881

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	858

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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

Nature	Type	ID	Name	Page
MemberOf		1214	Data Integrity Issues	2498

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Other	Other	
	<i>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.</i>	
Integrity	Other	
Other	<i>Data that is parsed and used may be corrupted.</i>	
Non-Repudiation	Hide Activities	
Other	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);
    clien = 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	2588
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	2438
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

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> >.2024-11-17.

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, 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		221	Information Loss or Omission	563

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

Nature	Type	ID	Name	Page
MemberOf		355	User Interface Security Issues	2341

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

Nature	Type	ID	Name	V	Page
MemberOf		996	SFP Secondary Cluster: Security	888	2439
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

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, 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		693	Protection Mechanism Failure	1529
ParentOf		450	Multiple Interpretations of UI Input	1085

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

Nature	Type	ID	Name	Page
MemberOf		355	User Interface Security Issues	2341

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		996	SFP Secondary Cluster: Security	888	2439
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

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		693	Protection Mechanism Failure	1529
ChildOf		573	Improper Following of Specification by Caller	1307
PeerOf		325	Missing Cryptographic Step	801
CanAlsoBe		290	Authentication Bypass by Spoofing	712
CanAlsoBe		345	Insufficient Verification of Data Authenticity	858

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

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

Nature	Type	ID	Name	V	Page
MemberOf		978	SFP Secondary Cluster: Implementation	888	2429
MemberOf		1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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.

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		200	Exposure of Sensitive Information to an Unauthorized Actor	511

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		199	Information Management Errors	2333

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 to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted.

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 + ":" + timestamp);
```

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

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

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	2335
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	2457
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

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.

Other

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.

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](http://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, 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		345	Insufficient Verification of Data Authenticity	858
ParentOf		422	Unprotected Windows Messaging Channel ('Shatter')	1029

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Gain Privileges or Assume Identity	
Confidentiality	Execute Unauthorized Code or Commands	
Availability		
Access Control	<i>If one trusts the system-event information and executes commands based on it, one could potentially take actions based on a spoofed identity.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf		949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2416
MemberOf		1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

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> >.2024-11-17.

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

Weakness ID : 362

Structure : Simple

Abstraction : Class

Description

The product contains a concurrent code sequence that 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 operating concurrently.

Extended Description

A race condition occurs within concurrent environments, and it 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.

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, 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		691	Insufficient Control Flow Management	1525
ParentOf		364	Signal Handler Race Condition	905
ParentOf		366	Race Condition within a Thread	910
ParentOf		367	Time-of-check Time-of-use (TOCTOU) Race Condition	913
ParentOf		368	Context Switching Race Condition	918
ParentOf		421	Race Condition During Access to Alternate Channel	1028
ParentOf		689	Permission Race Condition During Resource Copy	1521
ParentOf		1223	Race Condition for Write-Once Attributes	2011
ParentOf		1298	Hardware Logic Contains Race Conditions	2170
CanFollow		662	Improper Synchronization	1457
CanPrecede		416	Use After Free	1019
CanPrecede		476	NULL Pointer Dereference	1139

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

Nature	Type	ID	Name	Page
ParentOf		367	Time-of-check Time-of-use (TOCTOU) Race Condition	913

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Language : Java (Prevalence = Sometimes)

Technology : Mobile (Prevalence = Undetermined)

Technology : ICS/OT (Prevalence = Undetermined)

Alternate Terms

Race Condition :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) <i>When a race condition makes it possible to bypass a resource cleanup routine or trigger multiple initialization routines, it may lead to resource exhaustion.</i>	
Availability	DoS: Crash, Exit, or Restart DoS: Instability <i>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.</i>	
Confidentiality Integrity	Read Files or Directories Read Application Data <i>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).</i>	
Access Control	Execute Unauthorized Code or Commands Gain Privileges or Assume Identity Bypass Protection Mechanism <i>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.</i>	

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.

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

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)

```
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 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

Reference	Description
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
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

Reference	Description
	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2573
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	2373
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2375
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2387
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2401
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	2433
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2469
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1365	ICS Communications: Unreliability	1358	2523
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf	C	1376	ICS Engineering (Construction/Deployment): Security Gaps in Commissioning	1358	2533
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621

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. Dobbs. 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, 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		367	Time-of-check Time-of-use (TOCTOU) Race Condition	913
CanPrecede		59	Improper Link Resolution Before File Access ('Link Following')	112

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2372
MemberOf		988	SFP Secondary Cluster: Race Condition Window	888	2433
MemberOf		1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2484
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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, 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		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
ParentOf		432	Dangerous Signal Handler not Disabled During Sensitive Operations	1052
ParentOf		828	Signal Handler with Functionality that is not Asynchronous-Safe	1746
ParentOf		831	Signal Handler Function Associated with Multiple Signals	1758
CanPrecede		123	Write-what-where Condition	329
CanPrecede		415	Double Free	1015
CanPrecede		416	Use After Free	1019

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

Nature	Type	ID	Name	Page
MemberOf		387	Signal Errors	2343
MemberOf		557	Concurrency Issues	2350

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
	Execute Unauthorized Code or Commands	
	<i>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.</i>	
Access Control	Gain Privileges or Assume Identity	
	<i>If a signal handler interrupts code that is executing with privileges, it may be possible that the signal handler will also be executed with elevated privileges, possibly making subsequent exploits more severe.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2341
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

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> >.2024-11-17.
- [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, 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		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

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	Alter Execution Logic	
Other	Unexpected State	
	<i>The main problem is that -- if a lock is overcome -- data could be altered in a bad state.</i>	

Detection Methods**Automated Static Analysis**

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

*Effectiveness = High***Potential Mitigations****Phase: Architecture and Design**

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 class Race {
    static int foo = 0;
    public static void main() {
        new Threader().start();
        foo = 1;
    }
    public static class Threader extends Thread {
        public void run() {
            System.out.println(foo);
        }
    }
}
```

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	2372
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2387
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2401
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2469
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2483
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

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> >.2024-11-17.

[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, 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		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
ParentOf		363	Race Condition Enabling Link Following	904
PeerOf		386	Symbolic Name not Mapping to Correct Object	949
CanFollow		609	Double-Checked Locking	1371

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

Nature	Type	ID	Name	Page
ChildOf		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

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 <i>The attacker can gain access to otherwise unauthorized resources.</i>	
Integrity Other	Modify Application Data Modify Files or Directories Modify Memory Other <i>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.</i>	
Integrity Other	Other <i>The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.</i>	
Non-Repudiation	Hide Activities <i>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.</i>	
Non-Repudiation Other	Other <i>In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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

Nature	Type	ID	Name		Page
MemberOf		361	7PK - Time and State	700	2341
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf		884	CWE Cross-section	884	2588
MemberOf		988	SFP Secondary Cluster: Race Condition Window	888	2433
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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> >.2024-11-17.
- [REF-367]Dan Tsafir, 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/tsafir.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, 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		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
CanAlsoBe		364	Signal Handler Race Condition	905

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

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	Modify Application Data	
Confidentiality	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 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

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

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1507

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1507

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1507

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1507

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

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	2333

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
	<i>A Divide by Zero results in a crash.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

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

Nature	Type	ID	Name	V	Page
MemberOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf		738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf		739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2364
MemberOf		848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	2384
MemberOf		872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395
MemberOf		873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2396
MemberOf		884	CWE Cross-section	884	2588
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf		1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	2466
MemberOf		1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2477
MemberOf		1408	Comprehensive Categorization: Incorrect Calculation	1400	2555

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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](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, 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		299	Improper Check for Certificate Revocation	734
PeerOf		296	Improper Following of a Certificate's Chain of Trust	726
PeerOf		297	Improper Validation of Certificate with Host Mismatch	729
PeerOf		298	Improper Validation of Certificate Expiration	733

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

Nature	Type	ID	Name	Page
MemberOf		1014	Identify Actors	2450

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
	<i>Trust may be assigned to an entity who is not who it claims to be.</i>	
Integrity	Modify Application Data <i>Data from an untrusted (and possibly malicious) source may be integrated.</i>	
Confidentiality	Read Application Data <i>Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf		988	SFP Secondary Cluster: Race Condition Window	888	2433
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

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> >.2024-11-17.

[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, 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		664	Improper Control of a Resource Through its Lifetime	1463

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

Nature	Type	ID	Name	Page
MemberOf		371	State Issues	2342

Applicable Platforms




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

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		371	State Issues	2342

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 <i>Potentially data could be tampered with by another function which should not have been tampered with.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf		849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf		1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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> >.2024-11-17.
- [REF-374]Tony Sintès. "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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		371	State Issues	2342

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 <i>Potentially data could be tampered with by another function which should not have been tampered with.</i>	

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

Nature	Type	ID	Name		Page
MemberOf		849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf		1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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> >.2024-11-17.

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478
ParentOf		378	Creation of Temporary File With Insecure Permissions	935
ParentOf		379	Creation of Temporary File in Directory with Insecure Permissions	937

Applicable Platforms

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

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Nature	Type	ID	Name	V	Page
MemberOf		361	7PK - Time and State	700	2341
MemberOf		857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf		964	SFP Secondary Cluster: Exposure Temporary File	888	2423
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf		1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2483
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf		1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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(), tmpnam(), 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, 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		377	Insecure Temporary File	932

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

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2501

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
	<i>If the temporary file can be read by the attacker, sensitive information may be in that file which could be revealed.</i>	
Authorization Other	Other <i>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.</i>	
Integrity Other	Other <i>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.</i>	

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 <code>createTempFile()</code> , 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		964	SFP Secondary Cluster: Exposure Temporary File	888	2423
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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> >.2024-11-17.

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`, `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		377	Insecure Temporary File	932

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

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2501

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Requirements

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 <code>/tmp</code> , 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 <code>File.createTempFile()</code> 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf		964	SFP Secondary Cluster: Exposure Temporary File	888	2423
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Guessed or visible temporary file

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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> >.2024-11-17.

[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, 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		705	Incorrect Control Flow Scoping	1550

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		Page
MemberOf		361	7PK - Time and State	700	2341
MemberOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf		851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf		1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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, 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		695	Use of Low-Level Functionality	1533

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	<input checked="" type="checkbox"/>	Page
MemberOf		361	7PK - Time and State	700	2341
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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		346	Origin Validation Error	860
Requires		472	External Control of Assumed-Immutable Web Parameter	1131
Requires		441	Unintended Proxy or Intermediary ('Confused Deputy')	1072

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

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

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1373
CanFollow		340	Generation of Predictable Numbers or Identifiers	849

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

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1373

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

Nature	Type	ID	Name	Page
MemberOf		1018	Manage User Sessions	2453

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 properly 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	2341
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2356
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2410
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2457
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2515
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

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

CAPEC-ID	Attack Pattern Name
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, 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		514	Covert Channel	1227
CanFollow		208	Observable Timing Discrepancy	537

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

Nature	Type	ID	Name	Page
MemberOf		417	Communication Channel Errors	2347

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Other	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		968	SFP Secondary Cluster: Covert Channel	888	2425
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

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> >.2024-11-17.

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < <https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf> >.2024-11-17.

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

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

Nature	Type	ID	Name	Page
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1553
PeerOf		367	Time-of-check Time-of-use (TOCTOU) Race Condition	913
PeerOf		486	Comparison of Classes by Name	1172
PeerOf		610	Externally Controlled Reference to a Resource in Another Sphere	1373

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

Applicable Platforms



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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity <i>The attacker can gain access to otherwise unauthorized resources.</i>	
Integrity Confidentiality Other	Modify Application Data Modify Files or Directories Read Application Data Read Files or Directories Other <i>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.</i>	
Integrity Other	Modify Application Data Other <i>The resource in question, or other resources (through the corrupted one) may be changed in undesirable ways by a malicious user.</i>	
Non-Repudiation	Hide Activities <i>If a file or other resource is written in this method, as opposed to a valid way, logging of the activity may not occur.</i>	
Non-Repudiation Integrity	Modify Files or Directories <i>In some cases it may be possible to delete files that a malicious user might not otherwise have access to -- such as log files.</i>	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2430
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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> >.2024-11-17.

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, 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		755	Improper Handling of Exceptional Conditions	1585
PeerOf		600	Uncaught Exception in Servlet	1352
CanPrecede		401	Missing Release of Memory after Effective Lifetime	980

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State Alter Execution Logic	
	<i>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.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2359
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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> >.2024-11-17.

[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, 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		754	Improper Check for Unusual or Exceptional Conditions	1577

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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

Nature	Type	ID	Name	Page
ChildOf		703	Improper Check or Handling of Exceptional Conditions	1544

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

Nature	Type	ID	Name	Page
ChildOf		703	Improper Check or Handling of Exceptional Conditions	1544

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	2343
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2359
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2371
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	2478
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2482
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2484
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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> >.2024-11-17.

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, 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		684	Incorrect Provision of Specified Functionality	1514
ChildOf		755	Improper Handling of Exceptional Conditions	1585


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

Nature	Type	ID	Name	Page
MemberOf		1012	Cross Cutting	2448

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

Nature	Type	ID	Name	Page
ChildOf		703	Improper Check or Handling of Exceptional Conditions	1544

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

Nature	Type	ID	Name	Page
ChildOf		703	Improper Check or Handling of Exceptional Conditions	1544

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	
<i>Errors that are not properly reported could place the system in an unexpected state that could lead to unintended behaviors.</i>		

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

Nature	Type	ID	Name	V	Page
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	2388
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	2471
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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, 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		703	Improper Check or Handling of Exceptional Conditions	1544
ChildOf		684	Incorrect Provision of Specified Functionality	1514

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Alter Execution Logic	
	<i>This weakness could place the system in a state that could lead unexpected logic to be executed or other unintended behaviors.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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, 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		754	Improper Check for Unusual or Exceptional Conditions	1577

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

Applicable Platforms

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

Common Consequences







Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2359
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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, 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		755	Improper Handling of Exceptional Conditions	1585
ChildOf		705	Incorrect Control Flow Scoping	1550

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

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

Nature	Type	ID	Name	V	Page
MemberOf		388	7PK - Errors	700	2343
MemberOf		851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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, 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		221	Information Loss or Omission	563
ChildOf		755	Improper Handling of Exceptional Conditions	1585
ChildOf		705	Incorrect Control Flow Scoping	1550

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

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	Hide Activities	
Other	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		388	7PK - Errors	700	2343
MemberOf		960	SFP Secondary Cluster: Ambiguous Exception Type	888	2420
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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, 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		221	Information Loss or Omission	563
ChildOf		703	Improper Check or Handling of Exceptional Conditions	1544
ChildOf		705	Incorrect Control Flow Scoping	1550

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2344

Applicable Platforms

Language : C++ (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Other	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	2343
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf	C	960	SFP Secondary Cluster: Ambiguous Exception Type	888	2420
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf		405	Asymmetric Resource Consumption (Amplification)	993
ParentOf		770	Allocation of Resources Without Limits or Throttling	1622
ParentOf		771	Missing Reference to Active Allocated Resource	1631

Nature	Type	ID	Name	Page
ParentOf		779	Logging of Excessive Data	1651
ParentOf		920	Improper Restriction of Power Consumption	1832
ParentOf		1235	Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations	2029
ParentOf		1246	Improper Write Handling in Limited-write Non-Volatile Memories	2054
CanFollow		410	Insufficient Resource Pool	1005

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

Nature	Type	ID	Name	Page
ParentOf		770	Allocation of Resources Without Limits or Throttling	1622
ParentOf		920	Improper Restriction of Power Consumption	1832

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) <i>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.</i>	
Access Control Other	Bypass Protection Mechanism Other <i>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.</i>	

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)

```
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2390
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2391
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	2432
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	2472
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2474
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

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
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> >.2024-11-17.

[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, 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		772	Missing Release of Resource after Effective Lifetime	1632
CanFollow		390	Detection of Error Condition Without Action	950

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	987

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	987

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) <i>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.</i>	
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 14882:2003), `std::shared_ptr` and `std::weak_ptr` (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

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 <code>free()</code> 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2344
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2391
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2474
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1238	SFP Primary Cluster: Failure to Release Memory	888	2503
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

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> >.2024-11-17.

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478
ParentOf		403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	985
ParentOf		619	Dangling Database Cursor ('Cursor Injection')	1391

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

Nature	Type	ID	Name		Page
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2421

Nature	Type	ID	Name	V	Page
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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, 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		402	Transmission of Private Resources into a New Sphere ('Resource Leak')	984

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

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	Read Application Data	
Integrity	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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, 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		664	Improper Control of a Resource Through its Lifetime	1463
ParentOf		299	Improper Check for Certificate Revocation	734
ParentOf		459	Incomplete Cleanup	1106
ParentOf		763	Release of Invalid Pointer or Reference	1608
ParentOf		772	Missing Release of Resource after Effective Lifetime	1632
ParentOf		1266	Improper Scrubbing of Sensitive Data from Decommissioned Device	2104
PeerOf		405	Asymmetric Resource Consumption (Amplification)	993
PeerOf		239	Failure to Handle Incomplete Element	589
CanPrecede		619	Dangling Database Cursor ('Cursor Injection')	1391






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

Nature	Type	ID	Name	Page
ParentOf		401	Missing Release of Memory after Effective Lifetime	980
ParentOf		459	Incomplete Cleanup	1106
ParentOf		763	Release of Invalid Pointer or Reference	1608
ParentOf		772	Missing Release of Resource after Effective Lifetime	1632

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

Nature	Type	ID	Name	Page
ParentOf		401	Missing Release of Memory after Effective Lifetime	980
ParentOf		772	Missing Release of Resource after Effective Lifetime	1632
ParentOf		775	Missing Release of File Descriptor or Handle after Effective Lifetime	1640

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

Nature	Type	ID	Name	Page
ParentOf		761	Free of Pointer not at Start of Buffer	1601
ParentOf		762	Mismatched Memory Management Routines	1605
ParentOf		763	Release of Invalid Pointer or Reference	1608
ParentOf		772	Missing Release of Resource after Effective Lifetime	1632
ParentOf		775	Missing Release of File Descriptor or Handle after Effective Lifetime	1640

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 <i>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.</i>	
Confidentiality	Read Application Data <i>When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.</i>	

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 BufferedReader 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)
{
    BufferedReader fil = new BufferedReader(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)
```

```
{
  BufferedReader fil = new BufferedReader(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	2344
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2374
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2401
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	2431
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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

CAPEC-ID	Attack Pattern Name
494	TCP Fragmentation
495	UDP Fragmentation
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, 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		400	Uncontrolled Resource Consumption	971
ParentOf		406	Insufficient Control of Network Message Volume (Network Amplification)	997
ParentOf		407	Inefficient Algorithmic Complexity	999
ParentOf		408	Incorrect Behavior Order: Early Amplification	1002
ParentOf		409	Improper Handling of Highly Compressed Data (Data Amplification)	1004
ParentOf		776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1642
ParentOf		1050	Excessive Platform Resource Consumption within a Loop	1895
ParentOf		1072	Data Resource Access without Use of Connection Pooling	1921
ParentOf		1073	Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses	1922
ParentOf		1084	Invokable Control Element with Excessive File or Data Access Operations	1933
ParentOf		1089	Large Data Table with Excessive Number of Indices	1938
ParentOf		1094	Excessive Index Range Scan for a Data Resource	1943
ParentOf		1176	Inefficient CPU Computation	1980
PeerOf		404	Improper Resource Shutdown or Release	987

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) <i>Sometimes this is a factor in "flood" attacks, but other types of amplification exist.</i>	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 $(\backslash w+\backslash s?)^*\$$ 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 $\backslash 2$ which prevents the backtracking. The modified example is:

*Example Language: JavaScript**(Good)*

```
var test_string = "Bad characters: $@#";
var good_pattern = /^(?=(w+)\backslash 2\s?)*$/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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360

Nature	Type	ID	Name	V	Page
MemberOf		855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	2388
MemberOf		857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	2471
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		405	Asymmetric Resource Consumption (Amplification)	993
CanFollow		941	Incorrectly Specified Destination in a Communication Channel	1855

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) <i>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.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1382	ICS Operations (& Maintenance): Emerging Energy Technologies	1358	2538
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		405	Asymmetric Resource Consumption (Amplification)	993
ParentOf		1333	Inefficient Regular Expression Complexity	2243

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

Nature	Type	ID	Name	Page
ParentOf		1333	Inefficient Regular Expression Complexity	2243

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) <i>The typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur.</i>	

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

Nature	Type	ID	Name		Page
MemberOf		884	CWE Cross-section	884	2588
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		405	Asymmetric Resource Consumption (Amplification)	993
ChildOf		696	Incorrect Behavior Order	1535

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification	

Scope	Impact	Likelihood
	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) <i>System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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, 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		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		19	Data Processing Errors	2330

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) <i>System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf		845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	2383
MemberOf		884	CWE Cross-section	884	2588
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	2465
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		664	Improper Control of a Resource Through its Lifetime	1463
CanPrecede		400	Uncontrolled Resource Consumption	971

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Integrity	Other	
Other	<i>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.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf		855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	2388
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	2471
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		667	Improper Locking	1472
CanAlsoBe		410	Insufficient Resource Pool	1005

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>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.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf		361	7PK - Time and State	700	2341
MemberOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf		853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2387
MemberOf		989	SFP Secondary Cluster: Unrestricted Lock	888	2434
MemberOf		1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	2470
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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, 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		667	Improper Locking	1472
ParentOf		591	Sensitive Data Storage in Improperly Locked Memory	1338

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Availability	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 insufficient 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2387
MemberOf		853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2387
MemberOf		986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf		1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2469
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Locking

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	VNA00-J		Ensure visibility when accessing shared primitive variables
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, 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		667	Improper Locking	1472

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Availability	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.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2004-1056

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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, 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		666	Operation on Resource in Wrong Phase of Lifetime	1471
ChildOf		1341	Multiple Releases of Same Resource or Handle	2258
ChildOf		825	Expired Pointer Dereference	1741
PeerOf		123	Write-what-where Condition	329
PeerOf		416	Use After Free	1019
CanFollow		364	Signal Handler Race Condition	905

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

Nature	Type	ID	Name	Page
ChildOf		672	Operation on a Resource after Expiration or Release	1488

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	1488

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	1488

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Alternate Terms

Double-free :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	<i>Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.</i>	

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.








Reference	Description
	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
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		398	7PK - Code Quality	700	2344
MemberOf		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf		969	SFP Secondary Cluster: Faulty Memory Release	888	2425
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1237	SFP Primary Cluster: Faulty Resource Release	888	2503
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

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> >.2024-11-17.

CWE-416: Use After Free

Weakness ID : 416

Structure : Simple

Abstraction : Variant









Description

The product reuses or references memory after it has been freed. At some point afterward, the memory may be allocated again and saved in another pointer, while the original pointer references a location somewhere within the new allocation. Any operations using the original pointer are no longer valid because the memory "belongs" to the code that operates on the new pointer.

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		825	Expired Pointer Dereference	1741
PeerOf		415	Double Free	1015
CanFollow		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
CanFollow		364	Signal Handler Race Condition	905
CanFollow		754	Improper Check for Unusual or Exceptional Conditions	1577
CanFollow		1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	2100
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	310
CanPrecede		123	Write-what-where Condition	329

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

Nature	Type	ID	Name	Page
ChildOf		672	Operation on a Resource after Expiration or Release	1488

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	1488

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	1488

Weakness Ordinalities

Resultant : If the product accesses a previously-freed pointer, then it means that a separate weakness or error already occurred previously, such as a race condition, an unexpected or poorly handled error condition, confusion over which part of the program is responsible for freeing the memory, performing the free too soon, etc.

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Alternate Terms

Dangling pointer : a pointer that no longer points to valid memory, often after it has been freed

UAF : commonly used acronym for Use After Free

Use-After-Free :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory <i>The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.</i>	
Availability	DoS: Crash, Exit, or Restart <i>If chunk consolidation occurs after the use of previously freed data, the process may crash when invalid data is used as chunk information.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>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. 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.</i>	

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

Strategy = Language Selection

Choose a language that provides automatic memory management.

Phase: Implementation

Strategy = Attack Surface Reduction

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.

Effectiveness = Defense in Depth

If a bug causes an attempted access of this pointer, then a NULL dereference could still lead to a crash or other unexpected behavior, but it will reduce or eliminate the risk of code execution.

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 insufficient 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. 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















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-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
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		398	7PK - Code Quality	700	2344
MemberOf		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf		808	2010 Top 25 - Weaknesses On the Cusp	800	2376
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf		983	SFP Secondary Cluster: Faulty Resource Use	888	2431
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf		1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf		1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf		1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546
MemberOf		1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf		1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

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> >.2024-11-17.

[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, 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		923	Improper Restriction of Communication Channel to Intended Endpoints	1836

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		417	Communication Channel Errors	2347

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 functionality 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		956	SFP Secondary Cluster: Channel Attack	888	2418
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

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, 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		923	Improper Restriction of Communication Channel to Intended Endpoints	1836
ParentOf		421	Race Condition During Access to Alternate Channel	1028
ParentOf		422	Unprotected Windows Messaging Channel ('Shatter')	1029

Nature	Type	ID	Name	Page
ParentOf		1299	Missing Protection Mechanism for Alternate Hardware Interface	2174
PeerOf		288	Authentication Bypass Using an Alternate Path or Channel	707

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		417	Communication Channel Errors	2347

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;
        end
end
```



```

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

Nature	Type	ID	Name	V	Page
MemberOf		956	SFP Secondary Cluster: Channel Attack	888	2418
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

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, 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		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
ChildOf		420	Unprotected Alternate Channel	1025

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

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

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	2418
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

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

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

Nature	Type	ID	Name	Page
ChildOf	B	360	Trust of System Event Data	894
ChildOf	B	420	Unprotected Alternate Channel	1025

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. 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		953	SFP Secondary Cluster: Missing Endpoint Authentication	888	2418
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1529
ChildOf		638	Not Using Complete Mediation	1413
ParentOf		425	Direct Request ('Forced Browsing')	1032

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

Nature	Type	ID	Name		Page
MemberOf		945	SFP Secondary Cluster: Insecure Resource Access	888	2415
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504

Nature	Type	ID	Name	V	Page
MemberOf		1308	CISQ Quality Measures - Security	1305	2506
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1340	CISQ Data Protection Measures	1340	2611
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

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		288	Authentication Bypass Using an Alternate Path or Channel	707
ChildOf		424	Improper Protection of Alternate Path	1031
ChildOf		862	Missing Authorization	1789
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	242
CanPrecede		471	Modification of Assumed-Immutable Data (MAID)	1129

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

Nature	Type	ID	Name	Page
ChildOf		862	Missing Authorization	1789

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2497
MemberOf		417	Communication Channel Errors	2347

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	Read Application Data	
Integrity	Modify Application Data	
Availability	Execute Unauthorized Code or Commands	
Access Control	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.

Reference	Description
	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. 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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	2355
MemberOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2355
MemberOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2356
MemberOf		953	SFP Secondary Cluster: Missing Endpoint Authentication	888	2418
MemberOf		1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	2458
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

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'

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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, 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		673	External Influence of Sphere Definition	1492
ChildOf		642	External Control of Critical State Data	1422
PeerOf		427	Uncontrolled Search Path Element	1040
PeerOf		428	Unquoted Search Path or Element	1047

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2501

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	Gain Privileges or Assume Identity	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	<i>There is the potential for arbitrary code execution with privileges of the vulnerable program.</i>	
Access Control		
Availability	DoS: Crash, Exit, or Restart	
	<i>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.</i>	
Confidentiality	Read Files or Directories	
	<i>The program could send the output of unauthorized files to the attacker.</i>	

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:

The following code from a system utility uses the system property APPHOME to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

Example Language: Java

(Bad)

```
...
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

The code above allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

Example 3:

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 4:

The following code is from a web application that allows users access to an interface through which they can update their password on the system. 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
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	2369
MemberOf		752	2009 Top 25 - Risky Resource Management	750	2374
MemberOf		808	2010 Top 25 - Weaknesses On the Cusp	800	2376

Nature	Type	ID	Name	V	Page
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2399
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

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> >.2024-11-17.
- [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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478
PeerOf		426	Untrusted Search Path	1035

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2501

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 directories 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
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 malicious 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

Reference	Description
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
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478
PeerOf		426	Untrusted Search Path	1035

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		1219	File Handling Issues	2501

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

Nature	Type	ID	Name	View	Page
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2430

Nature	Type	ID	Name	V	Page
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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		691	Insufficient Control Flow Management	1525
PeerOf		434	Unrestricted Upload of File with Dangerous Type	1055
PeerOf		434	Unrestricted Upload of File with Dangerous Type	1055
CanPrecede		433	Unparsed Raw Web Content Delivery	1053

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

Nature	Type	ID	Name	Page
MemberOf		429	Handler Errors	2347

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	Varies by Context	
Other	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
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1525
CanPrecede	ⓧ	433	Unparsed Raw Web Content Delivery	1053

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

Nature	Type	ID	Name	Page
MemberOf	ⓐ	429	Handler Errors	2347

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

Nature	Type	ID	Name	V	Page
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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, 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		364	Signal Handler Race Condition	905

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

Nature	Type	ID	Name		Page
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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, 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		219	Storage of File with Sensitive Data Under Web Root	560
CanFollow		178	Improper Handling of Case Sensitivity	451
CanFollow		430	Deployment of Wrong Handler	1049
CanFollow		431	Missing Handler	1051

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

Reference	Description
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name		Page
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

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 upload or transfer of dangerous file types that are automatically processed within its 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, 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		669	Incorrect Resource Transfer Between Spheres	1480
PeerOf		351	Insufficient Type Distinction	873
PeerOf		430	Deployment of Wrong Handler	1049
PeerOf		436	Interpretation Conflict	1065
PeerOf		430	Deployment of Wrong Handler	1049

Nature	Type	ID	Name	Page
CanFollow		73	External Control of File Name or Path	133
CanFollow		183	Permissive List of Allowed Inputs	464
CanFollow		184	Incomplete List of Disallowed Inputs	466

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

Nature	Type	ID	Name	Page
ChildOf		669	Incorrect Resource Transfer Between Spheres	1480

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		429	Handler Errors	2347

Weakness Ordinalities

Primary : This can be primary when there is no check for the file type at all.

Resultant : This can be resultant when use of double extensions (e.g. ".php.gif") bypasses a 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 <i>Arbitrary code execution is possible if an uploaded file is interpreted and executed as code by the recipient. This is especially true for web-server extensions such as .asp and .php 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.</i>	

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("\n"));
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.write(line);
                        bw.newLine();
                        bw.flush();
                    }
                }
            } //end of for loop
            bw.close();
        } catch (IOException ex) {...}
        // output successful upload response HTML page
    }
    // output unsuccessful upload response HTML page
    else
```

```

    {...}
  }
  ...
}

```

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

Nature	Type	ID	Name		Page
MemberOf		714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	2352
MemberOf		801	2010 Top 25 - Insecure Interaction Between Components	800	2375

Nature	Type	ID	Name	V	Page
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2378
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2392
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

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, 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
MemberOf		1000	Research Concepts	2596
ParentOf		188	Reliance on Data/Memory Layout	476
ParentOf		436	Interpretation Conflict	1065
ParentOf		439	Behavioral Change in New Version or Environment	1068
ParentOf		1038	Insecure Automated Optimizations	1881

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	

Scope	Impact	Likelihood
-------	--------	------------

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

Nature	Type	ID	Name	V	Page
MemberOf		957	SFP Secondary Cluster: Protocol Error	888	2419
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2545

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, 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	I P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	1063
ParentOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	194
ParentOf	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Request/Response Splitting')	277
ParentOf	B	115	Misinterpretation of Input	286
ParentOf	B	437	Incomplete Model of Endpoint Features	1067
ParentOf	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling')	1075
ParentOf	V	626	Null Byte Interaction Error (Poison Null Byte)	1403
ParentOf	V	650	Trusting HTTP Permission Methods on the Server Side	1441
PeerOf	B	351	Insufficient Type Distinction	873
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	1055

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

Nature	Type	ID	Name	Page
ParentOf	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling')	1075

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	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
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		957	SFP Secondary Cluster: Protocol Error	888	2419
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	1065

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	2419
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	1063

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

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

Nature	Type	ID	Name		Page
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2545

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, 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		684	Incorrect Provision of Specified Functionality	1514

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

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 register 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
    end
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	2441

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2495
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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/b9ecdf6068445d76d6bee692d163fedef7a9d9b/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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1373
ParentOf		918	Server-Side Request Forgery (SSRF)	1829
ParentOf		1021	Improper Restriction of Rendered UI Layers or Frames	1869
PeerOf		611	Improper Restriction of XML External Entity Reference	1376
CanPrecede		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		1014	Identify Actors	2450

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	Gain Privileges or Assume Identity	
Access Control	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

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2009-0037

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	2418
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2491
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

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


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

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	1065

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

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	1065

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

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	Unexpected State	
Non-Repudiation	Hide Activities	

Scope	Impact	Likelihood
Access Control	<p>Bypass Protection Mechanism</p> <p><i>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).</i></p>	

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2545

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, 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		684	Incorrect Provision of Specified Functionality	1514
ParentOf		447	Unimplemented or Unsupported Feature in UI	1082
ParentOf		448	Obsolete Feature in UI	1083
ParentOf		449	The UI Performs the Wrong Action	1084

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

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	2439
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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, 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	G	446	UI Discrepancy for Security Feature	1081
ChildOf	G	671	Lack of Administrator Control over Security	1487

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

Nature	Type	ID	Name	Page
MemberOf		355	User Interface Security Issues	2341

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

Nature	Type	ID	Name		Page
MemberOf		995	SFP Secondary Cluster: Feature	888	2439
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

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, 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		446	UI Discrepancy for Security Feature	1081

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

Nature	Type	ID	Name	Page
MemberOf		355	User Interface Security Issues	2341

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

Nature	Type	ID	Name		Page
MemberOf		995	SFP Secondary Cluster: Feature	888	2439
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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, 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		446	UI Discrepancy for Security Feature	1081

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

Nature	Type	ID	Name	Page
MemberOf		355	User Interface Security Issues	2341

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

Nature	Type	ID	Name		Page
MemberOf		995	SFP Secondary Cluster: Feature	888	2439
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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, 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		357	Insufficient UI Warning of Dangerous Operations	887

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

Nature	Type	ID	Name		Page
MemberOf		995	SFP Secondary Cluster: Feature	888	2439
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

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, 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		221	Information Loss or Omission	563
ChildOf		684	Incorrect Provision of Specified Functionality	1514
ParentOf		1007	Insufficient Visual Distinction of Homoglyphs Presented to User	1866
ParentOf		1021	Improper Restriction of Rendered UI Layers or Frames	1869
PeerOf		346	Origin Validation Error	860

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Access Control	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

Reference	Description
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
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

Reference	Description
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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2439
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1379	ICS Operations (& Maintenance): Human factors in ICS environments	1358	2535
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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, 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		1188	Initialization of a Resource with an Insecure Default	1983

Applicable Platforms

Language : PHP (*Prevalence = Sometimes*)

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>An attacker could gain access to and modify sensitive data or system information.</i>	

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2424
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		1419	Incorrect Initialization of Resource	2292
CanAlsoBe		456	Missing Initialization of a Variable	1096

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

Nature	Type	ID	Name	Page
MemberOf		452	Initialization and Cleanup Errors	2348

Applicable Platforms

Language : PHP (*Prevalence = Sometimes*)

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>An attacker could gain access to and modify sensitive data or system information.</i>	

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)

```
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)

```
$debugEnabled = false;
if ($_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 Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf		808	2010 Top 25 - Weaknesses On the Cusp	800	2376
MemberOf		884	CWE Cross-section	884	2588
MemberOf		994	SFP Secondary Cluster: Tainted Input to Variable	888	2438
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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, 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		636	Not Failing Securely ('Failing Open')	1409
ChildOf		665	Improper Initialization	1465
ChildOf		705	Incorrect Control Flow Scoping	1550

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

Nature	Type	ID	Name	Page
MemberOf		452	Initialization and Cleanup Errors	2348

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Other	Alter Execution Logic	
	<i>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.</i>	

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

Nature	Type	ID	Name		Page
MemberOf		884	CWE Cross-section	884	2588
MemberOf		961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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, 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		909	Missing Initialization of Resource	1806

Nature	Type	ID	Name	Page
CanPrecede		89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	206
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	242
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	310
CanPrecede		457	Use of Uninitialized Variable	1102

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1465

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1465

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Quality Degradation	
	Varies by Context	
	<i>The uninitialized data may be invalid, causing logic errors within the program. In some cases, this could result in a security problem.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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);
```

Observed Examples

Reference	Description
CVE-2020-6078	Chain: The return value of a function returning a pointer is not checked for success (CWE-252) resulting in the later use of an uninitialized variable (CWE-456) and a null pointer dereference (CWE-476) https://www.cve.org/CVERecord?id=CVE-2020-6078
CVE-2009-2692	Chain: Use of an unimplemented network socket operation pointing to an uninitialized handler function (CWE-456) causes a crash because of a null pointer dereference (CWE-476). https://www.cve.org/CVERecord?id=CVE-2009-2692
CVE-2020-20739	A variable that has its value set in a conditional statement is sometimes used when the conditional fails, sometimes causing data leakage https://www.cve.org/CVERecord?id=CVE-2020-20739
CVE-2005-2978	Product uses uninitialized variables for size and index, leading to resultant buffer overflow. https://www.cve.org/CVERecord?id=CVE-2005-2978
CVE-2005-2109	Internal variable in PHP application is not initialized, allowing external modification. https://www.cve.org/CVERecord?id=CVE-2005-2109
CVE-2005-2193	Array variable not initialized in PHP application, leading to resultant SQL injection. https://www.cve.org/CVERecord?id=CVE-2005-2193

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2376
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2482
MemberOf	C	1180	SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1178	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Relationship

This weakness is a major factor in a number of resultant weaknesses, especially in web applications that allow global variable initialization (such as PHP) with libraries that can be directly requested.

Research Gap

It is highly likely that a large number of resultant weaknesses have missing initialization as a primary factor, but researcher reports generally do not provide this level of detail.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Initialization
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	ERR30-C	CWE More Abstract	Set errno to zero before calling a library function known to set errno, and check errno only after the function returns a value indicating failure
SEI CERT Perl Coding Standard	DCL04-PL	Exact	Always initialize local variables
SEI CERT Perl Coding Standard	DCL33-PL	Imprecise	Declare identifiers before using them
OMG ASCSM	ASCSM-CWE-456		
OMG ASCRM	ASCRM-CWE-456		

References

- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.
- [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-457: Use of Uninitialized Variable

Weakness ID : 457

Structure : Simple

Abstraction : Variant

Description

The code uses a variable that has not been initialized, leading to unpredictable or unintended results.



Extended Description

In some languages such as C and C++, stack variables are not initialized by default. They generally contain junk data with the contents of stack memory before the function was invoked. An attacker can sometimes control or read these contents. In other languages or conditions, a variable that is not explicitly initialized can be given a default value that has security implications, depending on the logic of the program. The presence of an uninitialized variable can sometimes indicate a typographic error in the code.

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		908	Use of Uninitialized Resource	1802
CanFollow		456	Missing Initialization of a Variable	1096

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1465

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1465

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Language : Perl (Prevalence = Often)

Language : PHP (Prevalence = Often)

Language : Not Language-Specific (Prevalence = Undetermined)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	Other	
Integrity	Initial variables usually contain junk, which can not be trusted for consistency. This can lead to denial of service conditions, or modify control flow in unexpected ways. In some cases, an attacker can "pre-initialize" the variable using previous actions, which might enable	
Other		

Scope	Impact	Likelihood
	code execution. This can cause a race condition if a lock variable check passes when it should not.	
Authorization Other	Other Strings that are not initialized are especially dangerous, since many functions expect a null at the end -- and only at the end -- of a string.	

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 = Attack Surface Reduction

Assign all variables to an initial value.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Most compilers will complain about the use of uninitialized variables if warnings are turned on.

Phase: Implementation

Phase: Operation

When using a language that does not require explicit declaration of variables, run or compile the software in a mode that reports undeclared or unknown variables. This may indicate the presence of a typographic error in the variable's name.

Phase: Requirements

The choice could be made to use a language that is not susceptible to these issues.

Phase: Architecture and Design

Mitigating technologies such as safe string libraries and container abstractions could be introduced.

Demonstrative Examples

Example 1:

This code prints a greeting using information stored in a POST request:

*Example Language: PHP**(Bad)*

```
if (isset($_POST['names'])) {
    $nameArray = $_POST['names'];
}
echo "Hello " . $nameArray['first'];
```

This code checks if the POST array 'names' is set before assigning it to the \$nameArray variable. However, if the array is not in the POST request, \$nameArray will remain uninitialized. This will cause an error when the array is accessed to print the greeting message, which could lead to further exploit.

Example 2:

The following switch statement is intended to set the values of the variables aN and bN before they are used:

*Example Language: C**(Bad)*

```
int aN, bN;
switch (ctl) {
    case -1:
        aN = 0;
        bN = 0;
        break;
    case 0:
        aN = i;
        bN = -i;
        break;
    case 1:
        aN = i + NEXT_SZ;
        bN = i - NEXT_SZ;
        break;
    default:
        aN = -1;
        aN = -1;
        break;
}
repaint(aN, bN);
```

In the default case of the switch statement, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value. Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

Example 3:

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);
```




Observed Examples

Reference	Description
CVE-2019-15900	Chain: sscanf() call is used to check if a username and group exists, but the return value of sscanf() call is not checked (CWE-252), causing an uninitialized variable to be checked (CWE-457), returning success to allow authorization bypass for executing a privileged (CWE-863). https://www.cve.org/CVERecord?id=CVE-2019-15900
CVE-2008-3688	Chain: A denial of service may be caused by an uninitialized variable (CWE-457) allowing an infinite loop (CWE-835) resulting from a connection to an unresponsive server. https://www.cve.org/CVERecord?id=CVE-2008-3688
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. https://www.cve.org/CVERecord?id=CVE-2008-0081
CVE-2007-4682	Crafted input triggers dereference of an uninitialized object pointer. https://www.cve.org/CVERecord?id=CVE-2007-4682
CVE-2007-3468	Crafted audio file triggers crash when an uninitialized variable is used. https://www.cve.org/CVERecord?id=CVE-2007-3468
CVE-2007-2728	Uninitialized random seed variable used. https://www.cve.org/CVERecord?id=CVE-2007-2728

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		Page
MemberOf		398	7PK - Code Quality	700	2344

Nature	Type	ID	Name	V	Page
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf		1180	SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1178	2486
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Uninitialized variable
7 Pernicious Kingdoms			Uninitialized Variable
Software Fault Patterns	SFP1		Glitch in computation
SEI CERT Perl Coding Standard	DCL33-PL	Imprecise	Declare identifiers before using them

References

- [REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.
- [REF-436]mercy. "Exploiting Uninitialized Data". 2006 January. < <http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip> >.
- [REF-437]Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008 March 1. < <https://msrc.microsoft.com/blog/2008/03/ms08-014-the-case-of-the-uninitialized-stack-variable-vulnerability/> >.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-459: Incomplete Cleanup

Weakness ID : 459

Structure : Simple

Abstraction : Base





Description

The product does not properly "clean up" and remove temporary or supporting resources after they have been used.

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		404	Improper Resource Shutdown or Release	987
ParentOf		226	Sensitive Information in Resource Not Removed Before Reuse	569
ParentOf		460	Improper Cleanup on Thrown Exception	1109
ParentOf		568	finalize() Method Without super.finalize()	1299

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	987

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

Nature	Type	ID	Name	Page
MemberOf		452	Initialization and Cleanup Errors	2348

Applicable Platforms

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

Alternate Terms

Insufficient Cleanup :

Common Consequences

Scope	Impact	Likelihood
Other	Other	
Confidentiality	Read Application Data	
Integrity	Modify Application Data	
	DoS: Resource Consumption (Other)	
	<i>It is possible to overflow the number of temporary files because directories typically have limits on the number of files allowed. This could create a denial of service problem.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Temporary files and other supporting resources should be deleted/released immediately after they are no longer needed.

Demonstrative Examples

Example 1:

Stream resources in a Java application should be released in a finally block, otherwise an exception thrown before the call to close() would result in an unreleased I/O resource. In the example below, the close() method is called in the try block (incorrect).

Example Language: Java

(Bad)

```
try {
    InputStream is = new FileInputStream(path);
    byte b[] = new byte[is.available()];
    is.read(b);
    is.close();
} catch (Throwable t) {
```

```
log.error("Something bad happened: " + t.getMessage());
}
```

Observed Examples









Reference	Description
CVE-2000-0552	World-readable temporary file not deleted after use. https://www.cve.org/CVERecord?id=CVE-2000-0552
CVE-2005-2293	Temporary file not deleted after use, leaking database usernames and passwords. https://www.cve.org/CVERecord?id=CVE-2005-2293
CVE-2002-0788	Interaction error creates a temporary file that can not be deleted due to strong permissions. https://www.cve.org/CVERecord?id=CVE-2002-0788
CVE-2002-2066	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). https://www.cve.org/CVERecord?id=CVE-2002-2066
CVE-2002-2067	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). https://www.cve.org/CVERecord?id=CVE-2002-2067
CVE-2002-2068	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). https://www.cve.org/CVERecord?id=CVE-2002-2068
CVE-2002-2069	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). https://www.cve.org/CVERecord?id=CVE-2002-2069
CVE-2002-2070	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). https://www.cve.org/CVERecord?id=CVE-2002-2070
CVE-2005-1744	Users not logged out when application is restarted after security-relevant changes were made. https://www.cve.org/CVERecord?id=CVE-2005-1744

Functional Areas

- File Processing

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2360
MemberOf		857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf		982	SFP Secondary Cluster: Failure to Release Resource	888	2431
MemberOf		1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504

Nature	Type	ID	Name	V	Page
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Relationship

CWE-459 is a child of CWE-404 because, while CWE-404 covers any type of improper shutdown or release of a resource, CWE-459 deals specifically with a multi-step shutdown process in which a crucial step for "proper" cleanup is omitted or impossible. That is, CWE-459 deals specifically with a cleanup or shutdown process that does not successfully remove all potentially sensitive data.

Relationship

Overlaps other categories such as permissions and containment. Concept needs further development. This could be primary (e.g. leading to infoleak) or resultant (e.g. resulting from unhandled error conditions or early termination).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Cleanup
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
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
The CERT Oracle Secure Coding Standard for Java (2011)	FIO00-J		Do not operate on files in shared directories
Software Fault Patterns	SFP14		Failure to release resource

CWE-460: Improper Cleanup on Thrown Exception

Weakness ID : 460

Structure : Simple

Abstraction : Base

Description

The product does not clean up its state or incorrectly cleans up its state when an exception is thrown, leading to unexpected state or control flow.



Extended Description

Often, when functions or loops become complicated, some level of resource cleanup is needed throughout execution. Exceptions can disturb the flow of the code and prevent the necessary cleanup from happening.

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		755	Improper Handling of Exceptional Conditions	1585
ChildOf		459	Incomplete Cleanup	1106

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

Nature	Type	ID	Name	Page
MemberOf		1012	Cross Cutting	2448

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
Other	Varies by Context <i>The code could be left in a bad state.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

If one breaks from a loop or function by throwing an exception, make sure that cleanup happens or that you should exit the program. Use throwing exceptions sparsely.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java

(Bad)

```
public class foo {
    public static final void main( String args[] ) {
        boolean returnValue;
        returnValue=doStuff();
    }
    public static final boolean doStuff( ) {
        boolean threadLock;
        boolean truthvalue=true;
        try {
            while(
                //check some condition
            ) {
                threadLock=true; //do some stuff to truthvalue
            }
        }
    }
}
```



```







        threadLock=false;
    }
}
catch (Exception e){
    System.err.println("You did something bad");
    if (something) return truthvalue;
}
return truthvalue;
}
}

```

In this case, a thread might be left locked accidentally.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf		880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf		961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf		1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper cleanup on thrown exception
The CERT Oracle Secure Coding Standard for Java (2011)	ERR03-J		Restore prior object state on method failure
The CERT Oracle Secure Coding Standard for Java (2011)	ERR05-J		Do not let checked exceptions escape from a finally block
SEI CERT Perl Coding Standard	EXP31-PL	Imprecise	Do not suppress or ignore exceptions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

CWE-462: Duplicate Key in Associative List (Alist)

Weakness ID : 462

Structure : Simple

Abstraction : Variant

Description

Duplicate keys in associative lists can lead to non-unique keys being mistaken for an error.

Extended Description

A duplicate key entry -- if the alist is designed properly -- could be used as a constant time replace function. However, duplicate key entries could be inserted by mistake. Because of this ambiguity, duplicate key entries in an association list are not recommended and should not be allowed.

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		694	Use of Multiple Resources with Duplicate Identifier	1531

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Use a hash table instead of an alist.

Phase: Architecture and Design

Use an alist which checks the uniqueness of hash keys with each entry before inserting the entry.

Demonstrative Examples

Example 1:

The following code adds data to a list and then attempts to sort the data.

Example Language: Python

(Bad)

```
alist = []
while (foo()): #now assume there is a string data with a key basename
    queue.append(basename,data)
    queue.sort()
```

Since basename is not necessarily unique, this may not sort how one would like it to be.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	2369
MemberOf		878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2399
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Duplicate key in associative list (alist)
CERT C Secure Coding	ENV02-C		Beware of multiple environment variables with the same effective name

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

CWE-463: Deletion of Data Structure Sentinel

Weakness ID : 463

Structure : Simple

Abstraction : Base

Description

The accidental deletion of a data-structure sentinel can cause serious programming logic problems.

Extended Description

Often times data-structure sentinels are used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list. It is dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the deletion or modification outside of some wrapper interface which provides safety.

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		707	Improper Neutralization	1554
PeerOf		464	Addition of Data Structure Sentinel	1115
PeerOf		170	Improper Null Termination	434

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2332

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Availability Other	Other <i>Generally this error will cause the data structure to not work properly.</i>	
Authorization Other	Other <i>If a control character, such as NULL is removed, one may cause resource access control problems.</i>	

Potential Mitigations

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Use OS-level preventative functionality. Not a complete solution.

Demonstrative Examples

Example 1:

This example creates a null terminated string and prints its contents.

Example Language: C


(Bad)

```
char *foo;
int counter;
foo=calloc(sizeof(char)*10);
for (counter=0;counter!=10;counter++) {
    foo[counter]='a';
    printf("%s\n",foo);
}
```

The string foo has space for 9 characters and a null terminator, but 10 characters are written to it. As a result, the string foo is not null terminated and calling printf() on it will have unpredictable and possibly dangerous results.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2553

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Deletion of data-structure sentinel

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

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

CWE-464: Addition of Data Structure Sentinel

Weakness ID : 464

Structure : Simple

Abstraction : Base

Description

The accidental addition of a data-structure sentinel can cause serious programming logic problems.




Extended Description

Data-structure sentinels are often used to mark the structure of data. A common example of this is the null character at the end of strings or a special sentinel to mark the end of a linked list. It is dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the addition or modification of sentinels.

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		138	Improper Neutralization of Special Elements	379
PeerOf		170	Improper Null Termination	434
PeerOf		463	Deletion of Data Structure Sentinel	1113

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2332

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>Generally this error will cause the data structure to not work properly by truncating the data.</i>	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Encapsulate the user from interacting with data sentinels. Validate user input to verify that sentinels are not present.

Phase: Implementation

Proper error checking can reduce the risk of inadvertently introducing sentinel values into data. For example, if a parsing function fails or encounters an error, it might return a value that is the same as the sentinel.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. This is not a complete solution.

Phase: Operation

Use OS-level preventative functionality. This is not a complete solution.

Demonstrative Examples

Example 1:

The following example assigns some character values to a list of characters and prints them each individually, and then as a string. The third character value is intended to be an integer taken from user input and converted to an int.

Example Language: C





(Bad)

```
char *foo;
foo=malloc(sizeof(char)*5);
foo[0]='a';
foo[1]='a';
foo[2]=atoi(getc(stdin));
foo[3]='c';
foo[4]='\0'
printf("%c %c %c %c %c\n",foo[0],foo[1],foo[2],foo[3],foo[4]);
printf("%s\n",foo);
```

The first print statement will print each character separated by a space. However, if a non-integer is read from stdin by `getc`, then `atoi` will not make a conversion and return 0. When `foo` is printed as a string, the 0 at character `foo[2]` will act as a NULL terminator and `foo[3]` will never be printed.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2366
MemberOf		875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2397
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2553

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Addition of data-structure sentinel
CERT C Secure Coding	STR03-C		Do not inadvertently truncate a null-terminated byte string

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	STR06-C		Do not assume that strtok() leaves the parse string unchanged

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

CWE-466: Return of Pointer Value Outside of Expected Range

Weakness ID : 466

Structure : Simple

Abstraction : Base


Description

A function can return a pointer to memory that is outside of the buffer that the pointer is expected to reference.

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		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		465	Pointer Issues	2349

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Applicable Platforms

Language : C (Prevalence = Undetermined)



Language : C++ (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Integrity	Modify Memory	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf		872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

Notes

Maintenance

This entry should have a chaining relationship with CWE-119 instead of a parent / child relationship, however the focus of this weakness does not map cleanly to any existing entries in CWE. A new parent is being considered which covers the more generic problem of incorrect return values. There is also an abstract relationship to weaknesses in which one component sends incorrect messages to another component; in this case, one routine is sending an incorrect value to another.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Illegal Pointer Value
Software Fault Patterns	SFP1		Glitch in computation

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.

CWE-467: Use of sizeof() on a Pointer Type

Weakness ID : 467

Structure : Simple

Abstraction : Variant

Description

The code calls sizeof() on a pointer type, which can be an incorrect calculation if the programmer intended to determine the size of the data that is being pointed to.

Extended Description

The use of sizeof() on a pointer can sometimes generate useful information. An obvious case is to find out the wordsize on a platform. More often than not, the appearance of sizeof(pointer) indicates a bug.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	131	Incorrect Calculation of Buffer Size	361

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Confidentiality	Read Memory	
<i>This error can often cause one to allocate a buffer that is much smaller than what is needed, leading to resultant weaknesses such as buffer overflows.</i>		

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Use expressions such as "sizeof(*pointer)" instead of "sizeof(pointer)", unless you intend to run sizeof() on a pointer type to gain some platform independence or if you are allocating a variable on the stack.

Demonstrative Examples

Example 1:

Care should be taken to ensure sizeof returns the size of the data structure itself, and not the size of the pointer to the data structure.

In this example, sizeof(foo) returns the size of the pointer.

Example Language: C

(Bad)

```
double *foo;
...
foo = (double *)malloc(sizeof(foo));
```

In this example, sizeof(*foo) returns the size of the data structure and not the size of the pointer.

Example Language: C

(Good)

```
double *foo;
...
foo = (double *)malloc(sizeof(*foo));
```

Example 2:

This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches

the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

Example Language:

(Bad)

```
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
    printf("Sizeof username = %d\n", sizeof(username));
    printf("Sizeof pass = %d\n", sizeof(pass));
    if (strcmp(username, inUser, sizeof(username))) {
        printf("Auth failure of username using sizeof\n");
        return(AUTH_FAIL);
    }
    /* Because of CWE-467, the sizeof returns 4 on many platforms and architectures. */
    if (! strcmp(pass, inPass, sizeof(pass))) {
        printf("Auth success of password using sizeof\n");
        return(AUTH_SUCCESS);
    }
    else {
        printf("Auth fail of password using sizeof\n");
        return(AUTH_FAIL);
    }
}
int main (int argc, char **argv)
{
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult != AUTH_SUCCESS) {
        ExitError("Authentication failed");
    }
    else {
        DoAuthenticatedTask(argv[1]);
    }
}
```

In AuthenticateUser(), because sizeof() is applied to a parameter with an array type, the sizeof() call might return 4 on many modern architectures. As a result, the strcmp() call only checks the first four characters of the input password, resulting in a partial comparison (CWE-187), leading to improper authentication (CWE-287).

Because of the partial comparison, any of these passwords would still cause authentication to succeed for the "admin" user:

Example Language:

(Attack)

```
pass5
passABCDEFGH
passWORD
```

Because only 4 characters are checked, this significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "adminXYZ" and "administrator" will succeed for the username.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2362
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	2365
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	2396
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	974	SFP Secondary Cluster: Incorrect Buffer Length Computation	888	2427
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1408	Comprehensive Categorization: Incorrect Calculation	1400	2555

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Use of sizeof() on a pointer type
CERT C Secure Coding	ARR01-C		Do not apply the sizeof operator to a pointer when taking the size of an array
CERT C Secure Coding	MEM35-C	CWE More Abstract	Allocate sufficient memory for an object
Software Fault Patterns	SFP10		Incorrect Buffer Length Computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

[REF-442]Robert Seacord. "EXP01-A. Do not take the sizeof a pointer to determine the size of a type". < <https://www.securecoding.cert.org/confluence/display/seccode/EXP01-A.+Do+not+take+the+sizeof+a+pointer+to+determine+the+size+of+a+type> >.

CWE-468: Incorrect Pointer Scaling

Weakness ID : 468

Structure : Simple

Abstraction : Base

Description

In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.

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	1507

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2349

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Memory Modify Memory	
<i>Incorrect pointer scaling will often result in buffer overflow conditions. Confidentiality can be compromised if the weakness is in the context of a buffer over-read or under-read.</i>		

Potential Mitigations

Phase: Architecture and Design

Use a platform with high-level memory abstractions.

Phase: Implementation

Always use array indexing instead of direct pointer manipulation.

Phase: Architecture and Design

Use technologies for preventing buffer overflows.

Demonstrative Examples

Example 1:

This example attempts to calculate the position of the second byte of a pointer.

Example Language: C

(Bad)

```
int *p = x;  
char * second_char = (char *) (p + 1);
```

In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2362
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2478
MemberOf	C	1408	Comprehensive Categorization: Incorrect Calculation	1400	2555

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Unintentional pointer scaling
CERT C Secure Coding	ARR39-C	Exact	Do not add or subtract a scaled integer to a pointer
CERT C Secure Coding	EXP08-C		Ensure pointer arithmetic is used correctly
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

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

CWE-469: Use of Pointer Subtraction to Determine Size

Weakness ID : 469

Structure : Simple

Abstraction : Base

Description

The product subtracts one pointer from another in order to determine size, but this calculation can be incorrect if the pointers do not exist in the same memory chunk.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1507

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2349

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

Medium

Common Consequences

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

Scope	Impact	Likelihood
	There is the potential for arbitrary code execution with privileges of the vulnerable program.	

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

Save an index variable. This is the recommended solution. Rather than subtract pointers from one another, use an index variable of the same size as the pointers in question. Use this variable to "walk" from one pointer to the other and calculate the difference. Always validate this number.

Demonstrative Examples

Example 1:

The following example contains the method size that is used to determine the number of nodes in a linked list. The method is passed a pointer to the head of the linked list.

Example Language: C

(Bad)

```
struct node {
    int data;
    struct node* next;
};
// Returns the number of nodes in a linked list from
// the given pointer to the head of the list.
int size(struct node* head) {
    struct node* current = head;
    struct node* tail;
    while (current != NULL) {
        tail = current;
        current = current->next;
    }
    return tail - head;
}
// other methods for manipulating the list
...
```

However, the method creates a pointer that points to the end of the list and uses pointer subtraction to determine the number of nodes in the list by subtracting the tail pointer from the head pointer. There no guarantee that the pointers exist in the same memory area, therefore using pointer subtraction in this way could return incorrect results and allow other unintended behavior. In this

example a counter should be used to determine the number of nodes in the list, as shown in the following code.

Example Language: C

(Good)

```
...
int size(struct node* head) {
    struct node* current = head;
    int count = 0;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	2365
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	2396
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	2426
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2478
MemberOf	C	1408	Comprehensive Categorization: Incorrect Calculation	1400	2555

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper pointer subtraction
CERT C Secure Coding	ARR36-C	Exact	Do not subtract or compare two pointers that do not refer to the same array
Software Fault Patterns	SFP7		Faulty Pointer Use

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')

Weakness ID : 470

Structure : Simple

Abstraction : Base

Description

The product uses external input with reflection to select which classes or code to use, but it does not sufficiently prevent the input from selecting improper classes or code.

Extended Description

If the product uses external inputs to determine which class to instantiate or which method to invoke, then an attacker could supply values to select unexpected classes or methods. If this occurs, then the attacker could create control flow paths that were not intended by the developer. These paths could bypass authentication or access control checks, or otherwise cause the product to behave in an unexpected manner. This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the product's classpath (CWE-427) or add new entries to the product's classpath (CWE-426). Under either of these conditions, the attacker can use reflection to introduce new, malicious behavior into 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1373
ChildOf		913	Improper Control of Dynamically-Managed Code Resources	1814

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

Nature	Type	ID	Name	Page
ChildOf		913	Improper Control of Dynamically-Managed Code Resources	1814

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Language : PHP (*Prevalence = Undetermined*)

Language : Interpreted (*Prevalence = Sometimes*)

Alternate Terms

Reflection Injection :

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Alter Execution Logic	
Availability	<i>The attacker might be able to execute code that is not directly accessible to the attacker. Alternately, the attacker could call unexpected code in the wrong place or the wrong time, possibly modifying critical system state.</i>	
Other		
Availability	DoS: Crash, Exit, or Restart	
Other	Other	
	<i>The attacker might be able to use reflection to call the wrong code, possibly with unexpected arguments that</i>	

Scope	Impact	Likelihood
	<i>violate the API (CWE-227). This could cause the product to exit or hang.</i>	
Confidentiality	Read Application Data <i>By causing the wrong code to be invoked, the attacker might be able to trigger a runtime error that leaks sensitive information in the error message, such as CWE-536.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Refactor your code to avoid using reflection.

Phase: Architecture and Design

Do not use user-controlled inputs to select and load classes or code.

Phase: Implementation

Apply strict input validation by using allowlists or indirect selection to ensure that the user is only selecting allowable classes or code.

Demonstrative Examples

Example 1:

A common reason that programmers use the reflection API is to implement their own command dispatcher. The following example shows a command dispatcher that does not use reflection:

Example Language: Java

(Good)

```
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
    ao = new AddCommand();
}
else if (ctl.equals("Modify")) {
    ao = new ModifyCommand();
}
else {
    throw new UnknownActionError();
}
ao.doAction(request);
```

A programmer might refactor this code to use reflection as follows:

Example Language: Java

(Bad)

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

The refactoring initially appears to offer a number of advantages. There are fewer lines of code, the if/else blocks have been entirely eliminated, and it is now possible to add new command types without modifying the command dispatcher. However, the refactoring allows an attacker to instantiate any object that implements the Worker interface. If the command dispatcher is still responsible for access control, then whenever programmers create a new class that implements the Worker interface, they must remember to modify the dispatcher's access control code. If they do not modify the access control code, then some Worker classes will not have any access control.

One way to address this access control problem is to make the Worker object responsible for performing the access control check. An example of the re-refactored code follows:

Example Language: Java

(Bad)

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
ao.doAction(request);
```








Although this is an improvement, it encourages a decentralized approach to access control, which makes it easier for programmers to make access control mistakes. This code also highlights another security problem with using reflection to build a command dispatcher. An attacker can invoke the default constructor for any kind of object. In fact, the attacker is not even constrained to objects that implement the Worker interface; the default constructor for any object in the system can be invoked. If the object does not implement the Worker interface, a ClassCastException will be thrown before the assignment to ao, but if the constructor performs operations that work in the attacker's favor, the damage will already have been done. Although this scenario is relatively benign in simple products, in larger products where complexity grows exponentially it is not unreasonable that an attacker could find a constructor to leverage as part of an attack.

Observed Examples

Reference	Description
CVE-2018-1000613	Cryptography API uses unsafe reflection when deserializing a private key https://www.cve.org/CVERecord?id=CVE-2018-1000613
CVE-2004-2331	Database system allows attackers to bypass sandbox restrictions by using the Reflection API. https://www.cve.org/CVERecord?id=CVE-2004-2331

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2390
MemberOf		884	CWE Cross-section	884	2588
MemberOf		991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf		1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Unsafe Reflection

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not use reflection to increase accessibility of classes, methods, or fields

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
138	Reflection Injection

References

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

CWE-471: Modification of Assumed-Immutable Data (MAID)

Weakness ID : 471

Structure : Simple

Abstraction : Base

Description

The product does not properly protect an assumed-immutable element from being modified by an attacker.











Extended Description

This occurs when a particular input is critical enough to the functioning of the application that it should not be modifiable at all, but it is. Certain resources are often assumed to be immutable when they are not, such as hidden form fields in web applications, cookies, and reverse DNS lookups.

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		664	Improper Control of a Resource Through its Lifetime	1463
ParentOf		291	Reliance on IP Address for Authentication	715
ParentOf		472	External Control of Assumed-Immutable Web Parameter	1131
ParentOf		473	PHP External Variable Modification	1134
ParentOf		607	Public Static Final Field References Mutable Object	1368
CanFollow		425	Direct Request ('Forced Browsing')	1032
CanFollow		602	Client-Side Enforcement of Server-Side Security	1359
CanFollow		621	Variable Extraction Error	1394
CanFollow		1282	Assumed-Immutable Data is Stored in Writable Memory	2139
CanFollow		1321	Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')	2216

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
	<i>Common data types that are attacked are environment variables, web application parameters, and HTTP headers.</i>	
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Phase: Implementation

When the data is stored or transmitted through untrusted sources that could modify the data, implement integrity checks to detect unauthorized modification, or store/transmit the data in a trusted location that is free from external influence.

Demonstrative Examples

Example 1:

In the code excerpt below, an array returned by a Java method is modified despite the fact that arrays are mutable.

Example Language: Java

(Bad)




```
String[] colors = car.getAllPossibleColors();
colors[0] = "Red";
```

Observed Examples

Reference	Description
CVE-2002-1757	Relies on \$PHP_SELF variable for authentication. https://www.cve.org/CVERecord?id=CVE-2002-1757
CVE-2005-1905	Gain privileges by modifying assumed-immutable code addresses that are accessed by a driver. https://www.cve.org/CVERecord?id=CVE-2005-1905

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Relationship

MAID issues can be primary to many other weaknesses, and they are a major factor in languages that provide easy access to internal program constructs, such as PHP's `register_globals` and similar features. However, MAID issues can also be resultant from weaknesses that modify internal state; for example, a program might validate some data and store it in memory, but a buffer overflow could overwrite that validated data, leading to a change in program logic.

Theoretical

There are many examples where the MUTABILITY property is a major factor in a vulnerability.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Modification of Assumed-Immutable Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
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

CWE-472: External Control of Assumed-Immutable Web Parameter

Weakness ID : 472

Structure : Simple

Abstraction : Base

Description

The web application does not sufficiently verify inputs that are assumed to be immutable but are actually externally controllable, such as hidden form fields.

Extended Description

If a web product does not properly protect assumed-immutable values from modification in hidden form fields, parameters, cookies, or URLs, this can lead to modification of critical data. Web applications often mistakenly make the assumption that data passed to the client in hidden fields or cookies is not susceptible to tampering. Improper validation of data that are user-controllable can lead to the application processing incorrect, and often malicious, input.

For example, custom cookies commonly store session data or persistent data across sessions. This kind of session data is normally involved in security related decisions on the server side, such as user authentication and access control. Thus, the cookies might contain sensitive data such as user credentials and privileges. This is a dangerous practice, as it can often lead to improper reliance on the value of the client-provided cookie by the server side application.

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		471	Modification of Assumed-Immutable Data (MAID)	1129
ChildOf		642	External Control of Critical State Data	1422
CanFollow		656	Reliance on Security Through Obscurity	1452

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

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

Nature	Type	ID	Name	Page
MemberOf		19	Data Processing Errors	2330

Applicable Platforms

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

Alternate Terms

Assumed-Immutable Parameter Tampering :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>Without appropriate protection mechanisms, the client can easily tamper with cookies and similar web data. Reliance on the cookies without detailed validation can lead to problems such as SQL injection. If you use cookie values for security related decisions on the server side, manipulating the cookies might lead to violations of security policies such as authentication bypassing, user impersonation and privilege escalation. In addition, storing sensitive data in the cookie without appropriate protection can also lead to disclosure of sensitive user data, especially data stored in persistent cookies.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

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

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:**

In this example, a web application uses the value of a hidden form field (accountID) without having done any input validation because it was assumed to be immutable.

*Example Language: Java**(Bad)*

```
String accountID = request.getParameter("accountID");
User user = getUserFromID(Long.parseLong(accountID));
```

Example 2:

Hidden fields should not be trusted as secure parameters.

An attacker can intercept and alter hidden fields in a post to the server as easily as user input fields. An attacker can simply parse the HTML for the substring:

*Example Language: HTML**(Bad)*

```
<input type="hidden"
```

or even just "hidden". Hidden field values displayed later in the session, such as on the following page, can open a site up to cross-site scripting attacks.

Observed Examples

Reference	Description
CVE-2002-0108	Forum product allows spoofed messages of other users via hidden form fields for name and e-mail address. https://www.cve.org/CVERecord?id=CVE-2002-0108
CVE-2000-0253	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0253
CVE-2000-0254	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0254
CVE-2000-0926	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0926
CVE-2000-0101	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0101
CVE-2000-0102	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0102
CVE-2000-0758	Allows admin access by modifying value of form field. https://www.cve.org/CVERecord?id=CVE-2000-0758
CVE-2002-1880	Read messages by modifying message ID parameter. https://www.cve.org/CVERecord?id=CVE-2002-1880
CVE-2000-1234	Send email to arbitrary users by modifying email parameter. https://www.cve.org/CVERecord?id=CVE-2000-1234
CVE-2005-1652	Authentication bypass by setting a parameter. https://www.cve.org/CVERecord?id=CVE-2005-1652
CVE-2005-1784	Product does not check authorization for configuration change admin script, leading to password theft via modified e-mail address field.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2005-1784
CVE-2005-2314	Logic error leads to password disclosure. https://www.cve.org/CVERecord?id=CVE-2005-2314
CVE-2005-1682	Modification of message number parameter allows attackers to read other people's messages. https://www.cve.org/CVERecord?id=CVE-2005-1682

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	2352
MemberOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2355
MemberOf		991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

Notes

Relationship

This is a primary weakness for many other weaknesses and functional consequences, including XSS, SQL injection, path disclosure, and file inclusion.

Theoretical

This is a technology-specific MAID problem.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Web Parameter Tampering
OWASP Top Ten 2007	A4		CWE More Specific Insecure Direct Object Reference
OWASP Top Ten 2004	A1		CWE More Specific Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
146	XML Schema Poisoning
226	Session Credential Falsification through Manipulation

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.

CWE-473: PHP External Variable Modification

Weakness ID : 473

Structure : Simple

Abstraction : Variant




Description

A PHP application does not properly protect against the modification of variables from external sources, such as query parameters or cookies. This can expose the application to numerous weaknesses that would not exist otherwise.

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		471	Modification of Assumed-Immutable Data (MAID)	1129
PeerOf		616	Incomplete Identification of Uploaded File Variables (PHP)	1385
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	242

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Applicable Platforms

Language : PHP (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Requirements

Phase: Implementation

Carefully identify which variables can be controlled or influenced by an external user, and consider adopting a naming convention to emphasize when externally modifiable variables are being used. An application should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking is performed when relying on input from outside a trust boundary. Do not allow your application to run with register_globals enabled. If you implement a register_globals emulator, be extremely careful of variable extraction, dynamic evaluation, and similar issues, since weaknesses in your emulation could allow external variable modification to take place even without register_globals.

Observed Examples

Reference	Description
CVE-2000-0860	File upload allows arbitrary file read by setting hidden form variables to match internal variable names. https://www.cve.org/CVERecord?id=CVE-2000-0860
CVE-2001-0854	Mistakenly trusts \$PHP_SELF variable to determine if include script was called by its parent. https://www.cve.org/CVERecord?id=CVE-2001-0854
CVE-2002-0764	PHP remote file inclusion by modified assumed-immutable variable. https://www.cve.org/CVERecord?id=CVE-2002-0764
CVE-2001-1025	Modify key variable when calling scripts that don't load a library that initializes it. https://www.cve.org/CVERecord?id=CVE-2001-1025

Reference	Description
CVE-2003-0754	Authentication bypass by modifying array used for authentication. https://www.cve.org/CVERecord?id=CVE-2003-0754

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

Notes

Relationship

This is a language-specific instance of Modification of Assumed-Immutable Data (MAID). This can be resultant from direct request (alternate path) issues. It can be primary to weaknesses such as PHP file inclusion, SQL injection, XSS, authentication bypass, and others.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PHP External Variable Modification

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
77	Manipulating User-Controlled Variables

CWE-474: Use of Function with Inconsistent Implementations

Weakness ID : 474

Structure : Simple

Abstraction : Base

Description

The code uses a function that has inconsistent implementations across operating systems and versions.

Extended Description

The use of inconsistent implementations can cause changes in behavior when the code is ported or built under a different environment than the programmer expects, which can lead to security problems in some cases.

The implementation of many functions varies by platform, and at times, even by different versions of the same platform. Implementation differences can include:



- Slight differences in the way parameters are interpreted leading to inconsistent results.
- Some implementations of the function carry significant security risks.
- The function might not be defined on all platforms.
- The function might change which return codes it can provide, or change the meaning of its return codes.

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		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591
ParentOf		589	Call to Non-ubiquitous API	1333

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

Nature	Type	ID	Name	Page
MemberOf		1228	API / Function Errors	2503

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C (Prevalence = Often)

Language : PHP (Prevalence = Often)

Language : Not Language-Specific (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations



Phase: Architecture and Design

Phase: Requirements

Do not accept inconsistent behavior from the API specifications when the deviant behavior increase the risk level.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		398	7PK - Code Quality	700	2344
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Inconsistent Implementations
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-475: Undefined Behavior for Input to API

Weakness ID : 475

Structure : Simple

Abstraction : Base


Description

The behavior of this function is undefined unless its control parameter is set to a specific value.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1307

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

Nature	Type	ID	Name	Page
MemberOf		1228	API / Function Errors	2503

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		398	7PK - Code Quality	700	2344
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Notes

Other

The Linux Standard Base Specification 2.0.1 for libc places constraints on the arguments to some internal functions [21]. If the constraints are not met, the behavior of the functions is not defined. It is unusual for this function to be called directly. It is almost always invoked through a macro defined in a system header file, and the macro ensures that the following constraints are met: The value 1 must be passed to the third parameter (the version number) of the following file system function: `__xmknod` The value 2 must be passed to the third parameter (the group argument) of the following wide character string functions: `__wcstod_internal` `__wcstof_internal` `__wcstol_internal` `__wcstold_internal` `__wcstoul_internal` The value 3 must be passed as the first parameter (the version number) of the following file system functions: `__xstat` `__lxstat` `__fxstat` `__xstat64` `__lxstat64` `__fxstat64`

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Undefined Behavior
Software Fault Patterns	SFP1		Glitch in computation

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-476: NULL Pointer Dereference

Weakness ID : 476**Structure :** Simple**Abstraction :** Base


Description

The product dereferences a pointer that it expects to be valid but is NULL.

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		754	Improper Check for Unusual or Exceptional Conditions	1577

Nature	Type	ID	Name	Page
ChildOf	I	710	Improper Adherence to Coding Standards	1558
CanFollow	B	252	Unchecked Return Value	613
CanFollow	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
CanFollow	V	789	Memory Allocation with Excessive Size Value	1683
CanFollow	B	1325	Improperly Controlled Sequential Memory Allocation	2222

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

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1577

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2349

Weakness Ordinalities

Resultant : NULL pointer dereferences are frequently resultant from rarely encountered error conditions and race conditions, since these are most likely to escape detection during the testing phases.

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Language : Go (Prevalence = Undetermined)

Alternate Terms

NPD : Common abbreviation for Null Pointer Dereference

null deref : Common abbreviation for Null Pointer Dereference

NPE : Common abbreviation for Null Pointer Exception

nil pointer dereference : used for access of nil in Go programs

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart <i>NULL pointer dereferences usually result in the failure of the process unless exception handling (on some platforms) is available and implemented. Even when exception handling is being used, it can still be very difficult to return the software to a safe state of operation.</i>	
Integrity Confidentiality	Execute Unauthorized Code or Commands Read Memory Modify Memory <i>In rare circumstances, when NULL is equivalent to the 0x0 memory address and privileged code can access it, then</i>	

Scope	Impact	Likelihood
	writing or reading memory is possible, which may lead to code execution.	

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.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

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

If all pointers that could have been modified are checked for NULL before use, nearly all NULL pointer dereferences can be prevented.

Phase: Requirements

Select a programming language that is not susceptible to these issues.

Phase: Implementation

Check the results of all functions that return a value and verify that the value is non-null before acting upon it.

Effectiveness = Moderate

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment. This solution does not handle the use of improperly initialized variables (CWE-665).

Phase: Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

Phase: Implementation

Explicitly initialize all variables and other data stores, either during declaration or just before the first usage.

Demonstrative Examples

Example 1:

While there are no complete fixes aside from conscientious programming, the following steps will go a long way to ensure that NULL pointer dereferences do not occur.

Example Language:

(Good)

```
if (pointer1 != NULL) {  
    /* make use of pointer1 */  
    /* ... */  
}
```

When working with a multithreaded or otherwise asynchronous environment, ensure that proper locking APIs are used to lock before the if statement; and unlock when it has finished.

Example 2:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C

(Bad)

```
void host_lookup(char *user_supplied_addr){  
    struct hostent *hp;  
    in_addr_t *addr;  
    char hostname[64];  
    in_addr_t inet_addr(const char *cp);  
    /*routine that ensures user_supplied_addr is in the right format for conversion */  
    validate_addr_form(user_supplied_addr);  
    addr = inet_addr(user_supplied_addr);  
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);  
    strcpy(hostname, hp->h_name);  
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to `gethostbyaddr()` will return NULL. Since the code does not check the return value from `gethostbyaddr` (CWE-252), a NULL pointer dereference (CWE-476) would then occur in the call to `strcpy()`.

Note that this code is also vulnerable to a buffer overflow (CWE-119).

Example 3:

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a NULL pointer exception when it attempts to call the `trim()` method.

Example Language: Java

(Bad)

```
String cmd = System.getProperty("cmd");  
cmd = cmd.trim();
```

Example 4:

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

Example Language: Java

(Bad)

```
...  
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");  
MyReceiver receiver = new MyReceiver();  
registerReceiver(receiver, filter);  
...  
public class UrlHandlerReceiver extends BroadcastReceiver {
```

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

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

Example 5:

Consider the following example of a typical client server exchange. The `HandleRequest` function is intended to perform a request and use a defer to close the connection whenever the function returns.

Example Language: Go

(Bad)

```
func HandleRequest(client http.Client, request *http.Request) (*http.Response, error) {
    response, err := client.Do(request)
    defer response.Body.Close()
    if err != nil {
        return nil, err
    }
    ...
}
```

If a user supplies a malformed request or violates the client policy, the `Do` method can return a nil response and a non-nil err.

This `HandleRequest` Function evaluates the close before checking the error. A deferred call's arguments are evaluated immediately, so the defer statement panics due to a nil response.

Observed Examples

Reference	Description
CVE-2005-3274	race condition causes a table to be corrupted if a timer activates while it is being modified, leading to resultant NULL dereference; also involves locking. https://www.cve.org/CVERecord?id=CVE-2005-3274
CVE-2002-1912	large number of packets leads to NULL dereference https://www.cve.org/CVERecord?id=CVE-2002-1912
CVE-2005-0772	packet with invalid error status value triggers NULL dereference https://www.cve.org/CVERecord?id=CVE-2005-0772
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-2020-29652	ssh component for Go allows clients to cause a denial of service (nil pointer dereference) against SSH servers. https://www.cve.org/CVERecord?id=CVE-2020-29652
CVE-2009-2692	Chain: Use of an unimplemented network socket operation pointing to an uninitialized handler function (CWE-456) causes a crash because of a null pointer dereference (CWE-476). https://www.cve.org/CVERecord?id=CVE-2009-2692
CVE-2009-3547	Chain: race condition (CWE-362) might allow resource to be released before operating on it, leading to NULL dereference (CWE-476) https://www.cve.org/CVERecord?id=CVE-2009-3547

Reference	Description
CVE-2009-3620	Chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference https://www.cve.org/CVERecord?id=CVE-2009-3620
CVE-2009-2698	Chain: IP and UDP layers each track the same value with different mechanisms that can get out of sync, possibly resulting in a NULL dereference https://www.cve.org/CVERecord?id=CVE-2009-2698
CVE-2009-2692	Chain: uninitialized function pointers can be dereferenced allowing code execution https://www.cve.org/CVERecord?id=CVE-2009-2692
CVE-2009-0949	Chain: improper initialization of memory can lead to NULL dereference https://www.cve.org/CVERecord?id=CVE-2009-0949
CVE-2008-3597	Chain: game server can access player data structures before initialization has happened leading to NULL dereference https://www.cve.org/CVERecord?id=CVE-2008-3597
CVE-2020-6078	Chain: The return value of a function returning a pointer is not checked for success (CWE-252) resulting in the later use of an uninitialized variable (CWE-456) and a null pointer dereference (CWE-476) https://www.cve.org/CVERecord?id=CVE-2020-6078
CVE-2008-0062	Chain: a message having an unknown message type may cause a reference to uninitialized memory resulting in a null pointer dereference (CWE-476) or dangling pointer (CWE-825), possibly crashing the system or causing heap corruption. https://www.cve.org/CVERecord?id=CVE-2008-0062
CVE-2008-5183	Chain: unchecked return value can lead to NULL dereference https://www.cve.org/CVERecord?id=CVE-2008-5183
CVE-2004-0079	SSL software allows remote attackers to cause a denial of service (crash) via a crafted SSL/TLS handshake that triggers a null dereference. https://www.cve.org/CVERecord?id=CVE-2004-0079
CVE-2004-0365	Network monitor allows remote attackers to cause a denial of service (crash) via a malformed RADIUS packet that triggers a null dereference. https://www.cve.org/CVERecord?id=CVE-2004-0365
CVE-2003-1013	Network monitor allows remote attackers to cause a denial of service (crash) via a malformed Q.931, which triggers a null dereference. https://www.cve.org/CVERecord?id=CVE-2003-1013
CVE-2003-1000	Chat client allows remote attackers to cause a denial of service (crash) via a passive DCC request with an invalid ID number, which causes a null dereference. https://www.cve.org/CVERecord?id=CVE-2003-1000
CVE-2004-0389	Server allows remote attackers to cause a denial of service (crash) via malformed requests that trigger a null dereference. https://www.cve.org/CVERecord?id=CVE-2004-0389
CVE-2004-0119	OS allows remote attackers to cause a denial of service (crash from null dereference) or execute arbitrary code via a crafted request during authentication protocol selection. https://www.cve.org/CVERecord?id=CVE-2004-0119
CVE-2004-0458	Game allows remote attackers to cause a denial of service (server crash) via a missing argument, which triggers a null pointer dereference. https://www.cve.org/CVERecord?id=CVE-2004-0458
CVE-2002-0401	Network monitor allows remote attackers to cause a denial of service (crash) or execute arbitrary code via malformed packets that cause a NULL pointer dereference. https://www.cve.org/CVERecord?id=CVE-2002-0401

Reference	Description
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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2344
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	2360
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2362
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2376
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	C	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	2395
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	2426
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	2466
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Null Dereference
CLASP			Null-pointer dereference
PLOVER			Null Dereference (Null Pointer Dereference)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	EXP34-C	Exact	Do not dereference null pointers
Software Fault Patterns	SFP7		Faulty Pointer 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> >.2024-11-17.

[REF-1031]"Null pointer / Null dereferencing". 2019 July 5. Wikipedia. < https://en.wikipedia.org/wiki/Null_pointer#Null_dereferencing >.

[REF-1032]"Null Reference Creation and Null Pointer Dereference". Apple. < <https://developer.apple.com/documentation/xcode/null-reference-creation-and-null-pointer-dereference> >.2023-04-07.

[REF-1033]"NULL Pointer Dereference [CWE-476]". 2012 September 1. ImmuniWeb. < <https://www.immuniweb.com/vulnerability/null-pointer-dereference.html> >.

CWE-477: Use of Obsolete Function

Weakness ID : 477

Structure : Simple

Abstraction : Base

Description

The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

Extended Description

As programming languages evolve, functions occasionally become obsolete due to:

- Advances in the language
- Improved understanding of how operations should be performed effectively and securely
- Changes in the conventions that govern certain operations

Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way.


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		710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1228	API / Function Errors	2503

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Debugger

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 Quality Analyzer Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Highly cost effective: Origin Analysis

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Refer to the documentation for the obsolete function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality.

Phase: Requirements

Consider seriously the security implications of using an obsolete function. Consider using alternate functions.

Demonstrative Examples

Example 1:

The following code uses the deprecated function `getpw()` to verify that a plaintext password matches a user's encrypted password. If the password is valid, the function sets `result` to 1; otherwise it is set to 0.

Example Language: C

(Bad)

```
...
getpw(uid, pwline);
for (i=0; i<3; i++){
    cryptpw=strtok(pwline, ":");
    pwline=0;
}
result = strcmp(crypt(plainpw,cryptpw), cryptpw) == 0;
...
```

Although the code often behaves correctly, using the `getpw()` function can be problematic from a security standpoint, because it can overflow the buffer passed to its second parameter. Because of this vulnerability, `getpw()` has been supplanted by `getpwuid()`, which performs the same lookup as `getpw()` but returns a pointer to a statically-allocated structure to mitigate the risk. Not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. Software security has not been a priority, or even a consideration, for very long. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby.

Example 2:

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a null pointer exception when it attempts to call the "Trim()" method.

Example Language: Java

(Bad)

```
String cmd = null;
...
cmd = Environment.GetEnvironmentVariable("cmd");
cmd = cmd.Trim();
```

Example 3:

The following code constructs a string object from an array of bytes and a value that specifies the top 8 bits of each 16-bit Unicode character.

Example Language: Java

(Bad)

```
...
String name = new String(nameBytes, highByte);
...
```

In this example, the constructor may not correctly convert bytes to characters depending upon which charset is used to encode the string represented by `nameBytes`. Due to the evolution of the charsets used to encode strings, this constructor was deprecated and replaced by a constructor that accepts as one of its parameters the name of the charset used to encode the bytes for conversion.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	2344
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1180	SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1178	2486
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Obsolete
Software Fault Patterns	SFP3		Use of an improper API
SEI CERT Perl Coding Standard	DCL30-PL	CWE More Specific	Do not import deprecated modules
SEI CERT Perl Coding Standard	EXP30-PL	CWE More Specific	Do not use deprecated or obsolete functions or modules

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-478: Missing Default Case in Multiple Condition Expression

Weakness ID : 478

Structure : Simple

Abstraction : Base

Description

The code does not have a default case in an expression with multiple conditions, such as a switch statement.

Extended Description

If a multiple-condition expression (such as a switch in C) omits the default case but does not consider or handle all possible values that could occur, then this might lead to complex logical errors and resultant weaknesses. Because of this, further decisions are made based on poor information, and cascading failure results. This cascading failure may result in any number of security issues, and constitutes a significant failure in the system.

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		1023	Incomplete Comparison with Missing Factors	1874

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Language : Python (Prevalence = Undetermined)

Language : JavaScript (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context Alter Execution Logic <i>Depending on the logical circumstances involved, any consequences may result: e.g., issues of confidentiality, authentication, authorization, availability, integrity, accountability, or non-repudiation.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Ensure that there are no cases unaccounted for when adjusting program flow or values based on the value of a given variable. In the case of switch style statements, the very simple act of creating a default case can, if done correctly, mitigate this situation. Often however, the default case is used simply to represent an assumed option, as opposed to working as a check for invalid input. This is poor practice and in some cases is as bad as omitting a default case entirely.

Demonstrative Examples

Example 1:

The following does not properly check the return code in the case where the security_check function returns a -1 value when an error occurs. If an attacker can supply data that will invoke an error, the attacker can bypass the security check:

Example Language: C

(Bad)

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
        printf("Security check failed!\n");
        exit(-1);
        //Break never reached because of exit()
        break;
    case PASSED:
        printf("Security check passed.\n");
        break;
}
// program execution continues...
...
```

Instead a default label should be used for unaccounted conditions:

Example Language: C

(Good)

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
        printf("Security check failed!\n");
        exit(-1);
        //Break never reached because of exit()
        break;
    case PASSED:
        printf("Security check passed.\n");
        break;
    default:
        printf("Unknown error (%d), exiting...\n",result);
        exit(-1);
}
```

This label is used because the assumption cannot be made that all possible cases are accounted for. A good practice is to reserve the default case for error handling.

Example 2:

In the following Java example the method `getInterestRate` retrieves the interest rate for the number of points for a mortgage. The number of points is provided within the input parameter and a switch statement will set the interest rate value to be returned based on the number of points.

Example Language: Java

(Bad)

```
public static final String INTEREST_RATE_AT_ZERO_POINTS = "5.00";
public static final String INTEREST_RATE_AT_ONE_POINTS = "4.75";
public static final String INTEREST_RATE_AT_TWO_POINTS = "4.50";
...
public BigDecimal getInterestRate(int points) {
    BigDecimal result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
    switch (points) {
        case 0:
            result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
            break;
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
            break;
```

```

    case 2:
        result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
        break;
    }
    return result;
}

```

However, this code assumes that the value of the points input parameter will always be 0, 1 or 2 and does not check for other incorrect values passed to the method. This can be easily accomplished by providing a default label in the switch statement that outputs an error message indicating an invalid value for the points input parameter and returning a null value.

Example Language: Java

(Good)

```

public static final String INTEREST_RATE_AT_ZERO_POINTS = "5.00";
public static final String INTEREST_RATE_AT_ONE_POINTS = "4.75";
public static final String INTEREST_RATE_AT_TWO_POINTS = "4.50";
...
public BigDecimal getInterestRate(int points) {
    BigDecimal result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
    switch (points) {
        case 0:
            result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
            break;
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
            break;
        case 2:
            result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
            break;
        default:
            System.err.println("Invalid value for points, must be 0, 1 or 2");
            System.err.println("Returning null value for interest rate");
            result = null;
    }
    return result;
}

```

Example 3:

In the following Python example the match-case statements (available in Python version 3.10 and later) perform actions based on the result of the `process_data()` function. The expected return is either 0 or 1. However, if an unexpected result (e.g., -1 or 2) is obtained then no actions will be taken potentially leading to an unexpected program state.

Example Language: Python

(Bad)

```

result = process_data(data)
match result:
    case 0:
        print("Properly handle zero case.")
    case 1:
        print("Properly handle one case.")
# program execution continues...

```

The recommended approach is to add a default case that captures any unexpected result conditions, regardless of how improbable these unexpected conditions might be, and properly handles them.

Example Language: Python

(Good)

```

result = process_data(data)
match result:
    case 0:
        print("Properly handle zero case.")

```



```

case 1:
    print("Properly handle one case.")
case _:
    print("Properly handle unexpected condition.")
# program execution continues...

```

Example 4:

In the following JavaScript example the switch-case statements (available in JavaScript version 1.2 and later) are used to process a given step based on the result of a calculation involving two inputs. The expected return is either 1, 2, or 3. However, if an unexpected result (e.g., 4) is obtained then no action will be taken potentially leading to an unexpected program state.

Example Language: JavaScript

(Bad)

```

let step = input1 + input2;
switch(step) {
  case 1:
    alert("Process step 1.");
    break;
  case 2:
    alert("Process step 2.");
    break;
  case 3:
    alert("Process step 3.");
    break;
}
// program execution continues...

```

The recommended approach is to add a default case that captures any unexpected result conditions and properly handles them.

Example Language: JavaScript

(Good)

```

let step = input1 + input2;
switch(step) {
  case 1:
    alert("Process step 1.");
    break;
  case 2:
    alert("Process step 2.");
    break;
  case 3:
    alert("Process step 3.");
    break;
  default:
    alert("Unexpected step encountered.");
}
// program execution continues...

```

Example 5:

The Finite State Machine (FSM) shown in the "bad" code snippet below assigns the output ("out") based on the value of state, which is determined based on the user provided input ("user_input").

Example Language: Verilog

(Bad)

```

module fsm_1(out, user_input, clk, rst_n);
input [2:0] user_input;
input clk, rst_n;
output reg [2:0] out;
reg [1:0] state;
always @ (posedge clk or negedge rst_n )
begin
    if (!rst_n)
        state = 3'h0;

```

```
else
case (user_input)
  3'h0:
  3'h1:
  3'h2:
  3'h3: state = 2'h3;
  3'h4: state = 2'h2;
  3'h5: state = 2'h1;
endcase
end
out <= {1'h1, state};
endmodule
```

The case statement does not include a default to handle the scenario when the user provides inputs of 3'h6 and 3'h7. Those inputs push the system to an undefined state and might cause a crash (denial of service) or any other unanticipated outcome.

Adding a default statement to handle undefined inputs mitigates this issue. This is shown in the "Good" code snippet below. The default statement is in bold.

Example Language: Verilog (Good)

```
case (user_input)
  3'h0:
  3'h1:
  3'h2:
  3'h3: state = 2'h3;
  3'h4: state = 2'h2;
  3'h5: state = 2'h1;
  default: state = 2'h0;
endcase
```

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	2588
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2544

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to account for default case in switch
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> >.2024-11-17.

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

CWE-479: Signal Handler Use of a Non-reentrant Function

Weakness ID : 479

Structure : Simple**Abstraction** : Variant

Description

The product defines a signal handler that calls a non-reentrant function.

Extended Description




Non-reentrant functions are functions that cannot safely be called, interrupted, and then recalled before the first call has finished without resulting in memory corruption. This can lead to an unexpected system state and unpredictable results with a variety of potential consequences depending on context, including denial of service and code execution.

Many functions are not reentrant, but some of them can result in the corruption of memory if they are used in a signal handler. The function call `syslog()` is an example of this. In order to perform its functionality, it allocates a small amount of memory as "scratch space." If `syslog()` is suspended by a signal call and the signal handler calls `syslog()`, the memory used by both of these functions enters an undefined, and possibly, exploitable state. Implementations of `malloc()` and `free()` manage metadata in global structures in order to track which memory is allocated versus which memory is available, but they are non-reentrant. Simultaneous calls to these functions can cause corruption of the metadata.

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		663	Use of a Non-reentrant Function in a Concurrent Context	1461
ChildOf		828	Signal Handler with Functionality that is not Asynchronous-Safe	1746
CanPrecede		123	Write-what-where Condition	329

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>It may be possible to execute arbitrary code through the use of a write-what-where condition.</i>	
Integrity	Modify Memory Modify Application Data <i>Signal race conditions often result in data corruption.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code)

without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Require languages or libraries that provide reentrant functionality, or otherwise make it easier to avoid this weakness.

Phase: Architecture and Design

Design signal handlers to only set flags rather than perform complex functionality.

Phase: Implementation

Ensure that non-reentrant functions are not found in signal handlers.

Phase: Implementation

Use sanity checks to reduce the timing window for exploitation of race conditions. This is only a partial solution, since many attacks might fail, but other attacks still might work within the narrower window, even accidentally.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

In this example, a signal handler uses syslog() to log a message:

Example Language: (Bad)

```
char *message;
void sh(int dummy) {
    syslog(LOG_NOTICE,"%s\n",message);
    sleep(10);
    exit(0);
}
int main(int argc,char* argv[]) {
    ...
    signal(SIGHUP,sh);
    signal(SIGTERM,sh);
    sleep(10);
    exit(0);
}
```

If the execution of the first call to the signal handler is suspended after invoking syslog(), and the signal handler is called a second time, the memory allocated by syslog() enters an undefined, and possibly, exploitable state.

Observed Examples







Reference	Description
CVE-2005-0893	signal handler calls function that ultimately uses malloc() https://www.cve.org/CVERecord?id=CVE-2005-0893
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

Affected Resources

- System Process

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		745	CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)	734	2370
MemberOf		847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	2384
MemberOf		879	CERT C++ Secure Coding Section 11 - Signals (SIG)	868	2400
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf		1166	SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)	1154	2481
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Unsafe function call from a signal handler
CERT C Secure Coding	SIG30-C	Exact	Call only asynchronous-safe functions within signal handlers
CERT C Secure Coding	SIG34-C		Do not call signal() from within interruptible signal handlers
The CERT Oracle Secure Coding Standard for Java (2011)	EXP01-J		Never dereference null pointers
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

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

CWE-480: Use of Incorrect Operator

Weakness ID : 480

Structure : Simple

Abstraction : Base

Description

The product accidentally uses the wrong operator, which changes the logic in security-relevant ways.

Extended Description

These types of errors are generally the result of a typo by the programmer.

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		670	Always-Incorrect Control Flow Implementation	1484
ParentOf		481	Assigning instead of Comparing	1161
ParentOf		482	Comparing instead of Assigning	1165
ParentOf		597	Use of Wrong Operator in String Comparison	1345

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

Nature	Type	ID	Name	Page
MemberOf		133	String Errors	2331
MemberOf		438	Behavioral Problems	2348
MemberOf		569	Expression Issues	2351

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Language : Perl (Prevalence = Sometimes)

Language : Not Language-Specific (Prevalence = Undetermined)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic <i>This weakness can cause unintended logic to be executed and other unexpected application behavior.</i>	

Detection Methods

Automated Static Analysis

This weakness can be found easily using static analysis. However in some cases an operator might appear to be incorrect, but is actually correct and reflects unusual logic within the program.

Manual Static Analysis

This weakness can be found easily using static analysis. However in some cases an operator might appear to be incorrect, but is actually correct and reflects unusual logic within the program.

Demonstrative Examples

Example 1:

The following C/C++ and C# examples attempt to validate an int input parameter against the integer value 100.

Example Language: C

(Bad)

```
int isValid(int value) {
    if (value=100) {
        printf("Value is valid\n");
        return(1);
    }
    printf("Value is not valid\n");
    return(0);
}
```

Example Language: C#

(Bad)

```
bool isValid(int value) {
    if (value=100) {
```

```
    Console.WriteLine("Value is valid.");  
    return true;  
}  
Console.WriteLine("Value is not valid.");  
return false;  
}
```

However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". The result of using the assignment operator instead of the comparison operator causes the int variable to be reassigned locally and the expression in the if statement will always evaluate to the value on the right hand side of the expression. This will result in the input value not being properly validated, which can cause unexpected results.

Example 2:

The following C/C++ example shows a simple implementation of a stack that includes methods for adding and removing integer values from the stack. The example uses pointers to add and remove integer values to the stack array variable.

Example Language: C

(Bad)

```
#define SIZE 50  
int *tos, *p1, stack[SIZE];  
void push(int i) {  
    p1++;  
    if(p1==(tos+SIZE)) {  
        // Print stack overflow error message and exit  
    }  
    *p1 == i;  
}  
int pop(void) {  
    if(p1==tos) {  
        // Print stack underflow error message and exit  
    }  
    p1--;  
    return *(p1+1);  
}  
int main(int argc, char *argv[]) {  
    // initialize tos and p1 to point to the top of stack  
    tos = stack;  
    p1 = stack;  
    // code to add and remove items from stack  
    ...  
    return 0;  
}
```

The push method includes an expression to assign the integer value to the location in the stack pointed to by the pointer variable.

However, this expression uses the comparison operator "==" rather than the assignment operator "=". The result of using the comparison operator instead of the assignment operator causes erroneous values to be entered into the stack and can cause unexpected results.

Example 3:

The example code below is taken from the CVA6 processor core of the HACK@DAC'21 buggy OpenPiton SoC. Debug access allows users to access internal hardware registers that are otherwise not exposed for user access or restricted access through access control protocols. Hence, requests to enter debug mode are checked and authorized only if the processor has sufficient privileges. In addition, debug accesses are also locked behind password checkers. Thus, the processor enters debug mode only when the privilege level requirement is met, and the correct debug password is provided.

The following code [REF-1377] illustrates an instance of a vulnerable implementation of debug mode. The core correctly checks if the debug requests have sufficient privileges and enables the

debug_mode_d and debug_mode_q signals. It also correctly checks for debug password and enables umode_i signal.

Example Language: Verilog (Bad)

```
module csr_regfile #(
...
// check that we actually want to enter debug depending on the privilege level we are currently in
unique case (priv_lvl_o)
  riscv::PRIV_LVL_M: begin
    debug_mode_d = dcsr_q.ebreakm;
...
  riscv::PRIV_LVL_U: begin
    debug_mode_d = dcsr_q.ebreaku;
...
assign priv_lvl_o = (debug_mode_q || umode_i) ? riscv::PRIV_LVL_M : priv_lvl_q;
...
debug_mode_q <= debug_mode_d;
...

```

However, it grants debug access and changes the privilege level, priv_lvl_o, even when one of the two checks is satisfied and the other is not. Because of this, debug access can be granted by simply requesting with sufficient privileges (i.e., debug_mode_q is enabled) and failing the password check (i.e., umode_i is disabled). This allows an attacker to bypass the debug password checking and gain debug access to the core, compromising the security of the processor.

A fix to this issue is to only change the privilege level of the processor when both checks are satisfied, i.e., the request has enough privileges (i.e., debug_mode_q is enabled) and the password checking is successful (i.e., umode_i is enabled) [REF-1378].

Example Language: Verilog (Good)

```
module csr_regfile #(
...
// check that we actually want to enter debug depending on the privilege level we are currently in
unique case (priv_lvl_o)
  riscv::PRIV_LVL_M: begin
    debug_mode_d = dcsr_q.ebreakm;
...
  riscv::PRIV_LVL_U: begin
    debug_mode_d = dcsr_q.ebreaku;
...
assign priv_lvl_o = (debug_mode_q && umode_i) ? riscv::PRIV_LVL_M : priv_lvl_q;
...
debug_mode_q <= debug_mode_d;
...

```

Observed Examples

Reference	Description
CVE-2022-3979	Chain: data visualization program written in PHP uses the "!=" operator instead of the type-strict "!===" operator (CWE-480) when validating hash values, potentially leading to an incorrect type conversion (CWE-704) https://www.cve.org/CVERecord?id=CVE-2022-3979
CVE-2021-3116	Chain: Python-based HTTP Proxy server uses the wrong boolean operators (CWE-480) causing an incorrect comparison (CWE-697) that identifies an authN failure if all three conditions are met instead of only one, allowing bypass of the proxy authentication (CWE-1390) https://www.cve.org/CVERecord?id=CVE-2021-3116

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf	C	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	2395
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using the wrong operator
CERT C Secure Coding	EXP45-C	CWE More Abstract	Do not perform assignments in selection statements
CERT C Secure Coding	EXP46-C	CWE More Abstract	Do not use a bitwise operator with a Boolean-like operand
Software Fault Patterns	SFP1		Glitch in Computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

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

[REF-1377]"csr_regfile.sv line 938". 2021. < https://github.com/HACK-EVENT/hackatdac19/blob/57e7b2109c1ea2451914878df2e6ca740c2dcf34/src/csr_regfile.sv#L938 >.2023-12-13.

[REF-1378]"Fix for csr_regfile.sv line 938". 2021. < https://github.com/HACK-EVENT/hackatdac19/blob/a7b61209e56c48eec585eeedea8413997ec71e4a/src/csr_regfile.sv#L938C31-L938C56 >.2023-12-13.

CWE-481: Assigning instead of Comparing

Weakness ID : 481

Structure : Simple

Abstraction : Variant

Description

The code uses an operator for assignment when the intention was to perform a comparison.

Extended Description



In many languages the compare statement is very close in appearance to the assignment statement and are often confused. This bug is generally the result of a typo and usually causes

obvious problems with program execution. If the comparison is in an if statement, the if statement will usually evaluate the value of the right-hand side of the predicate.

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		480	Use of Incorrect Operator	1157
CanPrecede		697	Incorrect Comparison	1538

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Other	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: Testing

Many IDEs and static analysis products will detect this problem.

Phase: Implementation

Place constants on the left. If one attempts to assign a constant with a variable, the compiler will produce an error.

Demonstrative Examples

Example 1:

The following C/C++ and C# examples attempt to validate an int input parameter against the integer value 100.

Example Language: C

(Bad)

```
int isValid(int value) {
    if (value=100) {
```

```

    printf("Value is valid\n");
    return(1);
}
printf("Value is not valid\n");
return(0);
}

```

Example Language: C#

(Bad)

```

bool isValid(int value) {
    if (value=100) {
        Console.WriteLine("Value is valid.");
        return true;
    }
    Console.WriteLine("Value is not valid.");
    return false;
}

```

However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". The result of using the assignment operator instead of the comparison operator causes the int variable to be reassigned locally and the expression in the if statement will always evaluate to the value on the right hand side of the expression. This will result in the input value not being properly validated, which can cause unexpected results.

Example 2:

In this example, we show how assigning instead of comparing can impact code when values are being passed by reference instead of by value. Consider a scenario in which a string is being processed from user input. Assume the string has already been formatted such that different user inputs are concatenated with the colon character. When the processString function is called, the test for the colon character will result in an insertion of the colon character instead, adding new input separators. Since the string was passed by reference, the data sentinels will be inserted in the original string (CWE-464), and further processing of the inputs will be altered, possibly malformed..

Example Language: C

(Bad)

```

void processString (char *str) {
    int i;
    for(i=0; i<strlen(str); i++) {
        if (isalnum(str[i])){
            processChar(str[i]);
        }
        else if (str[i] == ':') {
            movingToNewInput();
        }
    }
}

```

Example 3:

The following Java example attempts to perform some processing based on the boolean value of the input parameter. However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". As with the previous examples, the variable will be reassigned locally and the expression in the if statement will evaluate to true and unintended processing may occur.

Example Language: Java

(Bad)

```

public void checkValid(boolean isValid) {
    if (isValid = true) {
        System.out.println("Performing processing");
        doSomethingImportant();
    }
}

```

```
else {
    System.out.println("Not Valid, do not perform processing");
    return;
}
}
```

While most Java compilers will catch the use of an assignment operator when a comparison operator is required, for boolean variables in Java the use of the assignment operator within an expression is allowed. If possible, try to avoid using comparison operators on boolean variables in java. Instead, let the values of the variables stand for themselves, as in the following code.

Example Language: Java (Good)

```
public void checkValid(boolean isValid) {
    if (isValid) {
        System.out.println("Performing processing");
        doSomethingImportant();
    }
    else {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
}
```

Alternatively, to test for false, just use the boolean NOT operator.

Example Language: Java (Good)

```
public void checkValid(boolean isValid) {
    if (!isValid) {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
    System.out.println("Performing processing");
    doSomethingImportant();
}
```

Example 4:

The following example demonstrates the weakness.

Example Language: C (Bad)

```
void called(int foo){
    if (foo=1) printf("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

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	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Assigning instead of comparing
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	EXP45-C	CWE More Abstract	Do not perform assignments in selection statements

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

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

CWE-482: Comparing instead of Assigning

Weakness ID : 482

Structure : Simple

Abstraction : Variant

Description

The code uses an operator for comparison when the intention was to perform an assignment.

Extended Description

In many languages, the compare statement is very close in appearance to the assignment statement; they are often confused.

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		480	Use of Incorrect Operator	1157

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	Unexpected State <i>The assignment will not take place, which should cause obvious program execution problems.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code)

without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Testing

Many IDEs and static analysis products will detect this problem.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java

(Bad)

```
void called(int foo) {
    foo==1;
    if (foo==1) System.out.println("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

Example 2:

The following C/C++ example shows a simple implementation of a stack that includes methods for adding and removing integer values from the stack. The example uses pointers to add and remove integer values to the stack array variable.

Example Language: C

(Bad)





```
#define SIZE 50
int *tos, *p1, stack[SIZE];
void push(int i) {
    p1++;
    if(p1==(tos+SIZE)) {
        // Print stack overflow error message and exit
    }
    *p1 == i;
}
int pop(void) {
    if(p1==tos) {
        // Print stack underflow error message and exit
    }
    p1--;
    return *(p1+1);
}
int main(int argc, char *argv[]) {
    // initialize tos and p1 to point to the top of stack
    tos = stack;
    p1 = stack;
    // code to add and remove items from stack
    ...
    return 0;
}
```

The push method includes an expression to assign the integer value to the location in the stack pointed to by the pointer variable.

However, this expression uses the comparison operator "==" rather than the assignment operator "=". The result of using the comparison operator instead of the assignment operator causes erroneous values to be entered into the stack and can cause unexpected results.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf		883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf		886	SFP Primary Cluster: Unused entities	888	2403
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Comparing instead of assigning
Software Fault Patterns	SFP2		Unused Entities

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

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

CWE-483: Incorrect Block Delimitation

Weakness ID : 483

Structure : Simple

Abstraction : Base

Description

The code does not explicitly delimit a block that is intended to contain 2 or more statements, creating a logic error.

Extended Description

In some languages, braces (or other delimiters) are optional for blocks. When the delimiter is omitted, it is possible to insert a logic error in which a statement is thought to be in a block but is not. In some cases, the logic error can have security implications.


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		670	Always-Incorrect Control Flow Implementation	1484

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C (*Prevalence = Sometimes*)

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Alter Execution Logic <i>This is a general logic error which will often lead to obviously-incorrect behaviors that are quickly noticed and fixed. In lightly tested or untested code, this error may be introduced it into a production environment and provide additional attack vectors by creating a control flow path leading to an unexpected state in the application. The consequences will depend on the types of behaviors that are being incorrectly executed.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Always use explicit block delimitation and use static-analysis technologies to enforce this practice.

Demonstrative Examples

Example 1:

In this example, the programmer has indented the statements to call Do_X() and Do_Y(), as if the intention is that these functions are only called when the condition is true. However, because there are no braces to signify the block, Do_Y() will always be executed, even if the condition is false.

Example Language: C

(Bad)

```
if (condition==true)
    Do_X();
    Do_Y();
```

This might not be what the programmer intended. When the condition is critical for security, such as in making a security decision or detecting a critical error, this may produce a vulnerability.

Example 2:

In this example, the programmer has indented the Do_Y() statement as if the intention is that the function should be associated with the preceding conditional and should only be called when the condition is true. However, because Do_X() was called on the same line as the conditional and there are no braces to signify the block, Do_Y() will always be executed, even if the condition is false.

Example Language: C(Bad)

```
if (condition==true) Do_X();
    Do_Y();
```

This might not be what the programmer intended. When the condition is critical for security, such as in making a security decision or detecting a critical error, this may produce a vulnerability.

Observed Examples

Reference	Description
CVE-2014-1266	incorrect indentation of "goto" statement makes it more difficult to detect an incorrect goto (Apple's "goto fail") 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	2588
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Incorrect block delimitation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < <https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf> >.2024-11-17.

CWE-484: Omitted Break Statement in Switch

Weakness ID : 484
Structure : Simple
Abstraction : Base

Description

The product omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.


Extended Description

This can lead to critical code executing in situations where it should not.


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		670	Always-Incorrect Control Flow Implementation	1484
ChildOf		710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Language : PHP (Prevalence = Undetermined)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic <i>This weakness can cause unintended logic to be executed and other unexpected application behavior.</i>	

Detection Methods

White Box

Omission of a break statement might be intentional, in order to support fallthrough. Automated detection methods might therefore be erroneous. Semantic understanding of expected product behavior is required to interpret whether the code is correct.

Black Box

Since this weakness is associated with a code construct, it would be indistinguishable from other errors that produce the same behavior.

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

Omitting a break statement so that one may fall through is often indistinguishable from an error, and therefore should be avoided. If you need to use fall-through capabilities, make sure that you have clearly documented this within the switch statement, and ensure that you have examined all the logical possibilities.

Phase: Implementation

The functionality of omitting a break statement could be clarified with an if statement. This method is much safer.

Demonstrative Examples

Example 1:

In both of these examples, a message is printed based on the month passed into the function:

Example Language: Java

(Bad)

```
public void printMessage(int month){
    switch (month) {
        case 1: print("January");
        case 2: print("February");
        case 3: print("March");
        case 4: print("April");
        case 5: print("May");
        case 6: print("June");
        case 7: print("July");
        case 8: print("August");
        case 9: print("September");
        case 10: print("October");
        case 11: print("November");
        case 12: print("December");
    }
    println(" is a great month");
}
```

Example Language: C

(Bad)

```
void printMessage(int month){
    switch (month) {
        case 1: printf("January");
        case 2: printf("February");
        case 3: printf("March");
        case 4: printf("April");
        case 5: printf("May");
        case 6: printf("June");
        case 7: printf("July");
        case 8: printf("August");
        case 9: printf("September");
        case 10: printf("October");
        case 11: printf("November");
        case 12: printf("December");
    }
    printf(" is a great month");
}
```

Both examples do not use a break statement after each case, which leads to unintended fall-through behavior. For example, calling "printMessage(10)" will result in the text "OctoberNovemberDecember is a great month" being printed.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Omitted break statement
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> >.2024-11-17.

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

CWE-486: Comparison of Classes by Name

Weakness ID : 486

Structure : Simple

Abstraction : Variant

Description

The product compares classes by name, which can cause it to use the wrong class when multiple classes can have the same name.

Extended Description

If the decision to trust the methods and data of an object is based on the name of a class, it is possible for malicious users to send objects of the same name as trusted classes and thereby gain the trust afforded to known classes and types.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	1025	Comparison Using Wrong Factors	1878
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	949

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences