

```

<session-config>
  <session-timeout>-1</session-timeout>
</session-config>
</web-app>

```

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		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2335
MemberOf		930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2389
MemberOf		951	SFP Secondary Cluster: Insecure Authentication Policy	888	2396
MemberOf		1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2436
MemberOf		1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Other

The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept a session ID, possibly via a network sniffer or Cross-site Scripting attack. Although short session expiration times do not help if a stolen token is immediately used, they will protect against ongoing replaying of the session ID. In another scenario, a user might access a web site from a shared computer (such as at a library, Internet cafe, or open work environment). Insufficient Session Expiration could allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	47		Insufficient Session Expiration

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Weakness ID : 614

Structure : Simple

Abstraction : Variant

Description

The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over an HTTP session.

Relationships

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

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

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

Applicable Platforms

Technology : Web Based (*Prevalence = Undetermined*)

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

Potential Mitigations

Phase: Implementation

Always set the secure attribute when the cookie should sent via HTTPS only.

Demonstrative Examples

Example 1:

The snippet of code below, taken from a servlet doPost() method, sets an accountID cookie (sensitive) without calling setSecure(true).

Example Language: Java

(Bad)




```
Cookie c = new Cookie(ACCOUNT_ID, acctID);  
response.addCookie(c);
```

Observed Examples

Reference	Description
CVE-2004-0462	A product does not set the Secure attribute for sensitive cookies in HTTPS sessions, which could cause the user agent to send those cookies in plaintext over an HTTP session with the product. https://www.cve.org/CVERecord?id=CVE-2004-0462
CVE-2008-3663	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie. https://www.cve.org/CVERecord?id=CVE-2008-3663
CVE-2008-3662	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie. https://www.cve.org/CVERecord?id=CVE-2008-3662
CVE-2008-0128	A product does not set the secure flag for a cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie. https://www.cve.org/CVERecord?id=CVE-2008-0128

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		966	SFP Secondary Cluster: Other Exposures	888	2403
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2527

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
102	Session Sidejacking

CWE-615: Inclusion of Sensitive Information in Source Code Comments

Weakness ID : 615

Structure : Simple

Abstraction : Variant

Description

While adding general comments is very useful, some programmers tend to leave important data, such as: filenames related to the web application, old links or links which were not meant to be browsed by users, old code fragments, etc.

Extended Description

An attacker who finds these comments can map the application's structure and files, expose hidden parts of the site, and study the fragments of code to reverse engineer the application, which may help develop further attacks against the site.

Relationships

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		540	Inclusion of Sensitive Information in Source Code	1251
PeerOf		546	Suspicious Comment	1258

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

Potential Mitigations

Phase: Distribution

Remove comments which have sensitive information about the design/implementation of the application. Some of the comments may be exposed to the user and affect the security posture of the application.

Demonstrative Examples

Example 1:

The following comment, embedded in a JSP, will be displayed in the resulting HTML output.

Example Language: JSP

(Bad)

```
<!-- FIXME: calling this with more than 30 args kills the JDBC server -->
```

Observed Examples

Reference	Description
CVE-2007-6197	Version numbers and internal hostnames leaked in HTML comments. https://www.cve.org/CVERecord?id=CVE-2007-6197
CVE-2007-4072	CMS places full pathname of server in HTML comment. https://www.cve.org/CVERecord?id=CVE-2007-4072
CVE-2009-2431	blog software leaks real username in HTML comment. https://www.cve.org/CVERecord?id=CVE-2009-2431

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	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

CWE-616: Incomplete Identification of Uploaded File Variables (PHP)

Weakness ID : 616

Structure : Simple

Abstraction : Variant

Description

The PHP application uses an old method for processing uploaded files by referencing the four global variables that are set for each file (e.g. \$varname, \$varname_size, \$varname_name, \$varname_type). These variables could be overwritten by attackers, causing the application to process unauthorized files.

Extended Description



These global variables could be overwritten by POST requests, cookies, or other methods of populating or overwriting these variables. This could be used to read or process arbitrary files by providing values such as "/etc/passwd".

Relationships

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	851
PeerOf		473	PHP External Variable Modification	1127

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP (*Prevalence = Undetermined*)

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

Use PHP 4 or later.

Phase: Architecture and Design

If you must support older PHP versions, write your own version of `is_uploaded_file()` and run it against `$HTTP_POST_FILES['userfile']`)

Phase: Implementation

For later PHP versions, reference uploaded files using the `$HTTP_POST_FILES` or `$_FILES` variables, and use `is_uploaded_file()` or `move_uploaded_file()` to ensure that you are dealing with an uploaded file.

Demonstrative Examples

Example 1:

As of 2006, the "four globals" method is probably in sharp decline, but older PHP applications could have this issue.

In the "four globals" method, PHP sets the following 4 global variables (where "varname" is application-dependent):

Example Language: PHP

(Bad)

```
$varname = name of the temporary file on local machine
$varname_size = size of file
$varname_name = original name of file provided by client
$varname_type = MIME type of the file
```

Example 2:

"The global `$_FILES` exists as of PHP 4.1.0 (Use `$HTTP_POST_FILES` instead if using an earlier version). These arrays will contain all the uploaded file information."

Example Language: PHP

(Bad)

```
$_FILES['userfile']['name'] - original filename from client
$_FILES['userfile']['tmp_name'] - the temp filename of the file on the server
```

**** note:** 'userfile' is the field name from the web form; this can vary.

Observed Examples

Reference	Description
CVE-2002-1460	Forum does not properly verify whether a file was uploaded or if the associated variables were set by POST, allowing remote attackers to read arbitrary files. https://www.cve.org/CVERecord?id=CVE-2002-1460
CVE-2002-1759	Product doesn't check if the variables for an upload were set by uploading the file, or other methods such as \$_POST. https://www.cve.org/CVERecord?id=CVE-2002-1759
CVE-2002-1710	Product does not distinguish uploaded file from other files. https://www.cve.org/CVERecord?id=CVE-2002-1710

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	994	SFP Secondary Cluster: Tainted Input to Variable	888	2417
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Identification of Uploaded File Variables (PHP)
Software Fault Patterns	SFP25		Tainted input to variable

References

[REF-502]Shaun Clowes. "A Study in Scarlet - section 5, "File Upload"".

CWE-617: Reachable Assertion

Weakness ID : 617

Structure : Simple

Abstraction : Base

Description

The product contains an assert() or similar statement that can be triggered by an attacker, which leads to an application exit or other behavior that is more severe than necessary.

Extended Description

While assertion is good for catching logic errors and reducing the chances of reaching more serious vulnerability conditions, it can still lead to a denial of service.

For example, if a server handles multiple simultaneous connections, and an assert() occurs in one single connection that causes all other connections to be dropped, this is a reachable assertion that leads to a denial of service.

Relationships

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	1475
CanFollow		193	Off-by-one Error	486

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

Nature	Type	ID	Name	Page
ChildOf		670	Always-Incorrect Control Flow Implementation	1475

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2322

Weakness Ordinalities

Resultant :

Alternate Terms

assertion failure :

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart <i>An attacker that can trigger an assert statement can still lead to a denial of service if the relevant code can be triggered by an attacker, and if the scope of the assert() extends beyond the attacker's own session.</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

Make sensitive open/close operation non reachable by directly user-controlled data (e.g. open/close resources)

Phase: Implementation

Strategy = Input Validation

Perform input validation on user data.

Demonstrative Examples

Example 1:

In the excerpt below, an AssertionError (an unchecked exception) is thrown if the user hasn't entered an email address in an HTML form.

Example Language: Java

(Bad)

```
String email = request.getParameter("email_address");
```









```
assert email != null;
```

Observed Examples

Reference	Description
CVE-2023-49286	Chain: function in web caching proxy does not correctly check a return value (CWE-253) leading to a reachable assertion (CWE-617) https://www.cve.org/CVERecord?id=CVE-2023-49286
CVE-2006-6767	FTP server allows remote attackers to cause a denial of service (daemon abort) via crafted commands which trigger an assertion failure. https://www.cve.org/CVERecord?id=CVE-2006-6767
CVE-2006-6811	Chat client allows remote attackers to cause a denial of service (crash) via a long message string when connecting to a server, which causes an assertion failure. https://www.cve.org/CVERecord?id=CVE-2006-6811
CVE-2006-5779	Product allows remote attackers to cause a denial of service (daemon crash) via LDAP BIND requests with long authcid names, which triggers an assertion failure. https://www.cve.org/CVERecord?id=CVE-2006-5779
CVE-2006-4095	Product allows remote attackers to cause a denial of service (crash) via certain queries, which cause an assertion failure. https://www.cve.org/CVERecord?id=CVE-2006-4095
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion. https://www.cve.org/CVERecord?id=CVE-2006-4574
CVE-2004-0270	Anti-virus product has assert error when line length is non-numeric. https://www.cve.org/CVERecord?id=CVE-2004-0270

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		850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2364
MemberOf		884	CWE Cross-section	884	2567
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf		1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2447
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET01-J		Never use assertions to validate method arguments
Software Fault Patterns	SFP3		Use of an improper API

CWE-618: Exposed Unsafe ActiveX Method

Weakness ID : 618

Structure : Simple

Abstraction : Variant

Description

An ActiveX control is intended for use in a web browser, but it exposes dangerous methods that perform actions that are outside of the browser's security model (e.g. the zone or domain).

Extended Description

ActiveX controls can exercise far greater control over the operating system than typical Java or javascript. Exposed methods can be subject to various vulnerabilities, depending on the implemented behaviors of those methods, and whether input validation is performed on the provided arguments. If there is no integrity checking or origin validation, this method could be invoked by attackers.

Relationships

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		749	Exposed Dangerous Method or Function	1564
PeerOf		623	Unsafe ActiveX Control Marked Safe For Scripting	1389

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

Nature	Type	ID	Name	Page
MemberOf		275	Permission Issues	2317

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

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 you must expose a method, make sure to perform input validation on all arguments, and protect against all possible vulnerabilities.

Phase: Architecture and Design

Use code signing, although this does not protect against any weaknesses that are already in the control.

Phase: Architecture and Design

Phase: System Configuration



Where possible, avoid marking the control as safe for scripting.

Observed Examples

Reference	Description
CVE-2007-1120	download a file to arbitrary folders. https://www.cve.org/CVERecord?id=CVE-2007-1120
CVE-2006-6838	control downloads and executes a url in a parameter https://www.cve.org/CVERecord?id=CVE-2006-6838
CVE-2007-0321	resultant buffer overflow https://www.cve.org/CVERecord?id=CVE-2007-0321

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	2407
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

References

[REF-503]Microsoft. "Developing Secure ActiveX Controls". 2005 April 3. < [https://learn.microsoft.com/en-us/previous-versions/ms533046\(v=vs.85\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/ms533046(v=vs.85)?redirectedfrom=MSDN) >.2023-04-07.

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

CWE-619: Dangling Database Cursor ('Cursor Injection')

Weakness ID : 619

Structure : Simple

Abstraction : Base

Description

If a database cursor is not closed properly, then it could become accessible to other users while retaining the same privileges that were originally assigned, leaving the cursor "dangling."

Extended Description

For example, an improper dangling cursor could arise from unhandled exceptions. The impact of the issue depends on the cursor's role, but SQL injection attacks are commonly possible.

Relationships

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')	976
CanFollow		404	Improper Resource Shutdown or Release	980

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324

Weakness Ordinalities

Primary : This could be primary when the programmer never attempts to close the cursor when finished with it.

Resultant :

Applicable Platforms

Language : SQL (*Prevalence = Undetermined*)

Background Details

A cursor is a feature in Oracle PL/SQL and other languages that provides a handle for executing and accessing the results of SQL queries.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	




Potential Mitigations

Phase: Implementation

Close cursors immediately after access to them is complete. Ensure that you close cursors if exceptions occur.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-505]David Litchfield. "The Oracle Hacker's Handbook".

[REF-506]David Litchfield. "Cursor Injection". < <http://www.davidlitchfield.com/cursor-injection.pdf> >.2023-04-07.

CWE-620: Unverified Password Change

Weakness ID : 620

Structure : Simple

Abstraction : Base

Description

When setting a new password for a user, the product does not require knowledge of the original password, or using another form of authentication.

Extended Description

This could be used by an attacker to change passwords for another user, thus gaining the privileges associated with that user.

Relationships

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		1390	Weak Authentication	2267

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

Nature	Type	ID	Name	Page
MemberOf		1010	Authenticate Actors	2424

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

Nature	Type	ID	Name	Page
MemberOf		255	Credentials Management Errors	2315

Weakness Ordinalities

Primary :

Resultant :

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

When prompting for a password change, force the user to provide the original password in addition to the new password.

Phase: Architecture and Design

Do not use "forgotten password" functionality. But if you must, ensure that you are only providing information to the actual user, e.g. by using an email address or challenge question that the legitimate user already provided in the past; do not allow the current user to change this identity information until the correct password has been provided.

Demonstrative Examples

Example 1:

This code changes a user's password.

Example Language: PHP

(Bad)

```
$user = $_GET['user'];  
$pass = $_GET['pass'];  
$checkpass = $_GET['checkpass'];  
if ($pass == $checkpass) {  
    SetUserPassword($user, $pass);  
}
```

While the code confirms that the requesting user typed the same new password twice, it does not confirm that the user requesting the password change is the same user whose password will be changed. An attacker can request a change of another user's password and gain control of the victim's account.

Observed Examples

Reference	Description
CVE-2007-0681	Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions. https://www.cve.org/CVERecord?id=CVE-2007-0681
CVE-2000-0944	Web application password change utility doesn't check the original password. https://www.cve.org/CVERecord?id=CVE-2000-0944

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		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2335
MemberOf		930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2389
MemberOf		952	SFP Secondary Cluster: Missing Authentication	888	2396
MemberOf		1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2436
MemberOf		1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
Software Fault Patterns	SFP31		Missing authentication

References

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

CWE-621: Variable Extraction Error

Weakness ID : 621

Structure : Simple

Abstraction : Variant

Description

The product uses external input to determine the names of variables into which information is extracted, without verifying that the names of the specified variables are valid. This could cause the program to overwrite unintended variables.

Extended Description

For example, in PHP, extraction can be used to provide functionality similar to `register_globals`, a dangerous functionality that is frequently disabled in production systems. Calling `extract()` or



import_request_variables() without the proper arguments could allow arbitrary global variables to be overwritten, including superglobals.

Similar functionality is possible in other interpreted languages, including custom languages.

Relationships

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		914	Improper Control of Dynamically-Identified Variables	1807
CanPrecede		471	Modification of Assumed-Immutable Data (MAID)	1121

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP (*Prevalence = Undetermined*)

Alternate Terms

Variable overwrite :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>An attacker could modify sensitive data or program variables.</i>	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Use allowlists of variable names that can be extracted.

Phase: Implementation

Consider refactoring your code to avoid extraction routines altogether.

Phase: Implementation

In PHP, call extract() with options such as EXTR_SKIP and EXTR_PREFIX_ALL; call import_request_variables() with a prefix argument. Note that these capabilities are not present in all PHP versions.

Demonstrative Examples

Example 1:

This code uses the credentials sent in a POST request to login a user.

Example Language: PHP

(Bad)

```
//Log user in, and set $isAdmin to true if user is an administrator
function login($user,$pass){
    $query = buildQuery($user,$pass);
    mysql_query($query);
    if(getUserRole($user) == "Admin"){
        $isAdmin = true;
    }
}
```

```

    }
}
$isAdmin = false;
extract($_POST);
login(mysql_real_escape_string($user),mysql_real_escape_string($pass));

```

The call to `extract()` will overwrite the existing values of any variables defined previously, in this case `$isAdmin`. An attacker can send a POST request with an unexpected third value "isAdmin" equal to "true", thus gaining Admin privileges.

Observed Examples

Reference	Description
CVE-2006-7135	extract issue enables file inclusion https://www.cve.org/CVERecord?id=CVE-2006-7135
CVE-2006-7079	Chain: PHP app uses extract for register_globals compatibility layer (CWE-621), enabling path traversal (CWE-22) https://www.cve.org/CVERecord?id=CVE-2006-7079
CVE-2007-0649	extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect. https://www.cve.org/CVERecord?id=CVE-2007-0649
CVE-2006-6661	extract() enables static code injection https://www.cve.org/CVERecord?id=CVE-2006-6661
CVE-2006-2828	import_request_variables() buried in include files makes post-disclosure analysis confusing https://www.cve.org/CVERecord?id=CVE-2006-2828

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Research Gap

Probably under-reported for PHP. Seems under-studied for other interpreted languages.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-622: Improper Validation of Function Hook Arguments

Weakness ID : 622

Structure : Simple

Abstraction : Variant

Description

The product adds hooks to user-accessible API functions, but it does not properly validate the arguments. This could lead to resultant vulnerabilities.

Extended Description

Such hooks can be used in defensive software that runs with privileges, such as anti-virus or firewall, which hooks kernel calls. When the arguments are not validated, they could be used to bypass the protection scheme or attack the product itself.

Relationships

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		20	Improper Input Validation	20

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Ensure that all arguments are verified, as defined by the API you are protecting.

Phase: Architecture and Design




Drop privileges before invoking such functions, if possible.

Observed Examples

Reference	Description
CVE-2007-0708	DoS in firewall using standard Microsoft functions https://www.cve.org/CVERecord?id=CVE-2007-0708
CVE-2006-7160	DoS in firewall using standard Microsoft functions https://www.cve.org/CVERecord?id=CVE-2006-7160
CVE-2007-1376	function does not verify that its argument is the proper type, leading to arbitrary memory write https://www.cve.org/CVERecord?id=CVE-2007-1376
CVE-2007-1220	invalid syscall arguments bypass code execution limits https://www.cve.org/CVERecord?id=CVE-2007-1220
CVE-2006-4541	DoS in IDS via NULL argument https://www.cve.org/CVERecord?id=CVE-2006-4541

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		991	SFP Secondary Cluster: Tainted Input to Environment	888	2416
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP27		Tainted input to environment

CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

Weakness ID : 623

Structure : Simple

Abstraction : Variant

Description

An ActiveX control is intended for restricted use, but it has been marked as safe-for-scripting.

Extended Description

This might allow attackers to use dangerous functionality via a web page that accesses the control, which can lead to different resultant vulnerabilities, depending on the control's behavior.

Relationships

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		267	Privilege Defined With Unsafe Actions	641
PeerOf		618	Exposed Unsafe ActiveX Method	1380

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity		
Availability		

Potential Mitigations

Phase: Architecture and Design

During development, do not mark it as safe for scripting.

Phase: System Configuration



After distribution, you can set the kill bit for the control so that it is not accessible from Internet Explorer.

Observed Examples

Reference	Description
CVE-2007-0617	control allows attackers to add malicious email addresses to bypass spam limits https://www.cve.org/CVERecord?id=CVE-2007-0617
CVE-2007-0219	web browser uses certain COM objects as ActiveX https://www.cve.org/CVERecord?id=CVE-2007-0219
CVE-2006-6510	kiosk allows bypass to read files https://www.cve.org/CVERecord?id=CVE-2006-6510

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		978	SFP Secondary Cluster: Implementation	888	2408
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

References

[REF-503]Microsoft. "Developing Secure ActiveX Controls". 2005 April 3. < [https://learn.microsoft.com/en-us/previous-versions/ms533046\(v=vs.85\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/ms533046(v=vs.85)?redirectedfrom=MSDN) >.2023-04-07.

[REF-510]Microsoft. "How to stop an ActiveX control from running in Internet Explorer". < <https://support.microsoft.com/en-us/help/240797/how-to-stop-an-activex-control-from-running-in-internet-explorer> >.2023-04-07.

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

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

CWE-624: Executable Regular Expression Error

Weakness ID : 624

Structure : Simple

Abstraction : Base

Description

The product uses a regular expression that either (1) contains an executable component with user-controlled inputs, or (2) allows a user to enable execution by inserting pattern modifiers.

Extended Description

Case (2) is possible in the PHP preg_replace() function, and possibly in other languages when a user-controlled input is inserted into a string that is later parsed as a regular expression.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145


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

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

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

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145

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

Nature	Type	ID	Name	Page
MemberOf		19	Data Processing Errors	2309

Applicable Platforms

Language : PHP (Prevalence = Undetermined)

Language : Perl (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation




The regular expression feature in some languages allows inputs to be quoted or escaped before insertion, such as \Q and \E in Perl.

Observed Examples

Reference	Description
CVE-2006-2059	Executable regexp in PHP by inserting "e" modifier into first argument to preg_replace https://www.cve.org/CVERecord?id=CVE-2006-2059
CVE-2005-3420	Executable regexp in PHP by inserting "e" modifier into first argument to preg_replace https://www.cve.org/CVERecord?id=CVE-2005-3420
CVE-2006-2878	Complex curly syntax inserted into the replacement argument to PHP preg_replace(), which uses the "/e" modifier https://www.cve.org/CVERecord?id=CVE-2006-2878
CVE-2006-2908	Function allows remote attackers to execute arbitrary PHP code via the username field, which is used in a preg_replace function call with a /e (executable) modifier. https://www.cve.org/CVERecord?id=CVE-2006-2908

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Notes

Research Gap

Under-studied. The existing PHP reports are limited to highly skilled researchers, but there are few examples for other languages. It is suspected that this is under-reported for all languages. Usability factors might make it more prevalent in PHP, but this theory has not been investigated.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-625: Permissive Regular Expression

Weakness ID : 625

Structure : Simple

Abstraction : Base

Description

The product uses a regular expression that does not sufficiently restrict the set of allowed values.

Extended Description






This effectively causes the regexp to accept substrings that match the pattern, which produces a partial comparison to the target. In some cases, this can lead to other weaknesses. Common errors include:

- not identifying the beginning and end of the target string
- using wildcards instead of acceptable character ranges
- others


Relationships

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		185	Incorrect Regular Expression	463
ParentOf		777	Regular Expression without Anchors	1636
PeerOf		183	Permissive List of Allowed Inputs	458
PeerOf		184	Incomplete List of Disallowed Inputs	459
PeerOf		187	Partial String Comparison	467

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

Nature	Type	ID	Name	Page
MemberOf		19	Data Processing Errors	2309

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Perl (*Prevalence = Undetermined*)

Language : PHP (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

When applicable, ensure that the regular expression marks beginning and ending string patterns, such as `"/^string$/"` for Perl.

Demonstrative Examples

Example 1:

The following code takes phone numbers as input, and uses a regular expression to reject invalid phone numbers.

Example Language: Perl

(Bad)

```
$phone = GetPhoneNumber();
if ($phone =~ /\d+-\d+/) {
    # looks like it only has hyphens and digits
    system("lookup-phone $phone");
}
else {
    error("malformed number!");
}
```

An attacker could provide an argument such as: `"; ls -l ; echo 123-456"` This would pass the check, since `"123-456"` is sufficient to match the `"\d+-\d+"` portion of the regular expression.

Example 2:

This code uses a regular expression to validate an IP string prior to using it in a call to the "ping" command.

Example Language: Python

(Bad)

```
import subprocess
import re
def validate_ip_regex(ip: str):
    ip_validator = re.compile(r"((25[0-5])|(2[0-4])|1\d|[1-9])\d)\.?\b){4}")
    if ip_validator.match(ip):
        return ip
    else:
        raise ValueError("IP address does not match valid pattern.")
def run_ping_regex(ip: str):
    validated = validate_ip_regex(ip)
    # The ping command treats zero-prepended IP addresses as octal
    result = subprocess.call(["ping", validated])
    print(result)
```




Since the regular expression does not have anchors (CWE-777), i.e. is unbounded without `^` or `$` characters, then prepending a 0 or 0x to the beginning of the IP address will still result in a matched regex pattern. Since the ping command supports octal and hex prepended IP addresses, it will use the unexpectedly valid IP address (CWE-1389). For example, `"0x63.63.63.63"` would be considered equivalent to `"99.63.63.63"`. As a result, the attacker could potentially ping systems that the attacker cannot reach directly.

Observed Examples

Reference	Description
CVE-2021-22204	Chain: regex in EXIF processor code does not correctly determine where a string ends (CWE-625), enabling eval injection (CWE-95), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-22204
CVE-2006-1895	".*" regexp leads to static code injection https://www.cve.org/CVERecord?id=CVE-2006-1895
CVE-2002-2175	insertion of username into regexp results in partial comparison, causing wrong database entry to be updated when one username is a substring of another. https://www.cve.org/CVERecord?id=CVE-2002-2175
CVE-2006-4527	regexp intended to verify that all characters are legal, only checks that at least one is legal, enabling file inclusion. https://www.cve.org/CVERecord?id=CVE-2006-4527
CVE-2005-1949	Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters. https://www.cve.org/CVERecord?id=CVE-2005-1949
CVE-2002-2109	Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings. https://www.cve.org/CVERecord?id=CVE-2002-2109
CVE-2006-6511	regexp in .htaccess file allows access of files whose names contain certain substrings https://www.cve.org/CVERecord?id=CVE-2006-6511
CVE-2006-6629	allow load of macro files whose names contain certain substrings. https://www.cve.org/CVERecord?id=CVE-2006-6629

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	2362
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1397	Comprehensive Categorization: Comparison	1400	2523

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS08-J		Sanitize untrusted data passed to a regex

References

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

CWE-626: Null Byte Interaction Error (Poison Null Byte)

Weakness ID : 626

Structure : Simple

Abstraction : Variant

Description

The product does not properly handle null bytes or NUL characters when passing data between different representations or components.

Extended Description

A null byte (NUL character) can have different meanings across representations or languages. For example, it is a string terminator in standard C libraries, but Perl and PHP strings do not treat it as a terminator. When two representations are crossed - such as when Perl or PHP invokes underlying C functionality - this can produce an interaction error with unexpected results. Similar issues have been reported for ASP. Other interpreters written in C might also be affected.

The poison null byte is frequently useful in path traversal attacks by terminating hard-coded extensions that are added to a filename. It can play a role in regular expression processing in PHP.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	1057
ChildOf		147	Improper Neutralization of Input Terminators	389

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP (*Prevalence = Undetermined*)

Language : Perl (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Remove null bytes from all incoming strings.

Observed Examples

Reference	Description
CVE-2005-4155	NUL byte bypasses PHP regular expression check https://www.cve.org/CVERecord?id=CVE-2005-4155
CVE-2005-3153	inserting SQL after a NUL byte bypasses allowlist regexp, enabling SQL injection https://www.cve.org/CVERecord?id=CVE-2005-3153

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

Notes

Terminology

Current usage of "poison null byte" is typically related to this C/Perl/PHP interaction error, but the original term in 1998 was applied to an off-by-one buffer overflow involving a null byte.

Research Gap

There are not many CVE examples, because the poison NULL byte is a design limitation, which typically is not included in CVE by itself. It is typically used as a facilitator manipulation to widen the scope of potential attacks against other vulnerabilities.

References

[REF-514]Rain Forest Puppy. "Poison NULL byte". Phrack 55. < <https://insecure.org/news/P55-07.txt> >.2023-04-07.

[REF-515]Brett Moore. "0x00 vs ASP file upload scripts". < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

[REF-516]ShAnKaR. "ShAnKaR: multiple PHP application poison NULL byte vulnerability". < <https://seclists.org/fulldisclosure/2006/Sep/185> >.2023-04-07.

CWE-627: Dynamic Variable Evaluation

Weakness ID : 627

Structure : Simple

Abstraction : Variant

Description

In a language where the user can influence the name of a variable at runtime, if the variable names are not controlled, an attacker can read or write to arbitrary variables, or access arbitrary functions.

Extended Description

The resultant vulnerabilities depend on the behavior of the application, both at the crossover point and in any control/data flow that is reachable by the related variables or functions.

Relationships

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	914	Improper Control of Dynamically-Identified Variables	1807
PeerOf	B	183	Permissive List of Allowed Inputs	458

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP (*Prevalence = Undetermined*)

Language : Perl (*Prevalence = Undetermined*)

Background Details

Many interpreted languages support the use of a "\$\$varname" construct to set a variable whose name is specified by the \$varname variable. In PHP, these are referred to as "variable variables." Functions might also be invoked using similar syntax, such as \$\$funcname(arg1, arg2).

Alternate Terms

Dynamic evaluation :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Application Data	
Integrity	Execute Unauthorized Code or Commands	
Availability	<i>An attacker could gain unauthorized access to internal program variables and execute arbitrary code.</i>	

Potential Mitigations

Phase: Implementation

Strategy = Refactoring

Refactor the code to avoid dynamic variable evaluation whenever possible.

Phase: Implementation

Strategy = Input Validation

Use only allowlists of acceptable variable or function names.

Phase: Implementation

For function names, ensure that you are only calling functions that accept the proper number of arguments, to avoid unexpected null arguments.

Observed Examples

Reference	Description
CVE-2009-0422	Chain: Dynamic variable evaluation allows resultant remote file inclusion and path traversal. https://www.cve.org/CVERecord?id=CVE-2009-0422
CVE-2007-2431	Chain: dynamic variable evaluation in PHP program used to modify critical, unexpected \$_SERVER variable for resultant XSS. https://www.cve.org/CVERecord?id=CVE-2007-2431
CVE-2006-4904	Chain: dynamic variable evaluation in PHP program used to conduct remote file inclusion. https://www.cve.org/CVERecord?id=CVE-2006-4904
CVE-2006-4019	Dynamic variable evaluation in mail program allows reading and modifying attachments and preferences of other users. https://www.cve.org/CVERecord?id=CVE-2006-4019

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Research Gap

Under-studied, probably under-reported. Few researchers look for this issue; most public reports are for PHP, although other languages are affected. This issue is likely to grow in PHP as developers begin to implement functionality in place of register_globals.

References

[REF-517]Steve Christey. "Dynamic Evaluation Vulnerabilities in PHP applications". Full-Disclosure. 2006 May 3. < <https://seclists.org/fulldisclosure/2006/May/35> >.2023-04-07.

[REF-518]Shaun Clowes. "A Study In Scarlet: Exploiting Common Vulnerabilities in PHP Applications". < <https://securereality.com.au/study-in-scarlett/> >.2023-04-07.

CWE-628: Function Call with Incorrectly Specified Arguments

Weakness ID : 628

Structure : Simple

Abstraction : Base

Description

The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.

Extended Description







There are multiple ways in which this weakness can be introduced, including:

- the wrong variable or reference;
- an incorrect number of arguments;
- incorrect order of arguments;
- wrong type of arguments; or
- wrong value.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1298
ParentOf		683	Function Call With Incorrect Order of Arguments	1504
ParentOf		685	Function Call With Incorrect Number of Arguments	1507
ParentOf		686	Function Call With Incorrect Argument Type	1508
ParentOf		687	Function Call With Incorrectly Specified Argument Value	1510
ParentOf		688	Function Call With Incorrect Variable or Reference as Argument	1511

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Primary : This is usually primary to other weaknesses, but it can be resultant if the function's API or function prototype changes.

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other Access Control	Quality Degradation Gain Privileges or Assume Identity	
<i>This weakness can cause unintended behavior and can lead to additional weaknesses such as allowing an attacker to gain unintended access to system resources.</i>		

Detection Methods

Other

Since these bugs typically introduce incorrect behavior that is obvious to users, they are found quickly, unless they occur in rarely-tested code paths. Managing the correct number of arguments can be made more difficult in cases where format strings are used, or when variable numbers of arguments are supported.

Potential Mitigations

Phase: Build and Compilation

Once found, these issues are easy to fix. Use code inspection tools and relevant compiler features to identify potential violations. Pay special attention to code that is not likely to be exercised heavily during QA.

Phase: Architecture and Design

Make sure your API's are stable before you use them in production code.

Demonstrative Examples

Example 1:

The following PHP method authenticates a user given a username/password combination but is called with the parameters in reverse order.

Example Language: PHP

(Bad)

```
function authenticate($username, $password) {
    // authenticate user
    ...
}
authenticate($_POST['password'], $_POST['username']);
```

Example 2:

This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

Example Language: Perl

(Bad)

```
sub ReportAuth {
    my ($username, $result, $fatal) = @_;
    PrintLog("auth: username=%s, result=%d", $username, $result);
    if (($result ne "success") && $fatal) {
        die "Failed!\n";
    }
}
sub PrivilegedFunc
{
    my $result = CheckAuth($username);
```

```
ReportAuth($username, $result, 0);
DoReallyImportantStuff();
}
```

Example 3:

In the following Java snippet, the `accessGranted()` method is accidentally called with the static `ADMIN_ROLES` array rather than the user roles.

Example Language: Java

(Bad)










```
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
    String[] userRoles = getUserRoles(user);
    return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
    // grant or deny access based on user roles
    ...
}
```

Observed Examples

Reference	Description
CVE-2006-7049	The method calls the functions with the wrong argument order, which allows remote attackers to bypass intended access restrictions. https://www.cve.org/CVERecord?id=CVE-2006-7049

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		736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	2341
MemberOf		737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2341
MemberOf		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf		884	CWE Cross-section	884	2567
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf		1180	SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1178	2465
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL10-C		Maintain the contract between the writer and caller of variadic functions
CERT C Secure Coding	EXP37-C	CWE More Abstract	Call functions with the correct number and type of arguments
SEI CERT Perl Coding Standard	DCL00-PL	CWE More Abstract	Do not use subroutine prototypes

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	EXP33-PL	Imprecise	Do not invoke a function in a context for which it is not defined

CWE-636: Not Failing Securely ('Failing Open')

Weakness ID : 636

Structure : Simple

Abstraction : Class

Description

When the product encounters an error condition or failure, its design requires it to fall back to a state that is less secure than other options that are available, such as selecting the weakest encryption algorithm or using the most permissive access control restrictions.


Extended Description

By entering a less secure state, the product inherits the weaknesses associated with that state, making it easier to compromise. At the least, it causes administrators to have a false sense of security. This weakness typically occurs as a result of wanting to "fail functional" to minimize administration and support costs, instead of "failing safe."

Relationships

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	1576
ChildOf		657	Violation of Secure Design Principles	1446
ParentOf		455	Non-exit on Failed Initialization	1087
PeerOf		280	Improper Handling of Insufficient Permissions or Privileges	672

Weakness Ordinalities

Primary :

Applicable Platforms

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

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

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

Alternate Terms

Failing Open :

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>Intended access restrictions can be bypassed, which is often contradictory to what the product's administrator expects.</i>	

Potential Mitigations

Phase: Architecture and Design

Subdivide and allocate resources and components so that a failure in one part does not affect the entire product.

Demonstrative Examples

Example 1:





Switches may revert their functionality to that of hubs when the table used to map ARP information to the switch interface overflows, such as when under a spoofing attack. This results in traffic being broadcast to an eavesdropper, instead of being sent only on the relevant switch interface. To mitigate this type of problem, the developer could limit the number of ARP entries that can be recorded for a given switch interface, while other interfaces may keep functioning normally. Configuration options can be provided on the appropriate actions to be taken in case of a detected failure, but safe defaults should be used.

Observed Examples

Reference	Description
CVE-2007-5277	The failure of connection attempts in a web browser resets DNS pin restrictions. An attacker can then bypass the same origin policy by rebinding a domain name to a different IP address. This was an attempt to "fail functional." https://www.cve.org/CVERecord?id=CVE-2007-5277
CVE-2006-4407	Incorrect prioritization leads to the selection of a weaker cipher. Although it is not known whether this issue occurred in implementation or design, it is feasible that a poorly designed algorithm could be a factor. https://www.cve.org/CVERecord?id=CVE-2006-4407

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		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	2337
MemberOf		961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2399
MemberOf		1369	ICS Supply Chain: IT/OT Convergence/Expansion	1358	2506
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

Research Gap

Since design issues are hard to fix, they are rarely publicly reported, so there are few CVE examples of this problem as of January 2008. Most publicly reported issues occur as the result of an implementation error instead of design, such as CVE-2005-3177 (Improper handling of large numbers of resources) or CVE-2005-2969 (inadvertently disabling a verification step, leading to selection of a weaker protocol).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-522]Sean Barnum and Michael Gegick. "Failing Securely". 2005 December 5. < <https://web.archive.org/web/20221017053210/https://www.cisa.gov/uscrt/bsi/articles/knowledge/principles/failing-securely> >.2023-04-07.

CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')

Weakness ID : 637

Structure : Simple

Abstraction : Class

Description

The product uses a more complex mechanism than necessary, which could lead to resultant weaknesses when the mechanism is not correctly understood, modeled, configured, implemented, or used.

Extended Description

Security mechanisms should be as simple as possible. Complex security mechanisms may engender partial implementations and compatibility problems, with resulting mismatches in assumptions and implemented security. A corollary of this principle is that data specifications should be as simple as possible, because complex data specifications result in complex validation code. Complex tasks and systems may also need to be guarded by complex security checks, so simple systems should be preferred.

Relationships

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		657	Violation of Secure Design Principles	1446

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Unnecessary Complexity :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Avoid complex security mechanisms when simpler ones would meet requirements. Avoid complex data models, and unnecessarily complex operations. Adopt architectures that provide guarantees, simplify understanding through elegance and abstraction, and that can be implemented similarly. Modularize, isolate and do not trust complex code, and apply other secure programming principles on these modules (e.g., least privilege) to mitigate vulnerabilities.

Demonstrative Examples

Example 1:

The IPSEC specification is complex, which resulted in bugs, partial implementations, and incompatibilities between vendors.

Example 2:



HTTP Request Smuggling (CWE-444) attacks are feasible because there are not stringent requirements for how illegal or inconsistent HTTP headers should be handled. This can lead to inconsistent implementations in which a proxy or firewall interprets the same data stream as a different set of requests than the end points in that stream.

Observed Examples

Reference	Description
CVE-2007-6067	Support for complex regular expressions leads to a resultant algorithmic complexity weakness (CWE-407). https://www.cve.org/CVERecord?id=CVE-2007-6067
CVE-2007-1552	Either a filename extension and a Content-Type header could be used to infer the file type, but the developer only checks the Content-Type, enabling unrestricted file upload (CWE-434). https://www.cve.org/CVERecord?id=CVE-2007-1552
CVE-2007-6479	In Apache environments, a "filename.php.gif" can be redirected to the PHP interpreter instead of being sent as an image/gif directly to the user. Not knowing this, the developer only checks the last extension of a submitted filename, enabling arbitrary code execution. https://www.cve.org/CVERecord?id=CVE-2007-6479
CVE-2005-2148	The developer cleanses the \$_REQUEST superglobal array, but PHP also populates \$_GET, allowing attackers to bypass the protection mechanism and conduct SQL injection attacks against code that uses \$_GET. https://www.cve.org/CVERecord?id=CVE-2005-2148

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		975	SFP Secondary Cluster: Architecture	888	2406
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-524]Sean Barnum and Michael Gegick. "Economy of Mechanism". 2005 September 3. < <https://web.archive.org/web/20220126060058/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/economy-of-mechanism> >.2023-04-07.

CWE-638: Not Using Complete Mediation

Weakness ID : 638

Structure : Simple

Abstraction : Class

Description

The product does not perform access checks on a resource every time the resource is accessed by an entity, which can create resultant weaknesses if that entity's rights or privileges change over time.

Relationships

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		862	Missing Authorization	1780
ChildOf		657	Violation of Secure Design Principles	1446
ParentOf		424	Improper Protection of Alternate Path	1023

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Gain Privileges or Assume Identity	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	Bypass Protection Mechanism	
Access Control	Read Application Data	
Other	Other	
<i>A user might retain access to a critical resource even after privileges have been revoked, possibly allowing access to privileged functionality or sensitive information, depending on the role of the resource.</i>		

Potential Mitigations

Phase: Architecture and Design

Invalidate cached privileges, file handles or descriptors, or other access credentials whenever identities, processes, policies, roles, capabilities or permissions change. Perform complete authentication checks before accepting, caching and reusing data, dynamic content and code (scripts). Avoid caching access control decisions as much as possible.

Phase: Architecture and Design

Identify all possible code paths that might access sensitive resources. If possible, create and use a single interface that performs the access checks, and develop code standards that require use of this interface.

Demonstrative Examples

Example 1:

When executable library files are used on web servers, which is common in PHP applications, the developer might perform an access check in any user-facing executable, and omit the access check from the library file itself. By directly requesting the library file (CWE-425), an attacker can bypass this access check.

Example 2:

When a developer begins to implement input validation for a web application, often the validation is performed in each area of the code that uses externally-controlled input. In complex applications with many inputs, the developer often misses a parameter here or a cookie there. One frequently-applied solution is to centralize all input validation, store these validated inputs in a separate data structure, and require that all access of those inputs must be through that data structure. An alternate approach would be to use an external input validation framework such as Struts, which performs the validation before the inputs are ever processed by the code.

Observed Examples

Reference	Description
CVE-2007-0408	Server does not properly validate client certificates when reusing cached connections. https://www.cve.org/CVERecord?id=CVE-2007-0408

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	988	SFP Secondary Cluster: Race Condition Window	888	2412
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2505
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
104	Cross Zone Scripting

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-526]Sean Barnum and Michael Gegick. "Complete Mediation". 2005 September 2. < <https://web.archive.org/web/20221006191503/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/complete-mediation> >.2023-04-07.

CWE-639: Authorization Bypass Through User-Controlled Key

Weakness ID : 639

Structure : Simple

Abstraction : Base

Description

The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.

Extended Description

Retrieval of a user record occurs in the system based on some key value that is under user control. The key would typically identify a user-related record stored in the system and would be used to

lookup that record for presentation to the user. It is likely that an attacker would have to be an authenticated user in the system. However, the authorization process would not properly check the data access operation to ensure that the authenticated user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other authorization checks present in the system.



For example, attackers can look at places where user specific data is retrieved (e.g. search screens) and determine whether the key for the item being looked up is controllable externally. The key may be a hidden field in the HTML form field, might be passed as a URL parameter or as an unencrypted cookie variable, then in each of these cases it will be possible to tamper with the key value.

One manifestation of this weakness is when a system uses sequential or otherwise easily-guessable session IDs that would allow one user to easily switch to another user's session and read/modify their data.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		863	Incorrect Authorization	1787
ParentOf		566	Authorization Bypass Through User-Controlled SQL Primary Key	1286

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

Nature	Type	ID	Name	Page
ChildOf		863	Incorrect Authorization	1787



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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2476
MemberOf		840	Business Logic Errors	2360

Applicable Platforms

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

Alternate Terms

Insecure Direct Object Reference / IDOR : The "Insecure Direct Object Reference" term, as described in the OWASP Top Ten, is broader than this CWE because it also covers path traversal (CWE-22). Within the context of vulnerability theory, there is a similarity between the OWASP concept and CWE-706: Use of Incorrectly-Resolved Name or Reference.

Broken Object Level Authorization / BOLA : BOLA is used in the 2019 OWASP API Security Top 10 and is said to be the same as IDOR.

Horizontal Authorization : "Horizontal Authorization" is used to describe situations in which two users have the same privilege level, but must be prevented from accessing each other's resources. This is fairly common when using key-based access to resources in a multi-user context.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>Access control checks for specific user data or functionality can be bypassed.</i>	
Access Control	Gain Privileges or Assume Identity <i>Horizontal escalation of privilege is possible (one user can view/modify information of another user).</i>	
Access Control	Gain Privileges or Assume Identity <i>Vertical escalation of privilege is possible if the user-controlled key is actually a flag that indicates administrator status, allowing the attacker to gain administrative access.</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

For each and every data access, ensure that the user has sufficient privilege to access the record that is being requested.

Phase: Architecture and Design

Phase: Implementation

Make sure that the key that is used in the lookup of a specific user's record is not controllable externally by the user or that any tampering can be detected.

Phase: Architecture and Design

Use encryption in order to make it more difficult to guess other legitimate values of the key or associate a digital signature with the key so that the server can verify that there has been no tampering.

Demonstrative Examples

Example 1:

The following code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice

matching the specified identifier [1]. The identifier is selected from a list of all invoices associated with the current authenticated user.

Example Language: C#

(Bad)

```
...
conn = new SqlConnection(_ConnectionString);
conn.Open();
int16 id = System.Convert.ToInt16(invoiceID.Text);
SqlCommand query = new SqlCommand("SELECT * FROM invoices WHERE id = @id", conn);
query.Parameters.AddWithValue("@id", id);
SqlDataReader objReader = objCommand.ExecuteReader();
...
```









The problem is that the developer has not considered all of the possible values of id. Although the interface generates a list of invoice identifiers that belong to the current user, an attacker can bypass this interface to request any desired invoice. Because the code in this example does not check to ensure that the user has permission to access the requested invoice, it will display any invoice, even if it does not belong to the current user.

Observed Examples

Reference	Description
CVE-2021-36539	An educational application does not appropriately restrict file IDs to a particular user. The attacker can brute-force guess IDs, indicating IDOR. https://www.cve.org/CVERecord?id=CVE-2021-36539

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	2331
MemberOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf		813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf		932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	2390
MemberOf		945	SFP Secondary Cluster: Insecure Resource Access	888	2394
MemberOf		1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	2437
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

CWE-640: Weak Password Recovery Mechanism for Forgotten Password

Weakness ID : 640

Structure : Simple

Abstraction : Base

Description

The product contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak.

Extended Description

It is common for an application to have a mechanism that provides a means for a user to gain access to their account in the event they forget their password. Very often the password recovery mechanism is weak, which has the effect of making it more likely that it would be possible for a person other than the legitimate system user to gain access to that user's account. Weak password recovery schemes completely undermine a strong password authentication scheme.

This weakness may be that the security question is too easy to guess or find an answer to (e.g. because the question is too common, or the answers can be found using social media). Or there might be an implementation weakness in the password recovery mechanism code that may for instance trick the system into e-mailing the new password to an e-mail account other than that of the user. There might be no throttling done on the rate of password resets so that a legitimate user can be denied service by an attacker if an attacker tries to recover their password in a rapid succession. The system may send the original password to the user rather than generating a new temporary password. In summary, password recovery functionality, if not carefully designed and implemented can often become the system's weakest link that can be misused in a way that would allow an attacker to gain unauthorized access to the system.


Relationships

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		1390	Weak Authentication	2267



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

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	692

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

Nature	Type	ID	Name	Page
MemberOf		1010	Authenticate Actors	2424

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

Nature	Type	ID	Name	Page
MemberOf		255	Credentials Management Errors	2315
MemberOf		840	Business Logic Errors	2360

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity <i>An attacker could gain unauthorized access to the system by retrieving legitimate user's authentication credentials.</i>	
Availability	DoS: Resource Consumption (Other)	

Scope	Impact	Likelihood
	<i>An attacker could deny service to legitimate system users by launching a brute force attack on the password recovery mechanism using user ids of legitimate users.</i>	
Integrity	Other	
Other	<i>The system's security functionality is turned against the system by the attacker.</i>	

Potential Mitigations

Phase: Architecture and Design

Make sure that all input supplied by the user to the password recovery mechanism is thoroughly filtered and validated.

Phase: Architecture and Design

Do not use standard weak security questions and use several security questions.

Phase: Architecture and Design

Make sure that there is throttling on the number of incorrect answers to a security question.

Disable the password recovery functionality after a certain (small) number of incorrect guesses.

Phase: Architecture and Design

Require that the user properly answers the security question prior to resetting their password and sending the new password to the e-mail address of record.

Phase: Architecture and Design

Never allow the user to control what e-mail address the new password will be sent to in the password recovery mechanism.

Phase: Architecture and Design

Assign a new temporary password rather than revealing the original password.

Demonstrative Examples

Example 1:

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

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		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2335
MemberOf		930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2389
MemberOf		959	SFP Secondary Cluster: Weak Cryptography	888	2398
MemberOf		1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2436
MemberOf		1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494

Nature	Type	ID	Name	V	Page
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry might be reclassified as a category or "loose composite," since it lists multiple specific errors that can make the mechanism weak. However, under view 1000, it could be a weakness under protection mechanism failure, although it is different from most PMF issues since it is related to a feature that is designed to bypass a protection mechanism (specifically, the lack of knowledge of a password).

Maintenance

This entry probably needs to be split; see extended description.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	49		Insufficient Password Recovery

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
50	Password Recovery Exploitation

References

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

CWE-641: Improper Restriction of Names for Files and Other Resources

Weakness ID : 641

Structure : Simple

Abstraction : Base

Description

The product constructs the name of a file or other resource using input from an upstream component, but it does not restrict or incorrectly restricts the resulting name.

Extended Description

This may produce resultant weaknesses. For instance, if the names of these resources contain scripting characters, it is possible that a script may get executed in the client's browser if the application ever displays the name of the resource on a dynamically generated web page. Alternately, if the resources are consumed by some application parser, a specially crafted name can exploit some vulnerability internal to the parser, potentially resulting in execution of arbitrary code on the server machine. The problems will vary based on the context of usage of such malformed resource names and whether vulnerabilities are present in or assumptions are made by the targeted technology that would make code execution possible.

Relationships

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		99	Improper Control of Resource Identifiers ('Resource Injection')	243

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478
MemberOf		137	Data Neutralization Issues	2311
MemberOf		399	Resource Management Errors	2324

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	<i>Execution of arbitrary code in the context of usage of the resources with dangerous names.</i>	
Availability		
Confidentiality	Read Application Data	
Availability	DoS: Crash, Exit, or Restart	
	<i>Crash of the consumer code of these resources resulting in information leakage or denial of service.</i>	

Potential Mitigations

Phase: Architecture and Design

Do not allow users to control names of resources used on the server side.

Phase: Architecture and Design




Perform allowlist input validation at entry points and also before consuming the resources. Reject bad file names rather than trying to cleanse them.

Phase: Architecture and Design

Make sure that technologies consuming the resources are not vulnerable (e.g. buffer overflow, format string, etc.) in a way that would allow code execution if the name of the resource is malformed.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-642: External Control of Critical State Data

Weakness ID : 642

Structure : Simple

Abstraction : Class

Description

The product stores security-critical state information about its users, or the product itself, in a location that is accessible to unauthorized actors.

Extended Description

If an attacker can modify the state information without detection, then it could be used to perform unauthorized actions or access unexpected resources, since the application programmer does not expect that the state can be changed.

State information can be stored in various locations such as a cookie, in a hidden web form field, input parameter or argument, an environment variable, a database record, within a settings file, etc. All of these locations have the potential to be modified by an attacker. When this state information is used to control security or determine resource usage, then it may create a vulnerability. For example, an application may perform authentication, then save the state in an "authenticated=true" cookie. An attacker may simply create this cookie in order to bypass the authentication.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1469
ParentOf		15	External Control of System or Configuration Setting	17
ParentOf		73	External Control of File Name or Path	132
ParentOf		426	Untrusted Search Path	1028
ParentOf		472	External Control of Assumed-Immutable Web Parameter	1123
ParentOf		565	Reliance on Cookies without Validation and Integrity Checking	1283

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

Applicable Platforms

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

Technology : Web Server (*Prevalence = Often*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity <i>An attacker could potentially modify the state in malicious ways. If the state is related to the privileges or level of authentication that the user has, then state modification might allow the user to bypass authentication or elevate privileges.</i>	
Confidentiality	Read Application Data <i>The state variables may contain sensitive information that should not be known by the client.</i>	
Availability	DoS: Crash, Exit, or Restart <i>By modifying state variables, the attacker could violate the application's expectations for the contents of the state, leading to a denial of service due to an unexpected error condition.</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

Understand all the potential locations that are accessible to attackers. For example, some programmers assume that cookies and hidden form fields cannot be modified by an attacker, or they may not consider that environment variables can be modified before a privileged program is invoked.

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Store state information and sensitive data on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions. If information must be stored on the client, do not do so without encryption and integrity checking, or otherwise having a mechanism on the server side to catch tampering. Use a message authentication code (MAC) algorithm, such as Hash Message Authentication Code (HMAC) [REF-529]. Apply this against the state or sensitive data that has to be exposed, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that a strong hash function is used (CWE-328).

Phase: Architecture and Design

Store state information on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions.

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. With a stateless protocol such as HTTP, use some frameworks can maintain the state for you. Examples include ASP.NET View State and the OWASP ESAPI Session Management feature. Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

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: Operation**Phase: Implementation***Strategy = Environment Hardening*

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

Phase: Testing

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

Phase: Testing

Use dynamic tools and techniques that interact with the product using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The product's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Testing

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

Demonstrative Examples**Example 1:**

In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

Example Language: Java

(Bad)

```
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

Example 2:

The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files (CWE-22).

Example Language: Java

(Bad)

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 3:

The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

Example Language: Java

(Bad)

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

Example 4:

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 5:

The following code segment implements a basic server that uses the "ls" program to perform a directory listing of the directory that is listed in the "HOMEDIR" environment variable. The code intends to allow the user to specify an alternate "LANG" environment variable. This causes "ls" to customize its output based on a given language, which is an important capability when supporting internationalization.

Example Language: Perl (Bad)

```
$ENV{"HOMEDIR"} = "/home/mydir/public/";
my $stream = AcceptUntrustedInputStream();
while (<$stream>) {
    chomp;
    if (/^ENV ([\w_]+) (.*)/) {
        $ENV{$1} = $2;
    }
    elsif (/^QUIT/) { ... }
    elsif (/^LIST/) {
        open($fh, "/bin/ls -l $ENV{HOMEDIR}|");
        while (<$fh>) {
            SendOutput($stream, "FILEINFO: $_");
        }
        close($fh);
    }
}
```

The programmer takes care to call a specific "ls" program and sets the HOMEDIR to a fixed value. However, an attacker can use a command such as "ENV HOMEDIR /secret/directory" to specify an alternate directory, enabling a path traversal attack (CWE-22). At the same time, other attacks are enabled as well, such as OS command injection (CWE-78) by setting HOMEDIR to a value such as "/tmp; rm -rf /". In this case, the programmer never intends for HOMEDIR to be modified, so input validation for HOMEDIR is not the solution. A partial solution would be an allowlist that only allows the LANG variable to be specified in the ENV command. Alternately, assuming this is an authenticated user, the language could be stored in a local file so that no ENV command at all would be needed.

While this example may not appear realistic, this type of problem shows up in code fairly frequently. See CVE-1999-0073 in the observed examples for a real-world example with similar behaviors.







Observed Examples

Reference	Description
CVE-2005-2428	Mail client stores password hashes for unrelated accounts in a hidden form field. https://www.cve.org/CVERecord?id=CVE-2005-2428
CVE-2008-0306	Privileged program trusts user-specified environment variable to modify critical configuration settings. https://www.cve.org/CVERecord?id=CVE-2008-0306
CVE-1999-0073	Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution. https://www.cve.org/CVERecord?id=CVE-1999-0073
CVE-2007-4432	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable. https://www.cve.org/CVERecord?id=CVE-2007-4432
CVE-2006-7191	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable. https://www.cve.org/CVERecord?id=CVE-2006-7191
CVE-2008-5738	Calendar application allows bypass of authentication by setting a certain cookie value to 1. https://www.cve.org/CVERecord?id=CVE-2008-5738
CVE-2008-5642	Setting of a language preference in a cookie enables path traversal attack.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2008-5642
CVE-2008-5125	Application allows admin privileges by setting a cookie value to "admin." https://www.cve.org/CVERecord?id=CVE-2008-5125
CVE-2008-5065	Application allows admin privileges by setting a cookie value to "admin." https://www.cve.org/CVERecord?id=CVE-2008-5065
CVE-2008-4752	Application allows admin privileges by setting a cookie value to "admin." https://www.cve.org/CVERecord?id=CVE-2008-4752
CVE-2000-0102	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0102
CVE-2000-0253	Shopping cart allows price modification via hidden form field. https://www.cve.org/CVERecord?id=CVE-2000-0253
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

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		752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf		884	CWE Cross-section	884	2567
MemberOf		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Taxonomy Mappings

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
31	Accessing/Intercepting/Modifying HTTP Cookies

References

- [REF-528]OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.
- [REF-529]"HMAC". 2011 August 8. Wikipedia. < <https://en.wikipedia.org/wiki/HMAC> >.2023-04-07.
- [REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection')

Weakness ID : 643
Structure : Simple
Abstraction : Base

Description

The product uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.

Extended Description

The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Relationships

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		91	XML Injection (aka Blind XPath Injection)	215
ChildOf		943	Improper Neutralization of Special Elements in Data Query Logic	1850

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>Controlling application flow (e.g. bypassing authentication).</i>	
Confidentiality	Read Application Data <i>The attacker could read restricted XML content.</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 parameterized XPath queries (e.g. using XQuery). This will help ensure separation between data plane and control plane.

Phase: Implementation

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XPath queries is safe in that context.

Demonstrative Examples

Example 1:

Consider the following simple XML document that stores authentication information and a snippet of Java code that uses XPath query to retrieve authentication information:

Example Language: XML

(Informative)

```
<users>
  <user>
    <login>john</login>
    <password>abracadabra</password>
    <home_dir>/home/john</home_dir>
  </user>
  <user>
    <login>cbc</login>
    <password>1mgr8</password>
    <home_dir>/home/cbc</home_dir>
  </user>
</users>
```

The Java code used to retrieve the home directory based on the provided credentials is:

Example Language: Java

(Bad)

```
XPath xpath = XPathFactory.newInstance().newXPath();
XPathExpression xlogin = xpath.compile("//users/user[login/text()=' " + login.getUserName() + "' and password/text() = '" +
login.getPassword() + "']/home_dir/text()");
Document d = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new File("db.xml"));
String homedir = xlogin.evaluate(d);
```

Assume that user "john" wishes to leverage XPath Injection and login without a valid password. By providing a username "john" and password "" or "=" the XPath expression now becomes

Example Language:





(Attack)

```
//users/user[login/text()='john' or "=" and password/text() = " or "="]/home_dir/text()
```

This lets user "john" login without a valid password, thus bypassing authentication.

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		929	OWASP Top Ten 2013 Category A1 - Injection		2389
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command		2413
MemberOf		1308	CISQ Quality Measures - Security		2485
MemberOf		1340	CISQ Data Protection Measures		2590
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection		2490
MemberOf		1409	Comprehensive Categorization: Injection		2535

Notes

Relationship

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	39		XPath Injection
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-531]Web Application Security Consortium. "XPath Injection". < <http://projects.webappsec.org/w/page/13247005/XPath%20Injection> >.2023-04-07.

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

CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax

Weakness ID : 644

Structure : Simple

Abstraction : Variant

Description

The product does not neutralize or incorrectly neutralizes web scripting syntax in HTTP headers that can be used by web browser components that can process raw headers, such as Flash.

Extended Description


An attacker may be able to conduct cross-site scripting and other attacks against users who have these components enabled.

If a product does not neutralize user controlled data being placed in the header of an HTTP response coming from the server, the header may contain a script that will get executed in the client's browser context, potentially resulting in a cross site scripting vulnerability or possibly an HTTP response splitting attack. It is important to carefully control data that is being placed both in HTTP response header and in the HTTP response body to ensure that no scripting syntax is present, taking various encodings into account.

Relationships

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		116	Improper Encoding or Escaping of Output	281

Applicable Platforms

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

Technology : Web Based (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Run arbitrary code.	
Availability		
Confidentiality	Read Application Data	
	Attackers may be able to obtain sensitive information.	

Potential Mitigations

Phase: Architecture and Design

Perform output validation in order to filter/escape/encode unsafe data that is being passed from the server in an HTTP response header.

Phase: Architecture and Design

Disable script execution functionality in the clients' browser.

Demonstrative Examples

Example 1:

In the following Java example, user-controlled data is added to the HTTP headers and returned to the client. Given that the data is not subject to neutralization, a malicious user may be able to inject dangerous scripting tags that will lead to script execution in the client browser.

Example Language: Java

(Bad)






```
response.addHeader(HEADER_NAME, untrustedRawInputData);
```

Observed Examples

Reference	Description
CVE-2006-3918	Web server does not remove the Expect header from an HTTP request when it is reflected back in an error message, allowing a Flash SWF file to perform XSS attacks. https://www.cve.org/CVERecord?id=CVE-2006-3918

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		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711	2336
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2532

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-645: Overly Restrictive Account Lockout Mechanism

Weakness ID : 645

Structure : Simple

Abstraction : Base

Description

The product contains an account lockout protection mechanism, but the mechanism is too restrictive and can be triggered too easily, which allows attackers to deny service to legitimate users by causing their accounts to be locked out.


Extended Description

Account lockout is a security feature often present in applications as a countermeasure to the brute force attack on the password based authentication mechanism of the system. After a certain number of failed login attempts, the users' account may be disabled for a certain period of time or until it is unlocked by an administrator. Other security events may also possibly trigger account lockout. However, an attacker may use this very security feature to deny service to legitimate system users. It is therefore important to ensure that the account lockout security mechanism is not overly restrictive.

Relationships

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		287	Improper Authentication	692

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

Nature	Type	ID	Name	Page
MemberOf		1017	Lock Computer	2431

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

Nature	Type	ID	Name	Page
MemberOf		1211	Authentication Errors	2475
MemberOf		1216	Lockout Mechanism Errors	2478

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>Users could be locked out of accounts.</i>	

Potential Mitigations

Phase: Architecture and Design

Implement more intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name.

Phase: Architecture and Design

Implement a lockout timeout that grows as the number of incorrect login attempts goes up, eventually resulting in a complete lockout.

Phase: Architecture and Design

Consider alternatives to account lockout that would still be effective against password brute force attacks, such as presenting the user machine with a puzzle to solve (makes it do some computation).

Demonstrative Examples

Example 1:

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

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	951	SFP Secondary Cluster: Insecure Authentication Policy	888	2396
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
2	Inducing Account Lockout

CWE-646: Reliance on File Name or Extension of Externally-Supplied File

Weakness ID : 646

Structure : Simple

Abstraction : Variant

Description

The product allows a file to be uploaded, but it relies on the file name or extension of the file to determine the appropriate behaviors. This could be used by attackers to cause the file to be misclassified and processed in a dangerous fashion.

Extended Description

An application might use the file name or extension of a user-supplied file to determine the proper course of action, such as selecting the correct process to which control should be passed, deciding what data should be made available, or what resources should be allocated. If the attacker can cause the code to misclassify the supplied file, then the wrong action could occur. For example, an attacker could supply a file that ends in a ".php.gif" extension that appears to be a GIF image, but would be processed as PHP code. In extreme cases, code execution is possible, but the attacker could also cause exhaustion of resources, denial of service, exposure of debug or system data (including application source code), or being bound to a particular server side process. This weakness may be due to a vulnerability in any of the technologies used by the web and application servers, due to misconfiguration, or resultant from another flaw in the application itself.

Relationships

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	851

Applicable Platforms

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

Technology : Web Server (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>An attacker may be able to read sensitive data.</i>	
Availability	DoS: Crash, Exit, or Restart <i>An attacker may be able to cause a denial of service.</i>	
Access Control	Gain Privileges or Assume Identity <i>An attacker may be able to gain privileges.</i>	

Potential Mitigations

Phase: Architecture and Design

Make decisions on the server side based on file content and not on file name or extension.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf		1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
209	XSS Using MIME Type Mismatch

CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions

Weakness ID : 647

Structure : Simple

Abstraction : Variant

Description

The product defines policy namespaces and makes authorization decisions based on the assumption that a URL is canonical. This can allow a non-canonical URL to bypass the authorization.

Extended Description

If an application defines policy namespaces and makes authorization decisions based on the URL, but it does not require or convert to a canonical URL before making the authorization decision,

then it opens the application to attack. For example, if the application only wants to allow access to `http://www.example.com/mypage`, then the attacker might be able to bypass this restriction using equivalent URLs such as:

- `http://WWW.EXAMPLE.COM/mypage`
- `http://www.example.com/%6Dypage` (alternate encoding)
- `http://192.168.1.1/mypage` (IP address)
- `http://www.example.com/mypage/` (trailing /)
- `http://www.example.com:80/mypage`

Therefore it is important to specify access control policy that is based on the path information in some canonical form with all alternate encodings rejected (which can be accomplished by a default deny rule).

Relationships

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		863	Incorrect Authorization	1787

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

Applicable Platforms

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

Technology : Web Server (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>An attacker may be able to bypass the authorization mechanism to gain access to the otherwise-protected URL.</i>	
Confidentiality	Read Files or Directories <i>If a non-canonical URL is used, the server may choose to return the contents of the file, instead of pre-processing the file (e.g. as a program).</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

Make access control policy based on path information in canonical form. Use very restrictive regular expressions to validate that the path is in the expected form.

Phase: Architecture and Design

Reject all alternate path encodings that are not in the expected canonical form.

Demonstrative Examples

Example 1:

Example from CAPEC (CAPEC ID: 4, "Using Alternative IP Address Encodings"). An attacker identifies an application server that applies a security policy based on the domain and application name, so the access control policy covers authentication and authorization for anyone accessing `http://example.domain:8080/application`. However, by putting in the IP address of the host the application authentication and authorization controls may be bypassed `http://192.168.0.1:8080/application`. The attacker relies on the victim applying policy to the namespace abstraction and not having a default deny policy in place to manage exceptions.

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	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	2362
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2395
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS02-J		Canonicalize path names before validating them

CWE-648: Incorrect Use of Privileged APIs

Weakness ID : 648

Structure : Simple

Abstraction : Base

Description

The product does not conform to the API requirements for a function call that requires extra privileges. This could allow attackers to gain privileges by causing the function to be called incorrectly.

Extended Description

When a product contains certain functions that perform operations requiring an elevated level of privilege, the caller of a privileged API must be careful to:

- ensure that assumptions made by the APIs are valid, such as validity of arguments

- account for known weaknesses in the design/implementation of the API
- call the API from a safe context

If the caller of the API does not follow these requirements, then it may allow a malicious user or process to elevate their privilege, hijack the process, or steal sensitive data.

For instance, it is important to know if privileged APIs do not shed their privileges before returning to the caller or if the privileged function might make certain assumptions about the data, context or state information passed to it by the caller. It is important to always know when and how privileged APIs can be called in order to ensure that their elevated level of privilege cannot be exploited.

Relationships

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		269	Improper Privilege Management	646

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

Nature	Type	ID	Name	Page
MemberOf		265	Privilege Issues	2316

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity <i>An attacker may be able to elevate privileges.</i>	
Confidentiality	Read Application Data <i>An attacker may be able to obtain sensitive information.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>An attacker may be able to execute code.</i>	

Potential Mitigations

Phase: Implementation

Before calling privileged APIs, always ensure that the assumptions made by the privileged code hold true prior to making the call.

Phase: Architecture and Design

Know architecture and implementation weaknesses of the privileged APIs and make sure to account for these weaknesses before calling the privileged APIs to ensure that they can be called safely.

Phase: Implementation

If privileged APIs make certain assumptions about data, context or state validity that are passed by the caller, the calling code must ensure that these assumptions have been validated prior to making the call.

Phase: Implementation

If privileged APIs do not shed their privilege prior to returning to the calling code, then calling code needs to shed these privileges immediately and safely right after the call to the privileged APIs. In particular, the calling code needs to ensure that a privileged thread of execution will never be returned to the user or made available to user-controlled processes.

Phase: Implementation

Only call privileged APIs from safe, consistent and expected state.

Phase: Implementation

Ensure that a failure or an error will not leave a system in a state where privileges are not properly shed and privilege escalation is possible (i.e. fail securely with regards to handling of privileges).

Observed Examples

Reference	Description
CVE-2003-0645	A Unix utility that displays online help files, if installed setuid, could allow a local attacker to gain privileges when a particular file-opening function is called. https://www.cve.org/CVERecord?id=CVE-2003-0645

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
107	Cross Site Tracing
234	Hijacking a privileged process

CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking**Weakness ID :** 649**Structure :** Simple**Abstraction :** Base**Description**

The product uses obfuscation or encryption of inputs that should not be mutable by an external actor, but the product does not use integrity checks to detect if those inputs have been modified.

Extended Description

When an application relies on obfuscation or incorrectly applied / weak encryption to protect client-controllable tokens or parameters, that may have an effect on the user state, system state, or some decision made on the server. Without protecting the tokens/parameters for integrity, the application is vulnerable to an attack where an adversary traverses the space of possible values of the said token/parameter in order to attempt to gain an advantage. The goal of the attacker is to find another admissible value that will somehow elevate their privileges in the system, disclose information or change the behavior of the system in some way beneficial to the attacker. If the

application does not protect these critical tokens/parameters for integrity, it will not be able to determine that these values have been tampered with. Measures that are used to protect data for confidentiality should not be relied upon to provide the integrity service.

Relationships

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	851

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2434

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

Nature	Type	ID	Name	Page
MemberOf		1214	Data Integrity Issues	2477

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State <i>The inputs could be modified without detection, causing the product to have unexpected system state or make incorrect security decisions.</i>	

Potential Mitigations

Phase: Architecture and Design

Protect important client controllable tokens/parameters for integrity using PKI methods (i.e. digital signatures) or other means, and checks for integrity on the server side.

Phase: Architecture and Design

Repeated requests from a particular user that include invalid values of tokens/parameters (those that should not be changed manually by users) should result in the user account lockout.

Phase: Architecture and Design

Client side tokens/parameters should not be such that it would be easy/predictable to guess another valid state.

Phase: Architecture and Design

Obfuscation should not be relied upon. If encryption is used, it needs to be properly applied (i.e. proven algorithm and implementation, use padding, use random initialization vector, user proper encryption mode). Even with proper encryption where the ciphertext does not leak information about the plaintext or reveal its structure, compromising integrity is possible (although less likely) without the provision of the integrity service.

Observed Examples

Reference	Description
CVE-2005-0039	An IPSec configuration does not perform integrity checking of the IPSec packet as the result of either not configuring ESP properly to support the integrity service or using AH improperly. In either case, the security gateway receiving the IPSec packet would not validate the integrity of the packet to ensure that it was not changed. Thus if the packets were intercepted the attacker could undetectably change some of the bits in the packets. The meaningful bit flipping was possible due to the known weaknesses in the CBC encryption mode. Since the attacker knew the structure of the packet, they were able (in one variation of the attack) to use bit flipping to change the destination IP of the packet to the destination machine controlled by the attacker. And so the destination security gateway would decrypt the packet and then forward the plaintext to the machine controlled by the attacker. The attacker could then read the original message. For instance if VPN was used with the vulnerable IPSec configuration the attacker could read the victim's e-mail. This vulnerability demonstrates the need to enforce the integrity service properly when critical data could be modified by an attacker. This problem might have also been mitigated by using an encryption mode that is not susceptible to bit flipping attacks, but the preferred mechanism to address this problem still remains message verification for integrity. While this attack focuses on the network layer and requires an entity that controls part of the communication path such as a router, the situation is not much different at the software level, where an attacker can modify tokens/parameters used by the application. https://www.cve.org/CVERecord?id=CVE-2005-0039

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	975	SFP Secondary Cluster: Architecture	888	2406
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

CWE-650: Trusting HTTP Permission Methods on the Server Side

Weakness ID : 650

Structure : Simple

Abstraction : Variant

Description

The server contains a protection mechanism that assumes that any URI that is accessed using HTTP GET will not cause a state change to the associated resource. This might allow attackers to bypass intended access restrictions and conduct resource modification and deletion attacks, since some applications allow GET to modify state.

Extended Description

The HTTP GET method and some other methods are designed to retrieve resources and not to alter the state of the application or resources on the server side. Furthermore, the HTTP specification requires that GET requests (and other requests) should not have side effects.

Believing that it will be enough to prevent unintended resource alterations, an application may disallow the HTTP requests to perform DELETE, PUT and POST operations on the resource representation. However, there is nothing in the HTTP protocol itself that actually prevents the HTTP GET method from performing more than just query of the data. Developers can easily code programs that accept a HTTP GET request that do in fact create, update or delete data on the server. For instance, it is a common practice with REST based Web Services to have HTTP GET requests modifying resources on the server side. However, whenever that happens, the access control needs to be properly enforced in the application. No assumptions should be made that only HTTP DELETE, PUT, POST, and other methods have the power to alter the representation of the resource being accessed in the 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		436	Interpretation Conflict	1057

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity <i>An attacker could escalate privileges.</i>	
Integrity	Modify Application Data <i>An attacker could modify resources.</i>	
Confidentiality	Read Application Data <i>An attacker could obtain sensitive information.</i>	





Potential Mitigations

Phase: System Configuration

Configure ACLs on the server side to ensure that proper level of access control is defined for each accessible resource representation.

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	2394
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2524

Weakness ID : 651**Structure :** Simple**Abstraction :** Variant

Description

The Web services architecture may require exposing a Web Service Definition Language (WSDL) file that contains information on the publicly accessible services and how callers of these services should interact with them (e.g. what parameters they expect and what types they return).

Extended Description


An information exposure may occur if any of the following apply:

- The WSDL file is accessible to a wider audience than intended.
- The WSDL file contains information on the methods/services that should not be publicly accessible or information about deprecated methods. This problem is made more likely due to the WSDL often being automatically generated from the code.
- Information in the WSDL file helps guess names/locations of methods/resources that should not be publicly accessible.

Relationships

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		538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1248

Applicable Platforms

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

Technology : Web Server (*Prevalence = Often*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>The attacker may find sensitive information located in the WSDL file.</i>	

Potential Mitigations

Phase: Architecture and Design

Limit access to the WSDL file as much as possible. If services are provided only to a limited number of entities, it may be better to provide WSDL privately to each of these entities than to publish WSDL publicly.

Phase: Architecture and Design

Strategy = Separation of Privilege

Make sure that WSDL does not describe methods that should not be publicly accessible. Make sure to protect service methods that should not be publicly accessible with access controls.

Phase: Architecture and Design

Do not use method names in WSDL that might help an adversary guess names of private methods/resources used by the service.




Demonstrative Examples

Example 1:

The WSDL for a service providing information on the best price of a certain item exposes the following method: float getBestPrice(String ItemID) An attacker might guess that there is a method setBestPrice (String ItemID, float Price) that is available and invoke that method to try and change the best price of a given item to their advantage. The attack may succeed if the attacker correctly guesses the name of the method, the method does not have proper access controls around it and the service itself has the functionality to update the best price of the item.

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		966	SFP Secondary Cluster: Other Exposures	888	2403
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf		1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')

Weakness ID : 652

Structure : Simple

Abstraction : Base

Description

The product uses external input to dynamically construct an XQuery expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.



Extended Description

The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Relationships

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		91	XML Injection (aka Blind XPath Injection)	215
ChildOf		943	Improper Neutralization of Special Elements in Data Query Logic	1850

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2433

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>An attacker might be able to read sensitive information from the XML database.</i>	

Potential Mitigations

Phase: Implementation

Use parameterized queries. This will help ensure separation between data plane and control plane.

Phase: Implementation

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XQL queries is safe in that context.

Demonstrative Examples

Example 1:

An attacker may pass XQuery expressions embedded in an otherwise standard XML document. The attacker tunnels through the application entry point to target the resource access layer. The string below is an example of an attacker accessing the accounts.xml to request the service provider send all user names back. `doc(accounts.xml)//user[name=']` The attacks that are possible through XQuery are difficult to predict, if the data is not validated prior to executing the XQL.

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	929	OWASP Top Ten 2013 Category A1 - Injection	928	2389
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

Notes

Relationship

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	46		XQuery Injection
Software Fault Patterns	SFP24		Tainted input to command

CWE-653: Improper Isolation or Compartmentalization

Weakness ID : 653

Structure : Simple

Abstraction : Class

Description

The product does not properly compartmentalize or isolate functionality, processes, or resources that require different privilege levels, rights, or permissions.

Extended Description

When a weakness occurs in functionality that is accessible by lower-privileged users, then without strong boundaries, an attack might extend the scope of the damage to higher-privileged users.

Relationships

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	1520
ChildOf	ⓐ	657	Violation of Secure Design Principles	1446
ParentOf	ⓑ	1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)	1976
ParentOf	ⓑ	1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2225

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

Nature	Type	ID	Name	Page
MemberOf	Ⓒ	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	Ⓒ	1212	Authorization Errors	2476

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Separation of Privilege : Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. This node conflicts with the original definition of "Separation of Privilege" by Saltzer and Schroeder; that original definition is more closely associated with CWE-654. Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Scope	Impact	Likelihood
	<i>The exploitation of a weakness in low-privileged areas of the software can be leveraged to reach higher-privileged areas without having to overcome any additional obstacles.</i>	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Compare binary / bytecode to application permission manifest

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Break up privileges between different modules, objects, or entities. Minimize the interfaces between modules and require strong access control between them.

Demonstrative Examples

Example 1:

Single sign-on technology is intended to make it easier for users to access multiple resources or domains without having to authenticate each time. While this is highly convenient for the user and attempts to address problems with psychological acceptability, it also means that a compromise of a user's credentials can provide immediate access to all other resources or domains.

Example 2:

The traditional UNIX privilege model provides root with arbitrary access to all resources, but root is frequently the only user that has privileges. As a result, administrative tasks require root privileges, even if those tasks are limited to a small area, such as updating user manpages. Some UNIX flavors have a "bin" user that is the owner of system executables, but since root relies on executables owned by bin, a compromise of the bin account can be leveraged for root privileges by modifying a bin-owned executable, such as CVE-2007-4238.

Observed Examples

Reference	Description
CVE-2021-33096	Improper isolation of shared resource in a network-on-chip leads to denial of service https://www.cve.org/CVERecord?id=CVE-2021-33096
CVE-2019-6260	Baseboard Management Controller (BMC) device implements Advanced High-performance Bus (AHB) bridges that do not require authentication for arbitrary read and write access to the BMC's physical address space from the host, and possibly the network [REF-1138]. https://www.cve.org/CVERecord?id=CVE-2019-6260

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	901	SFP Primary Cluster: Privilege	888	2386
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

Relationship

There is a close association with CWE-250 (Execution with Unnecessary Privileges). CWE-653 is about providing separate components for each "privilege"; CWE-250 is about ensuring that each component has the least amount of privileges possible. In this fashion, compartmentalization becomes one mechanism for reducing privileges.

Terminology

The term "Separation of Privilege" is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (this node) and using only one factor in a security decision (CWE-654). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-535]Sean Barnum and Michael Gegick. "Separation of Privilege". 2005 December 6. < <https://web.archive.org/web/20220126060047/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/separation-of-privilege> >.2023-04-07.

[REF-1138]Stewart Smith. "CVE-2019-6260: Gaining control of BMC from the host processor". 2019. < <https://www.flamingspork.com/blog/2019/01/23/cve-2019-6260:-gaining-control-of-bmc-from-the-host-processor/> >.

CWE-654: Reliance on a Single Factor in a Security Decision

Weakness ID : 654

Structure : Simple

Abstraction : Base

Description

A protection mechanism relies exclusively, or to a large extent, on the evaluation of a single condition or the integrity of a single object or entity in order to make a decision about granting access to restricted resources or functionality.

Relationships

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	1520
ChildOf	ⓐ	657	Violation of Secure Design Principles	1446
ParentOf	ⓑ	308	Use of Single-factor Authentication	752
ParentOf	ⓑ	309	Use of Password System for Primary Authentication	754
PeerOf	ⓑ	1293	Missing Source Correlation of Multiple Independent Data	2149

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

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

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Separation of Privilege : Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. While this entry is closely associated with the original definition of "Separation of Privilege" by Saltzer and Schroeder, others use the same term to describe poor compartmentalization (CWE-653). Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity <i>If the single factor is compromised (e.g. by theft or spoofing), then the integrity of the entire security mechanism can be violated with respect to the user that is identified by that factor.</i>	
Non-Repudiation	Hide Activities <i>It can become difficult or impossible for the product to be able to distinguish between legitimate activities by the entity who provided the factor, versus illegitimate activities by an attacker.</i>	

Potential Mitigations

Phase: Architecture and Design

Use multiple simultaneous checks before granting access to critical operations or granting critical privileges. A weaker but helpful mitigation is to use several successive checks (multiple layers of security).

Phase: Architecture and Design

Use redundant access rules on different choke points (e.g., firewalls).

Demonstrative Examples

Example 1:

Password-only authentication is perhaps the most well-known example of use of a single factor. Anybody who knows a user's password can impersonate that user.

Example 2:




When authenticating, use multiple factors, such as "something you know" (such as a password) and "something you have" (such as a hardware-based one-time password generator, or a biometric device).

Observed Examples

Reference	Description
CVE-2022-35248	Chat application skips validation when Central Authentication Service (CAS) is enabled, effectively removing the second factor from two-factor authentication https://www.cve.org/CVERecord?id=CVE-2022-35248

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		975	SFP Secondary Cluster: Architecture	888	2406
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes**Maintenance**

This entry is closely associated with the term "Separation of Privilege." This term is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (CWE-653) and using only one factor in a security decision (this entry). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SD-3
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-1		Req SI-1

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
16	Dictionary-based Password Attack
49	Password Brute Forcing
55	Rainbow Table Password Cracking
70	Try Common or Default Usernames and Passwords
274	HTTP Verb Tampering
560	Use of Known Domain Credentials
565	Password Spraying
600	Credential Stuffing
652	Use of Known Kerberos Credentials
653	Use of Known Operating System Credentials

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-535]Sean Barnum and Michael Gegick. "Separation of Privilege". 2005 December 6. <
<https://web.archive.org/web/20220126060047/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/separation-of-privilege> >.2023-04-07.

CWE-655: Insufficient Psychological Acceptability

Weakness ID : 655

Structure : Simple

Abstraction : Class

Description

The product has a protection mechanism that is too difficult or inconvenient to use, encouraging non-malicious users to disable or bypass the mechanism, whether by accident or on purpose.

Relationships

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	1520
ChildOf	⬢	657	Violation of Secure Design Principles	1446

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>By bypassing the security mechanism, a user might leave the system in a less secure state than intended by the administrator, making it more susceptible to compromise.</i>	

Potential Mitigations

Phase: Testing

Where possible, perform human factors and usability studies to identify where your product's security mechanisms are difficult to use, and why.

Phase: Architecture and Design

Make the security mechanism as seamless as possible, while also providing the user with sufficient details when a security decision produces unexpected results.

Demonstrative Examples

Example 1:

In "Usability of Security: A Case Study" [REF-540], the authors consider human factors in a cryptography product. Some of the weakness relevant discoveries of this case study were: users accidentally leaked sensitive information, could not figure out how to perform some tasks, thought they were enabling a security option when they were not, and made improper trust decisions.

Example 2:

Enforcing complex and difficult-to-remember passwords that need to be frequently changed for access to trivial resources, e.g., to use a black-and-white printer. Complex password requirements can also cause users to store the passwords in an unsafe manner so they don't have to remember them, such as using a sticky note or saving them in an unencrypted file.

Example 3:

Some CAPTCHA utilities produce images that are too difficult for a human to read, causing user frustration.

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		995	SFP Secondary Cluster: Feature	888	2418
MemberOf		1379	ICS Operations (& Maintenance): Human factors in ICS environments	1358	2514
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

Other

This weakness covers many security measures causing user inconvenience, requiring effort or causing frustration, that are disproportionate to the risks or value of the protected assets, or that are perceived to be ineffective.

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
ISA/IEC 62443	Part 2-1		Req 4.3.3.6
ISA/IEC 62443	Part 4-1		Req SD-4

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-539]Sean Barnum and Michael Gegick. "Psychological Acceptability". 2005 September 5. < <https://web.archive.org/web/20221104163022/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/psychological-acceptability> >.2023-04-07.

[REF-540]J. D. Tygar and Alma Whitten. "Usability of Security: A Case Study". SCS Technical Report Collection, CMU-CS-98-155. 1998 December 5. < <http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-155.pdf> >.

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

CWE-656: Reliance on Security Through Obscurity

Weakness ID : 656

Structure : Simple

Abstraction : Class

Description

The product uses a protection mechanism whose strength depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to defeat the mechanism.







Extended Description

This reliance on "security through obscurity" can produce resultant weaknesses if an attacker is able to reverse engineer the inner workings of the mechanism. Note that obscurity can be one small part of defense in depth, since it can create more work for an attacker; however, it is a significant risk if used as the primary means of protection.

Relationships

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	1520
ChildOf		657	Violation of Secure Design Principles	1446
PeerOf		603	Use of Client-Side Authentication	1354
CanPrecede		259	Use of Hard-coded Password	623
CanPrecede		321	Use of Hard-coded Cryptographic Key	785
CanPrecede		472	External Control of Assumed-Immutable Web Parameter	1123

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Never Assuming your secrets are safe :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Other <i>The security mechanism can be bypassed easily.</i>	
Integrity		
Availability		
Other		

Potential Mitigations

Phase: Architecture and Design

Always consider whether knowledge of your code or design is sufficient to break it. Reverse engineering is a highly successful discipline, and financially feasible for motivated adversaries. Black-box techniques are established for binary analysis of executables that use obfuscation, runtime analysis of proprietary protocols, inferring file formats, and others.

Phase: Architecture and Design

When available, use publicly-vetted algorithms and procedures, as these are more likely to undergo more extensive security analysis and testing. This is especially the case with encryption and authentication.

Demonstrative Examples

Example 1:




The design of TCP relies on the secrecy of Initial Sequence Numbers (ISNs), as originally covered in CVE-1999-0077 [REF-542]. If ISNs can be guessed (due to predictability, CWE-330) or sniffed (due to lack of encryption during transmission, CWE-312), then an attacker can hijack or spoof connections. Many TCP implementations have had variations of this problem over the years, including CVE-2004-0641, CVE-2002-1463, CVE-2001-0751, CVE-2001-0328, CVE-2001-0288, CVE-2001-0163, CVE-2001-0162, CVE-2000-0916, and CVE-2000-0328.

Observed Examples

Reference	Description
CVE-2006-6588	Reliance on hidden form fields in a web application. Many web application vulnerabilities exist because the developer did not consider that "hidden" form fields can be processed using a modified client. https://www.cve.org/CVERecord?id=CVE-2006-6588
CVE-2006-7142	Hard-coded cryptographic key stored in executable program. https://www.cve.org/CVERecord?id=CVE-2006-7142
CVE-2005-4002	Hard-coded cryptographic key stored in executable program. https://www.cve.org/CVERecord?id=CVE-2005-4002
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks. https://www.cve.org/CVERecord?id=CVE-2006-4068

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		975	SFP Secondary Cluster: Architecture	888	2406
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

Relationship

Note that there is a close relationship between this weakness and CWE-603 (Use of Client-Side Authentication). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-544]Sean Barnum and Michael Gegick. "Never Assuming that Your Secrets Are Safe". 2005 September 4. < <https://web.archive.org/web/20220126060054/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/never-assuming-that-your-secrets-are-safe> >.2023-04-07.

[REF-542]Jon Postel, Editor. "RFC: 793, TRANSMISSION CONTROL PROTOCOL". 1981 September. Information Sciences Institute. < <https://www.ietf.org/rfc/rfc0793.txt> >.2023-04-07.

CWE-657: Violation of Secure Design Principles

Weakness ID : 657

Structure : Simple

Abstraction : Class

Description

The product violates well-established principles for secure design.

Extended Description

This can introduce resultant weaknesses or make it easier for developers to introduce related weaknesses during implementation. Because code is centered around design, it can be resource-intensive to fix design problems.

Relationships

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]	710	Improper Adherence to Coding Standards	1549
ParentOf	[B]	250	Execution with Unnecessary Privileges	599
ParentOf	[C]	636	Not Failing Securely ('Failing Open')	1401
ParentOf	[C]	637	Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')	1403
ParentOf	[C]	638	Not Using Complete Mediation	1404
ParentOf	[C]	653	Improper Isolation or Compartmentalization	1437
ParentOf	[B]	654	Reliance on a Single Factor in a Security Decision	1439
ParentOf	[C]	655	Insufficient Psychological Acceptability	1442
ParentOf	[C]	656	Reliance on Security Through Obscurity	1444
ParentOf	[C]	671	Lack of Administrator Control over Security	1478
ParentOf	[B]	1192	Improper Identifier for IP Block used in System-On-Chip (SOC)	1985
ParentOf	[C]	1395	Dependency on Vulnerable Third-Party Component	2277

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Demonstrative Examples

Example 1:

Switches may revert their functionality to that of hubs when the table used to map ARP information to the switch interface overflows, such as when under a spoofing attack. This results in traffic being broadcast to an eavesdropper, instead of being sent only on the relevant switch interface.

To mitigate this type of problem, the developer could limit the number of ARP entries that can

be recorded for a given switch interface, while other interfaces may keep functioning normally. Configuration options can be provided on the appropriate actions to be taken in case of a detected failure, but safe defaults should be used.

Example 2:

The IPSEC specification is complex, which resulted in bugs, partial implementations, and incompatibilities between vendors.

Example 3:

When executable library files are used on web servers, which is common in PHP applications, the developer might perform an access check in any user-facing executable, and omit the access check from the library file itself. By directly requesting the library file (CWE-425), an attacker can bypass this access check.

Example 4:

Single sign-on technology is intended to make it easier for users to access multiple resources or domains without having to authenticate each time. While this is highly convenient for the user and attempts to address problems with psychological acceptability, it also means that a compromise of a user's credentials can provide immediate access to all other resources or domains.

Example 5:

The design of TCP relies on the secrecy of Initial Sequence Numbers (ISNs), as originally covered in CVE-1999-0077 [REF-542]. If ISNs can be guessed (due to predictability, CWE-330) or sniffed (due to lack of encryption during transmission, CWE-312), then an attacker can hijack or spoof connections. Many TCP implementations have had variations of this problem over the years, including CVE-2004-0641, CVE-2002-1463, CVE-2001-0751, CVE-2001-0328, CVE-2001-0288, CVE-2001-0163, CVE-2001-0162, CVE-2000-0916, and CVE-2000-0328.

Example 6:

The "SweynTooth" vulnerabilities in Bluetooth Low Energy (BLE) software development kits (SDK) were found to affect multiple Bluetooth System-on-Chip (SoC) manufacturers. These SoCs were used by many products such as medical devices, Smart Home devices, wearables, and other IoT devices. [REF-1314] [REF-1315]

Observed Examples

Reference	Description
CVE-2019-6260	Baseboard Management Controller (BMC) device implements Advanced High-performance Bus (AHB) bridges that do not require authentication for arbitrary read and write access to the BMC's physical address space from the host, and possibly the network [REF-1138]. https://www.cve.org/CVERecord?id=CVE-2019-6260
CVE-2007-5277	The failure of connection attempts in a web browser resets DNS pin restrictions. An attacker can then bypass the same origin policy by rebinding a domain name to a different IP address. This was an attempt to "fail functional." https://www.cve.org/CVERecord?id=CVE-2007-5277
CVE-2006-7142	Hard-coded cryptographic key stored in executable program. https://www.cve.org/CVERecord?id=CVE-2006-7142
CVE-2007-0408	Server does not properly validate client certificates when reusing cached connections. https://www.cve.org/CVERecord?id=CVE-2007-0408

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	975	SFP Secondary Cluster: Architecture	888	2406
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

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
ISA/IEC 62443	Part 4-1		Req SD-3
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-1		Req SI-1

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < <http://web.mit.edu/Saltzer/www/publications/protection/> >.

[REF-546]Sean Barnum and Michael Gegick. "Design Principles". 2005 September 9. < <https://web.archive.org/web/20220126060046/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/design-principles> >.2023-04-07.

[REF-542]Jon Postel, Editor. "RFC: 793, TRANSMISSION CONTROL PROTOCOL". 1981 September. Information Sciences Institute. < <https://www.ietf.org/rfc/rfc0793.txt> >.2023-04-07.

[REF-1138]Stewart Smith. "CVE-2019-6260: Gaining control of BMC from the host processor". 2019. < <https://www.flamingspork.com/blog/2019/01/23/cve-2019-6260:-gaining-control-of-bmc-from-the-host-processor/> >.

[REF-1314]ICS-CERT. "ICS Alert (ICS-ALERT-20-063-01): SweynTooth Vulnerabilities". 2020 March 4. < <https://www.cisa.gov/news-events/ics-alerts/ics-alert-20-063-01> >.2023-04-07.

[REF-1315]Matheus E. Garbelini, Sudipta Chattopadhyay, Chundong Wang, Singapore University of Technology and Design. "Unleashing Mayhem over Bluetooth Low Energy". 2020 March 4. < <https://asset-group.github.io/disclosures/sweyntooth/> >.2023-01-25.

CWE-662: Improper Synchronization

Weakness ID : 662

Structure : Simple

Abstraction : Class

Description

The product utilizes multiple threads or processes to allow temporary access to a shared resource that can only be exclusive to one process at a time, but it does not properly synchronize these actions, which might cause simultaneous accesses of this resource by multiple threads or processes.

Extended Description

Synchronization refers to a variety of behaviors and mechanisms that allow two or more independently-operating processes or threads to ensure that they operate on shared resources in predictable ways that do not interfere with each other. Some shared resource operations cannot be executed atomically; that is, multiple steps must be guaranteed to execute sequentially, without any interference by other processes. Synchronization mechanisms vary widely, but they may include locking, mutexes, and semaphores. When a multi-step operation on a shared resource cannot be guaranteed to execute independent of interference, then the resulting behavior can be unpredictable. Improper synchronization could lead to data or memory corruption, denial of service, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	1517
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1452
ParentOf	C	667	Improper Locking	1464
ParentOf	B	820	Missing Synchronization	1720
ParentOf	B	821	Incorrect Synchronization	1722
ParentOf	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1893
CanPrecede	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888










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

Nature	Type	ID	Name	Page
ParentOf	C	667	Improper Locking	1464

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

Nature	Type	ID	Name	Page
ParentOf	B	366	Race Condition within a Thread	904
ParentOf	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1255
ParentOf	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1288
ParentOf	C	667	Improper Locking	1464
ParentOf	B	764	Multiple Locks of a Critical Resource	1604
ParentOf	B	820	Missing Synchronization	1720
ParentOf	B	821	Incorrect Synchronization	1722
ParentOf	B	833	Deadlock	1753
ParentOf	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1893
ParentOf	V	1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1936

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

Nature	Type	ID	Name	Page
ParentOf		366	Race Condition within a Thread	904
ParentOf		543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1255
ParentOf		567	Unsynchronized Access to Shared Data in a Multithreaded Context	1288
ParentOf		667	Improper Locking	1464
ParentOf		764	Multiple Locks of a Critical Resource	1604
ParentOf		820	Missing Synchronization	1720
ParentOf		821	Incorrect Synchronization	1722
ParentOf		1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1893
ParentOf		1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1936

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	
Other	Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Use industry standard APIs to synchronize your code.

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:

The following code intends to fork a process, then have both the parent and child processes print a single line.

Example Language: C

(Bad)

```
static void print (char * string) {
    char * word;
    int counter;
    for (word = string; counter = *word++; ) {
        putc(counter, stdout);
        fflush(stdout);
        /* Make timing window a little larger... */
        sleep(1);
    }
}

int main(void) {
    pid_t pid;
    pid = fork();
    if (pid == -1) {
        exit(-2);
    }
    else if (pid == 0) {
        print("child\n");
    }
    else {
        print("PARENT\n");
    }
    exit(0);
}
```

One might expect the code to print out something like:

PARENT
child

However, because the parent and child are executing concurrently, and stdout is flushed each time a character is printed, the output might be mixed together, such as:

PcAhRiEINdT
[blank line]
[blank line]

Observed Examples

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-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice https://www.cve.org/CVERecord?id=CVE-2009-0935

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	745	CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)	734	2349
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2366

Nature	Type	ID	Name	V	Page
MemberOf	C	879	CERT C++ Secure Coding Section 11 - Signals (SIG)	868	2379
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2448
MemberOf	C	1166	SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)	1154	2460
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG00-C		Mask signals handled by noninterruptible signal handlers
CERT C Secure Coding	SIG31-C	CWE More Abstract	Do not access shared objects in signal handlers
CLASP			State synchronization error
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
25	Forced Deadlock
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

CWE-663: Use of a Non-reentrant Function in a Concurrent Context

Weakness ID : 663

Structure : Simple

Abstraction : Base

Description





The product calls a non-reentrant function in a concurrent context in which a competing code sequence (e.g. thread or signal handler) may have an opportunity to call the same function or otherwise influence its state.

Relationships

1452

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		662	Improper Synchronization	1448
ParentOf		479	Signal Handler Use of a Non-reentrant Function	1147
ParentOf		558	Use of getlogin() in Multithreaded Application	1272
PeerOf		1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	2088

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2329

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Confidentiality	Read Memory	
Other	Modify Application Data	
	Read Application Data	
	Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Use reentrant functions if available.

Phase: Implementation

Add synchronization to your non-reentrant function.

Phase: Implementation

In Java, use the ReentrantLock Class.

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.

Example 2:

The following code relies on `getlogin()` to determine whether or not a user is trusted. It is easily subverted.

Example Language: C

(Bad)

```
pwd = getpwnam(getlogin());
if (isTrustedGroup(pwd->pw_gid)) {
    allow();
} else {
    deny();
}
```

Observed Examples

Reference	Description
CVE-2001-1349	unsafe calls to library functions from signal handler https://www.cve.org/CVERecord?id=CVE-2001-1349
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

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	2411
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Related Attack Patterns

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

References

[REF-547]SUN. "Java Concurrency API". < <https://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/locks/ReentrantLock.html> >.2023-04-07.

[REF-548]Dipak Jha, Software Engineer, IBM. "Use reentrant functions for safer signal handling". < <https://archive.ph/r11XR> >.2023-04-07.

CWE-664: Improper Control of a Resource Through its Lifetime

Weakness ID : 664

Structure : Simple

Abstraction : Pillar

Description

The product does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release.

Extended Description



















Resources often have explicit instructions on how to be created, used and destroyed. When code does not follow these instructions, it can lead to unexpected behaviors and potentially exploitable states.

Even without explicit instructions, various principles are expected to be adhered to, such as "Do not use an object until after its creation is complete," or "do not use an object after it has been slated for destruction."

Relationships

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	2575
ParentOf		118	Incorrect Access of Indexable Resource ('Range Error')	292
ParentOf		221	Information Loss or Omission	556
ParentOf		372	Incomplete Internal State Distinction	919
ParentOf		400	Uncontrolled Resource Consumption	964
ParentOf		404	Improper Resource Shutdown or Release	980
ParentOf		410	Insufficient Resource Pool	998
ParentOf		471	Modification of Assumed-Immutable Data (MAID)	1121
ParentOf		487	Reliance on Package-level Scope	1167
ParentOf		495	Private Data Structure Returned From A Public Method	1189
ParentOf		496	Public Data Assigned to Private Array-Typed Field	1192
ParentOf		501	Trust Boundary Violation	1203
ParentOf		580	clone() Method Without super.clone()	1311
ParentOf		610	Externally Controlled Reference to a Resource in Another Sphere	1364
ParentOf		662	Improper Synchronization	1448
ParentOf		665	Improper Initialization	1456
ParentOf		666	Operation on Resource in Wrong Phase of Lifetime	1462
ParentOf		668	Exposure of Resource to Wrong Sphere	1469
ParentOf		669	Incorrect Resource Transfer Between Spheres	1471
ParentOf		673	External Influence of Sphere Definition	1483
ParentOf		704	Incorrect Type Conversion or Cast	1538
ParentOf		706	Use of Incorrectly-Resolved Name or Reference	1544
ParentOf		911	Improper Update of Reference Count	1801
ParentOf		913	Improper Control of Dynamically-Managed Code Resources	1805
ParentOf		922	Insecure Storage of Sensitive Information	1825
ParentOf		1229	Creation of Emergent Resource	2006
ParentOf		1250	Improper Preservation of Consistency Between Independent Representations of Shared State	2052
ParentOf		1329	Reliance on Component That is Not Updateable	2219

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Testing

Use Static analysis tools to check for unreleased resources.

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-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

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	984	SFP Secondary Cluster: Life Cycle	888	2411
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1370	ICS Supply Chain: Common Mode Frailties	1358	2507
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

More work is needed on this entry and its children. There are perspective/layering issues; for example, one breakdown is based on lifecycle phase (CWE-404, CWE-665), while other children are independent of lifecycle, such as CWE-400. Others do not specify as many bases or variants, such as CWE-704, which primarily covers numbers at this stage.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO39-C	CWE More Abstract	Do not alternately input and output from a stream without an intervening flush or positioning call

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery
196	Session Credential Falsification through Forging

CWE-665: Improper Initialization

Weakness ID : 665

Structure : Simple

Abstraction : Class

Description

The product does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.

Extended Description

This can have security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not.

Relationships

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	1454
ParentOf	B	455	Non-exit on Failed Initialization	1087
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1613
ParentOf	B	908	Use of Uninitialized Resource	1792
ParentOf	C	909	Missing Initialization of Resource	1797
ParentOf	B	1279	Cryptographic Operations are run Before Supporting Units are Ready	2120
ParentOf	C	1419	Incorrect Initialization of Resource	2280

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

Nature	Type	ID	Name	Page
ParentOf	B	908	Use of Uninitialized Resource	1792
ParentOf	C	909	Missing Initialization of Resource	1797
ParentOf	B	1188	Initialization of a Resource with an Insecure Default	1974

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

Nature	Type	ID	Name	Page
ParentOf	V	456	Missing Initialization of a Variable	1089
ParentOf	V	457	Use of Uninitialized Variable	1094

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

Nature	Type	ID	Name	Page
ParentOf	V	456	Missing Initialization of a Variable	1089
ParentOf	V	457	Use of Uninitialized Variable	1094

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Scope	Impact	Likelihood
Access Control	Read Application Data <i>When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.</i>	
	Bypass Protection Mechanism <i>If security-critical decisions rely on a variable having a "0" or equivalent value, and the programming language performs this initialization on behalf of the programmer, then a bypass of security may occur.</i>	
Availability	DoS: Crash, Exit, or Restart <i>The uninitialized data may contain values that cause program flow to change in ways that the programmer did not intend. For example, if an uninitialized variable is used as an array index in C, then its previous contents may produce an index that is outside the range of the array, possibly causing a crash or an exit in other environments.</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. Initialization problems may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

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: 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, in Java, if the programmer does not explicitly

initialize a variable, then the code could produce a compile-time error (if the variable is local) or automatically initialize the variable to the default value for the variable's type. In Perl, if explicit initialization is not performed, then a default value of undef is assigned, which is interpreted as 0, false, or an equivalent value depending on the context in which the variable is accessed.

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

Phase: Implementation

Pay close attention to complex conditionals that affect initialization, since some conditions might not perform the initialization.

Phase: Implementation

Avoid race conditions (CWE-362) during initialization routines.

Phase: Build and Compilation

Run or compile your product with settings that generate warnings about uninitialized variables or data.

Phase: Testing

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

Demonstrative Examples**Example 1:**

Here, a boolean initialized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Example Language: Java

(Bad)

```
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
        ...
        initialized = true;
    }
}
```

Example 2:

The following code intends to limit certain operations to the administrator only.

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.

Example 3:

The following code intends to concatenate a string to a variable and print the string.

Example Language: C

(Bad)

```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.














Observed Examples

Reference	Description
CVE-2001-1471	chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection. https://www.cve.org/CVERecord?id=CVE-2001-1471
CVE-2008-3637	Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution. https://www.cve.org/CVERecord?id=CVE-2008-3637
CVE-2008-4197	Use of uninitialized memory may allow code execution. https://www.cve.org/CVERecord?id=CVE-2008-4197
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution. https://www.cve.org/CVERecord?id=CVE-2008-2934
CVE-2007-3749	OS kernel does not reset a port when starting a setuid program, allowing local users to access the port and gain privileges. https://www.cve.org/CVERecord?id=CVE-2007-3749
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak. https://www.cve.org/CVERecord?id=CVE-2008-0063
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free. https://www.cve.org/CVERecord?id=CVE-2008-0062
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. https://www.cve.org/CVERecord?id=CVE-2008-0081
CVE-2008-3688	chain: Uninitialized variable leads to infinite loop. https://www.cve.org/CVERecord?id=CVE-2008-3688
CVE-2008-3475	chain: Improper initialization leads to memory corruption. https://www.cve.org/CVERecord?id=CVE-2008-3475

Reference	Description
CVE-2008-5021	Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory. https://www.cve.org/CVERecord?id=CVE-2008-5021
CVE-2005-1036	Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap https://www.cve.org/CVERecord?id=CVE-2005-1036
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-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-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

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		740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	2344
MemberOf		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf		752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf		846	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL)	844	2362
MemberOf		874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	2375
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1135	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL)	1133	2444
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf		1308	CISQ Quality Measures - Security	1305	2485
MemberOf		1340	CISQ Data Protection Measures	1340	2590
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incorrect initialization
CERT C Secure Coding	ARR02-C		Explicitly specify array bounds, even if implicitly defined by an initializer

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	DCL00-J		Prevent class initialization cycles
Software Fault Patterns	SFP4		Unchecked Status 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-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-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-666: Operation on Resource in Wrong Phase of Lifetime

Weakness ID : 666

Structure : Simple

Abstraction : Class

Description

The product performs an operation on a resource at the wrong phase of the resource's lifecycle, which can lead to unexpected behaviors.







Extended Description

A resource's lifecycle includes several phases: initialization, use, and release. For each phase, it is important to follow the specifications outlined for how to operate on the resource and to ensure that the resource is in the expected phase. Otherwise, if a resource is in one phase but the operation is not valid for that phase (i.e., an incorrect phase of the resource's lifetime), then this can produce resultant weaknesses. For example, using a resource before it has been fully initialized could cause corruption or incorrect data to be 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		664	Improper Control of a Resource Through its Lifetime	1454
ParentOf		415	Double Free	1008
ParentOf		593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	1331
ParentOf		605	Multiple Binds to the Same Port	1356
ParentOf		672	Operation on a Resource after Expiration or Release	1479
ParentOf		826	Premature Release of Resource During Expected Lifetime	1734

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Follow the resource's lifecycle from creation to release.

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.

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

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	984	SFP Secondary Cluster: Life Cycle	888	2411
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory

CWE-667: Improper Locking

Weakness ID : 667

Structure : Simple

Abstraction : Class

Description

The product does not properly acquire or release a lock on a resource, leading to unexpected resource state changes and behaviors.













Extended Description

Locking is a type of synchronization behavior that ensures that multiple independently-operating processes or threads do not interfere with each other when accessing the same resource. All processes/threads are expected to follow the same steps for locking. If these steps are not followed precisely - or if no locking is done at all - then another process/thread could modify the shared resource in a way that is not visible or predictable to the original process. This can lead to data or memory corruption, denial of service, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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		662	Improper Synchronization	1448
ParentOf		412	Unrestricted Externally Accessible Lock	1000
ParentOf		413	Improper Resource Locking	1003
ParentOf		414	Missing Lock Check	1007
ParentOf		609	Double-Checked Locking	1362
ParentOf		764	Multiple Locks of a Critical Resource	1604
ParentOf		765	Multiple Unlocks of a Critical Resource	1605
ParentOf		832	Unlock of a Resource that is not Locked	1752
ParentOf		833	Deadlock	1753
ParentOf		1232	Improper Lock Behavior After Power State Transition	2010
ParentOf		1233	Security-Sensitive Hardware Controls with Missing Lock Bit Protection	2012
ParentOf		1234	Hardware Internal or Debug Modes Allow Override of Locks	2014

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1448

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) <i>Inconsistent locking discipline can lead to deadlock.</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 = Libraries or Frameworks

Use industry standard APIs to implement locking mechanism.

Demonstrative Examples

Example 1:

In the following Java snippet, methods are defined to get and set a long field in an instance of a class that is shared across multiple threads. Because operations on double and long are nonatomic in Java, concurrent access may cause unexpected behavior. Thus, all operations on long and double fields should be synchronized.

Example Language: Java

(Bad)

```
private long someLongValue;
public long getLongValue() {
    return someLongValue;
}
public void setLongValue(long l) {
    someLongValue = l;
}
```

Example 2:

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().

Example 3:

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 4:

It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...

Example Language: Java

(Bad)

```
if (helper == null) {  
    synchronized (this) {  
        if (helper == null) {  
            helper = new Helper();  
        }  
    }  
}  
return helper;
```

The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

Suppose that helper is not initialized. Then, thread A sees that helper==null and enters the synchronized block and begins to execute:

Example Language:

(Bad)

```
helper = new Helper();
```

If a second thread, thread B, takes over in the middle of this call and helper has not finished running the constructor, then thread B may make calls on helper while its fields hold incorrect values.












Observed Examples

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-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice https://www.cve.org/CVERecord?id=CVE-2009-0935
CVE-2010-4210	function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory. https://www.cve.org/CVERecord?id=CVE-2010-4210
CVE-2008-4302	Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash. https://www.cve.org/CVERecord?id=CVE-2008-4302
CVE-2009-1243	OS kernel performs an unlock in some incorrect circumstances, leading to panic. https://www.cve.org/CVERecord?id=CVE-2009-1243
CVE-2009-2857	OS deadlock https://www.cve.org/CVERecord?id=CVE-2009-2857
CVE-2009-1961	OS deadlock involving 3 separate functions https://www.cve.org/CVERecord?id=CVE-2009-1961
CVE-2009-2699	deadlock in library https://www.cve.org/CVERecord?id=CVE-2009-2699
CVE-2009-4272	deadlock triggered by packets that force collisions in a routing table https://www.cve.org/CVERecord?id=CVE-2009-4272
CVE-2002-1850	read/write deadlock between web server and script https://www.cve.org/CVERecord?id=CVE-2002-1850
CVE-2004-0174	web server deadlock involving multiple listening connections https://www.cve.org/CVERecord?id=CVE-2004-0174
CVE-2009-1388	multiple simultaneous calls to the same function trigger deadlock. https://www.cve.org/CVERecord?id=CVE-2009-1388
CVE-2006-5158	chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833). https://www.cve.org/CVERecord?id=CVE-2006-5158
CVE-2006-4342	deadlock when an operation is performed on a resource while it is being removed. https://www.cve.org/CVERecord?id=CVE-2006-4342
CVE-2006-2374	Deadlock in device driver triggered by using file handle of a related device. https://www.cve.org/CVERecord?id=CVE-2006-2374
CVE-2006-2275	Deadlock when large number of small messages cannot be processed quickly enough. https://www.cve.org/CVERecord?id=CVE-2006-2275
CVE-2005-3847	OS kernel has deadlock triggered by a signal during a core dump. https://www.cve.org/CVERecord?id=CVE-2005-3847
CVE-2005-3106	Race condition leads to deadlock. https://www.cve.org/CVERecord?id=CVE-2005-3106

Reference	Description
CVE-2005-2456	Chain: array index error (CWE-129) leads to deadlock (CWE-833) https://www.cve.org/CVERecord?id=CVE-2005-2456
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		748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2351
MemberOf		852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2366
MemberOf		853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2366
MemberOf		884	CWE Cross-section	884	2567
MemberOf		986	SFP Secondary Cluster: Missing Lock	888	2411
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf		1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2448
MemberOf		1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	2449
MemberOf		1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2462
MemberOf		1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2463
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662

and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	CON31-C	CWE More Abstract	Do not destroy a mutex while it is locked
CERT C Secure Coding	POS48-C	CWE More Abstract	Do not unlock or destroy another POSIX thread's mutex
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)	VNA05-J		Ensure atomicity when reading and writing 64-bit values
The CERT Oracle Secure Coding Standard for Java (2011)	LCK06-J		Do not use an instance lock to protect shared static data
Software Fault Patterns	SFP19		Missing Lock
OMG ASCSM	ASCSM-CWE-667		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links

References

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

CWE-668: Exposure of Resource to Wrong Sphere

Weakness ID : 668

Structure : Simple

Abstraction : Class

Description

The product exposes a resource to the wrong control sphere, providing unintended actors with inappropriate access to the resource.

Extended Description

Resources such as files and directories may be inadvertently exposed through mechanisms such as insecure permissions, or when a program accidentally operates on the wrong object. For example, a program may intend that private files can only be provided to a specific user. This effectively defines a control sphere that is intended to prevent attackers from accessing these private files. If the file permissions are insecure, then parties other than the user will be able to access those files.

A separate control sphere might effectively require that the user can only access the private files, but not any other files on the system. If the program does not ensure that the user is only requesting private files, then the user might be able to access other files on the system.

In either case, the end result is that a resource has been exposed to the wrong party.

Relationships

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	1454
ParentOf	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	6
ParentOf	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
ParentOf	B	134	Use of Externally-Controlled Format String	365
ParentOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	504
ParentOf	B	374	Passing Mutable Objects to an Untrusted Method	920
ParentOf	B	375	Returning a Mutable Object to an Untrusted Caller	923
ParentOf	C	377	Insecure Temporary File	925
ParentOf	C	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	976
ParentOf	B	427	Uncontrolled Search Path Element	1033
ParentOf	B	428	Unquoted Search Path or Element	1039
ParentOf	B	488	Exposure of Data Element to Wrong Session	1169
ParentOf	V	491	Public cloneable() Method Without Final ('Object Hijack')	1174
ParentOf	V	492	Use of Inner Class Containing Sensitive Data	1175
ParentOf	V	493	Critical Public Variable Without Final Modifier	1182
ParentOf	V	498	Cloneable Class Containing Sensitive Information	1196
ParentOf	V	499	Serializable Class Containing Sensitive Data	1198
ParentOf	C	522	Insufficiently Protected Credentials	1225
ParentOf	B	524	Use of Cache Containing Sensitive Information	1232
ParentOf	B	552	Files or Directories Accessible to External Parties	1265
ParentOf	V	582	Array Declared Public, Final, and Static	1314
ParentOf	V	583	finalize() Method Declared Public	1315
ParentOf	V	608	Struts: Non-private Field in ActionForm Class	1361
ParentOf	C	642	External Control of Critical State Data	1414
ParentOf	C	732	Incorrect Permission Assignment for Critical Resource	1551
ParentOf	B	767	Access to Critical Private Variable via Public Method	1610
ParentOf	V	927	Use of Implicit Intent for Sensitive Communication	1836
ParentOf	B	1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)	1976
ParentOf	B	1282	Assumed-Immutable Data is Stored in Writable Memory	2127
ParentOf	B	1327	Binding to an Unrestricted IP Address	2215
ParentOf	B	1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2225
CanFollow	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1064
CanFollow	V	942	Permissive Cross-domain Policy with Untrusted Domains	1847

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

Nature	Type	ID	Name	Page
ParentOf	B	134	Use of Externally-Controlled Format String	365
ParentOf	B	426	Untrusted Search Path	1028
ParentOf	B	427	Uncontrolled Search Path Element	1033
ParentOf	B	428	Unquoted Search Path or Element	1039
ParentOf	B	552	Files or Directories Accessible to External Parties	1265

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	
Other	Other	

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	963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Theoretical

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

References

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

CWE-669: Incorrect Resource Transfer Between Spheres

Weakness ID : 669

Structure : Simple

Abstraction : Class









Description

The product does not properly transfer a resource/behavior to another sphere, or improperly imports a resource/behavior from another sphere, in a manner that provides unintended control over that resource.






Relationships

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	1454
ParentOf		212	Improper Removal of Sensitive Information Before Storage or Transfer	544
ParentOf		243	Creation of chroot Jail Without Changing Working Directory	589
ParentOf		434	Unrestricted Upload of File with Dangerous Type	1048
ParentOf		494	Download of Code Without Integrity Check	1185
ParentOf		829	Inclusion of Functionality from Untrusted Control Sphere	1741
ParentOf		1420	Exposure of Sensitive Information during Transient Execution	2284
CanFollow		244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	591

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

Nature	Type	ID	Name	Page
ParentOf		212	Improper Removal of Sensitive Information Before Storage or Transfer	544
ParentOf		434	Unrestricted Upload of File with Dangerous Type	1048
ParentOf		494	Download of Code Without Integrity Check	1185
ParentOf		565	Reliance on Cookies without Validation and Integrity Checking	1283
ParentOf		829	Inclusion of Functionality from Untrusted Control Sphere	1741

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

Background Details

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	
	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: HTML

(Good)

```
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Example Language: Java

(Bad)

```
public class FileUploadServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ...
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf("\\"), pLine.lastIndexOf("\""));
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.write(line);
                        bw.newLine();
                        bw.flush();
                    }
                } //end of for loop
                bw.close();
            } catch (IOException ex) {...}
            // output successful upload response HTML page
        }
        // output unsuccessful upload response HTML page
        else
        {...}
    }
    ...
}
```

This code does not perform a check on the type of the file being uploaded (CWE-434). This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the `BufferedWriter` object is subject to relative path traversal (CWE-23). Since the code does not check the filename that is provided in the header, an attacker can use `"../"` sequences to write to files outside of the intended directory. Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Example 2:

This code includes an external script to get database credentials, then authenticates a user against the database, allowing access to the application.

Example Language: PHP

(Bad)

```
//assume the password is already encrypted, avoiding CWE-312
function authenticate($username,$password){
    include("http://external.example.com/dbInfo.php");
    //dbInfo.php makes $dbhost, $dbuser, $dbpass, $dbname available
    mysql_connect($dbhost, $dbuser, $dbpass) or die ('Error connecting to mysql');
    mysql_select_db($dbname);
    $query = 'Select * from users where username='.$username.' And password='.$password;
    $result = mysql_query($query);
    if(mysql_numrows($result) == 1){
        mysql_close();
        return true;
    }
    else{
        mysql_close();
        return false;
    }
}
```

This code does not verify that the external domain accessed is the intended one. An attacker may somehow cause the external domain name to resolve to an attack server, which would provide the information for a false database. The attacker may then steal the usernames and encrypted passwords from real user login attempts, or simply allow themselves to access the application without a real user account.

This example is also vulnerable to an Adversary-in-the-Middle (CWE-300) attack.

Example 3:

This code either generates a public HTML user information page or a JSON response containing the same user information.

Example Language: PHP

(Bad)

```
// API flag, output JSON if set
$json = $_GET['json']
$username = $_GET['user']
if(!$json)
{
    $record = getUserRecord($username);
    foreach($record as $fieldName => $fieldValue)
    {
        if($fieldName == "email_address") {
            // skip displaying user emails
            continue;
        }
        else{
            writeToHtmlPage($fieldName,$fieldValue);
        }
    }
}
else
{
    $record = getUserRecord($username);
```

```
echo json_encode($record);
}
```





The programmer is careful to not display the user's e-mail address when displaying the public HTML page. However, the e-mail address is not removed from the JSON response, exposing the user's e-mail address.

Observed Examples

Reference	Description
CVE-2021-22909	Chain: router's firmware update procedure uses curl with "-k" (insecure) option that disables certificate validation (CWE-295), allowing adversary-in-the-middle (AITM) compromise with a malicious firmware image (CWE-494). https://www.cve.org/CVERecord?id=CVE-2021-22909
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-2005-0406	Some image editors modify a JPEG image, but the original EXIF thumbnail image is left intact within the JPEG. (Also an interaction error). https://www.cve.org/CVERecord?id=CVE-2005-0406

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		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

CWE-670: Always-Incorrect Control Flow Implementation

Weakness ID : 670

Structure : Simple

Abstraction : Class

Description

The code contains a control flow path that does not reflect the algorithm that the path is intended to implement, leading to incorrect behavior any time this path is navigated.

Extended Description

This weakness captures cases in which a particular code segment is always incorrect with respect to the algorithm that it is implementing. For example, if a C programmer intends to include multiple statements in a single block but does not include the enclosing braces (CWE-483), then the logic is always incorrect. This issue is in contrast to most weaknesses in which the code usually behaves correctly, except when it is externally manipulated in malicious ways.

Relationships

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	1517
ParentOf	B	480	Use of Incorrect Operator	1150
ParentOf	B	483	Incorrect Block Delimitation	1160
ParentOf	B	484	Omitted Break Statement in Switch	1162
ParentOf	B	617	Reachable Assertion	1378
ParentOf	B	698	Execution After Redirect (EAR)	1533
ParentOf	B	783	Operator Precedence Logic Error	1650

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

Nature	Type	ID	Name	Page
ParentOf	B	617	Reachable Assertion	1378

Common Consequences

Scope	Impact	Likelihood
Other	Other Alter Execution Logic	

Demonstrative Examples**Example 1:**

This code queries a server and displays its status when a request comes from an authorized IP address.

Example Language: PHP

(Bad)

```
$requestingIP = $_SERVER['REMOTE_ADDR'];
if(!in_array($requestingIP,$ipAllowList)){
    echo "You are not authorized to view this page";
    http_redirect($errorPageURL);
}
$status = getServerStatus();
echo $status;
...
```

This code redirects unauthorized users, but continues to execute code after calling `http_redirect()`. This means even unauthorized users may be able to access the contents of the page or perform a DoS attack on the server being queried. Also, note that this code is vulnerable to an IP address spoofing attack (CWE-212).

Example 2:

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 3:

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.

Example 4:

In the excerpt below, an AssertionError (an unchecked exception) is thrown if the user hasn't entered an email address in an HTML form.

Example Language: Java

(Bad)

```
String email = request.getParameter("email_address");
assert email != null;
```




Observed Examples

Reference	Description
CVE-2021-3011	virtual interrupt controller in a virtualization product allows crash of host by writing a certain invalid value to a register, which triggers a fatal error instead of returning an error code https://www.cve.org/CVERecord?id=CVE-2021-3011

MemberOf Relationships

CWE-670: Always-Incorrect Control Flow Implementation

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	2407
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Notes

Maintenance

This node could possibly be split into lower-level nodes. "Early Return" is for returning control to the caller too soon (e.g., CWE-584). "Excess Return" is when control is returned too far up the call stack (CWE-600, CWE-395). "Improper control limitation" occurs when the product maintains control at a lower level of execution, when control should be returned "further" up the call stack (CWE-455). "Incorrect syntax" covers code that's "just plain wrong" such as CWE-484 and CWE-483.

CWE-671: Lack of Administrator Control over Security

Weakness ID : 671

Structure : Simple

Abstraction : Class

Description

The product uses security features in a way that prevents the product's administrator from tailoring security settings to reflect the environment in which the product is being used. This introduces resultant weaknesses or prevents it from operating at a level of security that is desired by the administrator.

Extended Description

If the product's administrator does not have the ability to manage security-related decisions at all times, then protecting the product from outside threats - including the product's developer - can become impossible. For example, a hard-coded account name and password cannot be changed by the administrator, thus exposing that product to attacks that the administrator can not prevent.

Relationships

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		657	Violation of Secure Design Principles	1446
ParentOf		447	Unimplemented or Unsupported Feature in UI	1075
ParentOf		798	Use of Hard-coded Credentials	1690

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Demonstrative Examples

Example 1:

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

Example Language: C

(Bad)

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

Example Language: Java

(Bad)

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



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

Observed Examples

Reference	Description
CVE-2022-29953	Condition Monitor firmware has a maintenance interface with hard-coded credentials https://www.cve.org/CVERecord?id=CVE-2022-29953
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

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		975	SFP Secondary Cluster: Architecture	888	2406
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

CWE-672: Operation on a Resource after Expiration or Release

Weakness ID : 672

Structure : Simple

Abstraction : Class











Description

The product uses, accesses, or otherwise operates on a resource after that resource has been expired, released, or revoked.

Relationships

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	1462
ParentOf		298	Improper Validation of Certificate Expiration	726
ParentOf		324	Use of a Key Past its Expiration Date	792
ParentOf		613	Insufficient Session Expiration	1371
ParentOf		825	Expired Pointer Dereference	1732
ParentOf		910	Use of Expired File Descriptor	1800
CanFollow		562	Return of Stack Variable Address	1278
CanFollow		826	Premature Release of Resource During Expected Lifetime	1734
CanFollow		911	Improper Update of Reference Count	1801
CanFollow		1341	Multiple Releases of Same Resource or Handle	2246

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

Nature	Type	ID	Name	Page
ParentOf		415	Double Free	1008
ParentOf		416	Use After Free	1012
ParentOf		613	Insufficient Session Expiration	1371

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

Nature	Type	ID	Name	Page
ParentOf		415	Double Free	1008
ParentOf		416	Use After Free	1012

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

Nature	Type	ID	Name	Page
ParentOf		415	Double Free	1008
ParentOf		416	Use After Free	1012

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	
	<i>If a released resource is subsequently reused or reallocated, then an attempt to use the original resource might allow access to sensitive data that is associated with a different user or entity.</i>	
Other	Other	

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
	When a resource is released it might not be in an expected state, later attempts to access the resource may lead to resultant errors that may lead to a crash.	

Demonstrative Examples

Example 1:

The following code shows a simple example of 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.

Example 2:

The following code shows a simple example of a double free error:

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 the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Example 3:

In the following C/C++ example the method processMessage is used to process a message received in the input array of char arrays. The input message array contains two char arrays: the first is the length of the message and the second is the body of the message. The length of the message is retrieved and used to allocate enough memory for a local char array, messageBody, to be created for the message body. The messageBody is processed in the method processMessageBody that will return an error if an error occurs while processing. If an error occurs then the return result variable is set to indicate an error and the messageBody char array memory is released using the method free and an error message is sent to the logError method.

Example Language: C

(Bad)

```
#define FAIL 0
```

```
#define SUCCESS 1
#define ERROR -1
#define MAX_MESSAGE_SIZE 32
int processMessage(char **message)
{
    int result = SUCCESS;
    int length = getMessageLength(message[0]);
    char *messageBody;
    if ((length > 0) && (length < MAX_MESSAGE_SIZE)) {
        messageBody = (char*)malloc(length*sizeof(char));
        messageBody = &message[1][0];
        int success = processMessageBody(messageBody);
        if (success == ERROR) {
            result = ERROR;
            free(messageBody);
        }
    }
    else {
        printf("Unable to process message; invalid message length");
        result = FAIL;
    }
    if (result == ERROR) {
        logError("Error processing message", messageBody);
    }
    return result;
}
```

However, the call to the method logError includes the messageBody after the memory for messageBody has been released using the free method. This can cause unexpected results and may lead to system crashes. A variable should never be used after its memory resources have been released.

Example Language: C (Good)

```
...
messageBody = (char*)malloc(length*sizeof(char));
messageBody = &message[1][0];
int success = processMessageBody(messageBody);
if (success == ERROR) {
    result = ERROR;
    logError("Error processing message", messageBody);
    free(messageBody);
}
...
```

Observed Examples

Reference	Description
CVE-2009-3547	Chain: race condition (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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	983	SFP Secondary Cluster: Faulty Resource Use	888	2410
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576

Nature	Type	ID	Name	V	Page
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2544

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP15		Faulty Resource Use
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory
OMG ASCSM	ASCSM-CWE-672		

References

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

CWE-673: External Influence of Sphere Definition

Weakness ID : 673

Structure : Simple

Abstraction : Class

Description

The product does not prevent the definition of control spheres from external actors.

Extended Description

Typically, a product defines its control sphere within the code itself, or through configuration by the product's administrator. In some cases, an external party can change the definition of the control sphere. This is typically a resultant weakness.

Relationships

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	1454
ParentOf	E	426	Untrusted Search Path	1028

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Demonstrative Examples

Example 1:

Consider a blog publishing tool, which might have three explicit control spheres: the creation of articles, only accessible to a "publisher;" commenting on articles, only accessible to a "commenter" who is a registered user; and reading articles, only accessible to an anonymous reader. Suppose that the application is deployed on a web server that is shared with untrusted parties. If a local user can modify the data files that define who a publisher is, then this user has modified the control sphere. In this case, the issue would be resultant from another weakness such as insufficient permissions.

Example 2:



In Untrusted Search Path (CWE-426), a user might be able to define the PATH environment variable to cause the product to search in the wrong directory for a library to load. The product's intended sphere of control would include "resources that are only modifiable by the person who installed the product." The PATH effectively changes the definition of this sphere so that it overlaps the attacker's sphere of control.

Observed Examples

Reference	Description
CVE-2008-2613	setuid program allows compromise using path that finds and loads a malicious library. https://www.cve.org/CVERecord?id=CVE-2008-2613

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	2416
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Theoretical

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

CWE-674: Uncontrolled Recursion

Weakness ID : 674
Structure : Simple
Abstraction : Class

Description

The product does not properly control the amount of recursion that takes place, consuming excessive resources, such as allocated memory or the program stack.

Relationships

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		834	Excessive Iteration	1754
ParentOf		776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1633

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

Nature	Type	ID	Name	Page
ParentOf		776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1633

Applicable Platforms

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

Alternate Terms

Stack Exhaustion :

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) <i>Resources including CPU, memory, and stack memory could be rapidly consumed or exhausted, eventually leading to an exit or crash.</i>	
Confidentiality	Read Application Data <i>In some cases, an application's interpreter might kill a process or thread that appears to be consuming too much resources, such as with PHP's <code>memory_limit</code> setting. When the interpreter kills the process/thread, it might report an error containing detailed information such as the application's installation path.</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 an end condition will be reached under all logic conditions. The end condition may include testing against the depth of recursion and exiting with an error if the recursion goes too deep. The complexity of the end condition contributes to the effectiveness of this action.

Effectiveness = Moderate

Phase: Implementation

Increase the stack size.

Effectiveness = Limited

Increasing the stack size might only be a temporary measure, since the stack typically is still not very large, and it might remain easy for attackers to cause an out-of-stack fault.

Demonstrative Examples

Example 1:

In this example a mistake exists in the code where the exit condition contained in `flag` is never called. This results in the function calling itself over and over again until the stack is exhausted.

Example Language: C

(Bad)

```
void do_something_recursive (int flag)
{
    ... // Do some real work here, but the value of flag is unmodified
    if (flag) { do_something_recursive (flag); } // flag is never modified so it is always TRUE - this call will continue until the stack explodes
}
int flag = 1; // Set to TRUE
do_something_recursive (flag);
```

Note that the only difference between the Good and Bad examples is that the recursion flag will change value and cause the recursive call to return.

Example Language: C

(Good)

```
void do_something_recursive (int flag)
{
    ... // Do some real work here
    // Modify value of flag on done condition
    if (flag) { do_something_recursive (flag); } // returns when flag changes to 0
}
int flag = 1; // Set to TRUE
do_something_recursive (flag);
```

Observed Examples

Reference	Description
CVE-2007-1285	Deeply nested arrays trigger stack exhaustion. https://www.cve.org/CVERecord?id=CVE-2007-1285
CVE-2007-3409	Self-referencing pointers create infinite loop and resultant stack exhaustion. https://www.cve.org/CVERecord?id=CVE-2007-3409
CVE-2016-10707	Javascript application accidentally changes input in a way that prevents a recursive call from detecting an exit condition. https://www.cve.org/CVERecord?id=CVE-2016-10707
CVE-2016-3627	An attempt to recover a corrupted XML file infinite recursion protection counter was not always incremented missing the exit condition. https://www.cve.org/CVERecord?id=CVE-2016-3627
CVE-2019-15118	USB-audio driver's descriptor code parsing allows unlimited recursion leading to stack exhaustion. https://www.cve.org/CVERecord?id=CVE-2019-15118

Affected Resources

- CPU

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	2339
MemberOf		884	CWE Cross-section	884	2567
MemberOf		985	SFP Secondary Cluster: Unrestricted Consumption	888	2411
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
Software Fault Patterns	SFP13		Unrestricted Consumption
OMG ASCRM	ASCRM-CWE-674		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
230	Serialized Data with Nested Payloads
231	Oversized Serialized Data Payloads

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < <http://www.omg.org/spec/ASCRM/1.0/> >.

CWE-675: Multiple Operations on Resource in Single-Operation Context

Weakness ID : 675

Structure : Simple

Abstraction : Class



Description








The product performs the same operation on a resource two or more times, when the operation should only be applied once.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1298
ParentOf		174	Double Decoding of the Same Data	437

Nature	Type	ID	Name	Page
ParentOf		605	Multiple Binds to the Same Port	1356
ParentOf		764	Multiple Locks of a Critical Resource	1604
ParentOf		765	Multiple Unlocks of a Critical Resource	1605
ParentOf		1341	Multiple Releases of Same Resource or Handle	2246
PeerOf		102	Struts: Duplicate Validation Forms	246
PeerOf		586	Explicit Call to Finalize()	1320
PeerOf		85	Doubled Character XSS Manipulations	188

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Other	

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:

This code binds a server socket to port 21, allowing the server to listen for traffic on that port.

Example Language: C

(Bad)

```
void bind_socket(void) {
    int server_sockfd;
    int server_len;
    struct sockaddr_in server_address;
    /*unlink the socket if already bound to avoid an error when bind() is called*/
    unlink("server_socket");
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 21;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_len = sizeof(struct sockaddr_in);
    bind(server_sockfd, (struct sockaddr *) &s1, server_len);
}
```





This code may result in two servers binding a socket to same port, thus receiving each other's traffic. This could be used by an attacker to steal packets meant for another process, such as a secure FTP server.

Observed Examples

Reference	Description
CVE-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice https://www.cve.org/CVERecord?id=CVE-2009-0935
CVE-2019-13351	file descriptor double close can cause the wrong file to be associated with a file descriptor. https://www.cve.org/CVERecord?id=CVE-2019-13351
CVE-2004-1939	XSS protection mechanism attempts to remove "/" that could be used to close tags, but it can be bypassed using double encoded slashes (%252F) https://www.cve.org/CVERecord?id=CVE-2004-1939

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		984	SFP Secondary Cluster: Life Cycle	888	2411
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Relationship

This weakness is probably closely associated with other issues related to doubling, such as CWE-462 (duplicate key in alist) or CWE-102 (Struts duplicate validation forms). It's usually a case of an API contract violation (CWE-227).

CWE-676: Use of Potentially Dangerous Function

Weakness ID : 676

Structure : Simple

Abstraction : Base

Description

The product invokes a potentially dangerous function that could introduce a vulnerability if it is used incorrectly, but the function can also be used safely.

Relationships


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		1177	Use of Prohibited Code	1972

Nature	Type	ID	Name	Page
ParentOf		785	Use of Path Manipulation Function without Maximum-sized Buffer	1656

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

Nature	Type	ID	Name	Page
MemberOf		1228	API / Function Errors	2482

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context Quality Degradation Unexpected State <i>If the function is used incorrectly, then it could result in security problems.</i>	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis Cost effective for partial coverage: Binary / Bytecode Quality Analysis Binary / Bytecode simple extractor - strings, ELF readers, etc.

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 Cost effective for partial coverage: 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 Cost effective for partial coverage: Warning Flags Source Code Quality Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Origin Analysis

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Build and Compilation

Phase: Implementation

Identify a list of prohibited API functions and prohibit developers from using these functions, providing safer alternatives. In some cases, automatic code analysis tools or the compiler can be instructed to spot use of prohibited functions, such as the "banned.h" include file from Microsoft's SDL. [REF-554] [REF-7]

Demonstrative Examples

Example 1:

The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

Example Language: C

(Bad)

```
void manipulate_string(char * string){
    char buf[24];
    strcpy(buf, string);
    ...
}
```

However, the programmer does not ensure that the size of the data pointed to by string will fit in the local buffer and copies the data with the potentially dangerous strcpy() function. This may result in a buffer overflow condition if an attacker can influence the contents of the string parameter.

Observed Examples

Reference	Description
CVE-2007-1470	Library has multiple buffer overflows using sprintf() and strcpy() https://www.cve.org/CVERecord?id=CVE-2007-1470
CVE-2009-3849	Buffer overflow using strcat() https://www.cve.org/CVERecord?id=CVE-2009-3849
CVE-2006-2114	Buffer overflow using strcpy() https://www.cve.org/CVERecord?id=CVE-2006-2114
CVE-2006-0963	Buffer overflow using strcpy() https://www.cve.org/CVERecord?id=CVE-2006-0963
CVE-2011-0712	Vulnerable use of strcpy() changed to use safer strncpy() https://www.cve.org/CVERecord?id=CVE-2011-0712

Reference	Description
CVE-2008-5005	Buffer overflow using strcpy() https://www.cve.org/CVERecord?id=CVE-2008-5005

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	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2342
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2350
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	2371
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2374
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	2458
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	2460
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2461
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2462
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2463
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Relationship

This weakness is different than CWE-242 (Use of Inherently Dangerous Function). CWE-242 covers functions with such significant security problems that they can never be guaranteed to be safe. Some functions, if used properly, do not directly pose a security risk, but can introduce a weakness if not called correctly. These are regarded as potentially dangerous. A well-known example is the strcpy() function. When provided with a destination buffer that is larger than its source, strcpy() will not overflow. However, it is so often misused that some developers prohibit strcpy() entirely.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Dangerous Functions
CERT C Secure Coding	CON33-C	CWE More Abstract	Avoid race conditions when using library functions
CERT C Secure Coding	ENV33-C	CWE More Abstract	Do not call system()
CERT C Secure Coding	ERR07-C		Prefer functions that support error checking over equivalent functions that don't

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ERR34-C	CWE More Abstract	Detect errors when converting a string to a number
CERT C Secure Coding	FIO01-C		Be careful using functions that use file names for identification
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers
CERT C Secure Coding	STR31-C	Imprecise	Guarantee that storage for strings has sufficient space for character data and the null terminator
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 >.

[REF-554]Michael Howard. "Security Development Lifecycle (SDL) Banned Function Calls". < [https://learn.microsoft.com/en-us/previous-versions/bb288454\(v=msdn.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/bb288454(v=msdn.10)?redirectedfrom=MSDN) >. 2023-04-07.

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

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

CWE-680: Integer Overflow to Buffer Overflow

Weakness ID : 680



Structure : Chain

Abstraction : Compound

Description

The product performs a calculation to determine how much memory to allocate, but an integer overflow can occur that causes less memory to be allocated than expected, leading to a buffer overflow.


Chain Components

Nature	Type	ID	Name	Page
StartsWith		190	Integer Overflow or Wraparound	472
FollowedBy		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

Relationships

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		190	Integer Overflow or Wraparound	472

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	

Demonstrative Examples

Example 1:

The following image processing code allocates a table for images.

Example Language: C

(Bad)

```
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```




This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Observed Examples

Reference	Description
CVE-2021-43537	Chain: in a web browser, an unsigned 64-bit integer is forcibly cast to a 32-bit integer (CWE-681) and potentially leading to an integer overflow (CWE-190). If an integer overflow occurs, this can cause heap memory corruption (CWE-122) https://www.cve.org/CVERecord?id=CVE-2021-43537
CVE-2017-1000121	chain: unchecked message size metadata allows integer overflow (CWE-190) leading to buffer overflow (CWE-119). https://www.cve.org/CVERecord?id=CVE-2017-1000121

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		1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2456
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2525

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	INT30-C	Imprecise	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	INT32-C	Imprecise	Ensure that operations on signed integers do not result in overflow

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM35-C	CWE More Abstract	Allocate sufficient memory for an object

Related Attack Patterns

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

CWE-681: Incorrect Conversion between Numeric Types

Weakness ID : 681

Structure : Simple

Abstraction : Base

Description

When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur.

Relationships

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		704	Incorrect Type Conversion or Cast	1538
ParentOf		192	Integer Coercion Error	482
ParentOf		194	Unexpected Sign Extension	491
ParentOf		195	Signed to Unsigned Conversion Error	494
ParentOf		196	Unsigned to Signed Conversion Error	498
ParentOf		197	Numeric Truncation Error	500
CanPrecede		682	Incorrect Calculation	1499

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





Nature	Type	ID	Name	Page
ChildOf		704	Incorrect Type Conversion or Cast	1538

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

Nature	Type	ID	Name	Page
ParentOf		194	Unexpected Sign Extension	491

Nature	Type	ID	Name	Page
ParentOf		195	Signed to Unsigned Conversion Error	494
ParentOf		196	Unsigned to Signed Conversion Error	498
ParentOf		197	Numeric Truncation Error	500

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

Nature	Type	ID	Name	Page
ParentOf		194	Unexpected Sign Extension	491
ParentOf		195	Signed to Unsigned Conversion Error	494
ParentOf		196	Unsigned to Signed Conversion Error	498
ParentOf		197	Numeric Truncation Error	500

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

Nature	Type	ID	Name	Page
MemberOf		136	Type Errors	2310
MemberOf		189	Numeric Errors	2312

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other Integrity	Unexpected State Quality Degradation <i>The program could wind up using the wrong number and generate incorrect results. If the number is used to allocate resources or make a security decision, then this could introduce a vulnerability.</i>	

Potential Mitigations

Phase: Implementation

Avoid making conversion between numeric types. Always check for the allowed ranges.

Demonstrative Examples

Example 1:

In the following Java example, a float literal is cast to an integer, thus causing a loss of precision.

Example Language: Java

(Bad)

```
int i = (int) 33457.8f;
```

Example 2:

This code adds a float and an integer together, casting the result to an integer.

Example Language: PHP

(Bad)

```
$floatVal = 1.8345;  
$intVal = 3;  
$result = (int)$floatVal + $intVal;
```

Normally, PHP will preserve the precision of this operation, making `$result = 4.8345`. After the cast to int, it is reasonable to expect PHP to follow rounding convention and set `$result = 5`. However,

the explicit cast to int always rounds DOWN, so the final value of \$result is 4. This behavior may have unintended consequences.

Example 3:

In this example the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned int, amount will be implicitly converted to unsigned.

Example Language: C

(Bad)

```
unsigned int readdata () {
    int amount = 0;
    ...
    if (result == ERROR)
        amount = -1;
    ...
    return amount;
}
```

If the error condition in the code above is met, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Example 4:

In this example, depending on the return value of accessmainframe(), the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned value, amount will be implicitly cast to an unsigned number.

Example Language: C

(Bad)

```
unsigned int readdata () {
    int amount = 0;
    ...
    amount = accessmainframe();
    ...
    return amount;
}
```

If the return value of accessmainframe() is -1, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Observed Examples

Reference	Description
CVE-2022-2639	Chain: integer coercion error (CWE-192) prevents a return value from indicating an error, leading to out-of-bounds write (CWE-787) https://www.cve.org/CVERecord?id=CVE-2022-2639
CVE-2021-43537	Chain: in a web browser, an unsigned 64-bit integer is forcibly cast to a 32-bit integer (CWE-681) and potentially leading to an integer overflow (CWE-190). If an integer overflow occurs, this can cause heap memory corruption (CWE-122) https://www.cve.org/CVERecord?id=CVE-2021-43537
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) https://www.cve.org/CVERecord?id=CVE-2007-4268
CVE-2007-4988	Chain: signed short width value in image processor is sign extended during conversion to unsigned int, which leads to integer overflow and heap-based buffer overflow. https://www.cve.org/CVERecord?id=CVE-2007-4988
CVE-2009-0231	Integer truncation of length value leads to heap-based buffer overflow. https://www.cve.org/CVERecord?id=CVE-2009-0231

Reference	Description
CVE-2008-3282	Size of a particular type changes for 64-bit platforms, leading to an integer truncation in document processor causes incorrect index to be generated. https://www.cve.org/CVERecord?id=CVE-2008-3282

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	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2342
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2343
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	2363
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2374
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2375
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	2445
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2456
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	2457
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FLP34-C	CWE More Abstract	Ensure that floating point conversions are within range of the new type
CERT C Secure Coding	INT15-C		Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types
CERT C Secure Coding	INT31-C	CWE More Abstract	Ensure that integer conversions do not result in lost or misinterpreted data
CERT C Secure Coding	INT35-C		Evaluate integer expressions in a larger size before comparing or assigning to that size
The CERT Oracle Secure Coding Standard for Java (2011)	NUM12-J		Ensure conversions of numeric types to narrower types do not result in lost or misinterpreted data
Software Fault Patterns	SFP1		Glitch in computation

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCSM	ASCSM- CWE-681		

References

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

CWE-682: Incorrect Calculation

Weakness ID : 682

Structure : Simple

Abstraction : Pillar

Description

The product performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
















Extended Description

When product performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things. Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.



Relationships

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	2575
ParentOf		128	Wrap-around Error	339
ParentOf		131	Incorrect Calculation of Buffer Size	355
ParentOf		135	Incorrect Calculation of Multi-Byte String Length	370
ParentOf		190	Integer Overflow or Wraparound	472
ParentOf		191	Integer Underflow (Wrap or Wraparound)	480
ParentOf		193	Off-by-one Error	486
ParentOf		369	Divide By Zero	913
ParentOf		468	Incorrect Pointer Scaling	1114
ParentOf		469	Use of Pointer Subtraction to Determine Size	1115
ParentOf		1335	Incorrect Bitwise Shift of Integer	2235
ParentOf		1339	Insufficient Precision or Accuracy of a Real Number	2242
CanFollow		681	Incorrect Conversion between Numeric Types	1495
CanFollow		839	Numeric Range Comparison Without Minimum Check	1767
CanPrecede		170	Improper Null Termination	428

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

Nature	Type	ID	Name	Page
ParentOf		131	Incorrect Calculation of Buffer Size	355
ParentOf		190	Integer Overflow or Wraparound	472

Nature	Type	ID	Name	Page
ParentOf	B	191	Integer Underflow (Wrap or Wraparound)	480
ParentOf	B	193	Off-by-one Error	486
ParentOf	B	369	Divide By Zero	913

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

Nature	Type	ID	Name	Page
ParentOf	B	131	Incorrect Calculation of Buffer Size	355
ParentOf	B	369	Divide By Zero	913

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

Nature	Type	ID	Name	Page
ParentOf	B	131	Incorrect Calculation of Buffer Size	355
ParentOf	B	369	Divide By Zero	913

Applicable Platforms

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

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart <i>If the incorrect calculation causes the program to move into an unexpected state, it may lead to a crash or impairment of service.</i>	
Integrity Confidentiality Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (Other) Execute Unauthorized Code or Commands <i>If the incorrect calculation is used in the context of resource allocation, it could lead to an out-of-bounds operation (CWE-119) leading to a crash or even arbitrary code execution. Alternatively, it may result in an integer overflow (CWE-190) and / or a resource consumption problem (CWE-400).</i>	
Access Control	Gain Privileges or Assume Identity <i>In the context of privilege or permissions assignment, an incorrect calculation can provide an attacker with access to sensitive resources.</i>	
Access Control	Bypass Protection Mechanism <i>If the incorrect calculation leads to an insufficient comparison (CWE-697), it may compromise a protection mechanism such as a validation routine and allow an attacker to bypass the security-critical code.</i>	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for

evaluating the correctness of allocation calculations. This can be useful for detecting overflow conditions (CWE-190) or similar weaknesses that might have serious security impacts on the program.

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.

Potential Mitigations

Phase: Implementation

Understand your programming language's underlying representation and how it interacts with numeric calculation. Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how your language handles numbers that are too large or too small for its underlying representation.

Phase: Implementation

Strategy = Input Validation

Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.

Phase: Implementation

Use the appropriate type for the desired action. For example, in C/C++, only use unsigned types for values that could never be negative, such as height, width, or other numbers related to quantity.

Phase: Architecture and Design

Strategy = Language Selection

Use languages, libraries, or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use languages, libraries, or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).

Phase: Implementation

Strategy = Compilation or Build Hardening

Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Phase: Testing

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

Phase: Testing

Use dynamic tools and techniques that interact with the product using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The

product's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Demonstrative Examples

Example 1:

The following image processing code allocates a table for images.

Example Language: C (Bad)

```
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 2:

This code attempts to calculate a football team's average number of yards gained per touchdown.

Example Language: Java (Bad)

```
...
int touchdowns = team.getTouchdowns();
int yardsGained = team.getTotalYardage();
System.out.println(team.getName() + " averages " + yardsGained / touchdowns + "yards gained for every touchdown scored");
...
```

The code does not consider the event that the team they are querying has not scored a touchdown, but has gained yardage. In that case, we should expect an ArithmeticException to be thrown by the JVM. This could lead to a loss of availability if our error handling code is not set up correctly.

Example 3:

This example 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.

Observed Examples

Reference	Description
CVE-2020-0022	chain: mobile phone Bluetooth implementation does not include offset when calculating packet length (CWE-682), leading to out-of-bounds write (CWE-787) https://www.cve.org/CVERecord?id=CVE-2020-0022

Reference	Description
CVE-2004-1363	substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed https://www.cve.org/CVERecord?id=CVE-2004-1363

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	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2342
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2343
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2353
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2374
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2375
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	2445
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2456
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	2457
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1408	Comprehensive Categorization: Incorrect Calculation	1400	2534

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FLP32-C	CWE More Abstract	Prevent or detect domain and range errors in math functions
CERT C Secure Coding	INT07-C		Use only explicitly signed or unsigned char type for numeric values
CERT C Secure Coding	INT13-C		Use bitwise operators only on unsigned operands
CERT C Secure Coding	INT33-C	CWE More Abstract	Ensure that division and remainder operations do not result in divide-by-zero errors
CERT C Secure Coding	INT34-C	CWE More Abstract	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
128	Integer Attacks
129	Pointer Manipulation

References

[REF-106]David LeBlanc and Niels Dekker. "SafeInt". < <http://safeint.codeplex.com/> >.

[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-683: Function Call With Incorrect Order of Arguments

Weakness ID : 683

Structure : Simple

Abstraction : Variant

Description

The product calls a function, procedure, or routine, but the caller specifies the arguments in an incorrect order, leading to resultant weaknesses.

Extended Description

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers or types of arguments, such as format strings in C. It also can occur in languages or environments that do not enforce strong typing.

Relationships

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		628	Function Call with Incorrectly Specified Arguments	1398

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Use the function, procedure, or routine as specified.

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the product. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally

produces the correct results or if the provided argument type is very similar to the expected argument type.

Demonstrative Examples

Example 1:

The following PHP method authenticates a user given a username/password combination but is called with the parameters in reverse order.

Example Language: PHP

(Bad)

```
function authenticate($username, $password) {
    // authenticate user
    ...
}
authenticate($_POST['password'], $_POST['username']);
```

Observed Examples

Reference	Description
CVE-2006-7049	Application calls functions with arguments in the wrong order, allowing attacker to bypass intended access restrictions. https://www.cve.org/CVERecord?id=CVE-2006-7049

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		998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-684: Incorrect Provision of Specified Functionality

Weakness ID : 684

Structure : Simple

Abstraction : Class

Description

The code does not function according to its published specifications, potentially leading to incorrect usage.


Extended Description








When providing functionality to an external party, it is important that the product behaves in accordance with the details specified. When requirements of nuances are not documented, the functionality may produce unintended behaviors for the caller, possibly leading to an exploitable 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		710	Improper Adherence to Coding Standards	1549

Nature	Type	ID	Name	Page
ParentOf		392	Missing Report of Error Condition	951
ParentOf		393	Return of Wrong Status Code	953
ParentOf		440	Expected Behavior Violation	1062
ParentOf		446	UI Discrepancy for Security Feature	1073
ParentOf		451	User Interface (UI) Misrepresentation of Critical Information	1079
ParentOf		912	Hidden Functionality	1803
ParentOf		1245	Improper Finite State Machines (FSMs) in Hardware Logic	2041

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Ensure that your code strictly conforms to specifications.

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;
}
```

Example 2:

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-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

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
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-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		Page
MemberOf		735	CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE)	734	2340
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	PRE09-C		Do not replace secure functions with less secure functions

CWE-685: Function Call With Incorrect Number of Arguments

Weakness ID : 685

Structure : Simple

Abstraction : Variant


Description

The product calls a function, procedure, or routine, but the caller specifies too many arguments, or too few arguments, which may lead to undefined behavior and resultant weaknesses.

Relationships

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		628	Function Call with Incorrectly Specified Arguments	1398

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : Perl (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Other

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in languages or environments that do not require that functions always be called with the correct number of arguments, such as Perl.

Potential Mitigations

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the product. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

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	998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	EXP37-C	CWE More Specific	Call functions with the correct number and type of arguments
CERT C Secure Coding	FIO47-C	Imprecise	Use valid format strings

CWE-686: Function Call With Incorrect Argument Type

Weakness ID : 686

Structure : Simple

Abstraction : Variant

Description

The product calls a function, procedure, or routine, but the caller specifies an argument that is the wrong data type, which may lead to resultant weaknesses.


Extended Description

This weakness is most likely to occur in loosely typed languages, or in strongly typed languages in which the types of variable arguments cannot be enforced at compilation time, or where there is implicit casting.

Relationships

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		628	Function Call with Incorrectly Specified Arguments	1398

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	













Potential Mitigations

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the product. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

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		736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	2341
MemberOf		739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2343
MemberOf		741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2344
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf		748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2351
MemberOf		873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2375
MemberOf		875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2376
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP37-C	CWE More Specific	Call functions with the correct number and type of arguments

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO47-C	Imprecise	Use valid format strings
CERT C Secure Coding	POS34-C		Do not call putenv() with a pointer to an automatic variable as the argument
CERT C Secure Coding	STR37-C		Arguments to character handling functions must be representable as an unsigned char
Software Fault Patterns	SFP1		Glitch in computation

CWE-687: Function Call With Incorrectly Specified Argument Value

Weakness ID : 687

Structure : Simple

Abstraction : Variant



Description

The product calls a function, procedure, or routine, but the caller specifies an argument that contains the wrong value, which may lead to resultant weaknesses.

Relationships

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		628	Function Call with Incorrectly Specified Arguments	1398
ParentOf		560	Use of umask() with chmod-style Argument	1274

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Manual Static Analysis

This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Demonstrative Examples

Example 1:

This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

Example Language: Perl

(Bad)

```
sub ReportAuth {
    my ($username, $result, $fatal) = @_;
    PrintLog("auth: username=%s, result=%d", $username, $result);
    if (($result ne "success") && $fatal) {
        die "Failed!\n";
    }
}
```





```

    }
}
sub PrivilegedFunc
{
    my $result = CheckAuth($username);
    ReportAuth($username, $result, 0);
    DoReallyImportantStuff();
}

```

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		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Relationship

When primary, this weakness is most likely to occur in rarely-tested code, since the wrong value can change the semantic meaning of the program's execution and lead to obviously-incorrect behavior. It can also be resultant from issues in which the program assigns the wrong value to a variable, and that variable is later used in a function call. In that sense, this issue could be argued as having chaining relationships with many implementation errors in CWE.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM04-C		Do not perform zero length allocations
Software Fault Patterns	SFP24		Tainted input to command

CWE-688: Function Call With Incorrect Variable or Reference as Argument

Weakness ID : 688

Structure : Simple

Abstraction : Variant

Description

The product calls a function, procedure, or routine, but the caller specifies the wrong variable or reference as one of the arguments, which may lead to undefined behavior and resultant weaknesses.

Relationships

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		628	Function Call with Incorrectly Specified Arguments	1398

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (*Prevalence = Undetermined*)

Language : Perl (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Other

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in loosely typed languages or environments. This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Potential Mitigations

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the product. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

Demonstrative Examples

Example 1:

In the following Java snippet, the `accessGranted()` method is accidentally called with the static `ADMIN_ROLES` array rather than the user roles.

Example Language: Java

(Bad)

```
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
    String[] userRoles = getUserRoles(user);
    return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
    // grant or deny access based on user roles
    ...
}
```

Observed Examples

Reference	Description
CVE-2005-2548	Kernel code specifies the wrong variable in first argument, leading to resultant NULL pointer dereference. https://www.cve.org/CVERecord?id=CVE-2005-2548

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		998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-689: Permission Race Condition During Resource Copy

Weakness ID : 689

Structure : Composite

Abstraction : Compound

Description

The product, while copying or cloning a resource, does not set the resource's permissions or access control until the copy is complete, leaving the resource exposed to other spheres while the copy is taking place.

Composite Components

Nature	Type	ID	Name	Page
Requires		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
Requires		732	Incorrect Permission Assignment for Critical Resource	1551

Relationships

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')	888

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : Perl (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	

Observed Examples

Reference	Description
CVE-2002-0760	Archive extractor decompresses files with world-readable permissions, then later sets permissions to what the archive specified. https://www.cve.org/CVERecord?id=CVE-2002-0760

Reference	Description
CVE-2005-2174	Product inserts a new object into database before setting the object's permissions, introducing a race condition. https://www.cve.org/CVERecord?id=CVE-2005-2174
CVE-2006-5214	Error file has weak permissions before a chmod is performed. https://www.cve.org/CVERecord?id=CVE-2006-5214
CVE-2005-2475	Archive permissions issue using hard link. https://www.cve.org/CVERecord?id=CVE-2005-2475
CVE-2003-0265	Database product creates files world-writable before initializing the setuid bits, leading to modification of executables. https://www.cve.org/CVERecord?id=CVE-2003-0265

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	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Research Gap

Under-studied. It seems likely that this weakness could occur in any situation in which a complex or large copy operation occurs, when the resource can be made available to other spheres as soon as it is created, but before its initialization is complete.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links

References

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

CWE-690: Unchecked Return Value to NULL Pointer Dereference

Weakness ID : 690

Structure : Chain

Abstraction : Compound

Description

The product does not check for an error after calling a function that can return with a NULL pointer if the function fails, which leads to a resultant NULL pointer dereference.

Chain Components

Nature	Type	ID	Name	Page
StartsWith	B	252	Unchecked Return Value	606
FollowedBy	B	476	NULL Pointer Dereference	1132

Extended Description

While unchecked return value weaknesses are not limited to returns of NULL pointers (see the examples in CWE-252), functions often return NULL to indicate an error status. When this error condition is not checked, a NULL pointer dereference can occur.

Relationships

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		252	Unchecked Return Value	606

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Read Memory	
Availability	Modify Memory	
<i>In rare circumstances, when NULL is equivalent to the 0x0 memory address and privileged code can access it, then writing or reading memory is possible, which may lead to code execution.</i>		

Detection Methods

Black Box

This typically occurs in rarely-triggered error conditions, reducing the chances of detection during black box testing.

White Box

Code analysis can require knowledge of API behaviors for library functions that might return NULL, reducing the chances of detection when unknown libraries are used.

Demonstrative Examples

Example 1:

The code below makes a call to the getUsername() function but doesn't check the return value before dereferencing (which may cause a NullPointerException).

Example Language: Java

(Bad)

```
String username = getUsername();
if (username.equals(ADMIN_USER)) {
    ...
}
```

Example 2:

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

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).

Observed Examples

Reference	Description
CVE-2008-1052	Large Content-Length value leads to NULL pointer dereference when malloc fails. https://www.cve.org/CVERecord?id=CVE-2008-1052
CVE-2006-6227	Large message length field leads to NULL pointer dereference when malloc fails. https://www.cve.org/CVERecord?id=CVE-2006-6227
CVE-2006-2555	Parsing routine encounters NULL dereference when input is missing a colon separator. https://www.cve.org/CVERecord?id=CVE-2006-2555
CVE-2003-1054	URI parsing API sets argument to NULL when a parsing failure occurs, such as when the Referer header is missing a hostname, leading to NULL dereference. https://www.cve.org/CVERecord?id=CVE-2003-1054
CVE-2008-5183	chain: unchecked return value can lead to NULL dereference https://www.cve.org/CVERecord?id=CVE-2008-5183

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	2365
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2525

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP34-C	CWE More Specific	Do not dereference null pointers
The CERT Oracle Secure Coding Standard for Java (2011)	ERR08-J		Do not catch NullPointerException or any of its ancestors

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	EXP32-PL	CWE More Specific	Do not ignore function return values

CWE-691: Insufficient Control Flow Management

Weakness ID : 691

Structure : Simple

Abstraction : Pillar
















Description

The code does not sufficiently manage its control flow during execution, creating conditions in which the control flow can be modified in unexpected ways.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf		1000	Research Concepts	2575
ParentOf		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888
ParentOf		430	Deployment of Wrong Handler	1042
ParentOf		431	Missing Handler	1043
ParentOf		662	Improper Synchronization	1448
ParentOf		670	Always-Incorrect Control Flow Implementation	1475
ParentOf		696	Incorrect Behavior Order	1527
ParentOf		705	Incorrect Control Flow Scoping	1542
ParentOf		768	Incorrect Short Circuit Evaluation	1612
ParentOf		799	Improper Control of Interaction Frequency	1699
ParentOf		834	Excessive Iteration	1754
ParentOf		841	Improper Enforcement of Behavioral Workflow	1772
ParentOf		1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	2088
ParentOf		1279	Cryptographic Operations are run Before Supporting Units are Ready	2120
ParentOf		1281	Sequence of Processor Instructions Leads to Unexpected Behavior	2124

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic	

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:

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 3:

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-2019-9805	Chain: Creation of the packet client occurs before initialization is complete (CWE-696) resulting in a read from uninitialized memory (CWE-908), causing memory corruption. https://www.cve.org/CVERecord?id=CVE-2019-9805
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
CVE-2011-1027	Chain: off-by-one error (CWE-193) leads to infinite loop (CWE-835) using invalid hex-encoded characters. https://www.cve.org/CVERecord?id=CVE-2011-1027

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	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	40		Insufficient Process Validation

Related Attack Patterns

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

CWE-692: Incomplete Denylist to Cross-Site Scripting

Weakness ID : 692

Structure : Chain

Abstraction : Compound

Description

The product uses a denylist-based protection mechanism to defend against XSS attacks, but the denylist is incomplete, allowing XSS variants to succeed.

Chain Components

Nature	Type	ID	Name	Page
StartsWith	B	184	Incomplete List of Disallowed Inputs	459
FollowedBy	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	163

Extended Description

While XSS might seem simple to prevent, web browsers vary so widely in how they parse web pages, that a denylist cannot keep track of all the variations. The "XSS Cheat Sheet" [REF-714] contains a large number of attacks that are intended to bypass incomplete denylists.

Relationships

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		184	Incomplete List of Disallowed Inputs	459

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Observed Examples

Reference	Description
CVE-2007-5727	Denylist only removes <SCRIPT> tag. https://www.cve.org/CVERecord?id=CVE-2007-5727
CVE-2006-3617	Denylist only removes <SCRIPT> tag. https://www.cve.org/CVERecord?id=CVE-2006-3617
CVE-2006-4308	Denylist only checks "javascript:" tag https://www.cve.org/CVERecord?id=CVE-2006-4308

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		1409	Comprehensive Categorization: Injection	1400	2535

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
71	Using Unicode Encoding to Bypass Validation Logic
80	Using UTF-8 Encoding to Bypass Validation Logic
85	AJAX Footprinting
120	Double Encoding
267	Leverage Alternate Encoding

References

[REF-714]RSnake. "XSS (Cross Site Scripting) Cheat Sheet". < <http://ha.ckers.org/xss.html> >.

CWE-693: Protection Mechanism Failure

Weakness ID : 693

Structure : Simple

Abstraction : Pillar**Description**

The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.















Extended Description

This weakness covers three distinct situations. A "missing" protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An "insufficient" protection mechanism might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended. Finally, an "ignored" mechanism occurs when a mechanism is available and in active use within the product, but the developer has not applied it in some code path.

Relationships

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	2575
ParentOf		182	Collapse of Data into Unsafe Value	455
ParentOf		184	Incomplete List of Disallowed Inputs	459
ParentOf		311	Missing Encryption of Sensitive Data	757
ParentOf		326	Inadequate Encryption Strength	796
ParentOf		327	Use of a Broken or Risky Cryptographic Algorithm	799
ParentOf		330	Use of Insufficiently Random Values	814
ParentOf		345	Insufficient Verification of Data Authenticity	851
ParentOf		357	Insufficient UI Warning of Dangerous Operations	880
ParentOf		358	Improperly Implemented Security Check for Standard	881
ParentOf		424	Improper Protection of Alternate Path	1023
ParentOf		602	Client-Side Enforcement of Server-Side Security	1350
ParentOf		653	Improper Isolation or Compartmentalization	1437
ParentOf		654	Reliance on a Single Factor in a Security Decision	1439
ParentOf		655	Insufficient Psychological Acceptability	1442
ParentOf		656	Reliance on Security Through Obscurity	1444
ParentOf		757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	1581
ParentOf		778	Insufficient Logging	1638
ParentOf		807	Reliance on Untrusted Inputs in a Security Decision	1714
ParentOf		1039	Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	1873
ParentOf		1248	Semiconductor Defects in Hardware Logic with Security-Sensitive Implications	2049
ParentOf		1253	Incorrect Selection of Fuse Values	2058
ParentOf		1269	Product Released in Non-Release Configuration	2098
ParentOf		1278	Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques	2118
ParentOf		1291	Public Key Re-Use for Signing both Debug and Production Code	2145

Nature	Type	ID	Name	Page
ParentOf	B	1318	Missing Support for Security Features in On-chip Fabrics or Buses	2197
ParentOf	B	1319	Improper Protection against Electromagnetic Fault Injection (EM-FI)	2199
ParentOf	B	1326	Missing Immutable Root of Trust in Hardware	2212
ParentOf	B	1338	Improper Protections Against Hardware Overheating	2240

Applicable Platforms

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

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection 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	C	975	SFP Secondary Cluster: Architecture	888	2406
MemberOf	C	1370	ICS Supply Chain: Common Mode Frailties	1358	2507
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Notes

Research Gap

The concept of protection mechanisms is well established, but protection mechanism failures have not been studied comprehensively. It is suspected that protection mechanisms can have significantly different types of weaknesses than the weaknesses that they are intended to prevent.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
17	Using Malicious Files
20	Encryption Brute Forcing
22	Exploiting Trust in Client
36	Using Unpublished Interfaces or Functionality
51	Poison Web Service Registry
57	Utilizing REST's Trust in the System Resource to Obtain Sensitive Data
59	Session Credential Falsification through Prediction
65	Sniff Application Code
74	Manipulating State
87	Forceful Browsing
107	Cross Site Tracing
127	Directory Indexing
237	Escaping a Sandbox by Calling Code in Another Language
477	Signature Spoofing by Mixing Signed and Unsigned Content
480	Escaping Virtualization

CAPEC-ID	Attack Pattern Name
668	Key Negotiation of Bluetooth Attack (KNOB)

CWE-694: Use of Multiple Resources with Duplicate Identifier

Weakness ID : 694

Structure : Simple

Abstraction : Base

Description

The product uses multiple resources that can have the same identifier, in a context in which unique identifiers are required.


Extended Description

If the product assumes that each resource has a unique identifier, the product could operate on the wrong resource if attackers can cause multiple resources to be associated with the same identifier.




Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1298
ChildOf		99	Improper Control of Resource Identifiers ('Resource Injection')	243
ParentOf		102	Struts: Duplicate Validation Forms	246
ParentOf		462	Duplicate Key in Associative List (Alist)	1104

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422
MemberOf		137	Data Neutralization Issues	2311
MemberOf		399	Resource Management Errors	2324

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>If unique identifiers are assumed when protecting sensitive resources, then duplicate identifiers might allow attackers to bypass the protection.</i>	
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Where possible, use unique identifiers. If non-unique identifiers are detected, then do not operate any resource with a non-unique identifier and report the error appropriately.

Demonstrative Examples

Example 1:

These two Struts validation forms have the same name.

Example Language: XML

(Bad)

```
<form-validation>
  <formset>
    <form name="ProjectForm"> ... </form>
    <form name="ProjectForm"> ... </form>
  </formset>
</form-validation>
```

It is not certain which form will be used by Struts. It is critically important that validation logic be maintained and kept in sync with the rest of the product.

Observed Examples

Reference	Description
CVE-2013-4787	chain: mobile OS verifies cryptographic signature of file in an archive, but then installs a different file with the same name that is also listed in the archive. https://www.cve.org/CVERecord?id=CVE-2013-4787

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		984	SFP Secondary Cluster: Life Cycle	888	2411
MemberOf		1409	Comprehensive Categorization: Injection	1400	2535

Notes**Relationship**

This weakness is probably closely associated with other issues related to doubling, such as CWE-675 (Duplicate Operations on Resource). It's often a case of an API contract violation (CWE-227).

CWE-695: Use of Low-Level Functionality

Weakness ID : 695

Structure : Simple

Abstraction : Base

Description

The product uses low-level functionality that is explicitly prohibited by the framework or specification under which the product is supposed to operate.

Extended Description

The use of low-level functionality can violate the specification in unexpected ways that effectively disable built-in protection mechanisms, introduce exploitable inconsistencies, or otherwise expose the functionality to attack.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1298
ParentOf		111	Direct Use of Unsafe JNI	266
ParentOf		245	J2EE Bad Practices: Direct Management of Connections	592
ParentOf		246	J2EE Bad Practices: Direct Use of Sockets	594
ParentOf		383	J2EE Bad Practices: Direct Use of Threads	935
ParentOf		574	EJB Bad Practices: Use of Synchronization Primitives	1300
ParentOf		575	EJB Bad Practices: Use of AWT Swing	1301
ParentOf		576	EJB Bad Practices: Use of Java I/O	1304

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

Nature	Type	ID	Name	Page
MemberOf		1228	API / Function Errors	2482

Common Consequences

Scope	Impact	Likelihood
Other	Other	

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 defines a class named Echo. The class declares one native method (defined below), which uses C to echo commands entered on the console back to the user. The following C code defines the native method implemented in the Echo class:

Example Language: Java

(Bad)

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

Example Language: C

(Bad)

```
#include <jni.h>
#include "Echo.h"//the java class above compiled with javah
#include <stdio.h>
JNIEXPORT void JNICALL
Java_Echo_runEcho(JNIEnv *env, jobject obj)
{
    char buf[64];
    gets(buf);
}
```

```
printf(buf);
}
```

Because the example is implemented in Java, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although Java does do a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in source code written in other languages that are accessed using the Java Native Interface. Despite the memory protections offered in Java, the C code in this example is vulnerable to a buffer overflow because it makes use of `gets()`, which does not check the length of its input.

The Sun Java(TM) Tutorial provides the following description of JNI [See Reference]: The JNI framework lets your native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects, including arrays and strings, and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code. A native method can even update Java objects that it created or that were passed to it, and these updated objects are available to the Java application. Thus, both the native language side and the Java side of an application can create, update, and access Java objects and then share these objects between them.

The vulnerability in the example above could easily be detected through a source code audit of the native method implementation. This may not be practical or possible depending on the availability of the C source code and the way the project is built, but in many cases it may suffice. However, the ability to share objects between Java and native methods expands the potential risk to much more insidious cases where improper data handling in Java may lead to unexpected vulnerabilities in native code or unsafe operations in native code corrupt data structures in Java. Vulnerabilities in native code accessed through a Java application are typically exploited in the same manner as they are in applications written in the native language. The only challenge to such an attack is for the attacker to identify that the Java application uses native code to perform certain operations. This can be accomplished in a variety of ways, including identifying specific behaviors that are often implemented with native code or by exploiting a system information exposure in the Java application that reveals its use of JNI [See Reference].

Example 2:

The following example opens a socket to connect to a remote server.

Example Language: Java



(Bad)

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Perform servlet tasks.
    ...
    // Open a socket to a remote server (bad).
    Socket sock = null;
    try {
        sock = new Socket(remoteHostname, 3000);
        // Do something with the socket.
        ...
    } catch (Exception e) {
        ...
    }
}
```

A `Socket` object is created directly within the Java servlet, which is a dangerous way to manage remote connections.

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		1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
36	Using Unpublished Interfaces or Functionality

CWE-696: Incorrect Behavior Order

Weakness ID : 696

Structure : Simple

Abstraction : Class








Description

The product performs multiple related behaviors, but the behaviors are performed in the wrong order in ways which may produce resultant weaknesses.

Relationships

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	1517
ParentOf		179	Incorrect Behavior Order: Early Validation	448
ParentOf		408	Incorrect Behavior Order: Early Amplification	995
ParentOf		551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1264
ParentOf		1190	DMA Device Enabled Too Early in Boot Phase	1978
ParentOf		1193	Power-On of Untrusted Execution Core Before Enabling Fabric Access Control	1986
ParentOf		1280	Access Control Check Implemented After Asset is Accessed	2122

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic	

Demonstrative Examples

Example 1:

The following code attempts to validate a given input path by checking it against an allowlist and then return the canonical path. In this specific case, the path is considered valid if it starts with the string `"/safe_dir/"`.

Example Language: Java

(Bad)

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



```

    return f.getCanonicalPath();
}

```

The problem with the above code is that the validation step occurs before canonicalization occurs. An attacker could provide an input path of `"/safe_dir/.."` that would pass the validation step. However, the canonicalization process sees the double dot as a traversal to the parent directory and hence when canonized the path would become just `"/"`.

To avoid this problem, validation should occur after canonicalization takes place. In this case canonicalization occurs during the initialization of the File object. The code below fixes the issue.

Example Language: Java

(Good)

```

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

```

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:

Assume that the module `foo_bar` implements a protected register. The register content is the asset. Only transactions made by user id (indicated by signal `usr_id`) 0x4 are allowed to modify the register contents. The signal `grant_access` is used to provide access.

Example Language: Verilog

(Bad)

```

module foo_bar(data_out, usr_id, data_in, clk, rst_n);
output reg [7:0] data_out;
input wire [2:0] usr_id;
input wire [7:0] data_in;
input wire clk, rst_n;
wire grant_access;
always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        data_out = 0;
    else
        data_out = (grant_access) ? data_in : data_out;
        assign grant_access = (usr_id == 3'h4) ? 1'b1 : 1'b0;
end

```

endmodule

This code uses Verilog blocking assignments for data_out and grant_access. Therefore, these assignments happen sequentially (i.e., data_out is updated to new value first, and grant_access is updated the next cycle) and not in parallel. Therefore, the asset data_out is allowed to be modified even before the access control check is complete and grant_access signal is set. Since grant_access does not have a reset value, it will be meta-stable and will randomly go to either 0 or 1.

Flipping the order of the assignment of data_out and grant_access should solve the problem. The correct snippet of code is shown below.

Example Language: Verilog

(Good)





```
always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        data_out = 0;
    else
        assign grant_access = (usr_id == 3'h4) ? 1'b1 : 1'b0;
        data_out = (grant_access) ? data_in : data_out;
    end
endmodule
```

Observed Examples

Reference	Description
CVE-2019-9805	Chain: Creation of the packet client occurs before initialization is complete (CWE-696) resulting in a read from uninitialized memory (CWE-908), causing memory corruption. https://www.cve.org/CVERecord?id=CVE-2019-9805
CVE-2007-5191	file-system management programs call the setuid and setgid functions in the wrong order and do not check the return values, allowing attackers to gain unintended privileges https://www.cve.org/CVERecord?id=CVE-2007-5191
CVE-2007-1588	C++ web server program calls Process::setuid before calling Process::setgid, preventing it from dropping privileges, potentially allowing CGI programs to be called with higher privileges than intended https://www.cve.org/CVERecord?id=CVE-2007-1588
CVE-2022-37734	Chain: lexer in Java-based GraphQL server does not enforce maximum of tokens early enough (CWE-696), allowing excessive CPU consumption (CWE-1176) https://www.cve.org/CVERecord?id=CVE-2022-37734

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	2351
MemberOf		977	SFP Secondary Cluster: Design	888	2407
MemberOf		1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2463
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	POS36-C	CWE More Abstract	Observe correct revocation order while relinquishing privileges

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

CWE-697: Incorrect Comparison

Weakness ID : 697

Structure : Simple

Abstraction : Pillar

Description

The product compares two entities in a security-relevant context, but the comparison is incorrect, which may lead to resultant weaknesses.

Extended Description












This Pillar covers several possibilities:

- the comparison checks one factor incorrectly;
- the comparison should consider multiple factors, but it does not check at least one of those factors at all;
- the comparison checks the wrong factor.

Relationships

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	2575
ParentOf		183	Permissive List of Allowed Inputs	458
ParentOf		185	Incorrect Regular Expression	463
ParentOf		581	Object Model Violation: Just One of Equals and Hashcode Defined	1312
ParentOf		1023	Incomplete Comparison with Missing Factors	1865
ParentOf		1024	Comparison of Incompatible Types	1867
ParentOf		1025	Comparison Using Wrong Factors	1868
ParentOf		1039	Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	1873
ParentOf		1077	Floating Point Comparison with Incorrect Operator	1917
ParentOf		1254	Incorrect Comparison Logic Granularity	2060
CanFollow		481	Assigning instead of Comparing	1154

Weakness Ordinalities

Primary :

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Demonstrative Examples

Example 1:

Consider an application in which Truck objects are defined to be the same if they have the same make, the same model, and were manufactured in the same year.

Example Language: Java

(Bad)

```
public class Truck {
    private String make;
    private String model;
    private int year;
    public boolean equals(Object o) {
        if (o == null) return false;
        if (o == this) return true;
        if (!(o instanceof Truck)) return false;
        Truck t = (Truck) o;
        return (this.make.equals(t.getMake()) && this.model.equals(t.getModel()));
    }
}
```

Here, the equals() method only checks the make and model of the Truck objects, but the year of manufacture is not included.

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: C

(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) {
    if (strcmp(username, inUser, strlen(inUser))) {
        logEvent("Auth failure of username using strlen of inUser");
        return(AUTH_FAIL);
    }
    if (! strcmp(pass, inPass, strlen(inPass))) {
        logEvent("Auth success of password using strlen of inUser");
        return(AUTH_SUCCESS);
    }
    else {
        logEvent("Auth fail of password using sizeof");
        return(AUTH_FAIL);
    }
}
int main (int argc, char **argv) {
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult == AUTH_SUCCESS) {
        DoAuthenticatedTask(argv[1]);
    }
}
```

```

else {
    ExitError("Authentication failed");
}
}

```

In `AuthenticateUser()`, the `strcmp()` call uses the string length of an attacker-provided `inPass` parameter in order to determine how many characters to check in the password. So, if the attacker only provides a password of length 1, the check will only examine the first byte of the application's password before determining success.

As a result, this partial comparison leads to improper authentication (CWE-287).

Any of these passwords would still cause authentication to succeed for the "admin" user:

Example Language:

(Attack)

```

p
pa
pas
pass

```

This significantly reduces the search space for an attacker, making brute force attacks more feasible.

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







While this demonstrative example may not seem realistic, see the Observed Examples for CVE entries that effectively reflect this same weakness.

Observed Examples

Reference	Description
CVE-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
CVE-2020-15811	Chain: Proxy uses a substring search instead of parsing the Transfer-Encoding header (CWE-697), allowing request splitting (CWE-113) and cache poisoning https://www.cve.org/CVERecord?id=CVE-2020-15811
CVE-2016-10003	Proxy performs incorrect comparison of request headers, leading to infoleak https://www.cve.org/CVERecord?id=CVE-2016-10003

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	2350
MemberOf		883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2381
MemberOf		977	SFP Secondary Cluster: Design	888	2407
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2447
MemberOf		1397	Comprehensive Categorization: Comparison	1400	2523

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.

Maintenance

This entry likely has some relationships with case sensitivity (CWE-178), but case sensitivity is a factor in other types of weaknesses besides comparison. Also, in cryptography, certain attacks are possible when certain comparison operations do not take place in constant time, causing a timing-related information leak (CWE-208).

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
6	Argument Injection
7	Blind SQL Injection
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
15	Command Delimiters
24	Filter Failure through Buffer Overflow
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads
43	Exploiting Multiple Input Interpretation Layers
44	Overflow Binary Resource File
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
73	User-Controlled Filename
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
88	OS Command Injection
92	Forced Integer Overflow
120	Double Encoding
182	Flash Injection
267	Leverage Alternate Encoding

CWE-698: Execution After Redirect (EAR)

Weakness ID : 698

Structure : Simple

Abstraction : Base


Description

The web application sends a redirect to another location, but instead of exiting, it executes additional 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		670	Always-Incorrect Control Flow Implementation	1475
ChildOf		705	Incorrect Control Flow Scoping	1542

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2326

Weakness Ordinalities

Primary :

Alternate Terms

Redirect Without Exit :

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic	
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	<i>This weakness could affect the control flow of the application and allow execution of untrusted code.</i>	
Availability		

Detection Methods

Black Box

This issue might not be detected if testing is performed using a web browser, because the browser might obey the redirect and move the user to a different page before the application has produced outputs that indicate something is amiss.

Demonstrative Examples

Example 1:

This code queries a server and displays its status when a request comes from an authorized IP address.

Example Language: PHP

(Bad)

```
$requestingIP = $_SERVER['REMOTE_ADDR'];
if(!in_array($requestingIP,$ipAllowList)){
    echo "You are not authorized to view this page";
    http_redirect($errorPageURL);
}
$status = getServerStatus();
echo $status;
...
```

This code redirects unauthorized users, but continues to execute code after calling `http_redirect()`. This means even unauthorized users may be able to access the contents of the page or perform a DoS attack on the server being queried. Also, note that this code is vulnerable to an IP address spoofing attack (CWE-212).

Observed Examples

Reference	Description
CVE-2013-1402	Execution-after-redirect allows access to application configuration details. https://www.cve.org/CVERecord?id=CVE-2013-1402
CVE-2009-1936	chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal. https://www.cve.org/CVERecord?id=CVE-2009-1936
CVE-2007-2713	Remote attackers can obtain access to administrator functionality through EAR. https://www.cve.org/CVERecord?id=CVE-2007-2713
CVE-2007-4932	Remote attackers can obtain access to administrator functionality through EAR. https://www.cve.org/CVERecord?id=CVE-2007-4932
CVE-2007-5578	Bypass of authentication step through EAR. https://www.cve.org/CVERecord?id=CVE-2007-5578
CVE-2007-2713	Chain: Execution after redirect triggers eval injection. https://www.cve.org/CVERecord?id=CVE-2007-2713
CVE-2007-6652	chain: execution after redirect allows non-administrator to perform static code injection. https://www.cve.org/CVERecord?id=CVE-2007-6652

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	977	SFP Secondary Cluster: Design	888	2407
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

References

[REF-565]Adam Doupé, Bryce Boe, Christopher Kruegel and Giovanni Vigna. "Fear the EAR: Discovering and Mitigating Execution After Redirect Vulnerabilities". < <http://cs.ucsb.edu/~bboe/public/pubs/fear-the-ear-ccs2011.pdf> >.

CWE-703: Improper Check or Handling of Exceptional Conditions

Weakness ID : 703

Structure : Simple

Abstraction : Pillar

Description







The product does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of 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
MemberOf	V	1000	Research Concepts	2575

Nature	Type	ID	Name	Page
ParentOf		228	Improper Handling of Syntactically Invalid Structure	568
ParentOf		393	Return of Wrong Status Code	953
ParentOf		397	Declaration of Throws for Generic Exception	961
ParentOf		754	Improper Check for Unusual or Exceptional Conditions	1568
ParentOf		755	Improper Handling of Exceptional Conditions	1576
ParentOf		1384	Improper Handling of Physical or Environmental Conditions	2257




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

Nature	Type	ID	Name	Page
MemberOf		1012	Cross Cutting	2427

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

Nature	Type	ID	Name	Page
ParentOf		248	Uncaught Exception	596
ParentOf		391	Unchecked Error Condition	948
ParentOf		392	Missing Report of Error Condition	951

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

Nature	Type	ID	Name	Page
ParentOf		248	Uncaught Exception	596
ParentOf		391	Unchecked Error Condition	948
ParentOf		392	Missing Report of Error Condition	951

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Availability	DoS: Crash, Exit, or Restart	
Integrity	Unexpected State	

Detection Methods

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fault Injection - source code Fault Injection - binary Cost effective for partial coverage: Forced Path Execution

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: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Demonstrative Examples

Example 1:

Consider the following code segment:

Example Language: C

(Bad)

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when `fgets()` returns, `buf` will contain a null-terminated string of length 9 or less. But if an I/O error occurs, `fgets()` will not null-terminate `buf`. Furthermore, if the end of the file is reached before any characters are read, `fgets()` returns without writing anything to `buf`. In both of these situations, `fgets()` signals that something unusual has happened by returning `NULL`, but in this code, the warning will not be noticed. The lack of a null terminator in `buf` can result in a buffer overflow in the subsequent call to `strcpy()`.

Example 2:

The following 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.

Observed Examples

Reference	Description
[REF-1374]	Chain: JavaScript-based cryptocurrency library can fall back to the insecure <code>Math.random()</code> 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-2022-22224	Chain: an operating system does not properly process malformed Open Shortest Path First (OSPF) Type/Length/Value Identifiers (TLV) (CWE-703), which can cause the process to enter an infinite loop (CWE-835) https://www.cve.org/CVERecord?id=CVE-2022-22224

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	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2365
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2379
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2399
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2448
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Notes

Relationship

This is a high-level class that might have some overlap with other classes. It could be argued that even "normal" weaknesses such as buffer overflows involve unusual or exceptional conditions. In that sense, this might be an inherent aspect of most other weaknesses within CWE, similar to API Abuse (CWE-227) and Indicator of Poor Code Quality (CWE-398). However, this entry is currently intended to unify disparate concepts that do not have other places within the Research Concepts view (CWE-1000).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	ERR06-J		Do not throw undeclared checked exceptions

References

[REF-567]Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995 August 1. < <http://ftp.cerias.purdue.edu/pub/papers/taimur-aslam/aslam-taxonomy-mstthesis.pdf> >.

[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.

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

[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-704: Incorrect Type Conversion or Cast

Weakness ID : 704

Structure : Simple

Abstraction : Class

Description

The product does not correctly convert an object, resource, or structure from one type to a different 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, 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	1454
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	1323
ParentOf	B	681	Incorrect Conversion between Numeric Types	1495
ParentOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1776
ParentOf	B	1389	Incorrect Parsing of Numbers with Different Radices	2263

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

Nature	Type	ID	Name	Page
ParentOf	B	681	Incorrect Conversion between Numeric Types	1495
ParentOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1776

Applicable Platforms

Language : C (Prevalence = Often)

Language : C++ (Prevalence = Often)

Language : Not Language-Specific (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Other	Other	

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 this example, depending on the return value of `accecssmainframe()`, the variable `amount` can hold a negative value when it is returned. Because the function is declared to return an unsigned value, `amount` will be implicitly cast to an unsigned number.

Example Language: C

(Bad)

```
unsigned int readdata () {
    int amount = 0;
    ...
}
```

```

    amount = accessmainframe();
    ...
    return amount;
}

```

If the return value of `accessmainframe()` is -1, then the return value of `readdata()` will be 4,294,967,295 on a system that uses 32-bit integers.

Example 2:

The following code uses a union to support the representation of different types of messages. It formats messages differently, depending on their type.

Example Language: C

(Bad)

```

#define NAME_TYPE 1
#define ID_TYPE 2
struct MessageBuffer
{
    int msgType;
    union {
        char *name;
        int nameID;
    };
};

int main (int argc, char **argv) {
    struct MessageBuffer buf;
    char *defaultMessage = "Hello World";
    buf.msgType = NAME_TYPE;
    buf.name = defaultMessage;
    printf("Pointer of buf.name is %p\n", buf.name);
    /* This particular value for nameID is used to make the code architecture-independent. If coming from untrusted input, it
    could be any value. */
    buf.nameID = (int)(defaultMessage + 1);
    printf("Pointer of buf.name is now %p\n", buf.name);
    if (buf.msgType == NAME_TYPE) {
        printf("Message: %s\n", buf.name);
    }
    else {
        printf("Message: Use ID %d\n", buf.nameID);
    }
}

```

The code intends to process the message as a `NAME_TYPE`, and sets the default message to "Hello World." However, since both `buf.name` and `buf.nameID` are part of the same union, they can act as aliases for the same memory location, depending on memory layout after compilation.

As a result, modification of `buf.nameID` - an `int` - can effectively modify the pointer that is stored in `buf.name` - a string.

Execution of the program might generate output such as:

```

Pointer of name is 10830
Pointer of name is now 10831
Message: ello World

```

Notice how the pointer for `buf.name` was changed, even though `buf.name` was not explicitly modified.

In this case, the first "H" character of the message is omitted. However, if an attacker is able to fully control the value of `buf.nameID`, then `buf.name` could contain an arbitrary pointer, leading to out-of-bounds reads or writes.

Observed Examples

Reference	Description
CVE-2021-43537	Chain: in a web browser, an unsigned 64-bit integer is forcibly cast to a 32-bit integer (CWE-681) and potentially leading to an integer overflow (CWE-190). If an integer overflow occurs, this can cause heap memory corruption (CWE-122) https://www.cve.org/CVERecord?id=CVE-2021-43537
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

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	2341
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2344
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2350
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2376
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2381
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2456
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	2458
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP05-C		Do not cast away a const qualification
CERT C Secure Coding	EXP39-C	CWE More Abstract	Do not access a variable through a pointer of an incompatible type
CERT C Secure Coding	INT31-C	CWE More Abstract	Ensure that integer conversions do not result in lost or misinterpreted data
CERT C Secure Coding	INT36-C	CWE More Abstract	Converting a pointer to integer or integer to pointer
CERT C Secure Coding	STR34-C	CWE More Abstract	Cast characters to unsigned types before converting to larger integer sizes

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	STR37-C	CWE More Abstract	Arguments to character handling functions must be representable as an unsigned char
Software Fault Patterns	SFP1		Glitch in computation
OMG ASCRM	ASCRM-CWE-704		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < <http://www.omg.org/spec/ASCRM/1.0/> >.

CWE-705: Incorrect Control Flow Scoping

Weakness ID : 705

Structure : Simple

Abstraction : Class

Description

The product does not properly return control flow to the proper location after it has completed a task or detected an unusual condition.

Relationships

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	1517
ParentOf	B	248	Uncaught Exception	596
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	933
ParentOf	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	957
ParentOf	B	396	Declaration of Catch for Generic Exception	959
ParentOf	B	397	Declaration of Throws for Generic Exception	961
ParentOf	B	455	Non-exit on Failed Initialization	1087
ParentOf	B	584	Return Inside Finally Block	1317
ParentOf	B	698	Execution After Redirect (EAR)	1533

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic Other	

Demonstrative Examples

Example 1:

The following example attempts to resolve a hostname.

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());
}
```

A DNS lookup failure will cause the Servlet to throw an exception.

Example 2:

This code queries a server and displays its status when a request comes from an authorized IP address.

Example Language: PHP

(Bad)

```
$requestingIP = $_SERVER['REMOTE_ADDR'];
if(!in_array($requestingIP,$ipAllowList)){
    echo "You are not authorized to view this page";
    http_redirect($errorPageURL);
}
$status = getServerStatus();
echo $status;
...
```

This code redirects unauthorized users, but continues to execute code after calling `http_redirect()`. This means even unauthorized users may be able to access the contents of the page or perform a DoS attack on the server being queried. Also, note that this code is vulnerable to an IP address spoofing attack (CWE-212).

Example 3:

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);
    }
}
```

Observed Examples

Reference	Description
CVE-2023-21087	Java code in a smartphone OS can encounter a "boot loop" due to an uncaught exception https://www.cve.org/CVERecord?id=CVE-2023-21087
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		744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	2348
MemberOf		746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2350
MemberOf		851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2365
MemberOf		854	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)	844	2367
MemberOf		878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2378
MemberOf		880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2379
MemberOf		977	SFP Secondary Cluster: Design	888	2407
MemberOf		1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2448
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf		1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	2460
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ENV32-C	CWE More Abstract	All exit handlers must return normally
CERT C Secure Coding	ERR04-C		Choose an appropriate termination strategy
The CERT Oracle Secure Coding Standard for Java (2011)	THI05-J		Do not use Thread.stop() to terminate threads
The CERT Oracle Secure Coding Standard for Java (2011)	ERR04-J		Do not complete abruptly from a finally block
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

CWE-706: Use of Incorrectly-Resolved Name or Reference

Weakness ID : 706

Structure : Simple

Abstraction : Class












Description

The product uses a name or reference to access a resource, but the name/reference resolves to a resource that is outside of the intended control sphere.




Relationships

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	1454
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
ParentOf		41	Improper Resolution of Path Equivalence	86
ParentOf		59	Improper Link Resolution Before File Access ('Link Following')	111
ParentOf		66	Improper Handling of File Names that Identify Virtual Resources	124
ParentOf		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	236
ParentOf		178	Improper Handling of Case Sensitivity	445
ParentOf		386	Symbolic Name not Mapping to Correct Object	942
ParentOf		827	Improper Control of Document Type Definition	1736
PeerOf		99	Improper Control of Resource Identifiers ('Resource Injection')	243
PeerOf		99	Improper Control of Resource Identifiers ('Resource Injection')	243

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

Nature	Type	ID	Name	Page
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
ParentOf		59	Improper Link Resolution Before File Access ('Link Following')	111
ParentOf		178	Improper Handling of Case Sensitivity	445

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	

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		932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	2390
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2409

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
48	Passing Local Filenames to Functions That Expect a URL
159	Redirect Access to Libraries
177	Create files with the same name as files protected with a higher classification
641	DLL Side-Loading

CWE-707: Improper Neutralization

Weakness ID : 707

Structure : Simple

Abstraction : Pillar

Description

The product does not ensure or incorrectly ensures that structured messages or data are well-formed and that certain security properties are met before being read from an upstream component or sent to a downstream component.

Extended Description

If a message is malformed, it may cause the message to be incorrectly interpreted.

Neutralization is an abstract term for any technique that ensures that input (and output) conforms with expectations and is "safe." This can be done by:

- checking that the input/output is already "safe" (e.g. validation)
- transformation of the input/output to be "safe" using techniques such as filtering, encoding/decoding, escaping/unescaping, quoting/unquoting, or canonicalization
- preventing the input/output from being directly provided by an attacker (e.g. "indirect selection" that maps externally-provided values to internally-controlled values)
- preventing the input/output from being processed at all









This weakness typically applies in cases where the product prepares a control message that another process must act on, such as a command or query, and malicious input that was intended as data, can enter the control plane instead. However, this weakness also applies to more general cases where there are not always control implications.

Relationships


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	V	1000	Research Concepts	2575
ParentOf	G	20	Improper Input Validation	20

Nature	Type	ID	Name	Page
ParentOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf		116	Improper Encoding or Escaping of Output	281
ParentOf		138	Improper Neutralization of Special Elements	373
ParentOf		170	Improper Null Termination	428
ParentOf		172	Encoding Error	433
ParentOf		228	Improper Handling of Syntactically Invalid Structure	568
ParentOf		240	Improper Handling of Inconsistent Structural Elements	583
ParentOf		463	Deletion of Data Structure Sentinel	1105

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2434

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*)

Common Consequences

Scope	Impact	Likelihood
Other	Other	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2413
MemberOf		1370	ICS Supply Chain: Common Mode Frailties	1358	2507
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2532

Notes

Maintenance

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
7	Blind SQL Injection
43	Exploiting Multiple Input Interpretation Layers
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
83	XPath Injection
84	XQuery Injection

CAPEC-ID	Attack Pattern Name
250	XML Injection
276	Inter-component Protocol Manipulation
277	Data Interchange Protocol Manipulation
278	Web Services Protocol Manipulation
279	SOAP Manipulation
468	Generic Cross-Browser Cross-Domain Theft

CWE-708: Incorrect Ownership Assignment

Weakness ID : 708

Structure : Simple

Abstraction : Base

Description

The product assigns an owner to a resource, but the owner is outside of the intended control sphere.

Extended Description

This may allow the resource to be manipulated by actors outside of the intended control sphere.

Relationships

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		282	Improper Ownership Management	676
CanAlsoBe		345	Insufficient Verification of Data Authenticity	851

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf		840	Business Logic Errors	2360

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	
<i>An attacker could read and modify data for which they do not have permissions to access directly.</i>		

Potential Mitigations

Phase: Policy

Periodically review the privileges and their owners.

Phase: Testing




Use automated tools to check for privilege settings.

Observed Examples

Reference	Description
CVE-2007-5101	File system sets wrong ownership and group when creating a new file. https://www.cve.org/CVERecord?id=CVE-2007-5101
CVE-2007-4238	OS installs program with bin owner/group, allowing modification. https://www.cve.org/CVERecord?id=CVE-2007-4238
CVE-2007-1716	Manager does not properly restore ownership of a reusable resource when a user logs out, allowing privilege escalation. https://www.cve.org/CVERecord?id=CVE-2007-1716
CVE-2005-3148	Backup software restores symbolic links with incorrect uid/gid. https://www.cve.org/CVERecord?id=CVE-2005-3148
CVE-2005-1064	Product changes the ownership of files that a symlink points to, instead of the symlink itself. https://www.cve.org/CVERecord?id=CVE-2005-1064
CVE-2011-1551	Component assigns ownership of sensitive directory tree to a user account, which can be leveraged to perform privileged operations. https://www.cve.org/CVERecord?id=CVE-2011-1551

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2335
MemberOf		884	CWE Cross-section	884	2567
MemberOf		944	SFP Secondary Cluster: Access Management	888	2393
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes**Maintenance**

This overlaps verification errors, permissions, and privileges. A closely related weakness is the incorrect assignment of groups to a resource. It is not clear whether it would fall under this entry or require a different entry.

CWE-710: Improper Adherence to Coding Standards

Weakness ID : 710

Structure : Simple

Abstraction : Pillar













Description

The product does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf		1000	Research Concepts	2575
ParentOf		476	NULL Pointer Dereference	1132
ParentOf		477	Use of Obsolete Function	1138
ParentOf		484	Omitted Break Statement in Switch	1162
ParentOf		489	Active Debug Code	1171
ParentOf		570	Expression is Always False	1292
ParentOf		571	Expression is Always True	1295
ParentOf		573	Improper Following of Specification by Caller	1298
ParentOf		657	Violation of Secure Design Principles	1446
ParentOf		684	Incorrect Provision of Specified Functionality	1505
ParentOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1582
ParentOf		1041	Use of Redundant Code	1875
ParentOf		1044	Architecture with Number of Horizontal Layers Outside of Expected Range	1879
ParentOf		1048	Invokable Control Element with Large Number of Outward Calls	1883
ParentOf		1059	Insufficient Technical Documentation	1894
ParentOf		1061	Insufficient Encapsulation	1898
ParentOf		1065	Runtime Resource Management Control Element in a Component Built to Run on Application Servers	1903
ParentOf		1066	Missing Serialization Control Element	1904
ParentOf		1068	Inconsistency Between Implementation and Documented Design	1906
ParentOf		1076	Insufficient Adherence to Expected Conventions	1916
ParentOf		1092	Use of Same Invokable Control Element in Multiple Architectural Layers	1932
ParentOf		1093	Excessively Complex Data Representation	1933
ParentOf		1101	Reliance on Runtime Component in Generated Code	1941
ParentOf		1120	Excessive Code Complexity	1960
ParentOf		1126	Declaration of Variable with Unnecessarily Wide Scope	1966
ParentOf		1127	Compilation with Insufficient Warnings or Errors	1966
ParentOf		1164	Irrelevant Code	1967
ParentOf		1177	Use of Prohibited Code	1972
ParentOf		1209	Failure to Disable Reserved Bits	1991
ParentOf		1357	Reliance on Insufficiently Trustworthy Component	2254

Applicable Platforms**Language** : Not Language-Specific (*Prevalence = Undetermined*)**Technology** : Not Technology-Specific (*Prevalence = Undetermined*)**Common Consequences**

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations**Phase: Implementation**






Document and closely follow coding standards.

Phase: Testing**Phase: Implementation**

Where possible, use automated tools to enforce the standards.

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	2408
MemberOf		1370	ICS Supply Chain: Common Mode Frailties	1358	2507
MemberOf		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2511
MemberOf		1383	ICS Operations (& Maintenance): Compliance/Conformance with Regulatory Requirements	1358	2517
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-732: Incorrect Permission Assignment for Critical Resource

Weakness ID : 732

Structure : Simple

Abstraction : Class

Description

The product specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.










Extended Description

When a resource is given a permission setting that provides access to a wider range of actors than required, it could lead to the exposure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution, or sensitive user data. For example, consider a misconfigured storage account for the cloud that can be read or written by a public or anonymous 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		668	Exposure of Resource to Wrong Sphere	1469
ChildOf		285	Improper Authorization	684
ParentOf		276	Incorrect Default Permissions	665
ParentOf		277	Insecure Inherited Permissions	668
ParentOf		278	Insecure Preserved Inherited Permissions	669
ParentOf		279	Incorrect Execution-Assigned Permissions	671
ParentOf		281	Improper Preservation of Permissions	674
ParentOf		766	Critical Data Element Declared Public	1607
ParentOf		1004	Sensitive Cookie Without 'HttpOnly' Flag	1854

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

Nature	Type	ID	Name	Page
ParentOf	B	276	Incorrect Default Permissions	665
ParentOf	B	281	Improper Preservation of Permissions	674

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

Applicable Platforms

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

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

Technology : Cloud Computing (*Prevalence = Often*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories <i>An attacker may be able to read sensitive information from the associated resource, such as credentials or configuration information stored in a file.</i>	
Access Control	Gain Privileges or Assume Identity <i>An attacker may be able to modify critical properties of the associated resource to gain privileges, such as replacing a world-writable executable with a Trojan horse.</i>	
Integrity Other	Modify Application Data Other <i>An attacker may be able to destroy or corrupt critical data in the associated resource, such as deletion of records from a database.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc. Automated techniques may be able to detect the use of library functions that modify permissions, then analyze function calls for arguments that contain potentially insecure values. However, since the software's intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated static analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes. When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated static analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

Automated Dynamic Analysis

Automated dynamic analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc. However, since the software's intended security policy might allow loose permissions for certain operations

(such as publishing a file on a web server), automated dynamic analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes. When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated dynamic analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Manual Static Analysis

Manual static analysis may be effective in detecting the use of custom permissions models and functions. The code could then be examined to identifying usage of the related functions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Manual Dynamic Analysis

Manual dynamic analysis may be effective in detecting the use of custom permissions models and functions. The program could then be executed with a focus on exercising code paths that are related to the custom permissions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Fuzzing

Fuzzing is not effective in detecting this weakness.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and watch for library functions or system calls on OS resources such as files, directories, and shared memory. Examine the arguments to these calls to infer which permissions are being used.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inter-application Flow Analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria 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: Host Application Interface Scanner Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Forced Path Execution

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: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations**Phase: Implementation**

When using a critical resource such as a configuration file, check to see if the resource has insecure permissions (such as being modifiable by any regular user) [REF-62], and generate an error or even exit the software if there is a possibility that the resource could have been modified by an unauthorized party.

Phase: Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully defining distinct user groups, privileges, and/or roles. Map these against data, functionality, and the related resources. Then set the permissions accordingly. This will allow you to maintain more fine-grained control over your resources. [REF-207]

Effectiveness = Moderate

This can be an effective strategy. However, in practice, it may be difficult or time consuming to define these areas when there are many different resources or user types, or if the applications features change rapidly.

Phase: Architecture and Design**Phase: Operation**

Strategy = Sandbox or Jail

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

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

Effectiveness = Limited

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

Phase: Implementation

Phase: Installation

During program startup, explicitly set the default permissions or umask to the most restrictive setting possible. Also set the appropriate permissions during program installation. This will prevent you from inheriting insecure permissions from any user who installs or runs the program.

Effectiveness = High

Phase: System Configuration

For all configuration files, executables, and libraries, make sure that they are only readable and writable by the software's administrator.

Effectiveness = High

Phase: Documentation

Do not suggest insecure configuration changes in documentation, especially if those configurations can extend to resources and other programs that are outside the scope of the application.

Phase: Installation

Do not assume that a system administrator will manually change the configuration to the settings that are recommended in the software's manual.

Phase: Operation

Phase: System Configuration

Strategy = Environment Hardening

Ensure that the software runs properly under the United States Government Configuration Baseline (USGCB) [REF-199] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Phase: Implementation

Phase: System Configuration

Phase: Operation

When storing data in the cloud (e.g., S3 buckets, Azure blobs, Google Cloud Storage, etc.), use the provider's controls to disable public access.

Demonstrative Examples

Example 1:

The following code sets the umask of the process to 0 before creating a file and writing "Hello world" into the file.

Example Language: C

(Bad)

```
#define OUTFILE "hello.out"
umask(0);
```



```
FILE *out;
/* Ignore link following (CWE-59) for brevity */
out = fopen(OUTFILE, "w");
if (out) {
    fprintf(out, "hello world!\n");
    fclose(out);
}
```

After running this program on a UNIX system, running the "ls -l" command might return the following output:

Example Language:

(Result)

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 hello.out
```

The "rw-rw-rw-" string indicates that the owner, group, and world (all users) can read the file and write to it.

Example 2:

This code creates a home directory for a new user, and makes that user the owner of the directory. If the new directory cannot be owned by the user, the directory is deleted.

Example Language: PHP

(Bad)

```
function createUserDir($username){
    $path = '/home/'.$username;
    if(!mkdir($path)){
        return false;
    }
    if(!chown($path,$username)){
        rmdir($path);
        return false;
    }
    return true;
}
```

Because the optional "mode" argument is omitted from the call to mkdir(), the directory is created with the default permissions 0777. Simply setting the new user as the owner of the directory does not explicitly change the permissions of the directory, leaving it with the default. This default allows any user to read and write to the directory, allowing an attack on the user's files. The code also fails to change the owner group of the directory, which may result in access by unexpected groups.

This code may also be vulnerable to Path Traversal (CWE-22) attacks if an attacker supplies a non alphanumeric username.

Example 3:

The following code snippet might be used as a monitor to periodically record whether a web site is alive. To ensure that the file can always be modified, the code uses chmod() to make the file world-writable.

Example Language: Perl

(Bad)

```
$fileName = "secretFile.out";
if (-e $fileName) {
    chmod 0777, $fileName;
}
my $outFH;
if (!open($outFH, ">>$fileName")) {
    ExitError("Couldn't append to $fileName: $!");
}
my $dateString = FormatCurrentTime();
my $status = IsHostAlive("cwe.mitre.org");
print $outFH "$dateString cwe status: $status!\n";
```

```
close($outFH);
```

The first time the program runs, it might create a new file that inherits the permissions from its environment. A file listing might look like:

Example Language:

(Result)

```
-rw-r--r-- 1 username 13 Nov 24 17:58 secretFile.out
```

This listing might occur when the user has a default umask of 022, which is a common setting. Depending on the nature of the file, the user might not have intended to make it readable by everyone on the system.

The next time the program runs, however - and all subsequent executions - the chmod will set the file's permissions so that the owner, group, and world (all users) can read the file and write to it:

Example Language:

(Result)

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 secretFile.out
```

Perhaps the programmer tried to do this because a different process uses different permissions that might prevent the file from being updated.

Example 4:

This program creates and reads from an admin file to determine privilege information.

If the admin file doesn't exist, the program will create one. In order to create the file, the program must have write privileges to write to the file. After the file is created, the permissions need to be changed to read only.

Example Language: Go

(Bad)

```
const adminFile = "/etc/admin-users"
func createAdminFileIfNotExists() error {
    file, err := os.Create(adminFile)
    if err != nil {
        return err
    }
    return nil
}
func changeModeOfAdminFile() error {
    fileMode := os.FileMode(0440)
    if err := os.Chmod(adminFile, fileMode); err != nil {
        return err
    }
    return nil
}
```

os.Create will create a file with 0666 permissions before umask if the specified file does not exist. A typical umask of 0022 would result in the file having 0644 permissions. That is, the file would have world-writable and world-readable permissions.

In this scenario, it is advised to use the more customizable method of os.OpenFile with the os.O_WRONLY and os.O_CREATE flags specifying 0640 permissions to create the admin file.

This is because on a typical system where the umask is 0022, the perm 0640 applied in os.OpenFile will result in a file of 0620 where only the owner and group can write.

Example 5:

The following command recursively sets world-readable permissions for a directory and all of its children:

*Example Language: Shell**(Bad)*

```
chmod -R ugo+r DIRNAME
```

If this command is run from a program, the person calling the program might not expect that all the files under the directory will be world-readable. If the directory is expected to contain private data, this could become a security problem.

Example 6:

The following Azure command updates the settings for a storage account:

*Example Language: Shell**(Bad)*

```
az storage account update --name <storage-account> --resource-group <resource-group> --allow-blob-public-access true
```

However, "Allow Blob Public Access" is set to true, meaning that anonymous/public users can access blobs.

The command could be modified to disable "Allow Blob Public Access" by setting it to false.

*Example Language: Shell**(Good)*

```
az storage account update --name <storage-account> --resource-group <resource-group> --allow-blob-public-access false
```

Example 7:

The following Google Cloud Storage command gets the settings for a storage account named 'BUCKET_NAME':

*Example Language: Shell**(Informative)*

```
gsutil iam get gs://BUCKET_NAME
```

Suppose the command returns the following result:

*Example Language: JSON**(Bad)*

```
{
  "bindings": [{
    "members": [
      "projectEditor: PROJECT-ID",
      "projectOwner: PROJECT-ID"
    ],
    "role": "roles/storage.legacyBucketOwner"
  },
  {
    "members": [
      "allUsers",
      "projectViewer: PROJECT-ID"
    ],
    "role": "roles/storage.legacyBucketReader"
  }
  ]
}
```

This result includes the "allUsers" or IAM role added as members, causing this policy configuration to allow public access to cloud storage resources. There would be a similar concern if "allAuthenticatedUsers" was present.

The command could be modified to remove "allUsers" and/or "allAuthenticatedUsers" as follows:

Example Language: Shell

(Good)

```
gsutil iam ch -d allUsers gs://BUCKET_NAME
gsutil iam ch -d allAuthenticatedUsers gs://BUCKET_NAME
```

Observed Examples

Reference	Description
CVE-2022-29527	Go application for cloud management creates a world-writable sudoers file that allows local attackers to inject sudo rules and escalate privileges to root by winning a race condition. https://www.cve.org/CVERecord?id=CVE-2022-29527
CVE-2009-3482	Anti-virus product sets insecure "Everyone: Full Control" permissions for files under the "Program Files" folder, allowing attackers to replace executables with Trojan horses. https://www.cve.org/CVERecord?id=CVE-2009-3482
CVE-2009-3897	Product creates directories with 0777 permissions at installation, allowing users to gain privileges and access a socket used for authentication. https://www.cve.org/CVERecord?id=CVE-2009-3897
CVE-2009-3489	Photo editor installs a service with an insecure security descriptor, allowing users to stop or start the service, or execute commands as SYSTEM. https://www.cve.org/CVERecord?id=CVE-2009-3489
CVE-2020-15708	socket created with insecure permissions https://www.cve.org/CVERecord?id=CVE-2020-15708
CVE-2009-3289	Library function copies a file to a new target and uses the source file's permissions for the target, which is incorrect when the source file is a symbolic link, which typically has 0777 permissions. https://www.cve.org/CVERecord?id=CVE-2009-3289
CVE-2009-0115	Device driver uses world-writable permissions for a socket file, allowing attackers to inject arbitrary commands. https://www.cve.org/CVERecord?id=CVE-2009-0115
CVE-2009-1073	LDAP server stores a cleartext password in a world-readable file. https://www.cve.org/CVERecord?id=CVE-2009-1073
CVE-2009-0141	Terminal emulator creates TTY devices with world-writable permissions, allowing an attacker to write to the terminals of other users. https://www.cve.org/CVERecord?id=CVE-2009-0141
CVE-2008-0662	VPN product stores user credentials in a registry key with "Everyone: Full Control" permissions, allowing attackers to steal the credentials. https://www.cve.org/CVERecord?id=CVE-2008-0662
CVE-2008-0322	Driver installs its device interface with "Everyone: Write" permissions. https://www.cve.org/CVERecord?id=CVE-2008-0322
CVE-2009-3939	Driver installs a file with world-writable permissions. https://www.cve.org/CVERecord?id=CVE-2009-3939
CVE-2009-3611	Product changes permissions to 0777 before deleting a backup; the permissions stay insecure for subsequent backups. https://www.cve.org/CVERecord?id=CVE-2009-3611
CVE-2007-6033	Product creates a share with "Everyone: Full Control" permissions, allowing arbitrary program execution. https://www.cve.org/CVERecord?id=CVE-2007-6033
CVE-2007-5544	Product uses "Everyone: Full Control" permissions for memory-mapped files (shared memory) in inter-process communication, allowing attackers to tamper with a session. https://www.cve.org/CVERecord?id=CVE-2007-5544
CVE-2005-4868	Database product uses read/write permissions for everyone for its shared memory, allowing theft of credentials.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2005-4868
CVE-2004-1714	Security product uses "Everyone: Full Control" permissions for its configuration files.
	https://www.cve.org/CVERecord?id=CVE-2004-1714
CVE-2001-0006	"Everyone: Full Control" permissions assigned to a mutex allows users to disable network connectivity.
	https://www.cve.org/CVERecord?id=CVE-2001-0006
CVE-2002-0969	Chain: database product contains buffer overflow that is only reachable through a .ini configuration file - which has "Everyone: Full Control" permissions.
	https://www.cve.org/CVERecord?id=CVE-2002-0969

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	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2347
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	2353
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	2355
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	2358
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2369
MemberOf	C	860	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)	844	2370
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2372
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	2394
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)	1133	2452
MemberOf	C	1150	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)	1133	2452
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	FIO03-J		Create files with appropriate access permission
The CERT Oracle Secure Coding Standard for Java (2011)	SEC01-J		Do not allow tainted variables in privileged blocks
The CERT Oracle Secure Coding Standard for Java (2011)	ENV03-J		Do not grant dangerous combinations of permissions
CERT C Secure Coding	FIO06-C		Create files with appropriate access permissions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
17	Using Malicious Files
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery
122	Privilege Abuse
127	Directory Indexing
180	Exploiting Incorrectly Configured Access Control Security Levels
206	Signing Malicious Code
234	Hijacking a privileged process
642	Replace Binaries

References

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

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

[REF-594]Jason Lam. "Top 25 Series - Rank 21 - Incorrect Permission Assignment for Critical Response". 2010 March 4. SANS Software Security Institute. < <http://software-security.sans.org/blog/2010/03/24/top-25-series-rank-21-incorrect-permission-assignment-for-critical-response> >.

[REF-199]NIST. "United States Government Configuration Baseline (USGCB)". < <https://csrc.nist.gov/Projects/United-States-Government-Configuration-Baseline> >.2023-03-28.

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

[REF-1307]Center for Internet Security. "CIS Microsoft Azure Foundations Benchmark version 1.5.0". 2022 August 6. < <https://www.cisecurity.org/benchmark/azure> >.2023-01-19.

[REF-1327]Center for Internet Security. "CIS Google Cloud Computing Platform Benchmark version 1.3.0". 2022 March 1. < https://www.cisecurity.org/benchmark/google_cloud_computing_platform >.2023-04-24.

CWE-733: Compiler Optimization Removal or Modification of Security-critical Code

Weakness ID : 733

Structure : Simple

Abstraction : Base



Description

The developer builds a security-critical protection mechanism into the software, but the compiler optimizes the program such that the mechanism is removed or modified.

Relationships

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		1038	Insecure Automated Optimizations	1872
ParentOf		14	Compiler Removal of Code to Clear Buffers	14

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2326

Applicable Platforms

Language : C (Prevalence = Often)

Language : C++ (Prevalence = Often)

Language : Compiled (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	
Other	Other	

Detection Methods

Black Box

This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box

This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Demonstrative Examples

Example 1:

The following code reads a password from the user, uses the password to connect to a back-end mainframe and then attempts to scrub the password from memory using memset().

Example Language: C

(Bad)

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

The code in the example will behave correctly if it is executed verbatim, but if the code is compiled using an optimizing compiler, such as Microsoft Visual C++ .NET or GCC 3.x, then the call to memset() will be removed as a dead store because the buffer pwd is not used after its value is overwritten [18]. Because the buffer pwd contains a sensitive value, the application may be vulnerable to attack if the data are left memory resident. If attackers are able to access the correct region of memory, they may use the recovered password to gain control of the system.

It is common practice to overwrite sensitive data manipulated in memory, such as passwords or cryptographic keys, in order to prevent attackers from learning system secrets. However, with the advent of optimizing compilers, programs do not always behave as their source code alone would suggest. In the example, the compiler interprets the call to memset() as dead code because the memory being written to is not subsequently used, despite the fact that there is clearly a security motivation for the operation to occur. The problem here is that many compilers, and in fact many programming languages, do not take this and other security concerns into consideration in their efforts to improve efficiency.

Attackers typically exploit this type of vulnerability by using a core dump or runtime mechanism to access the memory used by a particular application and recover the secret information. Once an attacker has access to the secret information, it is relatively straightforward to further exploit the system and possibly compromise other resources with which the application interacts.

Observed Examples

Reference	Description
CVE-2008-1685	C compiler optimization, as allowed by specifications, removes code that is used to perform checks to detect integer overflows. https://www.cve.org/CVERecord?id=CVE-2008-1685
CVE-2019-1010006	Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787). https://www.cve.org/CVERecord?id=CVE-2019-1010006

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		976	SFP Secondary Cluster: Compiler	888	2407
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2524

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
8	Buffer Overflow in an API Call

CAPEC-ID Attack Pattern Name

9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
24	Filter Failure through Buffer Overflow
46	Overflow Variables and Tags

References

[REF-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-749: Exposed Dangerous Method or Function**Weakness ID :** 749**Structure :** Simple**Abstraction :** Base**Description**

The product provides an Applications Programming Interface (API) or similar interface for interaction with external actors, but the interface includes a dangerous method or function that is not properly restricted.

Extended Description

This weakness can lead to a wide variety of resultant weaknesses, depending on the behavior of the exposed method. It can apply to any number of technologies and approaches, such as ActiveX controls, Java functions, IOCTLs, and so on.

The exposure can occur in a few different ways:

- The function/method was never intended to be exposed to outside actors.
- The function/method was only intended to be accessible to a limited set of actors, such as Internet-based access from a single web site.

Relationships

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	284	Improper Access Control	680
ParentOf	⚡	618	Exposed Unsafe ActiveX Method	1380
ParentOf	⚡	782	Exposed IOCTL with Insufficient Access Control	1648

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

Nature	Type	ID	Name	Page
MemberOf	⚡	1228	API / Function Errors	2482

Weakness Ordinalities

Primary :

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Gain Privileges or Assume Identity	
Confidentiality	Read Application Data	
Availability	Modify Application Data	
Access Control	Execute Unauthorized Code or Commands	
Other	Other	
<i>Exposing critical functionality essentially provides an attacker with the privilege level of the exposed functionality. This could result in the modification or exposure of sensitive data or possibly even execution of arbitrary code.</i>		

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

If you must expose a method, make sure to perform input validation on all arguments, limit access to authorized parties, and protect against all possible vulnerabilities.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Identify all exposed functionality. Explicitly list all functionality that must be exposed to some user or set of users. Identify which functionality may be: accessible to all users restricted to a small set of privileged users prevented from being directly accessible at all Ensure that the implemented code follows these expectations. This includes setting the appropriate access modifiers where applicable (public, private, protected, etc.) or not marking ActiveX controls safe-for-scripting.

Demonstrative Examples

Example 1:

In the following Java example the method removeDatabase will delete the database with the name specified in the input parameter.

Example Language: Java

(Bad)

```
public void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
    } catch (SQLException ex) {...}
}
```

The method in this example is declared public and therefore is exposed to any class in the application. Deleting a database should be considered a critical operation within an application and access to this potentially dangerous method should be restricted. Within Java this can be accomplished simply by declaring the method private thereby exposing it only to the enclosing class as in the following example.

Example Language: Java

(Good)

```
private void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
    } catch (SQLException ex) {...}
}
```

Example 2:

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

Example Language: Java

(Bad)

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

Example Language: Objective-C

(Bad)

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

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

Example Language: JavaScript

(Attack)

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

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

Example 3:

This application uses a WebView to display websites, and creates a Javascript interface to a Java object to allow enhanced functionality on a trusted website:

Example Language: Java

(Bad)

```
public class WebViewGUI extends Activity {
    WebView mainWebView;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mainWebView = new WebView(this);
        mainWebView.getSettings().setJavaScriptEnabled(true);
        mainWebView.addJavascriptInterface(new JavaScriptInterface(), "userInfoObject");
        mainWebView.loadUrl("file:///android_asset/www/index.html");
        setContentView(mainWebView);
    }
    final class JavaScriptInterface {
        JavaScriptInterface () {}
        public String getUserInfo() {
            return currentUser.Info();
        }
    }
}
```

Before Android 4.2 all methods, including inherited ones, are exposed to Javascript when using `addJavascriptInterface()`. This means that a malicious website loaded within this WebView can use reflection to acquire a reference to arbitrary Java objects. This will allow the website code to perform any action the parent application is authorized to.

For example, if the application has permission to send text messages:

Example Language: JavaScript

(Attack)

```
<script>
    userInfoObject.getClass().forName('android.telephony.SmsManager').getMethod('getDefault',null).sendTextMessage(attackNumber,
    null, attackMessage, null, null);
</script>
```

This malicious script can use the `userInfoObject` object to load the `SmsManager` object and send arbitrary text messages to any recipient.

Example 4:

After Android 4.2, only methods annotated with `@JavascriptInterface` are available in JavaScript, protecting usage of `getClass()` by default, as in this example:

Example Language: Java

(Bad)

```
final class JavaScriptInterface {
    JavaScriptInterface () {}
    @JavascriptInterface
    public String getUserInfo() {
        return currentUser.Info();
    }
}
```

This code is not vulnerable to the above attack, but still may expose user info to malicious pages loaded in the WebView. Even malicious iframes loaded within a trusted page may access the exposed interface:

Example Language: JavaScript

(Attack)

```
<script>
  var info = window.userInfoObject.getUserInfo();
  sendUserInfo(info);
</script>
```




This malicious code within an iframe is able to access the interface object and steal the user's data.

Observed Examples

Reference	Description
CVE-2007-6382	arbitrary Java code execution via exposed method https://www.cve.org/CVERecord?id=CVE-2007-6382
CVE-2007-1112	security tool ActiveX control allows download or upload of files https://www.cve.org/CVERecord?id=CVE-2007-1112

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	2355
MemberOf		975	SFP Secondary Cluster: Architecture	888	2406
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Research Gap

Under-reported and under-studied. This weakness could appear in any technology, language, or framework that allows the programmer to provide a functional interface to external parties, but it is not heavily reported. In 2007, CVE began showing a notable increase in reports of exposed method vulnerabilities in ActiveX applications, as well as IOCTL access to OS-level resources. These weaknesses have been documented for Java applications in various secure programming sources, but there are few reports in CVE, which suggests limited awareness in most parts of the vulnerability research community.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
500	WebView Injection

References

[REF-503]Microsoft. "Developing Secure ActiveX Controls". 2005 April 3. < [https://learn.microsoft.com/en-us/previous-versions/ms533046\(v=vs.85\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/ms533046(v=vs.85)?redirectedfrom=MSDN) >.2023-04-07.

[REF-510]Microsoft. "How to stop an ActiveX control from running in Internet Explorer". < <https://support.microsoft.com/en-us/help/240797/how-to-stop-an-activex-control-from-running-in-internet-explorer> >.2023-04-07.

CWE-754: Improper Check for Unusual or Exceptional Conditions

Weakness ID : 754**Structure** : Simple**Abstraction** : Class

Description

1568

The product does not check or incorrectly checks for unusual or exceptional conditions that are not expected to occur frequently during day to day operation of the product.

Extended Description

The programmer may assume that certain events or conditions will never occur or do not need to be worried about, such as low memory conditions, lack of access to resources due to restrictive permissions, or misbehaving clients or components. However, attackers may intentionally trigger these unusual conditions, thus violating the programmer's assumptions, possibly introducing instability, incorrect behavior, or a vulnerability.

Note that this entry is not exclusively about the use of exceptions and exception handling, which are mechanisms for both checking and handling unusual or unexpected conditions.

Relationships

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	703	Improper Check or Handling of Exceptional Conditions	1535
ParentOf	B	252	Unchecked Return Value	606
ParentOf	B	253	Incorrect Check of Function Return Value	613
ParentOf	B	273	Improper Check for Dropped Privileges	660
ParentOf	B	354	Improper Validation of Integrity Check Value	876
ParentOf	B	391	Unchecked Error Condition	948
ParentOf	B	394	Unexpected Status Code or Return Value	955
ParentOf	B	476	NULL Pointer Dereference	1132

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

Nature	Type	ID	Name	Page
ParentOf	B	252	Unchecked Return Value	606
ParentOf	B	273	Improper Check for Dropped Privileges	660
ParentOf	B	476	NULL Pointer Dereference	1132

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2427

Applicable Platforms

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

Background Details

Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
Availability	Unexpected State	
	<i>The data which were produced as a result of a function call could be in a bad state upon return. If the return value is not checked, then this bad data may be used in operations, possibly leading to a crash or other unintended behaviors.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis may be useful for detecting unusual conditions involving system resources or common programming idioms, but not for violations of business rules.

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.

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. Choose languages with features such as exception handling that force the programmer to anticipate unusual conditions that may generate exceptions. Custom exceptions may need to be developed to handle unusual business-logic conditions. Be careful not to pass sensitive exceptions back to the user (CWE-209, CWE-248).

Phase: Implementation

Check the results of all functions that return a value and verify that the value is expected.

Effectiveness = High

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

Phase: Implementation

If using exception handling, catch and throw specific exceptions instead of overly-general exceptions (CWE-396, CWE-397). Catch and handle exceptions as locally as possible so that exceptions do not propagate too far up the call stack (CWE-705). Avoid unchecked or uncaught exceptions where feasible (CWE-248).

Effectiveness = High

Using specific exceptions, and ensuring that exceptions are checked, helps programmers to anticipate and appropriately handle many unusual events that could occur.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience and no one else. The messages need to strike the balance between being too cryptic (which can confuse users) or being too detailed (which may reveal more than intended). The messages should not reveal the methods that were used to determine the error. Attackers can use detailed information to refine or optimize their original attack, thereby increasing their chances of success.

If errors must be captured in some detail, record them in log messages, but consider what could occur if the log messages can be viewed by attackers. Highly sensitive information such as passwords should never be saved to log files. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a user account exists or not. Exposing additional information to a potential attacker in the context of an exceptional condition can help the attacker determine what attack vectors are most likely to succeed beyond DoS.

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: Architecture and Design

Phase: Implementation

If the program must fail, ensure that it fails gracefully (fails closed). There may be a temptation to simply let the program fail poorly in cases such as low memory conditions, but an attacker may be able to assert control before the software has fully exited. Alternately, an uncontrolled failure could cause cascading problems with other downstream components; for example, the program could send a signal to a downstream process so the process immediately knows that a problem has occurred and has a better chance of recovery.

Phase: Architecture and Design

Use system limits, which should help to prevent resource exhaustion. However, the product should still handle low resource conditions since they may still occur.

Demonstrative Examples

Example 1:

Consider the following code segment:

Example Language: C

(Bad)

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when `fgets()` returns, `buf` will contain a null-terminated string of length 9 or less. But if an I/O error occurs, `fgets()` will not null-terminate `buf`. Furthermore, if the end of the file is reached before any characters are read, `fgets()` returns without writing anything to `buf`. In both of these situations, `fgets()` signals that something unusual has happened by returning `NULL`, but in this code, the warning will not be noticed. The lack of a null terminator in `buf` can result in a buffer overflow in the subsequent call to `strcpy()`.

Example 2:

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by `malloc()`.

Example Language: C

(Bad)

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation fault when it tries to dereference the null pointer." This argument ignores three important considerations:

- Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.
- It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.
- The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

Example 3:

The following examples read a file into a byte array.

Example Language: C#

(Bad)

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current()) {
    String userName = (String) i.Current();
    String pFileName = PFILE_ROOT + "/" + userName;
    StreamReader sr = new StreamReader(pFileName);
    sr.Read(byteArray,0,1024);//the file is always 1k bytes
    sr.Close();
    processPFile(userName, byteArray);
}
```

Example Language: Java

(Bad)

```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();){
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "/" + userName;
    FileInputStream fis = new FileInputStream(pFileName);
    fis.read(byteArray); // the file is always 1k bytes
    fis.close();
    processPFile(userName, byteArray);
}
```

The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

Example 4:

The following code does not check to see if the string returned by getParameter() is null before calling the member function compareTo(), potentially causing a NULL dereference.

Example Language: Java

(Bad)

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM) == 0) {
    ...
}
```

...

The following code does not check to see if the string returned by the Item property is null before calling the member function Equals(), potentially causing a NULL dereference.

Example Language: Java

(Bad)

```
String itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM)) {
    ...
}
...
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 5:

The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

Example Language: Java

(Bad)

```
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 6:

The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

Example Language: C#

(Bad)

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
Dim MyArray(50) As Byte
MyFile.Read(MyArray, 0, 50)
DoDangerousOperation(MyArray(20))
```

In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

Example 7:

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

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 8:

In the following C/C++ example the method `outputStringToFile` opens a file in the local filesystem and outputs a string to the file. The input parameters `output` and `filename` contain the string to output to the file and the name of the file respectively.

Example Language: C++

(Bad)

```
int outputStringToFile(char *output, char *filename) {
    openFileToWrite(filename);
    writeToFile(output);
    closeFile(filename);
}
```

However, this code does not check the return values of the methods `openFileToWrite`, `writeToFile`, `closeFile` to verify that the file was properly opened and closed and that the string was successfully written to the file. The return values for these methods should be checked to determine if the method was successful and allow for detection of errors or unexpected conditions as in the following example.

Example Language: C++

(Good)

```
int outputStringToFile(char *output, char *filename) {
    int isOutput = SUCCESS;
    int isOpen = openFileToWrite(filename);
    if (isOpen == FAIL) {
        printf("Unable to open file %s", filename);
        isOutput = FAIL;
    }
    else {
        int isWrite = writeToFile(output);
        if (isWrite == FAIL) {
            printf("Unable to write to file %s", filename);
            isOutput = FAIL;
        }
        int isClose = closeFile(filename);
        if (isClose == FAIL)
            isOutput = FAIL;
    }
    return isOutput;
}
```

Example 9:

In the following Java example the method `readFromFile` uses a `FileReader` object to read the contents of a file. The `FileReader` object is created using the `File` object `readFile`, the `readFile` object is initialized using the `setInputFile` method. The `setInputFile` method should be called before calling the `readFromFile` method.

Example Language: Java

(Bad)

```
private File readFile = null;
public void setInputFile(String inputFile) {
    // create readFile File object from string containing name of file
}
public void readFromFile() {
    try {
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
}
```

However, the `readFromFile` method does not check to see if the `readFile` object is null, i.e. has not been initialized, before creating the `FileReader` object and reading from the input file. The `readFromFile` method should verify whether the `readFile` object is null and output an error message and raise an exception if the `readFile` object is null, as in the following code.

Example Language: Java

(Good)












```
private File readFile = null;
public void setInputFile(String inputFile) {
    // create readFile File object from string containing name of file
}
public void readFromFile() {
    try {
        if (readFile == null) {
            System.err.println("Input file has not been set, call setInputFile method before calling openInputFile");
            throw NullPointerException;
        }
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
    catch (NullPointerException ex) {...}
}
```

Observed Examples

Reference	Description
CVE-2023-49286	Chain: function in web caching proxy does not correctly check a return value (CWE-253) leading to a reachable assertion (CWE-617) https://www.cve.org/CVERecord?id=CVE-2023-49286
CVE-2007-3798	Unchecked return value leads to resultant integer overflow and code execution. https://www.cve.org/CVERecord?id=CVE-2007-3798
CVE-2006-4447	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. https://www.cve.org/CVERecord?id=CVE-2006-4447
CVE-2006-2916	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. https://www.cve.org/CVERecord?id=CVE-2006-2916

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		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2345
MemberOf		802	2010 Top 25 - Risky Resource Management	800	2354
MemberOf		867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf		880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2379
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2448
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf		1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Notes

Relationship

Sometimes, when a return value can be used to indicate an error, an unchecked return value is a code-layer instance of a missing application-layer check for exceptional conditions. However, return values are not always needed to communicate exceptional conditions. For example, expiration of resources, values passed by reference, asynchronously modified data, sockets, etc. may indicate exceptional conditions without the use of a return value.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	EXP31-PL	CWE More Abstract	Do not suppress or ignore exceptions
ISA/IEC 62443	Part 4-2		Req CR 3.5
ISA/IEC 62443	Part 4-2		Req CR 3.7

References

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

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

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

[REF-622]Frank Kim. "Top 25 Series - Rank 15 - Improper Check for Unusual or Exceptional Conditions". 2010 March 5. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-15-improper-check-for-unusual-or-exceptional-conditions/> >.2023-04-07.

CWE-755: Improper Handling of Exceptional Conditions

Weakness ID : 755

Structure : Simple**Abstraction** : Class

Description

The product does not handle or incorrectly handles an exceptional condition.

Relationships

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	703	Improper Check or Handling of Exceptional Conditions	1535
ParentOf	B	209	Generation of Error Message Containing Sensitive Information	533
ParentOf	B	248	Uncaught Exception	596
ParentOf	B	274	Improper Handling of Insufficient Privileges	663
ParentOf	B	280	Improper Handling of Insufficient Permissions or Privileges	672
ParentOf	V	333	Improper Handling of Insufficient Entropy in TRNG	825
ParentOf	B	390	Detection of Error Condition Without Action	943
ParentOf	B	392	Missing Report of Error Condition	951
ParentOf	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	957
ParentOf	B	396	Declaration of Catch for Generic Exception	959
ParentOf	B	460	Improper Cleanup on Thrown Exception	1102
ParentOf	B	544	Missing Standardized Error Handling Mechanism	1256
ParentOf	C	636	Not Failing Securely ('Failing Open')	1401
ParentOf	B	756	Missing Custom Error Page	1579

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

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

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Demonstrative Examples

Example 1:

The following example attempts to resolve a hostname.

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());
}
```

A DNS lookup failure will cause the Servlet to throw an exception.

Example 2:

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 3:

The following code mistakenly catches a NullPointerException.

Example Language: Java (Bad)

```
try {
    mysteryMethod();
} catch (NullPointerException npe) {
}
```





Observed Examples

Reference	Description
CVE-2023-41151	SDK for OPC Unified Architecture (OPC UA) server has uncaught exception when a socket is blocked for writing but the server tries to send an error https://www.cve.org/CVERecord?id=CVE-2023-41151
[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-2021-3011	virtual interrupt controller in a virtualization product allows crash of host by writing a certain invalid value to a register, which triggers a fatal error instead of returning an error code https://www.cve.org/CVERecord?id=CVE-2021-3011

Reference	Description
CVE-2008-4302	Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash. https://www.cve.org/CVERecord?id=CVE-2008-4302

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		880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2379
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2400
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

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-756: Missing Custom Error Page

Weakness ID : 756

Structure : Simple

Abstraction : Base





Description

The product does not return custom error pages to the user, possibly exposing sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	1576
ParentOf		7	J2EE Misconfiguration: Missing Custom Error Page	4
ParentOf		12	ASP.NET Misconfiguration: Missing Custom Error Page	11
CanPrecede		209	Generation of Error Message Containing Sensitive Information	533

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

Nature	Type	ID	Name	Page
MemberOf		389	Error Conditions, Return Values, Status Codes	2322

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
	Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.	

Demonstrative Examples

Example 1:

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

Example Language: Java

(Bad)

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

Example 2:

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

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

Example Language: ASP.NET

(Bad)

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

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

Example Language: ASP.NET

(Good)

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

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

Example Language: ASP.NET



(Good)

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2400

Nature	Type	ID	Name	V	Page
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')

Weakness ID : 757

Structure : Simple

Abstraction : Base

Description

A protocol or its implementation supports interaction between multiple actors and allows those actors to negotiate which algorithm should be used as a protection mechanism such as encryption or authentication, but it does not select the strongest algorithm that is available to both parties.



Extended Description

When a security mechanism can be forced to downgrade to use a less secure algorithm, this can make it easier for attackers to compromise the product by exploiting weaker algorithm. The victim might not be aware that the less secure algorithm is being used. For example, if an attacker can force a communications channel to use cleartext instead of strongly-encrypted data, then the attacker could read the channel by sniffing, instead of going through extra effort of trying to decrypt the data using brute force techniques.

Relationships

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	1520
PeerOf		1328	Security Version Number Mutable to Older Versions	2217

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2428

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Observed Examples

Reference	Description
CVE-2006-4302	Attacker can select an older version of the software to exploit its vulnerabilities. https://www.cve.org/CVERecord?id=CVE-2006-4302
CVE-2006-4407	Improper prioritization of encryption ciphers during negotiation leads to use of a weaker cipher. https://www.cve.org/CVERecord?id=CVE-2006-4407
CVE-2005-2969	chain: SSL/TLS implementation disables a verification step (CWE-325) that enables a downgrade attack to a weaker protocol. https://www.cve.org/CVERecord?id=CVE-2005-2969
CVE-2001-1444	Telnet protocol implementation allows downgrade to weaker authentication and encryption using an Adversary-in-the-Middle MITM attack. https://www.cve.org/CVERecord?id=CVE-2001-1444
CVE-2002-1646	SSH server implementation allows override of configuration setting to use weaker authentication schemes. This may be a composite with CWE-642. https://www.cve.org/CVERecord?id=CVE-2002-1646

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	2398
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Notes

Relationship

This is related to CWE-300, although not all downgrade attacks necessarily require an entity that redirects or interferes with the network. See examples.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
220	Client-Server Protocol Manipulation
606	Weakening of Cellular Encryption
620	Drop Encryption Level

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

Weakness ID : 758

Structure : Simple

Abstraction : Class

Description

The product uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.

Extended Description

This can lead to resultant weaknesses when the required properties change, such as when the product is ported to a different platform or if an interaction error (CWE-435) occurs.

Relationships

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	710	Improper Adherence to Coding Standards	1549
ParentOf	B	474	Use of Function with Inconsistent Implementations	1128
ParentOf	B	562	Return of Stack Variable Address	1278
ParentOf	V	587	Assignment of a Fixed Address to a Pointer	1322
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	1323
ParentOf	C	1038	Insecure Automated Optimizations	1872
ParentOf	B	1102	Reliance on Machine-Dependent Data Representation	1942
ParentOf	B	1103	Use of Platform-Dependent Third Party Components	1943
ParentOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1945

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

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:

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

Example Language: C

(Bad)

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

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

Example 2:

The following function returns a stack address.

Example Language: C

(Bad)











```
char* getName() {
    char name[STR_MAX];
    fillName(name);
    return name;
}
```

Observed Examples

Reference	Description
CVE-2006-1902	Change in C compiler behavior causes resultant buffer overflows in programs that depend on behaviors that were undefined in the C standard. https://www.cve.org/CVERecord?id=CVE-2006-1902

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		1001	SFP Secondary Cluster: Use of an Improper API	888	2420
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf		1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2456
MemberOf		1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2457
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf		1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2461
MemberOf		1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2463
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR32-C	CWE More Abstract	Ensure size arguments for variable length arrays are in a valid range
CERT C Secure Coding	ERR34-C	Imprecise	Detect errors when converting a string to a number
CERT C Secure Coding	EXP30-C	CWE More Abstract	Do not depend on the order of evaluation for side effects
CERT C Secure Coding	EXP33-C	CWE More Abstract	Do not read uninitialized memory
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file
CERT C Secure Coding	INT34-C	CWE More Abstract	Do not shift an expression by a negative number of bits or by greater

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
			than or equal to the number of bits that exist in the operand
CERT C Secure Coding	INT36-C	CWE More Abstract	Converting a pointer to integer or integer to pointer
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory
CERT C Secure Coding	MSC14-C		Do not introduce unnecessary platform dependencies
CERT C Secure Coding	MSC15-C		Do not depend on undefined behavior
CERT C Secure Coding	MSC37-C	CWE More Abstract	Ensure that control never reaches the end of a non-void function

CWE-759: Use of a One-Way Hash without a Salt

Weakness ID : 759

Structure : Simple

Abstraction : Variant

Description

The product uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the product does not also use a salt as part of the input.

Extended Description


This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables.

It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

Relationships

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		916	Use of Password Hash With Insufficient Computational Effort	1813

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2428

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity <i>If an attacker can gain access to the hashes, then the lack of a salt makes it easier to conduct brute force attacks using techniques such as rainbow tables.</i>	

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

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

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

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the

high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Effectiveness = High

Phase: Architecture and Design

If a technique that requires extra computational effort can not be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password.

Effectiveness = Limited

Be aware that salts will not reduce the workload of a targeted attack against an individual hash (such as the password for a critical person), and in general they are less effective than other hashing techniques such as increasing the computation time or memory overhead. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing and GPU, ASIC, or FPGA hardware.

Phase: Implementation

Phase: Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Demonstrative Examples

Example 1:

In both of these examples, a user is logged in if their given password matches a stored password:

Example Language: C

(Bad)

```
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

Example Language: Java

(Bad)

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code relies exclusively on a password mechanism (CWE-309) using only one factor of authentication (CWE-308). If an attacker can steal or guess a user's password, they are given full

access to their account. Note this code also uses SHA-1, which is a weak hash (CWE-328). It also does not use a salt (CWE-759).

Example 2:

In this example, a new user provides a new username and password to create an account. The program hashes the new user's password then stores it in a database.

Example Language: Python

(Bad)

```
def storePassword(userName,Password):
    hasher = hashlib.new('md5')
    hasher.update(Password)
    hashedPassword = hasher.digest()
    # UpdateUserLogin returns True on success, False otherwise
    return updateUserLogin(userName,hashedPassword)
```

While it is good to avoid storing a cleartext password, the program does not provide a salt to the hashing function, thus increasing the chances of an attacker being able to reverse the hash and discover the original password if the database is compromised.

Fixing this is as simple as providing a salt to the hashing function on initialization:

Example Language: Python

(Good)

```
def storePassword(userName,Password):
    hasher = hashlib.new('md5',b'SaltGoesHere')
    hasher.update(Password)
    hashedPassword = hasher.digest()
    # UpdateUserLogin returns True on success, False otherwise
    return updateUserLogin(userName,hashedPassword)
```



Note that regardless of the usage of a salt, the md5 hash is no longer considered secure, so this example still exhibits CWE-327.

Observed Examples

Reference	Description
CVE-2008-1526	Router does not use a salt with a hash, making it easier to crack passwords. https://www.cve.org/CVERecord?id=CVE-2008-1526
CVE-2006-1058	Router does not use a salt with a hash, making it easier to crack passwords. https://www.cve.org/CVERecord?id=CVE-2006-1058

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		816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	2359
MemberOf		866	2011 Top 25 - Porous Defenses	900	2372
MemberOf		958	SFP Secondary Cluster: Broken Cryptography	888	2398
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2527

References

[REF-291]Johnny Shelley. "bcrypt". < <http://bcrypt.sourceforge.net/> >.

[REF-292]Colin Percival. "Tarsnap - The script key derivation function and encryption utility". < <http://www.tarsnap.com/scrypt.html> >.

- [REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < <https://www.rfc-editor.org/rfc/rfc2898> >.2023-04-07.
- [REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < <https://codahale.com/how-to-safely-store-a-password/> >.2023-04-07.
- [REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < <https://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/> >.2023-04-07.
- [REF-296]Solar Designer. "Password security: past, present, future". 2012. < <https://www.openwall.com/presentations/PHDays2012-Password-Security/> >.2023-04-07.
- [REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < <https://www.troyhunt.com/our-password-hashing-has-no-clothes/> >.2023-04-07.
- [REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < <https://web.archive.org/web/20120629144851/http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/> >.2023-04-07.
- [REF-631]OWASP. "Password Storage Cheat Sheet". < https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html >.2023-04-07.
- [REF-632]Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007 September 0. < <http://hashphp.org/hashing.html> >.2023-04-07.
- [REF-633]Robert Graham. "The Importance of Being Canonical". 2009 February 2. < <https://blog.erratasec.com/2009/02/importance-of-being-canonical.html#.ZCbyY7LMJPY> >.2023-04-07.
- [REF-634]James McGlinn. "Password Hashing". < <https://privacyaustralia.net/phpsec/articles/password-hashing/> >.2023-04-07.
- [REF-635]Jeff Atwood. "Rainbow Hash Cracking". 2007 September 8. < <https://blog.codinghorror.com/rainbow-hash-cracking/> >.2023-04-07.
- [REF-636]Jeff Atwood. "Speed Hashing". 2012 April 6. < <https://blog.codinghorror.com/speed-hashing/> >.2023-04-07.
- [REF-637]"Rainbow table". 2009 March 3. Wikipedia. < https://en.wikipedia.org/wiki/Rainbow_table >.2023-04-07.
- [REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.
- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-760: Use of a One-Way Hash with a Predictable Salt

Weakness ID : 760

Structure : Simple

Abstraction : Variant

Description

The product uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the product uses a predictable salt as part of the input.

Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables, effectively disabling the protection that an unpredictable salt would provide.

It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

Relationships

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		916	Use of Password Hash With Insufficient Computational Effort	1813

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2428

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with

hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Effectiveness = High

Phase: Implementation

If a technique that requires extra computational effort can not be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password.

Effectiveness = Limited




Be aware that salts will not reduce the workload of a targeted attack against an individual hash (such as the password for a critical person), and in general they are less effective than other hashing techniques such as increasing the computation time or memory overhead. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing and GPU, ASIC, or FPGA hardware.

Observed Examples

Reference	Description
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash. https://www.cve.org/CVERecord?id=CVE-2008-4905
CVE-2002-1657	Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks. https://www.cve.org/CVERecord?id=CVE-2002-1657
CVE-2001-0967	Server uses a constant salt when encrypting passwords, simplifying brute force attacks. https://www.cve.org/CVERecord?id=CVE-2001-0967
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass. https://www.cve.org/CVERecord?id=CVE-2005-0408

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		958	SFP Secondary Cluster: Broken Cryptography	888	2398
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2527

Notes

Maintenance

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

References

- [REF-291]Johnny Shelley. "bcrypt". < <http://bcrypt.sourceforge.net/> >.
- [REF-292]Colin Percival. "Tarsnap - The scrypt key derivation function and encryption utility". < <http://www.tarsnap.com/scrypt.html> >.
- [REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < <https://www.rfc-editor.org/rfc/rfc2898> >.2023-04-07.
- [REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < <https://codahale.com/how-to-safely-store-a-password/> >.2023-04-07.
- [REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < <https://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/> >.2023-04-07.
- [REF-296]Solar Designer. "Password security: past, present, future". 2012. < <https://www.openwall.com/presentations/PHDays2012-Password-Security/> >.2023-04-07.
- [REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < <https://www.troyhunt.com/our-password-hashing-has-no-clothes/> >.2023-04-07.
- [REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < <https://web.archive.org/web/20120629144851/http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/> >.2023-04-07.
- [REF-631]OWASP. "Password Storage Cheat Sheet". < https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html >.2023-04-07.
- [REF-632]Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007 September 0. < <http://hashphp.org/hashing.html> >.2023-04-07.
- [REF-633]Robert Graham. "The Importance of Being Canonical". 2009 February 2. < <https://blog.erratasec.com/2009/02/importance-of-being-canonical.html#.ZCbyY7LMJPY> >.2023-04-07.
- [REF-634]James McGlinn. "Password Hashing". < <https://privacyaustralia.net/phpsec/articles/password-hashing/> >.2023-04-07.
- [REF-635]Jeff Atwood. "Rainbow Hash Cracking". 2007 September 8. < <https://blog.codinghorror.com/rainbow-hash-cracking/> >.2023-04-07.
- [REF-636]Jeff Atwood. "Speed Hashing". 2012 April 6. < <https://blog.codinghorror.com/speed-hashing/> >.2023-04-07.
- [REF-637]"Rainbow table". 2009 March 3. Wikipedia. < https://en.wikipedia.org/wiki/Rainbow_table >.2023-04-07.
- [REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.
- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-761: Free of Pointer not at Start of Buffer

Weakness ID : 761

Structure : Simple

Abstraction : Variant

Description

The product calls free() on a pointer to a memory resource that was allocated on the heap, but the pointer is not at the start of the buffer.

Extended Description

This can cause the product to crash, or in some cases, modify critical program variables or execute code.

This weakness often occurs when the memory is allocated explicitly on the heap with one of the malloc() family functions and free() is called, but pointer arithmetic has caused the pointer to be in the interior or end of the buffer.

Relationships

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		763	Release of Invalid Pointer or Reference	1599

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

When utilizing pointer arithmetic to traverse a buffer, use a separate variable to track progress through memory and preserve the originally allocated address for later freeing.

Phase: Implementation

When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory.

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, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples

Example 1:

In this example, the programmer dynamically allocates a buffer to hold a string and then searches for a specific character. After completing the search, the programmer attempts to release the allocated memory and return SUCCESS or FAILURE to the caller. Note: for simplification, this example uses a hard-coded "Search Me!" string and a constant string length of 20.

Example Language: C

(Bad)

```

#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( *str != NULL){
        if( *str == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        str = str + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}

```

However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it.

Instead of freeing the pointer in the middle of the buffer, the programmer can use an indexing pointer to step through the memory or abstract the memory calculations by using array indexing.

Example Language: C

(Good)

```

#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    int i = 0;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( i < strlen(str) ){
        if( str[i] == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        i = i + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}

```

Example 2:

This code attempts to tokenize a string and place it into an array using the `strsep` function, which inserts a `\0` byte in place of whitespace or a tab character. After finishing the loop, each string in the AP array points to a location within the input string.

Example Language: C

(Bad)

```

char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
    if (*ap != '\0')
        if (++ap >= &argv[10])
            break;
    /*...*/
free(ap[4]);

```

Since `strsep` is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of `inputstring`.

Example 3:

Consider the following code in the context of a parsing application to extract commands out of user data. The intent is to parse each command and add it to a queue of commands to be executed, discarding each malformed entry.

Example Language: C

(Bad)

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok;
char* sep = " \\t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( isMalformed( tok ) ){
        /* ignore and discard bad data */
        free( tok );
    }
    else{
        add_to_command_queue( tok );
    }
    tok = strtok( NULL, sep));
}
```

While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

One way to fix this problem would be to copy the commands into a new memory location before placing them in the queue. Then, after all commands have been processed, the memory can safely be freed.

Example Language: C

(Good)

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok, *command;
char* sep = " \\t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( !isMalformed( command ) ){
        /* copy and enqueue good data */
        command = (char*) malloc( (strlen(tok) + 1) * sizeof(char) );
        strcpy( command, tok );
        add_to_command_queue( command );
    }
    tok = strtok( NULL, sep));
}
free( input )
```

Observed Examples

Reference	Description
CVE-2019-11930	function "internally calls 'calloc' and returns a pointer at an index... inside the allocated buffer. This led to freeing invalid memory." https://www.cve.org/CVERecord?id=CVE-2019-11930

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	969	SFP Secondary Cluster: Faulty Memory Release	888	2404
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Maintenance

Currently, CWE-763 is the parent, however it may be desirable to have an intermediate parent which is not function-specific, similar to how CWE-762 is an intermediate parent between CWE-763 and CWE-590.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-657]"boost C++ Library Smart Pointers". < https://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.2023-04-07.

[REF-480]"Valgrind". < <http://valgrind.org/> >.

CWE-762: Mismatched Memory Management Routines

Weakness ID : 762

Structure : Simple

Abstraction : Variant

Description

The product attempts to return a memory resource to the system, but it calls a release function that is not compatible with the function that was originally used to allocate that resource.

Extended Description

This weakness can be generally described as mismatching memory management routines, such as:

- The memory was allocated on the stack (automatically), but it was deallocated using the memory management routine `free()` (CWE-590), which is intended for explicitly allocated heap memory.
- The memory was allocated explicitly using one set of memory management functions, and deallocated using a different set. For example, memory might be allocated with `malloc()` in C++ instead of the `new` operator, and then deallocated with the `delete` operator.

When the memory management functions are mismatched, the consequences may be as severe as code execution, memory corruption, or program crash. Consequences and ease of exploit will vary depending on the implementation of the routines and the object being managed.

Relationships

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	763	Release of Invalid Pointer or Reference	1599
ParentOf	V	590	Free of Memory not on the Heap	1326

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

Nature	Type	ID	Name	Page
ChildOf	G	404	Improper Resource Shutdown or Release	980

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	

Potential Mitigations**Phase: Implementation**

Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with malloc(), dispose of the original pointer with free().

Phase: Implementation

Strategy = Libraries or Frameworks

Choose a language or tool that provides automatic memory management, or makes manual memory management less error-prone. For example, glibc in Linux provides protection against free of invalid pointers. When using Xcode to target OS X or iOS, enable automatic reference counting (ARC) [REF-391]. To help correctly and consistently manage memory when programming in C++, consider using a smart pointer class such as std::auto_ptr (defined by ISO/IEC ISO/IEC 14882:2003), std::shared_ptr and std::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

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, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples**Example 1:**

This example allocates a BarObj object using the new operator in C++, however, the programmer then deallocates the object using free(), which may lead to unexpected behavior.

Example Language: C++

(Bad)

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    free(ptr);
}
```

Instead, the programmer should have either created the object with one of the malloc family functions, or else deleted the object with the delete operator.

Example Language: C++

(Good)

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    delete ptr;
}
```

Example 2:

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 3:

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;
}
```

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	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	C	1172	SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)	1154	2464
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2482
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Applicable Platform

This weakness is possible in any programming language that allows manual management of memory.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	WIN30-C	Exact	Properly pair allocation and deallocation functions
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-657]"boost C++ Library Smart Pointers". < https://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.2023-04-07.

[REF-480]"Valgrind". < <http://valgrind.org/> >.

[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.

CWE-763: Release of Invalid Pointer or Reference

Weakness ID : 763

Structure : Simple

Abstraction : Base

Description

The product attempts to return a memory resource to the system, but it calls the wrong release function or calls the appropriate release function incorrectly.

Extended Description

This weakness can take several forms, such as:

- The memory was allocated, explicitly or implicitly, via one memory management method and deallocated using a different, non-compatible function (CWE-762).
- The function calls or memory management routines chosen are appropriate, however they are used incorrectly, such as in CWE-761.

Relationships

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	980
ParentOf		761	Free of Pointer not at Start of Buffer	1592
ParentOf		762	Mismatched Memory Management Routines	1596

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	980

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324
MemberOf		465	Pointer Issues	2328

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	
<p><i>This weakness may result in the corruption of memory, and perhaps instructions, possibly leading to a crash. If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code.</i></p>		

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

Potential Mitigations

Phase: Implementation

Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with malloc(), dispose of the original pointer with free().

Phase: Implementation

When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory.

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, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples**Example 1:**

This code attempts to tokenize a string and place it into an array using the `strsep` function, which inserts a `\0` byte in place of whitespace or a tab character. After finishing the loop, each string in the `AP` array points to a location within the input string.

Example Language: C

(Bad)

```
char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
    if (**ap != '\0')
        if (++ap >= &argv[10])
            break;
/.../
free(ap[4]);
```

Since `strsep` is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of `inputstring`.

Example 2:

This example allocates a `BarObj` object using the `new` operator in C++, however, the programmer then deallocates the object using `free()`, which may lead to unexpected behavior.

Example Language: C++

(Bad)

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    free(ptr);
}
```

Instead, the programmer should have either created the object with one of the `malloc` family functions, or else deleted the object with the `delete` operator.

Example Language: C++

(Good)

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    delete ptr;
}
```

Example 3:

In this example, the programmer dynamically allocates a buffer to hold a string and then searches for a specific character. After completing the search, the programmer attempts to release the allocated memory and return `SUCCESS` or `FAILURE` to the caller. Note: for simplification, this example uses a hard-coded "Search Me!" string and a constant string length of 20.

Example Language: C

(Bad)

```

#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( *str != NULL){
        if( *str == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        str = str + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}

```

However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it.

Instead of freeing the pointer in the middle of the buffer, the programmer can use an indexing pointer to step through the memory or abstract the memory calculations by using array indexing.

Example Language: C

(Good)

```

#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    int i = 0;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( i < strlen(str) ){
        if( str[i] == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        i = i + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}

```

Example 4:

Consider the following code in the context of a parsing application to extract commands out of user data. The intent is to parse each command and add it to a queue of commands to be executed, discarding each malformed entry.

Example Language: C

(Bad)

```

//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok;
char* sep = " \t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){

```

```
if( isMalformed( tok ) ){
    /* ignore and discard bad data */
    free( tok );
}
else{
    add_to_command_queue( tok );
}
tok = strtok( NULL, sep));
}
```

While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

One way to fix this problem would be to copy the commands into a new memory location before placing them in the queue. Then, after all commands have been processed, the memory can safely be freed.

Example Language: C (Good)

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok, *command;
char* sep = " \t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( !isMalformed( command ) ){
        /* copy and enqueue good data */
        command = (char*) malloc( (strlen(tok) + 1) * sizeof(char) );
        strcpy( command, tok );
        add_to_command_queue( command );
    }
    tok = strtok( NULL, sep));
}
free( input )
```

Observed Examples

Reference	Description
CVE-2019-11930	function "internally calls 'calloc' and returns a pointer at an index... inside the allocated buffer. This led to freeing invalid memory." https://www.cve.org/CVERecord?id=CVE-2019-11930

Affected Resources

- Memory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	2404
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2482
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Maintenance

The view-1000 subtree that is associated with this weakness needs additional work. Several entries will likely be created in this branch. Currently the focus is on free() of memory, but delete and other related release routines may require the creation of intermediate entries that are not specific to a particular function. In addition, the role of other types of invalid pointers, such as an expired pointer, i.e. CWE-415 Double Free and release of uninitialized pointers, related to CWE-457.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-657]"boost C++ Library Smart Pointers". < https://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.2023-04-07.

[REF-480]"Valgrind". < <http://valgrind.org/> >.

CWE-764: Multiple Locks of a Critical Resource

Weakness ID : 764

Structure : Simple

Abstraction : Base

Description

The product locks a critical resource more times than intended, leading to an unexpected state in the system.

Extended Description

When a product is operating in a concurrent environment and repeatedly locks a critical resource, the consequences will vary based on the type of lock, the lock's implementation, and the resource being protected. In some situations such as with semaphores, the resources are pooled and extra locking calls will reduce the size of the total available pool, possibly leading to degraded performance or a denial of service. If this can be triggered by an attacker, it will be similar to an unrestricted lock (CWE-412). In the context of a binary lock, it is likely that any duplicate locking attempts will never succeed since the lock is already held and progress may not be possible.

Relationships

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	🟢	675	Multiple Operations on Resource in Single-Operation Context	1487
ChildOf	🟢	667	Improper Locking	1464

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

Nature	Type	ID	Name	Page
ChildOf	🟢	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf	🟢	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2325

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	
Integrity	DoS: Crash, Exit, or Restart Unexpected State	




Potential Mitigations

Phase: Implementation

When locking and unlocking a resource, try to be sure that all control paths through the code in which the resource is locked one or more times correspond to exactly as many unlocks. If the software acquires a lock and then determines it is not able to perform its intended behavior, be sure to release the lock(s) before waiting for conditions to improve. Reacquire the lock(s) before trying again.

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		987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	2412
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Maintenance

An alternate way to think about this weakness is as an imbalance between the number of locks / unlocks in the control flow. Over the course of execution, if each lock call is not followed by a subsequent call to unlock in a reasonable amount of time, then system performance may be degraded or at least operating at less than peak levels if there is competition for the locks. This entry may need to be modified to reflect these concepts in the future.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP21		Multiple locks/unlocks

CWE-765: Multiple Unlocks of a Critical Resource

Weakness ID : 765

Structure : Simple

Abstraction : Base

Description

The product unlocks a critical resource more times than intended, leading to an unexpected state in the system.

Extended Description



When the product is operating in a concurrent environment and repeatedly unlocks a critical resource, the consequences will vary based on the type of lock, the lock's implementation, and the resource being protected. In some situations such as with semaphores, the resources are

pooled and extra calls to unlock will increase the count for the number of available resources, likely resulting in a crash or unpredictable behavior when the system nears capacity.

Relationships

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		675	Multiple Operations on Resource in Single-Operation Context	1487
ChildOf		667	Improper Locking	1464

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2325

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Integrity	Modify Memory Unexpected State	

Potential Mitigations

Phase: Implementation

When locking and unlocking a resource, try to be sure that all control paths through the code in which the resource is locked one or more times correspond to exactly as many unlocks. If the product acquires a lock and then determines it is not able to perform its intended behavior, be sure to release the lock(s) before waiting for conditions to improve. Reacquire the lock(s) before trying again.

Observed Examples

Reference	Description
CVE-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice https://www.cve.org/CVERecord?id=CVE-2009-0935

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		987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	2412
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Maintenance

An alternate way to think about this weakness is as an imbalance between the number of locks / unlocks in the control flow. Over the course of execution, if each lock call is not followed by a subsequent call to unlock in a reasonable amount of time, then system performance may be degraded or at least operating at less than peak levels if there is competition for the locks. This entry may need to be modified to reflect these concepts in the future.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP21		Multiple locks/unlocks

CWE-766: Critical Data Element Declared Public

Weakness ID : 766

Structure : Simple

Abstraction : Base

Description

The product declares a critical variable, field, or member to be public when intended security policy requires it to be private.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1061	Insufficient Encapsulation	1898
ChildOf		732	Incorrect Permission Assignment for Critical Resource	1551

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

Nature	Type	ID	Name	Page
MemberOf		275	Permission Issues	2317

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C++ (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Integrity	Read Application Data	
Confidentiality	Modify Application Data	
	<i>Making a critical variable public allows anyone with access to the object in which the variable is contained to alter or read the value.</i>	
Other	Reduce Maintainability	

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

Data should be private, static, and final whenever possible. This will assure that your code is protected by instantiating early, preventing access, and preventing tampering.

Demonstrative Examples

Example 1:

The following example declares a critical variable public, making it accessible to anyone with access to the object in which it is contained.

Example Language: C++

(Bad)

```
public: char* password;
```

Instead, the critical data should be declared private.

Example Language: C++

(Good)

```
private: char* password;
```

Even though this example declares the password to be private, there are other possible issues with this implementation, such as the possibility of recovering the password from process memory (CWE-257).

Example 2:

The following example shows a basic user account class that includes member variables for the username and password as well as a public constructor for the class and a public method to authorize access to the user account.

Example Language: C++

(Bad)

```
#define MAX_PASSWORD_LENGTH 15
#define MAX_USERNAME_LENGTH 15
class UserAccount
{
public:
    UserAccount(char *username, char *password)
    {
        if ((strlen(username) > MAX_USERNAME_LENGTH) ||
            (strlen(password) > MAX_PASSWORD_LENGTH)) {
            ExitError("Invalid username or password");
        }
        strcpy(this->username, username);
        strcpy(this->password, password);
    }
    int authorizeAccess(char *username, char *password)
    {
        if ((strlen(username) > MAX_USERNAME_LENGTH) ||
            (strlen(password) > MAX_PASSWORD_LENGTH)) {
```

```
        ExitError("Invalid username or password");
    }
    // if the username and password in the input parameters are equal to
    // the username and password of this account class then authorize access
    if (strcmp(this->username, username) ||
        strcmp(this->password, password))
        return 0;
    // otherwise do not authorize access
    else
        return 1;
}
char username[MAX_USERNAME_LENGTH+1];
char password[MAX_PASSWORD_LENGTH+1];
};
```

However, the member variables `username` and `password` are declared public and therefore will allow access and changes to the member variables to anyone with access to the object. These member variables should be declared private as shown below to prevent unauthorized access and changes.

Example Language: C++

(Good)






```
class UserAccount
{
public:
    ...
private:
    char username[MAX_USERNAME_LENGTH+1];
    char password[MAX_PASSWORD_LENGTH+1];
};
```

Observed Examples

Reference	Description
CVE-2010-3860	variables declared public allow remote read of system properties such as user name and home directory. https://www.cve.org/CVERecord?id=CVE-2010-3860

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	2364
MemberOf		1002	SFP Secondary Cluster: Unexpected Entry Points	888	2421
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf		1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2446
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to protect stored data from modification
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ01-J		Declare data members as private and provide accessible wrapper methods
Software Fault Patterns	SFP28		Unexpected access points

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-15		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < <https://www.omg.org/spec/ASCMM/> >.2023-04-07.

CWE-767: Access to Critical Private Variable via Public Method

Weakness ID : 767

Structure : Simple

Abstraction : Base

Description

The product defines a public method that reads or modifies a private variable.


Extended Description

If an attacker modifies the variable to contain unexpected values, this could violate assumptions from other parts of the code. Additionally, if an attacker can read the private variable, it may expose sensitive information or make it easier to launch further attacks.

Relationships

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

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

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

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

Nature	Type	ID	Name	Page
MemberOf		275	Permission Issues	2317

Applicable Platforms

Language : C++ (Prevalence = Undetermined)

Language : C# (Prevalence = Undetermined)

Language : Java (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Other	Other	

Potential Mitigations

Phase: Implementation

Use class accessor and mutator methods appropriately. Perform validation when accepting data from a public method that is intended to modify a critical private variable. Also be sure that appropriate access controls are being applied when a public method interfaces with critical data.

Demonstrative Examples

Example 1:

The following example declares a critical variable to be private, and then allows the variable to be modified by public methods.

Example Language: C++

(Bad)

```
private: float price;
public: void changePrice(float newPrice) {
    price = newPrice;
}
```

Example 2:

The following example could be used to implement a user forum where a single user (UID) can switch between multiple profiles (PID).

Example Language: Java




(Bad)

```
public class Client {
    private int UID;
    public int PID;
    private String userName;
    public Client(String userName){
        PID = getDefaultProfileID();
        UID = mapUserNameToUID( userName );
        this.userName = userName;
    }
    public void setPID(int ID) {
        UID = ID;
    }
}
```

The programmer implemented setPID with the intention of modifying the PID variable, but due to a typo, accidentally specified the critical variable UID instead. If the program allows profile IDs to be between 1 and 10, but a UID of 1 means the user is treated as an admin, then a user could gain administrative privileges as a result of this typo.

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		963	SFP Secondary Cluster: Exposed Data	888	2400
MemberOf		1184	SEI CERT Perl Coding Standard - Guidelines 06. Object-Oriented Programming (OOP)	1178	2467
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Maintenance

This entry is closely associated with access control for public methods. If the public methods are restricted with proper access controls, then the information in the private variable will not be exposed to unexpected parties. There may be chaining or composite relationships between improper access controls and this weakness.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to protect stored data from modification
Software Fault Patterns	SFP23		Exposed Data

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	OOP31-PL	Imprecise	Do not access private variables or subroutines in other packages

CWE-768: Incorrect Short Circuit Evaluation

Weakness ID : 768

Structure : Simple

Abstraction : Variant

Description

The product contains a conditional statement with multiple logical expressions in which one of the non-leading expressions may produce side effects. This may lead to an unexpected state in the program after the execution of the conditional, because short-circuiting logic may prevent the side effects from occurring.

Extended Description

Usage of short circuit evaluation, though well-defined in the C standard, may alter control flow in a way that introduces logic errors that are difficult to detect, possibly causing errors later during the product's execution. If an attacker can discover such an inconsistency, it may be exploitable to gain arbitrary control over a system.

If the first condition of an "or" statement is assumed to be true under normal circumstances, or if the first condition of an "and" statement is assumed to be false, then any subsequent conditional may contain its own logic errors that are not detected during code review or testing.

Finally, the usage of short circuit evaluation may decrease the maintainability of the code.

Relationships

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	1517

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Varies by Context <i>Widely varied consequences are possible if an attacker is aware of an unexpected state in the product after a conditional. It may lead to information exposure, a system crash, or even complete attacker control of the system.</i>	

Potential Mitigations

Phase: Implementation

Minimizing the number of statements in a conditional that produce side effects will help to prevent the likelihood of short circuit evaluation to alter control flow in an unexpected way.

Demonstrative Examples

Example 1:

The following function attempts to take a size value from a user and allocate an array of that size (we ignore bounds checking for simplicity). The function tries to initialize each spot with the value of its index, that is, $A[\text{len}-1] = \text{len} - 1$; $A[\text{len}-2] = \text{len} - 2$; ... $A[1] = 1$; $A[0] = 0$; However, since the programmer uses the prefix decrement operator, when the conditional is evaluated with $i == 1$, the decrement will result in a 0 value for the first part of the predicate, causing the second portion to be bypassed via short-circuit evaluation. This means we cannot be sure of what value will be in $A[0]$ when we return the array to the user.

Example Language: C

(Bad)

```
#define PRIV_ADMIN 0
#define PRIV_REGULAR 1
typedef struct{
    int privileges;
    int id;
} user_t;
user_t *Add_Regular_Users(int num_users){
    user_t* users = (user_t*)calloc(num_users, sizeof(user_t));
    int i = num_users;
    while( --i && (users[i].privileges = PRIV_REGULAR) ){
        users[i].id = i;
    }
    return users;
}
int main(){
    user_t* test;
    int i;
    test = Add_Regular_Users(25);
    for(i = 0; i < 25; i++) printf("user %d has privilege level %d\n", test[i].id, test[i].privileges);
}
```

When compiled and run, the above code will output a privilege level of 1, or PRIV_REGULAR for every user but the user with id 0 since the prefix increment operator used in the if statement will reach zero and short circuit before setting the 0th user's privilege level. Since we used calloc, this privilege will be set to 0, or PRIV_ADMIN.

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		871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	2374
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2419
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to protect stored data from modification
Software Fault Patterns	SFP1		Glitch in computation

CWE-770: Allocation of Resources Without Limits or Throttling

Weakness ID : 770**Structure :** Simple**Abstraction :** Base

Description

The product allocates a reusable resource or group of resources on behalf of an actor without imposing any restrictions on the size or number of resources that can be allocated, in violation of the intended security policy for that actor.







Extended Description

Code frequently has to work with limited resources, so programmers must be careful to ensure that resources are not consumed too quickly, or too easily. Without use of quotas, resource limits, or other protection mechanisms, it can be easy for an attacker to consume many resources by rapidly making many requests, or causing larger resources to be used than is needed. When too many resources are allocated, or if a single resource is too large, then it can prevent the code from working correctly, possibly leading to a denial of service.


Relationships

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		665	Improper Initialization	1456
ChildOf		400	Uncontrolled Resource Consumption	964
ParentOf		774	Allocation of File Descriptors or Handles Without Limits or Throttling	1630
ParentOf		789	Memory Allocation with Excessive Size Value	1674
ParentOf		1325	Improperly Controlled Sequential Memory Allocation	2210
CanFollow		20	Improper Input Validation	20



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

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	964

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324
MemberOf		840	Business Logic Errors	2360

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) <i>When allocating resources without limits, an attacker could prevent other systems, applications, or processes from accessing the same type of resource.</i>	

Detection Methods

Manual Static Analysis

Manual static analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. If denial-of-service is not considered a significant risk, or if there is strong emphasis on consequences such as code execution, then manual analysis may not focus on this weakness at all.

Fuzzing

While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently find uncontrolled resource allocation problems. This can occur when the fuzzer generates a large number of test cases but does not restart the targeted product in between test cases. If an individual test case produces a crash, but it does not do so reliably, then an inability to limit resource allocation may be the cause. When the allocation is directly affected by numeric inputs, then fuzzing may produce indications of this weakness.

Effectiveness = Opportunistic

Automated Dynamic Analysis

Certain automated dynamic analysis techniques may be effective in producing side effects of uncontrolled resource allocation problems, especially with resources such as processes, memory, and connections. The technique may involve generating a large number of requests to the product within a short time frame. Manual analysis is likely required to interpret the results.

Automated Static Analysis

Specialized configuration or tuning may be required to train automated tools to recognize this weakness. Automated static analysis typically has limited utility in recognizing unlimited allocation problems, except for the missing release of program-independent system resources such as files, sockets, and processes, or unchecked arguments to memory. For system resources, automated static analysis may be able to detect circumstances in which resources are not released after they have expired, or if too much of a resource is requested at once, as can occur with memory. Automated analysis of configuration files may be able to detect settings that do not specify a maximum value. Automated static analysis tools will not be appropriate for detecting exhaustion of custom resources, such as an intended security policy in which a bulletin board user is only allowed to make a limited number of posts per day.

Potential Mitigations

Phase: Requirements

Clearly specify the minimum and maximum expectations for capabilities, and dictate which behaviors are acceptable when resource allocation reaches limits.

Phase: Architecture and Design

Limit the amount of resources that are accessible to unprivileged users. Set per-user limits for resources. Allow the system administrator to define these limits. Be careful to avoid CWE-410.

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, and it will help the administrator to identify who is committing the abuse. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.

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: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Mitigation of resource exhaustion attacks requires that the target system either: recognizes the attack and denies that user further access for a given amount of time, typically by using increasing time delays uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed. The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, they may be able to prevent the user from accessing the server in question. The second solution can be difficult to effectively institute -- and even when properly done, it does not provide a full solution. It simply requires more resources on the part of the attacker.

Phase: Architecture and Design

Ensure that protocols have specific limits of scale placed on them.

Phase: Architecture and Design**Phase: Implementation**

If the program must fail, ensure that it fails gracefully (fails closed). There may be a temptation to simply let the program fail poorly in cases such as low memory conditions, but an attacker may be able to assert control before the software has fully exited. Alternately, an uncontrolled failure could cause cascading problems with other downstream components; for example, the program could send a signal to a downstream process so the process immediately knows that a problem has occurred and has a better chance of recovery. Ensure that all failures in resource allocation place the system into a safe posture.

Phase: Operation**Phase: Architecture and Design**

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Demonstrative Examples**Example 1:**

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 2:

In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method `openSocketConnection` establishes a server socket to accept requests from a client. When a client establishes a connection to this service the `getNextMessage` method is first used to retrieve from the socket the name of the file to store the data, the `openFileToWrite` method will validate the filename and open a file to write to on the local file system. The `getNextMessage` is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

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 3:

In the following example, the `processMessage` method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The `getMessageLength` method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

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 4:

In the following example, a server object creates a server socket and accepts client connections to the socket. For every client connection to the socket a separate thread object is generated using the `ClientSocketThread` class that handles request made by the client through the socket.

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 5:

An unnamed web site allowed a user to purchase tickets for an event. A menu option allowed the user to purchase up to 10 tickets, but the back end did not restrict the actual number of tickets that could be purchased.

Example 6:

Here the problem is that every time a connection is made, more memory is allocated. So if one just opened up more and more connections, eventually the machine would run out of memory.

Example Language: C

(Bad)

```

bar connection() {
    foo = malloc(1024);
    return foo;
}
endConnection(bar foo) {
    free(foo);
}
int main() {
    while(1) {
        foo=connection();
    }
    endConnection(foo)
}











```





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-2009-4017	Language interpreter does not restrict the number of temporary files being created when handling a MIME request with a large number of parts.. https://www.cve.org/CVERecord?id=CVE-2009-4017
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-2009-2054	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
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-2005-4650	CMS does not restrict the number of searches that can occur simultaneously, leading to resource exhaustion. https://www.cve.org/CVERecord?id=CVE-2005-4650
CVE-2020-15100	web application scanner attempts to read an excessively large file created by a user, causing process termination https://www.cve.org/CVERecord?id=CVE-2020-15100
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

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		802	2010 Top 25 - Risky Resource Management	800	2354
MemberOf		857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2368
MemberOf		858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2368
MemberOf		861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf		867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2377
MemberOf		884	CWE Cross-section	884	2567
MemberOf		985	SFP Secondary Cluster: Unrestricted Consumption	888	2411

Nature	Type	ID	Name	V	Page
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2450
MemberOf		1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	2451
MemberOf		1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

This entry is different from uncontrolled resource consumption (CWE-400) in that there are other weaknesses that are related to inability to control resource consumption, such as holding on to a resource too long after use, or not correctly keeping track of active resources so that they can be managed and released when they are finished (CWE-771).

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 one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Close resources when they are no longer needed
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
ISA/IEC 62443	Part 4-2		Req CR 7.2
ISA/IEC 62443	Part 4-2		Req CR 2.7
ISA/IEC 62443	Part 4-1		Req SI-1
ISA/IEC 62443	Part 4-1		Req SI-2
ISA/IEC 62443	Part 3-3		Req SR 7.2
ISA/IEC 62443	Part 3-3		Req SR 2.7

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
125	Flooding
130	Excessive Allocation
147	XML Ping of the Death
197	Exponential Data Expansion
229	Serialized Data Parameter Blowup
230	Serialized Data with Nested Payloads
231	Oversized Serialized Data Payloads
469	HTTP DoS
482	TCP Flood
486	UDP Flood
487	ICMP Flood

CAPEC-ID	Attack Pattern Name
488	HTTP Flood
489	SSL Flood
490	Amplification
491	Quadratic Data Expansion
493	SOAP Array Blowup
494	TCP Fragmentation
495	UDP Fragmentation
496	ICMP Fragmentation
528	XML Flood

References

[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> >.

[REF-667]Rafal Los. "Real-Life Example of a 'Business Logic Defect' (Screen Shots!)". 2011. < <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581> >.

[REF-672]Frank Kim. "Top 25 Series - Rank 22 - Allocation of Resources Without Limits or Throttling". 2010 March 3. SANS Software Security Institute. < <https://web.archive.org/web/20170113055136/https://software-security.sans.org/blog/2010/03/23/top-25-series-rank-22-allocation-of-resources-without-limits-or-throttling/> >.2023-04-07.

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

CWE-771: Missing Reference to Active Allocated Resource

Weakness ID : 771

Structure : Simple

Abstraction : Base

Description

The product does not properly maintain a reference to a resource that has been allocated, which prevents the resource from being reclaimed.

Extended Description



This does not necessarily apply in languages or frameworks that automatically perform garbage collection, since the removal of all references may act as a signal that the resource is ready to be reclaimed.

Relationships

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	964
ParentOf		773	Missing Reference to Active File Descriptor or Handle	1629

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>An attacker that can influence the allocation of resources that are not properly maintained could deplete the available resource pool and prevent all other processes from accessing the same type of resource.</i>	

Potential Mitigations

Phase: Operation






Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

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		982	SFP Secondary Cluster: Failure to Release Resource	888	2410
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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
Software Fault Patterns	SFP14		Failure to Release Resource
ISA/IEC 62443	Part 3-3		Req SR 7.2
ISA/IEC 62443	Part 4-1		Req SVV-1
ISA/IEC 62443	Part 4-2		Req CR 7.2

CWE-772: Missing Release of Resource after Effective Lifetime

Weakness ID : 772

Structure : Simple

Abstraction : Base

Description

The product does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.






Extended Description

When a resource is not released after use, it can allow attackers to cause a denial of service by causing the allocation of resources without triggering their release. Frequently-affected resources include memory, CPU, disk space, power or battery, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	980
ParentOf		401	Missing Release of Memory after Effective Lifetime	973
ParentOf		775	Missing Release of File Descriptor or Handle after Effective Lifetime	1631
ParentOf		1091	Use of Object without Invoking Destructor Method	1931
CanFollow		911	Improper Update of Reference Count	1801

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	980

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324

Applicable Platforms

Technology : Mobile (*Prevalence = Undetermined*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>An attacker that can influence the allocation of resources that are not properly released could deplete the available resource pool and prevent all other processes from accessing the same type of resource.</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. 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 resources in a function. If you allocate resources that you intend to free upon completion of the function, you must be sure to free the resources at all exit points for that function including error conditions.

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

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:

The following code attempts to open a new connection to a database, process the results returned by the database, and close the allocated SqlConnection object.

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();

```

The problem with the above code is that if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

Example 3:

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 4:

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 5:

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;
}
```

Observed Examples

Reference	Description
CVE-2007-0897	Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans. https://www.cve.org/CVERecord?id=CVE-2007-0897
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-1999-1127	Does not shut down named pipe connections if malformed data is sent. https://www.cve.org/CVERecord?id=CVE-1999-1127
CVE-2009-2858	Chain: memory leak (CWE-404) leads to resource exhaustion. https://www.cve.org/CVERecord?id=CVE-2009-2858

Reference	Description
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-2008-2122	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-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-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 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	2355
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2380
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	2410
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

"Resource exhaustion" (CWE-400) is currently treated as a weakness, although it is more like a category of weaknesses that all have the same type of consequence. While this entry treats CWE-400 as a parent in view 1000, the relationship is probably more appropriately described as a chain.

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 one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM31-C	CWE More Abstract	Free dynamically allocated memory when no longer needed
OMG ASCSM	ASCSM-CWE-772		
OMG ASCRM	ASCRM-CWE-772		
Software Fault Patterns	SFP14		Failure to Release Resource

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
469	HTTP DoS

References

[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-773: Missing Reference to Active File Descriptor or Handle

Weakness ID : 773

Structure : Simple

Abstraction : Variant

Description

The product does not properly maintain references to a file descriptor or handle, which prevents that file descriptor/handle from being reclaimed.


Extended Description

This can cause the product to consume all available file descriptors or handles, which can prevent other processes from performing critical file processing operations.

Relationships

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		771	Missing Reference to Active Allocated Resource	1622

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>An attacker that can influence the allocation of resources that are not properly maintained could deplete the available resource pool and prevent all other processes from accessing the same type of resource.</i>	

Potential Mitigations

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

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	982	SFP Secondary Cluster: Failure to Release Resource	888	2410
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
Software Fault Patterns	SFP14		Failure to Release Resource

CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling**Weakness ID :** 774**Structure :** Simple**Abstraction :** Variant**Description**

The product allocates file descriptors or handles on behalf of an actor without imposing any restrictions on how many descriptors can be allocated, in violation of the intended security policy for that actor.

Extended Description

This can cause the product to consume all available file descriptors or handles, which can prevent other processes from performing critical file processing operations.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	E	770	Allocation of Resources Without Limits or Throttling	1613

Alternate Terms**File Descriptor Exhaustion :****Likelihood Of Exploit**

Low

Common Consequences



Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.</i>	

Potential Mitigations**Phase: Operation****Phase: Architecture and Design***Strategy = Resource Limitation*

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

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		985	SFP Secondary Cluster: Unrestricted Consumption	888	2411
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP13		Unrestricted Consumption

References

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

CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime**Weakness ID :** 775**Structure :** Simple**Abstraction :** Variant**Description**

The product does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed.

Extended Description

When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.

Relationships

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	1624

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>An attacker that can influence the allocation of resources that are not properly released could deplete the available resource pool and prevent all other processes from accessing the same type of resource.</i>	

Potential Mitigations

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation




Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Observed Examples

Reference	Description
CVE-2007-0897	Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans. 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		982	SFP Secondary Cluster: Failure to Release Resource	888	2410
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
Software Fault Patterns	SFP14		Failure to Release Resource

References

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

CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')

Weakness ID : 776

Structure : Simple

Abstraction : Base

Description

The product uses XML documents and allows their structure to be defined with a Document Type Definition (DTD), but it does not properly control the number of recursive definitions of entities.




Extended Description

If the DTD contains a large number of nested or recursive entities, this can lead to explosive growth of data when parsed, causing a denial of service.


Relationships

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)	986
ChildOf		674	Uncontrolled Recursion	1484
CanFollow		827	Improper Control of Document Type Definition	1736

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

Nature	Type	ID	Name	Page
ChildOf		674	Uncontrolled Recursion	1484

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

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	2309

Applicable Platforms

Language : XML (*Prevalence = Undetermined*)

Alternate Terms

XEE : XEE is the acronym commonly used for XML Entity Expansion.

Billion Laughs Attack :

XML Bomb : While the "XML Bomb" term was used in the early years of knowledge of this issue, the XEE term seems to be more commonly used.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) <i>If parsed, recursive entity references allow the attacker to expand data exponentially, quickly consuming all system resources.</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: Operation

If possible, prohibit the use of DTDs or use an XML parser that limits the expansion of recursive DTD entities.

Phase: Implementation

Before parsing XML files with associated DTDs, scan for recursive entity declarations and do not continue parsing potentially explosive content.

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-2008-3281	XEE in XML-parsing library. https://www.cve.org/CVERecord?id=CVE-2008-3281
CVE-2011-3288	XML bomb / XEE in enterprise communication product. https://www.cve.org/CVERecord?id=CVE-2011-3288
CVE-2011-1755	"Billion laughs" attack in XMPP server daemon. https://www.cve.org/CVERecord?id=CVE-2011-1755
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		Page
MemberOf		1030	OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)	1026	2437
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	44		XML Entity Expansion

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
197	Exponential Data Expansion

References

[REF-676]Amit Klein. "Multiple vendors XML parser (and SOAP/WebServices server) Denial of Service attack using DTD". 2002 December 6. < <https://seclists.org/fulldisclosure/2002/Dec/229> >.2023-04-07.

[REF-677]Rami Jaamour. "XML security: Preventing XML bombs". 2006 February 2. < http://searchsoftwarequality.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid92_gci1168442,00.html?asrc=SS_CLA_302%20%20558&psrc=CLT_92# >.

[REF-678]Didier Stevens. "Dismantling an XML-Bomb". 2008 September 3. < <https://blog.didierstevens.com/2008/09/23/dismantling-an-xml-bomb/> >.2023-04-07.

[REF-679]Robert Auger. "XML Entity Expansion". < <http://projects.webappsec.org/w/page/13247002/XML%20Entity%20Expansion> >.2023-04-07.

[REF-680]Elliott Rusty Harold. "Tip: Configure SAX parsers for secure processing". 2005 May 7. < <https://web.archive.org/web/20101005080451/http://www.ibm.com/developerworks/xml/library/x-tipcfsx.html> >.2023-04-07.

[REF-500]Bryan Sullivan. "XML Denial of Service Attacks and Defenses". 2009 September. < <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/november/xml-denial-of-service-attacks-and-defenses> >.2023-04-07.

[REF-682]Blaise Doughan. "Preventing Entity Expansion Attacks in JAXB". 2011 March 1. < <http://blog.bdoughan.com/2011/03/preventing-entity-expansion-attacks-in.html> >.2023-04-07.

CWE-777: Regular Expression without Anchors

Weakness ID : 777

Structure : Simple

Abstraction : Variant

Description

The product uses a regular expression to perform neutralization, but the regular expression is not anchored and may allow malicious or malformed data to slip through.

Extended Description

When performing tasks such as validating against a set of allowed inputs (allowlist), data is examined and possibly modified to ensure that it is well-formed and adheres to a list of safe values. If the regular expression is not anchored, malicious or malformed data may be included before or after any string matching the regular expression. The type of malicious data that is allowed will depend on the context of the application and which anchors are omitted from the regular expression.

Relationships

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		625	Permissive Regular Expression	1392

Background Details

Regular expressions are typically used to match a pattern of text. Anchors are used in regular expressions to specify where the pattern should match: at the beginning, the end, or both (the whole input).

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability Confidentiality Access Control	Bypass Protection Mechanism <i>An unanchored regular expression in the context of an allowlist will possibly result in a protection mechanism failure, allowing malicious or malformed data to enter trusted regions of the program. The specific consequences will depend on what functionality the allowlist was protecting.</i>	

Potential Mitigations

Phase: Implementation

Be sure to understand both what will be matched and what will not be matched by a regular expression. Anchoring the ends of the expression will allow the programmer to define an allowlist strictly limited to what is matched by the text in the regular expression. If you are using a package that only matches one line by default, ensure that you can match multi-line inputs if necessary.

Demonstrative Examples

Example 1:

Consider a web application that supports multiple languages. It selects messages for an appropriate language by using the lang parameter.

Example Language: PHP

(Bad)

```
$dir = "/home/cwe/languages";
$lang = $_GET['lang'];
if (preg_match("/[A-Za-z0-9]+/", $lang)) {
    include("$dir/$lang");
}
else {
    echo "You shall not pass!\n";
}
```

The previous code attempts to match only alphanumeric values so that language values such as "english" and "french" are valid while also protecting against path traversal, CWE-22. However, the regular expression anchors are omitted, so any text containing at least one alphanumeric character will now pass the validation step. For example, the attack string below will match the regular expression.

Example Language:

(Attack)

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

If the attacker can inject code sequences into a file, such as the web server's HTTP request log, then the attacker may be able to redirect the lang parameter to the log file and execute arbitrary code.

Example 2:

This code uses a regular expression to validate an IP string prior to using it in a call to the "ping" command.

Example Language: Python

(Bad)

```
import subprocess
import re
def validate_ip_regex(ip: str):
    ip_validator = re.compile(r"((25[0-5])|(2[0-4]|1\d|[1-9])\d)\.?\b{4}")
    if ip_validator.match(ip):
        return ip
    else:
        raise ValueError("IP address does not match valid pattern.")
def run_ping_regex(ip: str):
    validated = validate_ip_regex(ip)
    # The ping command treats zero-prepended IP addresses as octal
    result = subprocess.call(["ping", validated])
    print(result)
```

Since the regular expression does not have anchors (CWE-777), i.e. is unbounded without ^ or \$ characters, then prepending a 0 or 0x to the beginning of the IP address will still result in a matched regex pattern. Since the ping command supports octal and hex prepended IP addresses, it will use the unexpectedly valid IP address (CWE-1389). For example, "0x63.63.63.63" would be considered equivalent to "99.63.63.63". As a result, the attacker could potentially ping systems that the attacker cannot reach directly.

Observed Examples

Reference	Description
CVE-2022-30034	Chain: Web UI for a Python RPC framework does not use regex anchors to validate user login emails (CWE-777), potentially allowing bypass of OAuth (CWE-1390). https://www.cve.org/CVERecord?id=CVE-2022-30034

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		1397	Comprehensive Categorization: Comparison	1400	2523

CWE-778: Insufficient Logging

Weakness ID : 778

Structure : Simple

Abstraction : Base

Description

When a security-critical event occurs, the product either does not record the event or omits important details about the event when logging it.

Extended Description



When security-critical events are not logged properly, such as a failed login attempt, this can make malicious behavior more difficult to detect and may hinder forensic analysis after an attack succeeds.

As organizations adopt cloud storage resources, these technologies often require configuration changes to enable detailed logging information, since detailed logging can incur additional costs. This could lead to telemetry gaps in critical audit logs. For example, in Azure, the default value for logging is disabled.

Relationships

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	1520
ChildOf		223	Omission of Security-relevant Information	559

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

Nature	Type	ID	Name	Page
MemberOf		1009	Audit	2424

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

Nature	Type	ID	Name	Page
MemberOf		1210	Audit / Logging Errors	2475

Applicable Platforms

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

Technology : Cloud Computing (*Prevalence = Undetermined*)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
	<i>If security critical information is not recorded, there will be no trail for forensic analysis and discovering the cause of problems or the source of attacks may become more difficult or impossible.</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 a centralized logging mechanism that supports multiple levels of detail.

Phase: Implementation

Ensure that all security-related successes and failures can be logged. When storing data in the cloud (e.g., AWS S3 buckets, Azure blobs, Google Cloud Storage, etc.), use the provider's controls to enable and capture detailed logging information.

Phase: Operation

Be sure to set the level of logging appropriately in a production environment. Sufficient data should be logged to enable system administrators to detect attacks, diagnose errors, and recover from attacks. At the same time, logging too much data (CWE-779) can cause the same problems, including unexpected costs when using a cloud environment.

Phase: Operation

To enable storage logging using Azure's Portal, navigate to the name of the Storage Account, locate Monitoring (CLASSIC) section, and select Diagnostic settings (classic). For each of the various properties (blob, file, table, queue), ensure the status is properly set for the desired logging data. If using PowerShell, the `Set-AzStorageServiceLoggingProperty` command could be called using appropriate `-ServiceType`, `-LoggingOperations`, and `-RetentionDays` arguments.

Demonstrative Examples

Example 1:

The example below shows a configuration for the service security audit feature in the Windows Communication Foundation (WCF).

Example Language: XML

(Bad)

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
          suppressAuditFailure="false"
          serviceAuthorizationAuditLevel="None"
          messageAuthenticationAuditLevel="None" />
      ...
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

The previous configuration file has effectively disabled the recording of security-critical events, which would force the administrator to look to other sources during debug or recovery efforts.

Logging failed authentication attempts can warn administrators of potential brute force attacks. Similarly, logging successful authentication events can provide a useful audit trail when a legitimate account is compromised. The following configuration shows appropriate settings, assuming that the site does not have excessive traffic, which could fill the logs if there are a large number of success or failure events (CWE-779).

Example Language: XML

(Good)

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
          suppressAuditFailure="false"
          serviceAuthorizationAuditLevel="SuccessAndFailure"
          messageAuthenticationAuditLevel="SuccessAndFailure" />
      ...
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Example 2:

In the following Java example the code attempts to authenticate the user. If the login fails a retry is made. Proper restrictions on the number of login attempts are of course part of the retry functionality. Unfortunately, the failed login is not recorded and there would be no record of an adversary attempting to brute force the program.

Example Language: Java

(Bad)

```
if LoginUser(){
    // Login successful
    RunProgram();
} else {
    // Login unsuccessful
    LoginRetry();
}
```

It is recommended to log the failed login action. Note that unneutralized usernames should not be part of the log message, and passwords should never be part of the log message.

Example Language: Java

(Good)

```
if LoginUser(){
    // Login successful
```

```
log.warn("Login by user successful.");
RunProgram();
} else {
    // Login unsuccessful
    log.warn("Login attempt by user failed, trying again.");
    LoginRetry();
}
```

Example 3:

Consider this command for updating Azure's Storage Logging for Blob service, adapted from [REF-1307]:

Example Language: Shell

(Bad)

```
az storage logging update --account-name --account-key --services b --log d --retention 90
```

The "--log d" portion of the command says to log deletes. However, the argument does not include the logging of writes and reads. Adding the "rw" arguments to the -log parameter will fix the issue:

Example Language: Shell

(Good)

```
az storage logging update --account-name --account-key --services b --log rwd --retention 90
```

To enable Azure's storage analytic logs programmatically using PowerShell:

Example Language: Shell

(Good)

```
Set-AzStorageServiceLoggingProperty -ServiceType Queue -LoggingOperations read,write,delete -RetentionDays 5 -
Context $MyContextObject
```

Notice that here, the retention has been limited to 5 days.

Observed Examples

Reference	Description
CVE-2008-4315	server does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected https://www.cve.org/CVERecord?id=CVE-2008-4315
CVE-2008-1203	admin interface does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected https://www.cve.org/CVERecord?id=CVE-2008-1203
CVE-2007-3730	default configuration for POP server does not log source IP or username for login attempts https://www.cve.org/CVERecord?id=CVE-2007-3730
CVE-2007-1225	proxy does not log requests without "http://" in the URL, allowing web surfers to access restricted web content without detection https://www.cve.org/CVERecord?id=CVE-2007-1225
CVE-2003-1566	web server does not log requests for a non-standard request type https://www.cve.org/CVERecord?id=CVE-2003-1566

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		1036	OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring	1026	2439
MemberOf		1308	CISQ Quality Measures - Security	1305	2485

Nature	Type	ID	Name	V	Page
MemberOf		1355	OWASP Top Ten 2021 Category A09:2021 - Security Logging and Monitoring Failures	1344	2496
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

References

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

[REF-1307]Center for Internet Security. "CIS Microsoft Azure Foundations Benchmark version 1.5.0". 2022 August 6. < <https://www.cisecurity.org/benchmark/azure> >.2023-01-19.

[REF-1308]Microsoft. "Enable and manage Azure Storage Analytics logs (classic)". 2023 January 3. < <https://learn.microsoft.com/en-us/azure/storage/common/manage-storage-analytics-logs> >.2023-01-24.

CWE-779: Logging of Excessive Data

Weakness ID : 779

Structure : Simple

Abstraction : Base

Description

The product logs too much information, making log files hard to process and possibly hindering recovery efforts or forensic analysis after an attack.


Extended Description

While logging is a good practice in general, and very high levels of logging are appropriate for debugging stages of development, too much logging in a production environment might hinder a system administrator's ability to detect anomalous conditions. This can provide cover for an attacker while attempting to penetrate a system, clutter the audit trail for forensic analysis, or make it more difficult to debug problems in a production environment.

Relationships

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	964

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

Nature	Type	ID	Name	Page
MemberOf		1009	Audit	2424

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

Nature	Type	ID	Name	Page
MemberOf		1210	Audit / Logging Errors	2475

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Other) <i>Log files can become so large that they consume excessive resources, such as disk and CPU, which can hinder the performance of the system.</i>	
Non-Repudiation	Hide Activities <i>Logging too much information can make the log files of less use to forensics analysts and developers when trying to diagnose a problem or recover from an attack.</i>	
Non-Repudiation	Hide Activities <i>If system administrators are unable to effectively process log files, attempted attacks may go undetected, possibly leading to eventual system compromise.</i>	

Potential Mitigations**Phase: Architecture and Design**

Suppress large numbers of duplicate log messages and replace them with periodic summaries. For example, syslog may include an entry that states "last message repeated X times" when recording repeated events.

Phase: Architecture and Design

Support a maximum size for the log file that can be controlled by the administrator. If the maximum size is reached, the admin should be notified. Also, consider reducing functionality of the product. This may result in a denial-of-service to legitimate product users, but it will prevent the product from adversely impacting the entire system.

Phase: Implementation



Adjust configurations appropriately when the product is transitioned from a debug state to production.

Observed Examples

Reference	Description
CVE-2007-0421	server records a large amount of data to the server log when it receives malformed headers https://www.cve.org/CVERecord?id=CVE-2007-0421
CVE-2002-1154	chain: application does not restrict access to front-end for updates, which allows attacker to fill the error log https://www.cve.org/CVERecord?id=CVE-2002-1154

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		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 3-3		Req SR 7.2

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SD-1
ISA/IEC 62443	Part 4-1		Req SVV-3
ISA/IEC 62443	Part 4-2		Req CR 7.2

CWE-780: Use of RSA Algorithm without OAEP

Weakness ID : 780

Structure : Simple

Abstraction : Variant

Description

The product uses the RSA algorithm but does not incorporate Optimal Asymmetric Encryption Padding (OAEP), which might weaken the encryption.


Extended Description

Padding schemes are often used with cryptographic algorithms to make the plaintext less predictable and complicate attack efforts. The OAEP scheme is often used with RSA to nullify the impact of predictable common text.

Relationships

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		327	Use of a Broken or Risky Cryptographic Algorithm	799

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2428

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>Without OAEP in RSA encryption, it will take less work for an attacker to decrypt the data or to infer patterns from the ciphertext.</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

Demonstrative Examples

Example 1:

The example below attempts to build an RSA cipher.

Example Language: Java

(Bad)

```
public Cipher getRSACipher() {
    Cipher rsa = null;
    try {
        rsa = javax.crypto.Cipher.getInstance("RSA/NONE/NoPadding");
    }
    catch (java.security.NoSuchAlgorithmException e) {
        log("this should never happen", e);
    }
    catch (javax.crypto.NoSuchPaddingException e) {
        log("this should never happen", e);
    }
    return rsa;
}
```

While the previous code successfully creates an RSA cipher, the cipher does not use padding. The following code creates an RSA cipher using OAEP.

Example Language: Java

(Good)

```
public Cipher getRSACipher() {
    Cipher rsa = null;
    try {
        rsa = javax.crypto.Cipher.getInstance("RSA/ECB/OAEPWithMD5AndMGF1Padding");
    }
    catch (java.security.NoSuchAlgorithmException e) {
        log("this should never happen", e);
    }
    catch (javax.crypto.NoSuchPaddingException e) {
        log("this should never happen", e);
    }
    return rsa;
}
```

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	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1402	Comprehensive Categorization: Encryption	1400	2527

Notes

Maintenance

This entry could probably have a new parent related to improper padding, however the role of padding in cryptographic algorithms can vary, such as hiding the length of the plaintext and providing additional random bits for the cipher. In general, cryptographic problems in CWE are not well organized and further research is needed.

References

[REF-694]Ronald L. Rivest and Burt Kaliski. "RSA Problem". 2003 December 0. < <http://people.csail.mit.edu/rivest/RivestKaliski-RSAProblem.pdf> >.

[REF-695]"Optimal Asymmetric Encryption Padding". 2009 July 8. Wikipedia. < https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding >.2023-04-07.

CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code

Weakness ID : 781

Structure : Simple

Abstraction : Variant

Description

The product defines an IOCTL that uses METHOD_NEITHER for I/O, but it does not validate or incorrectly validates the addresses that are provided.




Extended Description

When an IOCTL uses the METHOD_NEITHER option for I/O control, it is the responsibility of the IOCTL to validate the addresses that have been supplied to it. If validation is missing or incorrect, attackers can supply arbitrary memory addresses, leading to code execution or a denial of service.

Relationships

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		1285	Improper Validation of Specified Index, Position, or Offset in Input	2132
CanFollow		782	Exposed IOCTL with Insufficient Access Control	1648
CanPrecede		822	Untrusted Pointer Dereference	1723

Applicable Platforms

Language : C (Prevalence = Often)

Language : C++ (Prevalence = Often)

Operating_System : Windows NT (Prevalence = Sometimes)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	Read Memory	
Confidentiality	Execute Unauthorized Code or Commands	
	DoS: Crash, Exit, or Restart	
	An attacker may be able to access memory that belongs to another process or user. If the attacker can control the contents that the IOCTL writes, it may lead to code execution at high privilege levels. At the least, a crash can occur.	

Potential Mitigations

Phase: Implementation

If METHOD_NEITHER is required for the IOCTL, then ensure that all user-space addresses are properly validated before they are first accessed. The ProbeForRead and ProbeForWrite routines

are available for this task. Also properly protect and manage the user-supplied buffers, since the I/O Manager does not do this when METHOD_NEITHER is being used. See References.

Phase: Architecture and Design

If possible, avoid using METHOD_NEITHER in the IOCTL and select methods that effectively control the buffer size, such as METHOD_BUFFERED, METHOD_IN_DIRECT, or METHOD_OUT_DIRECT.

Phase: Architecture and Design

Phase: Implementation


If the IOCTL is part of a driver that is only intended to be accessed by trusted users, then use proper access control for the associated device or device namespace. See References.

Observed Examples

Reference	Description
CVE-2006-2373	Driver for file-sharing and messaging protocol allows attackers to execute arbitrary code. https://www.cve.org/CVERecord?id=CVE-2006-2373
CVE-2009-0686	Anti-virus product does not validate addresses, allowing attackers to gain SYSTEM privileges. https://www.cve.org/CVERecord?id=CVE-2009-0686
CVE-2009-0824	DVD software allows attackers to cause a crash. https://www.cve.org/CVERecord?id=CVE-2009-0824
CVE-2008-5724	Personal firewall allows attackers to gain SYSTEM privileges. https://www.cve.org/CVERecord?id=CVE-2008-5724
CVE-2007-5756	chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error. https://www.cve.org/CVERecord?id=CVE-2007-5756

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		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Notes

Applicable Platform

Because IOCTL functionality is typically performing low-level actions and closely interacts with the operating system, this weakness may only appear in code that is written in low-level languages.

Research Gap

While this type of issue has been known since 2006, it is probably still under-studied and under-reported. Most of the focus has been on high-profile software and security products, but other kinds of system software also use drivers. Since exploitation requires the development of custom code, it requires some skill to find this weakness. Because exploitation typically requires local privileges, it might not be a priority for active attackers. However, remote exploitation may be possible for software such as device drivers. Even when remote vectors are not available, it may be useful as the final privilege-escalation step in multi-stage remote attacks against application-layer software, or as the primary attack by a local user on a multi-user system.

References

[REF-696]Ruben Santamarta. "Exploiting Common Flaws in Drivers". 2007 July 1. < http://reversemode.com/index.php?option=com_content&task=view&id=38&Itemid=1 >.

[REF-697]Yuriy Bulygin. "Remote and Local Exploitation of Network Drivers". 2007 August 1. < <https://www.blackhat.com/presentations/bh-usa-07/Bulygin/Presentation/bh-usa-07-bulygin.pdf> >.

[REF-698]Anibal Sacco. "Windows driver vulnerabilities: the METHOD_NEITHER odyssey". 2008 October. < <http://www.net-security.org/dl/insecure/INSECURE-Mag-18.pdf> >.

[REF-699]Microsoft. "Buffer Descriptions for I/O Control Codes". < <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/buffer-descriptions-for-i-o-control-codes> >.2023-04-07.

[REF-700]Microsoft. "Using Neither Buffered Nor Direct I/O". < <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/using-neither-buffered-nor-direct-i-o> >.2023-04-07.

[REF-701]Microsoft. "Securing Device Objects". < <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/controlling-device-access> >.2023-04-07.

[REF-702]Piotr Bania. "Exploiting Windows Device Drivers". < <https://www.piotrbania.com/all/articles/ewdd.pdf> >.2023-04-07.

CWE-782: Exposed IOCTL with Insufficient Access Control

Weakness ID : 782

Structure : Simple

Abstraction : Variant

Description

The product implements an IOCTL with functionality that should be restricted, but it does not properly enforce access control for the IOCTL.

Extended Description



When an IOCTL contains privileged functionality and is exposed unnecessarily, attackers may be able to access this functionality by invoking the IOCTL. Even if the functionality is benign, if the programmer has assumed that the IOCTL would only be accessed by a trusted process, there may be little or no validation of the incoming data, exposing weaknesses that would never be reachable if the attacker cannot call the IOCTL directly.

The implementations of IOCTLs will differ between operating system types and versions, so the methods of attack and prevention may vary widely.

Relationships

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		749	Exposed Dangerous Method or Function	1564
CanPrecede		781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1646

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2425

Applicable Platforms

Language : C (Prevalence = Often)

Language : C++ (Prevalence = Often)

Operating_System : Unix (Prevalence = Undetermined)

Operating_System : Windows (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Integrity Availability Confidentiality	Varies by Context <i>Attackers can invoke any functionality that the IOCTL offers. Depending on the functionality, the consequences may include code execution, denial-of-service, and theft of data.</i>	

Potential Mitigations

Phase: Architecture and Design


In Windows environments, use proper access control for the associated device or device namespace. See References.

Observed Examples

Reference	Description
CVE-2009-2208	Operating system does not enforce permissions on an IOCTL that can be used to modify network settings. https://www.cve.org/CVERecord?id=CVE-2009-2208
CVE-2008-3831	Device driver does not restrict ioctl calls to its direct rendering manager. https://www.cve.org/CVERecord?id=CVE-2008-3831
CVE-2008-3525	ioctl does not check for a required capability before processing certain requests. https://www.cve.org/CVERecord?id=CVE-2008-3525
CVE-2008-0322	Chain: insecure device permissions allows access to an IOCTL, allowing arbitrary memory to be overwritten. https://www.cve.org/CVERecord?id=CVE-2008-0322
CVE-2007-4277	Chain: anti-virus product uses weak permissions for a device, leading to resultant buffer overflow in an exposed IOCTL. https://www.cve.org/CVERecord?id=CVE-2007-4277
CVE-2007-1400	Chain: sandbox allows opening of a TTY device, enabling shell commands through an exposed ioctl. https://www.cve.org/CVERecord?id=CVE-2007-1400
CVE-2006-4926	Anti-virus product uses insecure security descriptor for a device driver, allowing access to a privileged IOCTL. https://www.cve.org/CVERecord?id=CVE-2006-4926
CVE-1999-0728	Unauthorized user can disable keyboard or mouse by directly invoking a privileged IOCTL. https://www.cve.org/CVERecord?id=CVE-1999-0728

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		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Relationship

This can be primary to many other weaknesses when the programmer assumes that the IOCTL can only be accessed by trusted parties. For example, a program or driver might not validate incoming addresses in METHOD_NEITHER IOCTLs in Windows environments (CWE-781), which could allow buffer overflow and similar attacks to take place, even when the attacker never should have been able to access the IOCTL at all.

Applicable Platform

Because IOCTL functionality is typically performing low-level actions and closely interacts with the operating system, this weakness may only appear in code that is written in low-level languages.

References

[REF-701]Microsoft. "Securing Device Objects". < <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/controlling-device-access> >.2023-04-07.

CWE-783: Operator Precedence Logic Error

Weakness ID : 783

Structure : Simple

Abstraction : Base

Description

The product uses an expression in which operator precedence causes incorrect logic to be used.

Extended Description

While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.



Relationships

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	1475

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2326
MemberOf		569	Expression Issues	2330

Applicable Platforms

Language : C (Prevalence = Rarely)

Language : C++ (Prevalence = Rarely)

Language : Not Language-Specific (Prevalence = Rarely)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Varies by Context Unexpected State <i>The consequences will vary based on the context surrounding the incorrect precedence. In a security decision, integrity or confidentiality are the most likely results. Otherwise, a crash may occur due to the software reaching an unexpected state.</i>	

Potential Mitigations

Phase: Implementation

Regularly wrap sub-expressions in parentheses, especially in security-critical code.

Demonstrative Examples

Example 1:

In the following example, the method `validateUser` makes a call to another method to authenticate a username and password for a user and returns a success or failure code.

Example Language: C

(Bad)

```
#define FAIL 0
#define SUCCESS 1
...
int validateUser(char *username, char *password) {
    int isUser = FAIL;
    // call method to authenticate username and password
    // if authentication fails then return failure otherwise return success
    if (isUser = AuthenticateUser(username, password) == FAIL) {
        return isUser;
    }
    else {
        isUser = SUCCESS;
    }
    return isUser;
}
```

However, the method that authenticates the username and password is called within an if statement with incorrect operator precedence logic. Because the comparison operator `"=="` has a higher precedence than the assignment operator `"="`, the comparison operator will be evaluated first and if the method returns `FAIL` then the comparison will be true, the return variable will be set to true and `SUCCESS` will be returned. This operator precedence logic error can be easily resolved by properly using parentheses within the expression of the if statement, as shown below.

Example Language: C

(Good)

```
...
if ((isUser = AuthenticateUser(username, password)) == FAIL) {
    ...
}
```

Example 2:

In this example, the method calculates the return on investment for an accounting/financial application. The return on investment is calculated by subtracting the initial investment costs from the current value and then dividing by the initial investment costs.

Example Language: Java

(Bad)

```
public double calculateReturnOnInvestment(double currentValue, double initialInvestment) {
    double returnROI = 0.0;
    // calculate return on investment
}
```

```

returnROI = currentValue - initialInvestment / initialInvestment;
return returnROI;
}

```

However, the return on investment calculation will not produce correct results because of the incorrect operator precedence logic in the equation. The divide operator has a higher precedence than the minus operator, therefore the equation will divide the initial investment costs by the initial investment costs which will only subtract one from the current value. Again this operator precedence logic error can be resolved by the correct use of parentheses within the equation, as shown below.

Example Language: Java

(Good)

```

...
returnROI = (currentValue - initialInvestment) / initialInvestment;
...

```







Note that the initialInvestment variable in this example should be validated to ensure that it is greater than zero to avoid a potential divide by zero error (CWE-369).

Observed Examples

Reference	Description
CVE-2008-2516	Authentication module allows authentication bypass because it uses "(x = call(args) == SUCCESS)" instead of "((x = call(args)) == SUCCESS)". https://www.cve.org/CVERecord?id=CVE-2008-2516
CVE-2008-0599	Chain: Language interpreter calculates wrong buffer size (CWE-131) by using "size = ptr ? X : Y" instead of "size = (ptr ? X : Y)" expression. https://www.cve.org/CVERecord?id=CVE-2008-0599
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions. https://www.cve.org/CVERecord?id=CVE-2001-1155

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		737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2341
MemberOf		884	CWE Cross-section	884	2567
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2466
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf		1308	CISQ Quality Measures - Security	1305	2485
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP00-C	Exact	Use parentheses for precedence of operation
SEI CERT Perl Coding Standard	EXP04-PL	CWE More Abstract	Do not mix the early-precedence logical operators with late-precedence logical operators