

CWE-838: Inappropriate Encoding for Output Context

Weakness ID : 838
Structure : Simple
Abstraction : Base

Description

The product uses or specifies an encoding when generating output to a downstream component, but the specified encoding is not the same as the encoding that is expected by the downstream component.

Extended Description

This weakness can cause the downstream component to use a decoding method that produces different data than what the product intended to send. When the wrong encoding is used - even if closely related - the downstream component could decode the data incorrectly. This can have security consequences when the provided boundaries between control and data are inadvertently broken, because the resulting data could introduce control characters or special elements that were not sent by the product. The resulting data could then be used to bypass protection mechanisms such as input validation, and enable injection attacks.

While using output encoding is essential for ensuring that communications between components are accurate, the use of the wrong encoding - even if closely related - could cause the downstream component to misinterpret the output.


For example, HTML entity encoding is used for elements in the HTML body of a web page. However, a programmer might use entity encoding when generating output for that is used within an attribute of an HTML tag, which could contain functional Javascript that is not affected by the HTML encoding.

While web applications have received the most attention for this problem, this weakness could potentially apply to any type of product that uses a communications stream that could support multiple encodings.


Relationships

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


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

Nature	Type	ID	Name	Page
ChildOf		116	Improper Encoding or Escaping of Output	287

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

Nature	Type	ID	Name	Page
ChildOf		116	Improper Encoding or Escaping of Output	287

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2332

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	<i>An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.</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 = Output Encoding

Use context-aware encoding. That is, understand which encoding is being used by the downstream component, and ensure that this encoding is used. If an encoding can be specified, do so, instead of assuming that the default encoding is the same as the default being assumed by the downstream component.

Phase: Architecture and Design

Strategy = Output Encoding

Where possible, use communications protocols or data formats that provide strict boundaries between control and data. If this is not feasible, ensure that the protocols or formats allow the communicating components to explicitly state which encoding/decoding method is being used. Some template frameworks provide built-in support.

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, consider using the ESAPI Encoding control [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. Note that some template mechanisms provide built-in support for the appropriate encoding.

Demonstrative Examples

Example 1:

This code dynamically builds an HTML page using POST data:

Example Language: PHP

(Bad)

```
$username = $_POST['username'];
$picSource = $_POST['picsource'];
$picAltText = $_POST['picalttext'];
...
echo "<title>Welcome, " . htmlentities($username) . "</title>";
echo "<img src=\"" . htmlentities($picSource) . "\" alt=\"" . htmlentities($picAltText) . "\" />";
...
```

The programmer attempts to avoid XSS exploits (CWE-79) by encoding the POST values so they will not be interpreted as valid HTML. However, the `htmlentities()` encoding is not appropriate when the data are used as HTML attributes, allowing more attributes to be injected.

For example, an attacker can set `picAltText` to:

Example Language:

(Attack)

```
"altTextHere' onload='alert(document.cookie)"
```

This will result in the generated HTML image tag:

Example Language: HTML

(Result)

```
<img src='pic.jpg' alt='altTextHere' onload='alert(document.cookie)' />
```

The attacker can inject arbitrary javascript into the tag due to this incorrect encoding.

Observed Examples

Reference	Description
CVE-2009-2814	Server does not properly handle requests that do not contain UTF-8 data; browser assumes UTF-8, allowing XSS. https://www.cve.org/CVERecord?id=CVE-2009-2814

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	2383
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	1138	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR)	1133	2467
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS13-J		Use compatible encodings on both sides of file or network IO

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
468	Generic Cross-Browser Cross-Domain Theft

References

[REF-786]Jim Manico. "Injection-safe templating languages". 2010 June 0. < https://manicode.blogspot.com/2010/06/injection-safe-templating-languages_30.html >.2023-04-07.

[REF-787]Dinis Cruz. "Can we please stop saying that XSS is boring and easy to fix!". 2010 September 5. < <http://diniscruz.blogspot.com/2010/09/can-we-please-stop-saying-that-xss-is.html> >.

[REF-788]Ivan Ristic. "Canoe: XSS prevention via context-aware output encoding". 2010 September 4. < <https://blog.ivanristic.com/2010/09/introducing-canoe-context-aware-output-encoding-for-xss-prevention.html> >.2023-04-07.

[REF-789]Jim Manico. "What is the Future of Automated XSS Defense Tools?". 2011 March 8. < <http://software-security.sans.org/downloads/appsec-2011-files/manico-appsec-future-tools.pdf> >.

[REF-709]Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". 2007. Syngress.

[REF-725]OWASP. "DOM based XSS Prevention Cheat Sheet". < http://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet >.

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

CWE-839: Numeric Range Comparison Without Minimum Check

Weakness ID : 839

Structure : Simple

Abstraction : Base

Description

The product checks a value to ensure that it is less than or equal to a maximum, but it does not also verify that the value is greater than or equal to the minimum.

Extended Description

Some products use signed integers or floats even when their values are only expected to be positive or 0. An input validation check might assume that the value is positive, and only check for the maximum value. If the value is negative, but the code assumes that the value is positive, this can produce an error. The error may have security consequences if the negative value is used for memory allocation, array access, buffer access, etc. Ultimately, the error could lead to a buffer overflow or other type of memory corruption.

The use of a negative number in a positive-only context could have security implications for other types of resources. For example, a shopping cart might check that the user is not requesting more than 10 items, but a request for -3 items could cause the application to calculate a negative price and credit the attacker's account.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1023	Incomplete Comparison with Missing Factors	1874
CanPrecede		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299
CanPrecede		124	Buffer Underwrite ('Buffer Underflow')	332
CanPrecede		195	Signed to Unsigned Conversion Error	501
CanPrecede		682	Incorrect Calculation	1507

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

Nature	Type	ID	Name	Page
MemberOf		189	Numeric Errors	2333

Applicable Platforms

Language : C (*Prevalence = Often*)

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

Alternate Terms

Signed comparison : The "signed comparison" term is often used to describe when the product uses a signed variable and checks it to ensure that it is less than a maximum value (typically a maximum buffer size), but does not verify that it is greater than 0.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Execute Unauthorized Code or Commands	
Availability	<i>An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.</i>	
Availability	DoS: Resource Consumption (Other)	
	<i>in some contexts, a negative value could lead to resource consumption.</i>	
Confidentiality	Modify Memory	
Integrity	Read Memory	
	<i>If a negative value is used to access memory, buffers, or other indexable structures, it could access memory outside the bounds of the buffer.</i>	

Potential Mitigations

Phase: Implementation

Strategy = Enforcement by Conversion

If the number to be used is always expected to be positive, change the variable type from signed to unsigned or size_t.

Phase: Implementation

Strategy = Input Validation

If the number to be used could have a negative value based on the specification (thus requiring a signed value), but the number should only be positive to preserve code correctness, then include a check to ensure that the value is positive.

Demonstrative Examples

Example 1:

The following code is intended to read an incoming packet from a socket and extract one or more headers.

Example Language: C

(Bad)

```
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
```

```

}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);

```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 2:

The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

Example Language: C

(Bad)

```

int GetUntrustedInt () {
    return(0x0000FFFF);
}

void main (int argc, char **argv) {
    char path[256];
    char *input;
    int i;
    short s;
    unsigned int sz;
    i = GetUntrustedInt();
    s = i;
    /* s is -1 so it passes the safety check - CWE-697 */
    if (s > 256) {
        DiePainfully("go away!\n");
    }
    /* s is sign-extended and saved in sz */
    sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
    printf("i=%d, s=%d, sz=%u\n", i, s, sz);
    input = Get userInput("Enter pathname:");
    /* strncpy interprets s as unsigned int, so it's treated as MAX_INT
    (CWE-195), enabling buffer overflow (CWE-119) */
    strncpy(path, input, s);
    path[255] = '\0'; /* don't want CWE-170 */
    printf("Path is: %s\n", path);
}

```

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

Example 3:

In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

Example Language: C

(Bad)

```

int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array

```

```

if (index < len) {
    // get the value at the specified index of the array
    value = array[index];
}
// if array index is invalid then output error message
// and return value indicating error
else {
    printf("Value is: %d\n", array[index]);
    value = -1;
}
return value;
}

```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

Example Language: C

(Good)

```

...
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...

```

Example 4:

The following code shows a simple BankAccount class with deposit and withdraw methods.

Example Language: Java

(Bad)

```

public class BankAccount {
    public final int MAXIMUM_WITHDRAWAL_LIMIT = 350;
    // variable for bank account balance
    private double accountBalance;
    // constructor for BankAccount
    public BankAccount() {
        accountBalance = 0;
    }
    // method to deposit amount into BankAccount
    public void deposit(double depositAmount) {...}
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        if (withdrawAmount < MAXIMUM_WITHDRAWAL_LIMIT) {
            double newBalance = accountBalance - withdrawAmount;
            accountBalance = newBalance;
        }
        else {
            System.err.println("Withdrawal amount exceeds the maximum limit allowed, please try again...");
            ...
        }
    }
    // other methods for accessing the BankAccount object
    ...
}

```

The withdraw method includes a check to ensure that the withdrawal amount does not exceed the maximum limit allowed, however the method does not check to ensure that the withdrawal amount is greater than a minimum value (CWE-129). Performing a range check on a value that does not include a minimum check can have significant security implications, in this case not including a minimum range check can allow a negative value to be used which would cause the financial

application using this class to deposit money into the user account rather than withdrawing. In this example the if statement should be modified to include a minimum range check, as shown below.

Example Language: Java

(Good)

```
public class BankAccount {
    public final int MINIMUM_WITHDRAWAL_LIMIT = 0;
    public final int MAXIMUM_WITHDRAWAL_LIMIT = 350;
    ...
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        if (withdrawAmount < MAXIMUM_WITHDRAWAL_LIMIT &&
            withdrawAmount > MINIMUM_WITHDRAWAL_LIMIT) {
            ...
        }
    }
}
```

Note that this example does not protect against concurrent access to the BankAccount balance variable, see CWE-413 and CWE-362.

While it is out of scope for this example, note that the use of doubles or floats in financial calculations may be subject to certain kinds of attacks where attackers use rounding errors to steal money.

Observed Examples

Reference	Description
CVE-2010-1866	Chain: integer overflow (CWE-190) causes a negative signed value, which later bypasses a maximum-only check (CWE-839), leading to heap-based buffer overflow (CWE-122). https://www.cve.org/CVERecord?id=CVE-2010-1866
CVE-2009-1099	Chain: 16-bit counter can be interpreted as a negative value, compared to a 32-bit maximum value, leading to buffer under-write. https://www.cve.org/CVERecord?id=CVE-2009-1099
CVE-2011-0521	Chain: kernel's lack of a check for a negative value leads to memory corruption. https://www.cve.org/CVERecord?id=CVE-2011-0521
CVE-2010-3704	Chain: parser uses atoi() but does not check for a negative value, which can happen on some platforms, leading to buffer under-write. https://www.cve.org/CVERecord?id=CVE-2010-3704
CVE-2010-2530	Chain: Negative value stored in an int bypasses a size check and causes allocation of large amounts of memory. https://www.cve.org/CVERecord?id=CVE-2010-2530
CVE-2009-3080	Chain: negative offset value to IOCTL bypasses check for maximum index, then used as an array index for buffer under-read. https://www.cve.org/CVERecord?id=CVE-2009-3080
CVE-2008-6393	chain: file transfer client performs signed comparison, leading to integer overflow and heap-based buffer overflow. https://www.cve.org/CVERecord?id=CVE-2008-6393
CVE-2008-4558	chain: negative ID in media player bypasses check for maximum index, then used as an array index for buffer under-read. https://www.cve.org/CVERecord?id=CVE-2008-4558

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1397	Comprehensive Categorization: Comparison	1400	2544

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.

CWE-841: Improper Enforcement of Behavioral Workflow

Weakness ID : 841

Structure : Simple

Abstraction : Base

Description

The product supports a session in which more than one behavior must be performed by an actor, but it does not properly ensure that the actor performs the behaviors in the required sequence.

Extended Description

By performing actions in an unexpected order, or by omitting steps, an attacker could manipulate the business logic of the product or cause it to enter an invalid state. In some cases, this can also expose resultant weaknesses.

For example, a file-sharing protocol might require that an actor perform separate steps to provide a username, then a password, before being able to transfer files. If the file-sharing server accepts a password command followed by a transfer command, without any username being provided, the product might still perform the transfer.

Note that this is different than CWE-696, which focuses on when the product performs actions in the wrong sequence; this entry is closely related, but it is focused on ensuring that the actor performs actions in the correct sequence.

Workflow-related behaviors include:

- Steps are performed in the expected order.
- Required steps are not omitted.
- Steps are not interrupted.
- Steps are performed in a timely fashion.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf	[C]	1018	Manage User Sessions	2453

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

Nature	Type	ID	Name	Page
MemberOf	[C]	1217	User Session Errors	2500

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348
MemberOf		840	Business Logic Errors	2381

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic <i>An attacker could cause the product to skip critical steps or perform them in the wrong order, bypassing its intended business logic. This can sometimes have security implications.</i>	

Demonstrative Examples

Example 1:

This code is part of an FTP server and deals with various commands that could be sent by a user. It is intended that a user must successfully login before performing any other action such as retrieving or listing files.

Example Language: Python

(Bad)

```
def dispatchCommand(command, user, args):
    if command == 'Login':
        loginUser(args)
        return
    # user has requested a file
    if command == 'Retrieve_file':
        if authenticated(user) and ownsFile(user,args):
            sendFile(args)
            return
    if command == 'List_files':
        listFiles(args)
        return
    ...
```

The server correctly avoids sending files to a user that isn't logged in and doesn't own the file. However, the server will incorrectly list the files in any directory without confirming the command came from an authenticated user, and that the user is authorized to see the directory's contents.

Here is a fixed version of the above example:

Example Language: Python

(Good)

```
def dispatchCommand(command, user, args):
    ...
    if command == 'List_files':
        if authenticated(user) and ownsDirectory(user,args):
            listFiles(args)
            return
    ...
```

Observed Examples

Reference	Description
CVE-2011-0348	Bypass of access/billing restrictions by sending traffic to an unrestricted destination before sending to a restricted destination. https://www.cve.org/CVERecord?id=CVE-2011-0348
CVE-2007-3012	Attacker can access portions of a restricted page by canceling out of a dialog. https://www.cve.org/CVERecord?id=CVE-2007-3012
CVE-2009-5056	Ticket-tracking system does not enforce a permission setting. https://www.cve.org/CVERecord?id=CVE-2009-5056

Reference	Description
CVE-2004-2164	Shopping cart does not close a database connection when user restores a previous order, leading to connection exhaustion. https://www.cve.org/CVERecord?id=CVE-2004-2164
CVE-2003-0777	Chain: product does not properly handle dropped connections, leading to missing NULL terminator (CWE-170) and segmentation fault. https://www.cve.org/CVERecord?id=CVE-2003-0777
CVE-2005-3327	Chain: Authentication bypass by skipping the first startup step as required by the protocol. https://www.cve.org/CVERecord?id=CVE-2005-3327
CVE-2004-0829	Chain: File server crashes when sent a "find next" request without an initial "find first." https://www.cve.org/CVERecord?id=CVE-2004-0829
CVE-2010-2620	FTP server allows remote attackers to bypass authentication by sending (1) LIST, (2) RETR, (3) STOR, or other commands without performing the required login steps first. https://www.cve.org/CVERecord?id=CVE-2010-2620
CVE-2005-3296	FTP server allows remote attackers to list arbitrary directories as root by running the LIST command before logging in. https://www.cve.org/CVERecord?id=CVE-2005-3296

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	867	2011 Top 25 - Weaknesses On the Cusp	900	2393
MemberOf	V	884	CWE Cross-section	884	2588
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

Notes

Research Gap

This weakness is typically associated with business logic flaws, except when it produces resultant weaknesses. The classification of business logic flaws has been under-studied, although exploitation of business flaws frequently happens in real-world systems, and many applied vulnerability researchers investigate them. The greatest focus is in web applications. There is debate within the community about whether these problems represent particularly new concepts, or if they are variations of well-known principles. Many business logic flaws appear to be oriented toward business processes, application flows, and sequences of behaviors, which are not as well-represented in CWE as weaknesses related to input validation, memory management, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	40		Insufficient Process Validation

References

[REF-795]Jeremiah Grossman. "Business Logic Flaws and Yahoo Games". 2006 December 8. <
<https://blog.jeremiahgrossman.com/2006/12/business-logic-flaws.html> >.2023-04-07.

[REF-796]Jeremiah Grossman. "Seven Business Logic Flaws That Put Your Website At Risk". 2007 October. < <https://docplayer.net/10021793-Seven-business-logic-flaws-that-put-your-website-at-risk.html> >.2023-04-07.

[REF-797]WhiteHat Security. "Business Logic Flaws". < https://web.archive.org/web/20080720171327/http://www.whitehatsec.com/home/solutions/BL_auction.html >.2023-04-07.

[REF-806]WASC. "Insufficient Process Validation". < <http://projects.webappsec.org/w/page/13246943/Insufficient-Process-Validation> >.

[REF-799]Rafal Los and Prajakta Jagdale. "Defying Logic: Theory, Design, and Implementation of Complex Systems for Testing Application Logic". 2011. < <https://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation> >.2023-04-07.

[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-801]Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel and Giovanni Vigna. "Toward Automated Detection of Logic Vulnerabilities in Web Applications". USENIX Security Symposium 2010. 2010 August. < https://www.usenix.org/legacy/events/sec10/tech/full_papers/Felmetsger.pdf >.2023-04-07.

[REF-802]Faisal Nabi. "Designing a Framework Method for Secure Business Application Logic Integrity in e-Commerce Systems". International Journal of Network Security, Vol.12, No.1. 2011. < <http://ijns.femto.com.tw/contents/ijns-v12-n1/ijns-2011-v12-n1-p29-41.pdf> >.

CWE-842: Placement of User into Incorrect Group

Weakness ID : 842

Structure : Simple

Abstraction : Base

Description

The product or the administrator places a user into an incorrect group.

Extended Description

If the incorrect group has more access or privileges than the intended group, the user might be able to bypass intended security policy to access unexpected resources or perform unexpected actions. The access-control system might not be able to detect malicious usage of this group membership.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		286	Incorrect User Management	698

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2497

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Observed Examples

Reference	Description
CVE-1999-1193	Operating system assigns user to privileged wheel group, allowing the user to gain root privileges. https://www.cve.org/CVERecord?id=CVE-1999-1193
CVE-2010-3716	Chain: drafted web request allows the creation of users with arbitrary group membership. https://www.cve.org/CVERecord?id=CVE-2010-3716
CVE-2008-5397	Chain: improper processing of configuration options causes users to contain unintended group memberships. https://www.cve.org/CVERecord?id=CVE-2008-5397
CVE-2007-6644	CMS does not prevent remote administrators from promoting other users to the administrator group, in violation of the intended security model. https://www.cve.org/CVERecord?id=CVE-2007-6644
CVE-2007-3260	Product assigns members to the root group, allowing escalation of privileges. https://www.cve.org/CVERecord?id=CVE-2007-3260
CVE-2002-0080	Chain: daemon does not properly clear groups before dropping privileges. https://www.cve.org/CVERecord?id=CVE-2002-0080

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')

Weakness ID : 843

Structure : Simple

Abstraction : Base

Description

The product allocates or initializes a resource such as a pointer, object, or variable using one type, but it later accesses that resource using a type that is incompatible with the original type.

Extended Description

When the product accesses the resource using an incompatible type, this could trigger logical errors because the resource does not have expected properties. In languages without memory safety, such as C and C++, type confusion can lead to out-of-bounds memory access.

While this weakness is frequently associated with unions when parsing data with many different embedded object types in C, it can be present in any application that can interpret the same variable or memory location in multiple ways.

This weakness is not unique to C and C++. For example, errors in PHP applications can be triggered by providing array parameters when scalars are expected, or vice versa. Languages such as Perl, which perform automatic conversion of a variable of one type when it is accessed as if it were another type, can also contain these issues.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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
PeerOf		1287	Improper Validation of Specified Type of Input	2150
CanPrecede		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	299

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 "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		136	Type Errors	2331

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Alternate Terms

Object Type Confusion :

Common Consequences

Scope	Impact	Likelihood
Availability	Read Memory	
Integrity	Modify Memory	
Confidentiality	Execute Unauthorized Code or Commands	
	DoS: Crash, Exit, or Restart	
	<i>When a memory buffer is accessed using the wrong type, it could read or write memory out of the bounds of the buffer, if the allocated buffer is smaller than the type that the code is attempting to access, leading to a crash and possibly code execution.</i>	

Demonstrative Examples

Example 1:

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.

Example 2:

The following PHP code accepts a value, adds 5, and prints the sum.

Example Language: PHP

(Bad)

```

$value = $_GET['value'];
$sum = $value + 5;
echo "value parameter is '$value'<p>";
echo "SUM is $sum";

```

When called with the following query string:

```
value=123
```

the program calculates the sum and prints out:

```
SUM is 128
```

However, the attacker could supply a query string such as:

```
value[]=123
```

The "[]" array syntax causes \$value to be treated as an array type, which then generates a fatal error when calculating \$sum:

Fatal error: Unsupported operand types in program.php on line 2

Example 3:

The following Perl code is intended to look up the privileges for user ID's between 0 and 3, by performing an access of the \$UserPrivilegeArray reference. It is expected that only userID 3 is an admin (since this is listed in the third element of the array).

Example Language: Perl

(Bad)

```
my $UserPrivilegeArray = ["user", "user", "admin", "user"];
my $userID = get_current_user_ID();
if ($UserPrivilegeArray eq "user") {
    print "Regular user!\n";
}
else {
    print "Admin!\n";
}
print "\$UserPrivilegeArray = $UserPrivilegeArray\n";
```

In this case, the programmer intended to use "\$UserPrivilegeArray->{\$userID}" to access the proper position in the array. But because the subscript was omitted, the "user" string was compared to the scalar representation of the \$UserPrivilegeArray reference, which might be of the form "ARRAY(0x229e8)" or similar.

Since the logic also "fails open" (CWE-636), the result of this bug is that all users are assigned administrator privileges.



While this is a forced example, it demonstrates how type confusion can have security consequences, even in memory-safe languages.

Observed Examples

Reference	Description
CVE-2010-4577	Type confusion in CSS sequence leads to out-of-bounds read. https://www.cve.org/CVERecord?id=CVE-2010-4577
CVE-2011-0611	Size inconsistency allows code execution, first discovered when it was actively exploited in-the-wild. https://www.cve.org/CVERecord?id=CVE-2011-0611
CVE-2010-0258	Improperly-parsed file containing records of different types leads to code execution when a memory location is interpreted as a different object than intended. https://www.cve.org/CVERecord?id=CVE-2010-0258

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Applicable Platform

This weakness is possible in any type-unsafe programming language.

Research Gap

Type confusion weaknesses have received some attention by applied researchers and major software vendors for C and C++ code. Some publicly-reported vulnerabilities probably have type confusion as a root-cause weakness, but these may be described as "memory corruption" instead. For other languages, there are very few public reports of type confusion weaknesses.

These are probably under-studied. Since many programs rely directly or indirectly on loose typing, a potential "type confusion" behavior might be intentional, possibly requiring more manual analysis.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP39-C	Exact	Do not access a variable through a pointer of an incompatible type

References

[REF-811]Mark Dowd, Ryan Smith and David Dewey. "Attacking Interoperability". 2009. < http://hustlelabs.com/stuff/bh2009_dowd_smith_dewey.pdf >.2023-04-07.

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

CWE-862: Missing Authorization

Weakness ID : 862

Structure : Simple

Abstraction : Class






Description

The product does not perform an authorization check when an actor attempts to access a resource or perform an action.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		285	Improper Authorization	691
ParentOf		425	Direct Request ('Forced Browsing')	1032
ParentOf		638	Not Using Complete Mediation	1413
ParentOf		939	Improper Authorization in Handler for Custom URL Scheme	1849
ParentOf		1314	Missing Write Protection for Parametric Data Values	2199

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

Nature	Type	ID	Name	Page
ParentOf		425	Direct Request ('Forced Browsing')	1032

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	687

Applicable Platforms

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

Technology : Web Server (Prevalence = Often)

Technology : Database Server (Prevalence = Often)

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Alternate Terms

AuthZ : "AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is distinct from "AuthN" (or, sometimes, "AuthC") which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories <i>An attacker could read sensitive data, either by reading the data directly from a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to read the data.</i>	
Integrity	Modify Application Data Modify Files or Directories <i>An attacker could modify sensitive data, either by writing the data directly to a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to write the data.</i>	
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism <i>An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.</i>	
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) <i>An attacker could gain unauthorized access to resources on the system and excessively consume those resources, leading to a denial of service.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers,

or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Effectiveness = Limited

Automated Dynamic Analysis

Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic.

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 custom authorization mechanisms.

Effectiveness = Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

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: Host Application Interface Scanner Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: 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

Divide the product into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) [REF-229] to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor [REF-7].

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, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45].

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phase: System Configuration

Phase: Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Demonstrative Examples

Example 1:

This function runs an arbitrary SQL query on a given database, returning the result of the query.

Example Language: PHP

(Bad)

```
function runEmployeeQuery($dbName, $name){
    mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database ".$dbName);
    //Use a prepared statement to avoid CWE-89
    $preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');
    $preparedStatement->execute(array(':name' => $name));
    return $preparedStatement->fetchAll();
}
//...
$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending the query is authorized to do so. An attacker may be able to obtain sensitive employee information from the database.

Example 2:

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that `LookupMessageObject()` ensures that the `$id` argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

Example Language: Perl

(Bad)

```
sub DisplayPrivateMessage {
    my($id) = @_ ;
    my $Message = LookupMessageObject($id);
    print "From: " . encodeHTML($Message->{from}) . "<br>\n";
    print "Subject: " . encodeHTML($Message->{subject}) . "\n";
    print "<hr>\n";
    print "Body: " . encodeHTML($Message->{body}) . "\n";
}
my $q = new CGI;
# For purposes of this example, assume that CWE-309 and
# CWE-523 do not apply.
if (!AuthenticateUser($q->param('username'), $q->param('password'))) {
    ExitError("invalid username or password");
}
my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

Reference	Description
CVE-2022-24730	Go-based continuous deployment product does not check that a user has certain privileges to update or create an app, allowing adversaries to read sensitive repository information https://www.cve.org/CVERecord?id=CVE-2022-24730
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords. https://www.cve.org/CVERecord?id=CVE-2009-3168
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request. https://www.cve.org/CVERecord?id=CVE-2009-3597
CVE-2009-2282	Terminal server does not check authorization for guest access. https://www.cve.org/CVERecord?id=CVE-2009-2282
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms. https://www.cve.org/CVERecord?id=CVE-2008-5027
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files. https://www.cve.org/CVERecord?id=CVE-2009-3781

Reference	Description
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files. https://www.cve.org/CVERecord?id=CVE-2008-6548
CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users. https://www.cve.org/CVERecord?id=CVE-2009-2960
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations. https://www.cve.org/CVERecord?id=CVE-2009-3230
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings. https://www.cve.org/CVERecord?id=CVE-2009-2213
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges. https://www.cve.org/CVERecord?id=CVE-2009-0034
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect. https://www.cve.org/CVERecord?id=CVE-2008-6123
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client. https://www.cve.org/CVERecord?id=CVE-2008-7109
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access. https://www.cve.org/CVERecord?id=CVE-2008-3424
CVE-2005-1036	Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap https://www.cve.org/CVERecord?id=CVE-2005-1036
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions. https://www.cve.org/CVERecord?id=CVE-2008-4577
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries. https://www.cve.org/CVERecord?id=CVE-2007-2925
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header. https://www.cve.org/CVERecord?id=CVE-2006-6679
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files. https://www.cve.org/CVERecord?id=CVE-2005-3623
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied. https://www.cve.org/CVERecord?id=CVE-2005-2801
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
CVE-2020-17533	Chain: unchecked return value (CWE-252) of some functions for policy enforcement leads to authorization bypass (CWE-862) https://www.cve.org/CVERecord?id=CVE-2020-17533

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	2380
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2393
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	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

Terminology

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 2-1		Req 4.3.3.7
ISA/IEC 62443	Part 3-3		Req SR 2.1
ISA/IEC 62443	Part 4-2		Req CR 2.1

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
665	Exploitation of Thunderbolt Protection Flaws

References

[REF-229]NIST. "Role Based Access Control and Role Based Security". < <https://csrc.nist.gov/projects/role-based-access-control> >.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-231]Frank Kim. "Top 25 Series - Rank 5 - Improper Access Control (Authorization)". 2010 March 4. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-5-improper-access-control-authorization/> >.2023-04-07.

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

[REF-233]Rahul Bhattacharjee. "Authentication using JAAS". < <https://javaranch.com/journal/2008/04/authentication-using-JAAS.html> >.2023-04-07.

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

CWE-863: Incorrect Authorization

Weakness ID : 863

Structure : Simple

Abstraction : Class








Description

The product performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		285	Improper Authorization	691
ParentOf		551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1273
ParentOf		639	Authorization Bypass Through User-Controlled Key	1415
ParentOf		647	Use of Non-Canonical URL Paths for Authorization Decisions	1435
ParentOf		804	Guessable CAPTCHA	1710
ParentOf		942	Permissive Cross-domain Policy with Untrusted Domains	1857
ParentOf		1244	Internal Asset Exposed to Unsafe Debug Access Level or State	2048

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

Nature	Type	ID	Name	Page
ParentOf		639	Authorization Bypass Through User-Controlled Key	1415

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	687

Applicable Platforms

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

Technology : Web Server (*Prevalence = Often*)

Technology : Database Server (*Prevalence = Often*)

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner,

group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Alternate Terms

AuthZ : "AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is distinct from "AuthN" (or, sometimes, "AuthC") which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories <i>An attacker could bypass intended access restrictions to read sensitive data, either by reading the data directly from a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.</i>	
Integrity	Modify Application Data Modify Files or Directories <i>An attacker could bypass intended access restrictions to modify sensitive data, either by writing the data directly to a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.</i>	
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism <i>An attacker could bypass intended access restrictions to gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.</i>	
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands <i>An attacker could use elevated privileges to execute unauthorized commands or code.</i>	
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) <i>An attacker could gain unauthorized access to resources on the system and excessively consume those resources, leading to a denial of service.</i>	

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. Even if they can be customized to

recognize these schemes, they might not be able to tell whether the scheme correctly performs the authorization in a way that cannot be bypassed or subverted by an attacker.

Effectiveness = Limited

Automated Dynamic Analysis

Automated dynamic analysis may not be able to find interfaces that are protected by authorization checks, even if those checks contain weaknesses.

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 custom authorization mechanisms.

Effectiveness = Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

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: Host Application Interface Scanner Fuzz Tester Framework-based Fuzzer 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: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

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

Divide the product into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) [REF-229] to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor [REF-7].

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, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45].

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phase: System Configuration

Phase: Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Demonstrative Examples

Example 1:

The following code could be for a medical records application. It displays a record to already authenticated users, confirming the user's authorization using a value stored in a cookie.

Example Language: PHP

(Bad)

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

```
die("You are not Authorized to view this record\n");
}
```










The programmer expects that the cookie will only be set when `getRole()` succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie. However, the attacker can easily set the "role" cookie to the value "Reader". As a result, the `$role` variable is "Reader", and `getRole()` is never invoked. The attacker has bypassed the authorization system.

Observed Examples

Reference	Description
CVE-2021-39155	Chain: A microservice integration and management platform compares the hostname in the HTTP Host header in a case-sensitive way (CWE-178, CWE-1289), allowing bypass of the authorization policy (CWE-863) using a hostname with mixed case or other variations. https://www.cve.org/CVERecord?id=CVE-2021-39155
CVE-2019-15900	Chain: <code>sscanf()</code> call is used to check if a username and group exists, but the return value of <code>sscanf()</code> call is not checked (CWE-252), causing an uninitialized variable to be checked (CWE-457), returning success to allow authorization bypass for executing a privileged (CWE-863). https://www.cve.org/CVERecord?id=CVE-2019-15900
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings. https://www.cve.org/CVERecord?id=CVE-2009-2213
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges. https://www.cve.org/CVERecord?id=CVE-2009-0034
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect. https://www.cve.org/CVERecord?id=CVE-2008-6123
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client. https://www.cve.org/CVERecord?id=CVE-2008-7109
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access. https://www.cve.org/CVERecord?id=CVE-2008-3424
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions. https://www.cve.org/CVERecord?id=CVE-2008-4577
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header. https://www.cve.org/CVERecord?id=CVE-2006-6679
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied. https://www.cve.org/CVERecord?id=CVE-2005-2801
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		813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2378
MemberOf		817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	2380
MemberOf		866	2011 Top 25 - Porous Defenses	900	2393
MemberOf		884	CWE Cross-section	884	2588
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		1396	Comprehensive Categorization: Access Control	1400	2540
MemberOf		1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf		1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

Notes

Terminology

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-2		Req CR 2.1
ISA/IEC 62443	Part 4-2		Req CR 2.2
ISA/IEC 62443	Part 3-3		Req SR 2.1
ISA/IEC 62443	Part 3-3		Req SR 2.2
ISA/IEC 62443	Part 4-1		Req SVV-1
ISA/IEC 62443	Part 4-1		Req SVV-4
ISA/IEC 62443	Part 4-1		Req SD-1

References

[REF-229]NIST. "Role Based Access Control and Role Based Security". < <https://csrc.nist.gov/projects/role-based-access-control> >.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-231]Frank Kim. "Top 25 Series - Rank 5 - Improper Access Control (Authorization)". 2010 March 4. SANS Software Security Institute. < <https://www.sans.org/blog/top-25-series-rank-5-improper-access-control-authorization/> >.2023-04-07.

[REF-233]Rahul Bhattacharjee. "Authentication using JAAS". < <https://javaranch.com/journal/2008/04/authentication-using-JAAS.html> >.2023-04-07.

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

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

CWE-908: Use of Uninitialized Resource

Weakness ID : 908
Structure : Simple
Abstraction : Base

Description

The product uses or accesses a resource that has not been initialized.


Extended Description

When a resource has not been properly initialized, the product may behave unexpectedly. This may lead to a crash or invalid memory access, but the consequences vary depending on the type of resource and how it is used within 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		665	Improper Initialization	1465
ParentOf		457	Use of Uninitialized Variable	1102
CanFollow		909	Missing Initialization of Resource	1806

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1465

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data <i>When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.</i>	
Availability	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	<i>The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.</i>	

Potential Mitigations

Phase: Implementation

Explicitly initialize the resource before use. If this is performed through an API function or standard procedure, follow all required steps.

Phase: Implementation

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

Phase: Implementation

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

Phase: Build and Compilation

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

Demonstrative Examples

Example 1:

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

Example Language: Java

(Bad)

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

Example 2:

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

Example Language: Perl

(Bad)

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the \$uid variable will not be explicitly set by the programmer. This will cause \$uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

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

Example Language: C

(Bad)

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

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

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

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

Example 4:

This example will leave `test_string` in an unknown condition when `i` is the same value as `err_val`, because `test_string` is not initialized (CWE-456). Depending on where this code segment appears (e.g. within a function body), `test_string` might be random if it is stored on the heap or stack. If the variable is declared in static memory, it might be zero or NULL. Compiler optimization might contribute to the unpredictability of this address.

Example Language: C

(Bad)

```
char *test_string;
if (i != err_val)
{
    test_string = "Hello World!";
}
printf("%s", test_string);
```

When the `printf()` is reached, `test_string` might be an unexpected address, so the `printf` might print junk strings (CWE-457).

To fix this code, there are a couple approaches to making sure that `test_string` has been properly set once it reaches the `printf()`.

One solution would be to set `test_string` to an acceptable default before the conditional:

Example Language: C

(Good)

```
char *test_string = "Done at the beginning";
if (i != err_val)
{
    test_string = "Hello World!";
}
printf("%s", test_string);
```

Another solution is to ensure that each branch of the conditional - including the default/else branch - could ensure that `test_string` is set:

Example Language: C

(Good)

```

char *test_string;
if (i != err_val)
{
    test_string = "Hello World!";
}
else {
    test_string = "Done on the other side!";
}
printf("%s", test_string);

```

Observed Examples

Reference	Description
CVE-2019-9805	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-2008-4197	Use of uninitialized memory may allow code execution. https://www.cve.org/CVERecord?id=CVE-2008-4197
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution. https://www.cve.org/CVERecord?id=CVE-2008-2934
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak. https://www.cve.org/CVERecord?id=CVE-2008-0063
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free. https://www.cve.org/CVERecord?id=CVE-2008-0062
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. https://www.cve.org/CVERecord?id=CVE-2008-0081
CVE-2008-3688	Chain: Uninitialized variable leads to infinite loop. https://www.cve.org/CVERecord?id=CVE-2008-3688
CVE-2008-3475	Chain: Improper initialization leads to memory corruption. https://www.cve.org/CVERecord?id=CVE-2008-3475
CVE-2005-1036	Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap https://www.cve.org/CVERecord?id=CVE-2005-1036
CVE-2008-3597	Chain: game server can access player data structures before initialization has happened leading to NULL dereference https://www.cve.org/CVERecord?id=CVE-2008-3597
CVE-2009-2692	Chain: uninitialized function pointers can be dereferenced allowing code execution https://www.cve.org/CVERecord?id=CVE-2009-2692
CVE-2009-0949	Chain: improper initialization of memory can lead to NULL dereference https://www.cve.org/CVERecord?id=CVE-2009-0949
CVE-2009-3620	Chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference https://www.cve.org/CVERecord?id=CVE-2009-3620

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476

Nature	Type	ID	Name	V	Page
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	EXP33-C	CWE More Abstract	Do not read uninitialized memory

References

[REF-436]mercy. "Exploiting Uninitialized Data". 2006 January. < <http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip> >.

CWE-909: Missing Initialization of Resource

Weakness ID : 909

Structure : Simple

Abstraction : Class

Description

The product does not initialize a critical resource.

Extended Description

Many resources require initialization before they can be properly used. If a resource is not initialized, it could contain unpredictable or expired data, or it could be initialized to defaults that are invalid. This can have security implications when the resource is expected to have certain properties or values.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	665	Improper Initialization	1465
ParentOf	V	456	Missing Initialization of a Variable	1096
ParentOf	B	1271	Uninitialized Value on Reset for Registers Holding Security Settings	2115
CanPrecede	B	908	Use of Uninitialized Resource	1802

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

Nature	Type	ID	Name	Page
ChildOf	G	665	Improper Initialization	1465

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2345

Weakness Ordinalities

Primary :

1806

Resultant :

Applicable Platforms

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

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data <i>When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.</i>	
Availability	DoS: Crash, Exit, or Restart <i>The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.</i>	

Potential Mitigations

Phase: Implementation

Explicitly initialize the resource before use. If this is performed through an API function or standard procedure, follow all specified steps.

Phase: Implementation

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

Phase: Implementation

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

Phase: Build and Compilation

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

Demonstrative Examples

Example 1:

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

Example Language: Java

(Bad)

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

Example 2:

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

Example Language: Perl

(Bad)

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the \$uid variable will not be explicitly set by the programmer. This will cause \$uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

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

Example Language: C

(Bad)

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

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

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

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

Example 4:

This example will leave test_string in an unknown condition when i is the same value as err_val, because test_string is not initialized (CWE-456). Depending on where this code segment appears (e.g. within a function body), test_string might be random if it is stored on the heap or stack. If the variable is declared in static memory, it might be zero or NULL. Compiler optimization might contribute to the unpredictability of this address.

Example Language: C

(Bad)

```
char *test_string;
if (i != err_val)
{
    test_string = "Hello World!";
}
printf("%s", test_string);
```

When the printf() is reached, test_string might be an unexpected address, so the printf might print junk strings (CWE-457).

To fix this code, there are a couple approaches to making sure that test_string has been properly set once it reaches the printf().

One solution would be to set test_string to an acceptable default before the conditional:

Example Language: C

(Good)

```
char *test_string = "Done at the beginning";
if (i != err_val)
{
    test_string = "Hello World!";
}
printf("%s", test_string);
```

Another solution is to ensure that each branch of the conditional - including the default/else branch - could ensure that test_string is set:

Example Language: C

(Good)


```
char *test_string;
if (i != err_val)
{
    test_string = "Hello World!";
}
else {
    test_string = "Done on the other side!";
}
printf("%s", test_string);
```

Observed Examples

Reference	Description
CVE-2020-20739	A variable that has its value set in a conditional statement is sometimes used when the conditional fails, sometimes causing data leakage https://www.cve.org/CVERecord?id=CVE-2020-20739
CVE-2005-1036	Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap https://www.cve.org/CVERecord?id=CVE-2005-1036

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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

CWE-910: Use of Expired File Descriptor

Weakness ID : 910

Structure : Simple

Abstraction : Base

Description

The product uses or accesses a file descriptor after it has been closed.

Extended Description

After a file descriptor for a particular file or device has been released, it can be reused. The code might not write to the original file, since the reused file descriptor might reference a different file or device.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Language : Not Language-Specific (Prevalence = Undetermined)

Alternate Terms

Stale file descriptor :

Likelihood Of Exploit


Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories <i>The program could read data from the wrong file.</i>	
Availability	DoS: Crash, Exit, or Restart <i>Accessing a file descriptor that has been closed can cause a crash.</i>	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

Taxonomy Mappings

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

CWE-911: Improper Update of Reference Count

Weakness ID : 911

Structure : Simple

Abstraction : Base

Description

The product uses a reference count to manage a resource, but it does not update or incorrectly updates the reference count.




Extended Description

Reference counts can be used when tracking how many objects contain a reference to a particular resource, such as in memory management or garbage collection. When the reference count reaches zero, the resource can be de-allocated or reused because there are no more objects that use it. If the reference count accidentally reaches zero, then the resource might be released too soon, even though it is still in use. If all objects no longer use the resource, but the reference count is not zero, then the resource might not ever be released.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		664	Improper Control of a Resource Through its Lifetime	1463
CanPrecede		672	Operation on a Resource after Expiration or Release	1488
CanPrecede		772	Missing Release of Resource after Effective Lifetime	1632

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (Prevalence = Sometimes)

Language : C++ (Prevalence = Sometimes)

Language : Not Language-Specific (Prevalence = Undetermined)

Likelihood Of Exploit

Medium

Observed Examples

Reference	Description
CVE-2002-0574	chain: reference count is not decremented, leading to memory leak in OS by sending ICMP packets. https://www.cve.org/CVERecord?id=CVE-2002-0574
CVE-2004-0114	Reference count for shared memory not decremented when a function fails, potentially allowing unprivileged users to read kernel memory. https://www.cve.org/CVERecord?id=CVE-2004-0114
CVE-2006-3741	chain: improper reference count tracking leads to file descriptor consumption https://www.cve.org/CVERecord?id=CVE-2006-3741

Reference	Description
CVE-2007-1383	chain: integer overflow in reference counter causes the same variable to be destroyed twice. https://www.cve.org/CVERecord?id=CVE-2007-1383
CVE-2007-1700	Incorrect reference count calculation leads to improper object destruction and code execution. https://www.cve.org/CVERecord?id=CVE-2007-1700
CVE-2008-2136	chain: incorrect update of reference count leads to memory leak. https://www.cve.org/CVERecord?id=CVE-2008-2136
CVE-2008-2785	chain/composite: use of incorrect data type for a reference counter allows an overflow of the counter, leading to a free of memory that is still in use. https://www.cve.org/CVERecord?id=CVE-2008-2785
CVE-2008-5410	Improper reference counting leads to failure of cryptographic operations. https://www.cve.org/CVERecord?id=CVE-2008-5410
CVE-2009-1709	chain: improper reference counting in a garbage collection routine leads to use-after-free https://www.cve.org/CVERecord?id=CVE-2009-1709
CVE-2009-3553	chain: reference count not correctly maintained when client disconnects during a large operation, leading to a use-after-free. https://www.cve.org/CVERecord?id=CVE-2009-3553
CVE-2009-3624	Reference count not always incremented, leading to crash or code execution. https://www.cve.org/CVERecord?id=CVE-2009-3624
CVE-2010-0176	improper reference counting leads to expired pointer dereference. https://www.cve.org/CVERecord?id=CVE-2010-0176
CVE-2010-0623	OS kernel increments reference count twice but only decrements once, leading to resource consumption and crash. https://www.cve.org/CVERecord?id=CVE-2010-0623
CVE-2010-2549	OS kernel driver allows code execution https://www.cve.org/CVERecord?id=CVE-2010-2549
CVE-2010-4593	improper reference counting leads to exhaustion of IP addresses https://www.cve.org/CVERecord?id=CVE-2010-4593
CVE-2011-0695	Race condition causes reference counter to be decremented prematurely, leading to the destruction of still-active object and an invalid pointer dereference. https://www.cve.org/CVERecord?id=CVE-2011-0695
CVE-2012-4787	improper reference counting leads to use-after-free https://www.cve.org/CVERecord?id=CVE-2012-4787

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		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

References

[REF-884]Mateusz "j00ru" Jurczyk. "Windows Kernel Reference Count Vulnerabilities - Case Study". 2012 November. < <https://j00ru.vexillium.org/slides/2012/zeronights.pdf> >.2023-04-07.

CWE-912: Hidden Functionality

Weakness ID : 912

Structure : Simple
Abstraction : Class

Description

The product contains functionality that is not documented, not part of the specification, and not accessible through an interface or command sequence that is obvious to the product's users or administrators.



Extended Description

Hidden functionality can take many forms, such as intentionally malicious code, "Easter Eggs" that contain extraneous functionality such as games, developer-friendly shortcuts that reduce maintenance or support costs such as hard-coded accounts, etc. From a security perspective, even when the functionality is not intentionally malicious or damaging, it can increase the product's attack surface and expose additional weaknesses beyond what is already exposed by the intended functionality. Even if it is not easily accessible, the hidden functionality could be useful for attacks that modify the control flow of the application.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		684	Incorrect Provision of Specified Functionality	1514
ParentOf		506	Embedded Malicious Code	1218

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	
Integrity	Alter Execution Logic	

Potential Mitigations

Phase: Installation

Always verify the integrity of the product that is being installed.

Phase: Testing

Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

Observed Examples

Reference	Description
CVE-2022-31260	Chain: a digital asset management program has an undisclosed backdoor in the legacy version of a PHP script (CWE-912) that could allow an unauthenticated user to export metadata (CWE-306) https://www.cve.org/CVERecord?id=CVE-2022-31260
CVE-2022-3203	A wireless access point manual specifies that the only method of configuration is via web interface (CWE-1059), but there is an undisclosed telnet server that was activated by default (CWE-912). https://www.cve.org/CVERecord?id=CVE-2022-3203

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2529
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
133	Try All Common Switches
190	Reverse Engineer an Executable to Expose Assumed Hidden Functionality

CWE-913: Improper Control of Dynamically-Managed Code Resources

Weakness ID : 913

Structure : Simple

Abstraction : Class

Description

The product does not properly restrict reading from or writing to dynamically-managed code resources such as variables, objects, classes, attributes, functions, or executable instructions or statements.

Extended Description

Many languages offer powerful features that allow the programmer to dynamically create or modify existing code, or resources used by code such as variables and objects. While these features can offer significant flexibility and reduce development time, they can be extremely dangerous if attackers can directly influence these code resources 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
ChildOf	I P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')	225
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1125
ParentOf	B	502	Deserialization of Untrusted Data	1212
ParentOf	B	914	Improper Control of Dynamically-Identified Variables	1816
ParentOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1818

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

Nature	Type	ID	Name	Page
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1125

Nature	Type	ID	Name	Page
ParentOf		502	Deserialization of Untrusted Data	1212
ParentOf		1321	Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')	2216

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	
Other Integrity	Varies by Context Alter Execution Logic	

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

Strategy = Input Validation

For any externally-influenced input, check the input against an allowlist of acceptable values.

Phase: Implementation

Phase: Architecture and Design

Strategy = Refactoring

Refactor the code so that it does not need to be dynamically managed.



Observed Examples

Reference	Description
CVE-2022-2054	Python compiler uses eval() to execute malicious strings as Python code. https://www.cve.org/CVERecord?id=CVE-2022-2054
CVE-2018-1000613	Cryptography API uses unsafe reflection when deserializing a private key https://www.cve.org/CVERecord?id=CVE-2018-1000613
CVE-2015-8103	Deserialization issue in commonly-used Java library allows remote execution. https://www.cve.org/CVERecord?id=CVE-2015-8103
CVE-2006-7079	Chain: extract used for register_globals compatibility layer, enables path traversal (CWE-22) https://www.cve.org/CVERecord?id=CVE-2006-7079
CVE-2012-2055	Source version control product allows modification of trusted key using mass assignment. https://www.cve.org/CVERecord?id=CVE-2012-2055

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597

Nature	Type	ID	Name	V	Page
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

CWE-914: Improper Control of Dynamically-Identified Variables

Weakness ID : 914

Structure : Simple

Abstraction : Base

Description

The product does not properly restrict reading from or writing to dynamically-identified variables.





Extended Description

Many languages offer powerful features that allow the programmer to access arbitrary variables that are specified by an input string. While these features can offer significant flexibility and reduce development time, they can be extremely dangerous if attackers can modify unintended variables that have security implications.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		913	Improper Control of Dynamically-Managed Code Resources	1814
ChildOf		99	Improper Control of Resource Identifiers ('Resource Injection')	249
ParentOf		621	Variable Extraction Error	1394
ParentOf		627	Dynamic Variable Evaluation	1405

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>An attacker could modify sensitive data or program variables.</i>	
Integrity	Execute Unauthorized Code or Commands	
Other Integrity	Varies by Context Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

For any externally-influenced input, check the input against an allowlist of internal program variables that are allowed to be modified.

Phase: Implementation**Phase: Architecture and Design***Strategy = Refactoring*

Refactor the code so that internal program variables do not need to be dynamically identified.

Demonstrative Examples**Example 1:**

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

Example Language: PHP

(Bad)

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

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

Observed Examples

Reference	Description
CVE-2006-7135	extract issue enables file inclusion https://www.cve.org/CVERecord?id=CVE-2006-7135
CVE-2006-7079	Chain: extract used for register_globals compatibility layer, enables path traversal (CWE-22) https://www.cve.org/CVERecord?id=CVE-2006-7079
CVE-2007-0649	extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect. https://www.cve.org/CVERecord?id=CVE-2007-0649
CVE-2006-6661	extract() enables static code injection https://www.cve.org/CVERecord?id=CVE-2006-6661
CVE-2006-2828	import_request_variables() buried in include files makes post-disclosure analysis confusing https://www.cve.org/CVERecord?id=CVE-2006-2828
CVE-2009-0422	Chain: Dynamic variable evaluation allows resultant remote file inclusion and path traversal. https://www.cve.org/CVERecord?id=CVE-2009-0422
CVE-2007-2431	Chain: dynamic variable evaluation in PHP program used to modify critical, unexpected \$_SERVER variable for resultant XSS. https://www.cve.org/CVERecord?id=CVE-2007-2431
CVE-2006-4904	Chain: dynamic variable evaluation in PHP program used to conduct remote file inclusion. https://www.cve.org/CVERecord?id=CVE-2006-4904

Reference	Description
CVE-2006-4019	Dynamic variable evaluation in mail program allows reading and modifying attachments and preferences of other users. https://www.cve.org/CVERecord?id=CVE-2006-4019

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1409	Comprehensive Categorization: Injection	1400	2556

CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

Weakness ID : 915

Structure : Simple

Abstraction : Base

Description

The product receives input from an upstream component that specifies multiple attributes, properties, or fields that are to be initialized or updated in an object, but it does not properly control which attributes can be modified.

Extended Description





If the object contains attributes that were only intended for internal use, then their unexpected modification could lead to a vulnerability.

This weakness is sometimes known by the language-specific mechanisms that make it possible, such as mass assignment, autobinding, or object injection.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		913	Improper Control of Dynamically-Managed Code Resources	1814
ParentOf		1321	Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')	2216
PeerOf		502	Deserialization of Untrusted Data	1212
PeerOf		502	Deserialization of Untrusted Data	1212

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Ruby (*Prevalence = Undetermined*)

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

Language : PHP (*Prevalence = Undetermined*)

Language : Python (*Prevalence = Undetermined*)

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

Alternate Terms

Mass Assignment : "Mass assignment" is the name of a feature in Ruby on Rails that allows simultaneous modification of multiple object attributes.

AutoBinding : The "Autobinding" term is used in frameworks such as Spring MVC and ASP.NET MVC.

PHP Object Injection : Some PHP application researchers use this term for attacking unsafe use of the unserialize() function, but it is also used for CWE-502.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>An attacker could modify sensitive data or program variables.</i>	
Integrity	Execute Unauthorized Code or Commands	
Other	Varies by Context	
Integrity	Alter Execution Logic	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

If available, use features of the language or framework that allow specification of allowlists of attributes or fields that are allowed to be modified. If possible, prefer allowlists over denylists. For applications written with Ruby on Rails, use the attr_accessible (allowlist) or attr_protected (denylist) macros in each class that may be used in mass assignment.

Phase: Architecture and Design

Phase: Implementation

If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

Phase: Implementation

Strategy = Input Validation

For any externally-influenced input, check the input against an allowlist of internal object attributes or fields that are allowed to be modified.

Phase: Implementation

Phase: Architecture and Design

Strategy = Refactoring

Refactor the code so that object attributes or fields do not need to be dynamically identified, and only expose getter/setter functionality for the intended attributes.

Demonstrative Examples

Example 1:

This function sets object attributes based on a dot-separated path.

Example Language: JavaScript (Bad)

```
function setValueByPath (object, path, value) {
  const pathArray = path.split(".");
  const attributeToSet = pathArray.pop();
  let objectToModify = object;
  for (const attr of pathArray) {
    if (typeof objectToModify[attr] !== 'object') {
      objectToModify[attr] = {};
    }
    objectToModify = objectToModify[attr];
  }
  objectToModify[attributeToSet] = value;
  return object;
}
```

This function does not check if the attribute resolves to the object prototype. These codes can be used to add "isAdmin: true" to the object prototype.

Example Language: JavaScript (Bad)

```
setValueByPath({}, "__proto__.isAdmin", true)
setValueByPath({}, "constructor.prototype.isAdmin", true)
```

By using a denylist of dangerous attributes, this weakness can be eliminated.

Example Language: JavaScript (Good)

```
function setValueByPath (object, path, value) {
  const pathArray = path.split(".");
  const attributeToSet = pathArray.pop();
  let objectToModify = object;
  for (const attr of pathArray) {
    // Ignore attributes which resolve to object prototype
    if (attr === "__proto__" || attr === "constructor" || attr === "prototype") {
      continue;
    }
    if (typeof objectToModify[attr] !== "object") {
      objectToModify[attr] = {};
    }
    objectToModify = objectToModify[attr];
  }
  objectToModify[attributeToSet] = value;
  return object;
}
```

Observed Examples

Reference	Description
CVE-2024-3283	Application for using LLMs allows modification of a sensitive variable using mass assignment.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2024-3283
CVE-2012-2054	Mass assignment allows modification of arbitrary attributes using modified URL. https://www.cve.org/CVERecord?id=CVE-2012-2054
CVE-2012-2055	Source version control product allows modification of trusted key using mass assignment. https://www.cve.org/CVERecord?id=CVE-2012-2055
CVE-2008-7310	Attackers can bypass payment step in e-commerce product. https://www.cve.org/CVERecord?id=CVE-2008-7310
CVE-2013-1465	Use of PHP unserialize function on untrusted input allows attacker to modify application configuration. https://www.cve.org/CVERecord?id=CVE-2013-1465
CVE-2012-3527	Use of PHP unserialize function on untrusted input in content management system might allow code execution. https://www.cve.org/CVERecord?id=CVE-2012-3527
CVE-2012-0911	Use of PHP unserialize function on untrusted input in content management system allows code execution using a crafted cookie value. https://www.cve.org/CVERecord?id=CVE-2012-0911
CVE-2012-0911	Content management system written in PHP allows unserialize of arbitrary objects, possibly allowing code execution. https://www.cve.org/CVERecord?id=CVE-2012-0911
CVE-2011-4962	Content management system written in PHP allows code execution through page comments. https://www.cve.org/CVERecord?id=CVE-2011-4962
CVE-2009-4137	Use of PHP unserialize function on cookie value allows remote code execution or upload of arbitrary files. https://www.cve.org/CVERecord?id=CVE-2009-4137
CVE-2007-5741	Content management system written in Python interprets untrusted data as pickles, allowing code execution. https://www.cve.org/CVERecord?id=CVE-2007-5741
CVE-2011-2520	Python script allows local users to execute code via pickled data. https://www.cve.org/CVERecord?id=CVE-2011-2520
CVE-2005-2875	Python script allows remote attackers to execute arbitrary code using pickled objects. https://www.cve.org/CVERecord?id=CVE-2005-2875
CVE-2013-0277	Ruby on Rails allows deserialization of untrusted YAML to execute arbitrary code. https://www.cve.org/CVERecord?id=CVE-2013-0277
CVE-2011-2894	Spring framework allows deserialization of objects from untrusted sources to execute arbitrary code. https://www.cve.org/CVERecord?id=CVE-2011-2894
CVE-2012-1833	Grails allows binding of arbitrary parameters to modify arbitrary object properties. https://www.cve.org/CVERecord?id=CVE-2012-1833
CVE-2010-3258	Incorrect deserialization in web browser allows escaping the sandbox. https://www.cve.org/CVERecord?id=CVE-2010-3258
CVE-2008-1013	Media library allows deserialization of objects by untrusted Java applets, leading to arbitrary code execution. https://www.cve.org/CVERecord?id=CVE-2008-1013

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2565

Notes

Maintenance

The relationships between CWE-502 and CWE-915 need further exploration. CWE-915 is more narrowly scoped to object modification, and is not necessarily used for deserialization.

References

[REF-885]Stefan Esser. "Shocking News in PHP Exploitation". 2009. < <https://owasp.org/www-pdf-archive/POC2009-ShockingNewsInPHPExploitation.pdf> >.2023-04-07.

[REF-886]Dinis Cruz. "'Two Security Vulnerabilities in the Spring Framework's MVC" pdf (from 2008)". < <http://diniscruz.blogspot.com/2011/07/two-security-vulnerabilities-in-spring.html> >.2023-04-07.

[REF-887]Ryan Berg and Dinis Cruz. "Two Security Vulnerabilities in the Spring Framework's MVC". < https://o2platform.files.wordpress.com/2011/07/ounce_springframework_vulnerabilities.pdf >.2023-04-07.

[REF-888]ASPNETUE. "Best Practices for ASP.NET MVC". 2010 September 7. < https://web.archive.org/web/20100921074010/http://blogs.msdn.com/b/aspnetue/archive/2010/09/17/second_2d00_post.aspx >.2023-04-07.

[REF-889]Michael Hartl. "Mass assignment in Rails applications". 2008 September 1. < <https://web.archive.org/web/20090808163156/http://blog.mhartl.com/2008/09/21/mass-assignment-in-rails-applications/> >.2023-04-07.

[REF-890]Tobi. "Secure your Rails apps!". 2012 March 6. < <https://pragatob.wordpress.com/2012/03/06/secure-your-rails-apps/> >.2023-04-07.

[REF-891]Heiko Webers. "Ruby On Rails Security Guide". < <https://guides.rubyonrails.org/security.html#mass-assignment> >.2023-04-07.

[REF-892]Josh Bush. "Mass Assignment Vulnerability in ASP.NET MVC". 2012 March 5. < <https://web.archive.org/web/20120309022539/http://freshbrewedcode.com/joshbush/2012/03/05/mass-assignment-aspnet-mvc> >.2023-04-07.

[REF-893]K. Scott Allen. "6 Ways To Avoid Mass Assignment in ASP.NET MVC". 2012 March 2. < <https://odetocode.com/blogs/scott/archive/2012/03/11/complete-guide-to-mass-assignment-in-asp-net-mvc.aspx> >.2023-04-07.

[REF-894]Egidio Romano. "PHP Object Injection". 2013 January 2. < https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection >.2023-04-07.

[REF-464]Heine Deelstra. "Unserializing user-supplied data, a bad idea". 2010 August 5. < <https://drupal.sun.com/heine/2010/08/25/unserializing-user-supplied-data-bad-idea> >.2023-04-07.

[REF-466]Nadia Alramli. "Why Python Pickle is Insecure". 2009 September 9. < <http://michael-rushanan.blogspot.com/2012/10/why-python-pickle-is-insecure.html> >.2023-04-07.

CWE-916: Use of Password Hash With Insufficient Computational Effort

Weakness ID : 916

Structure : Simple
Abstraction : Base

Description

The product generates a hash for a password, but it uses a scheme that does not provide a sufficient level of computational effort that would make password cracking attacks infeasible or expensive.

Extended Description

Many password storage mechanisms compute a hash and store the hash, instead of storing the original password in plaintext. In this design, authentication involves accepting an incoming password, computing its hash, and comparing it to the stored hash.

Many hash algorithms are designed to execute quickly with minimal overhead, even cryptographic hashes. However, this efficiency is a problem for password storage, because it can reduce an attacker's workload for brute-force password cracking. If an attacker can obtain the hashes through some other method (such as SQL injection on a database that stores hashes), then the attacker can store the hashes offline and use various techniques to crack the passwords by computing hashes efficiently. 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 (such as cloud computing) and GPU, ASIC, or FPGA hardware. In such a scenario, an efficient hash algorithm helps the attacker.

There are several properties of a hash scheme that are relevant to its strength against an offline, massively-parallel attack:


- The amount of CPU time required to compute the hash ("stretching")
- The amount of memory required to compute the hash ("memory-hard" operations)
- Including a random value, along with the password, as input to the hash computation ("salting")
- Given a hash, there is no known way of determining an input (e.g., a password) that produces this hash value, other than by guessing possible inputs ("one-way" hashing)
- Relative to the number of all possible hashes that can be generated by the scheme, there is a low likelihood of producing the same hash for multiple different inputs ("collision resistance")

Note that the security requirements for the product may vary depending on the environment and the value of the passwords. Different schemes might not provide all of these properties, yet may still provide sufficient security for the environment. Conversely, a solution might be very strong in preserving one property, which still being very weak for an attack against another property, or it might not be able to significantly reduce the efficiency of a massively-parallel 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		328	Use of Weak Hash	813
ParentOf		759	Use of a One-Way Hash without a Salt	1593
ParentOf		760	Use of a One-Way Hash with a Predictable Salt	1598



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

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		1010	Authenticate Actors	2445

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

Nature	Type	ID	Name	Page
MemberOf		255	Credentials Management Errors	2336
MemberOf		310	Cryptographic Issues	2339

Weakness Ordinalities

Primary :

Applicable Platforms

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

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 sufficient computational effort will make it easier to conduct brute force attacks using techniques such as rainbow tables, or specialized hardware such as GPUs, which can be much faster than general-purpose CPUs for computing hashes.</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: 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 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
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		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2509
MemberOf		1402	Comprehensive Categorization: Encryption	1400	2548

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
55	Rainbow Table Password Cracking

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-636]Jeff Atwood. "Speed Hashing". 2012 April 6. < <https://blog.codinghorror.com/speed-hashing/> >.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-908]Solar Designer. "Password hashing at scale". 2012 October 1. < <https://www.openwall.com/presentations/YaC2012-Password-Hashing-At-Scale/> >.2023-04-07.
- [REF-909]Solar Designer. "New developments in password hashing: ROM-port-hard functions". 2012 November. < <https://www.openwall.com/presentations/ZeroNights2012-New-In-Password-Hashing/> >.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.

CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

Weakness ID : 917

Structure : Simple

Abstraction : Base

Description

The product constructs all or part of an expression language (EL) statement in a framework such as a Java Server Page (JSP) using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended EL statement before it is executed.

Extended Description

Frameworks such as Java Server Page (JSP) allow a developer to insert executable expressions within otherwise-static content. When the developer is not aware of the executable nature of these expressions and/or does not disable them, then if an attacker can inject expressions, this could lead to code execution or other unexpected behaviors.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	148

Nature	Type	ID	Name	Page
PeerOf		1336	Improper Neutralization of Special Elements Used in a Template Engine	2250

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

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	138


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

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	148

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

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	148

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2332

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java (Prevalence = Undetermined)

Alternate Terms

EL Injection :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Avoid adding user-controlled data into an expression interpreter when possible.

Phase: Implementation

If user-controlled data must be added to an expression interpreter, one or more of the following should be performed: Validate that the user input will not evaluate as an expression Encode the user input in a way that ensures it is not evaluated as an expression

Phase: System Configuration

Phase: Operation

The framework or tooling might allow the developer to disable or deactivate the processing of EL expressions, such as setting the isELIgnored attribute for a JSP page to "true".

Observed Examples

Reference	Description
CVE-2021-44228	Product does not neutralize \${xyz} style expressions, allowing remote code execution. (log4shell vulnerability in log4j) https://www.cve.org/CVERecord?id=CVE-2021-44228

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2456
MemberOf		1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf		1409	Comprehensive Categorization: Injection	1400	2556

Notes

Maintenance

The interrelationships and differences between CWE-917 and CWE-1336 need to be further clarified.

Relationship

In certain versions of Spring 3.0.5 and earlier, there was a vulnerability (CVE-2011-2730) in which Expression Language tags would be evaluated twice, which effectively exposed any application to EL injection. However, even for later versions, this weakness is still possible depending on configuration.

References

[REF-911]Stefano Di Paola and Arshan Dabirsiaghi. "Expression Language Injection". 2011 September 2. < <https://mindedsecurity.com/wp-content/uploads/2020/10/ExpressionLanguageInjection.pdf> >.2023-04-07.

[REF-912]Dan Amodio. "Remote Code with Expression Language Injection". 2012 December 4. < <http://danamodio.com/appsec/research/spring-remote-code-with-expression-language-injection/> >.2023-04-07.

[REF-1279]CWE/CAPEC. "Neutralizing Your Inputs: A Log4Shell Weakness Story". < https://medium.com/@CWE_CAPEC/neutralizing-your-inputs-a-log4shell-weakness-story-89954c8b25c9 >.

[REF-1280]OWASP. "Expression Language Injection". < https://owasp.org/www-community/vulnerabilities/Expression_Language_Injection >.

CWE-918: Server-Side Request Forgery (SSRF)

Weakness ID : 918

Structure : Simple


Abstraction : Base**Description**

The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		441	Unintended Proxy or Intermediary ('Confused Deputy')	1072

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

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

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

Nature	Type	ID	Name	Page
MemberOf		417	Communication Channel Errors	2347

Applicable Platforms

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

Technology : Web Server (*Prevalence = Undetermined*)

Alternate Terms

XSPA : Cross Site Port Attack

SSRF : Server-Side Request Forgery

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	
Access Control	Bypass Protection Mechanism	
	<p><i>By providing URLs to unexpected hosts or ports, attackers can make it appear that the server is sending the request, possibly bypassing access controls such as firewalls that prevent the attackers from accessing the URLs directly. The server can be used as a proxy to conduct port scanning of hosts in internal networks, use other URLs such as that can access documents on the system (using file://), or use other protocols such as gopher:// or tftp://, which may provide greater control over the contents of requests.</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

Observed Examples

Reference	Description
CVE-2023-32786	Chain: LLM integration framework has prompt injection (CWE-1427) that allows an attacker to force the service to retrieve data from an arbitrary URL, essentially providing SSRF (CWE-918) and potentially injecting content into downstream tasks. https://www.cve.org/CVERecord?id=CVE-2023-32786
CVE-2021-26855	Server Side Request Forgery (SSRF) in mail server, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-26855
CVE-2021-21973	Server Side Request Forgery in cloud platform, as exploited in the wild per CISA KEV. https://www.cve.org/CVERecord?id=CVE-2021-21973
CVE-2016-4029	Chain: incorrect validation of intended decimal-based IP address format (CWE-1286) enables parsing of octal or hexadecimal formats (CWE-1389), allowing bypass of an SSRF protection mechanism (CWE-918). https://www.cve.org/CVERecord?id=CVE-2016-4029
CVE-2002-1484	Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning. https://www.cve.org/CVERecord?id=CVE-2002-1484
CVE-2004-2061	CGI script accepts and retrieves incoming URLs. https://www.cve.org/CVERecord?id=CVE-2004-2061
CVE-2010-1637	Web-based mail program allows internal network scanning using a modified POP3 port number. https://www.cve.org/CVERecord?id=CVE-2010-1637
CVE-2009-0037	URL-downloading library automatically follows redirects to file:// and scp:// URLs https://www.cve.org/CVERecord?id=CVE-2009-0037

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf		1356	OWASP Top Ten 2021 Category A10:2021 - Server-Side Request Forgery (SSRF)	1344	2518
MemberOf		1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540
MemberOf		1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2621
MemberOf		1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses	1430	2622

Notes

Relationship

CWE-918 (SSRF) and CWE-611 (XXE) are closely related, because they both involve web-related technologies and can launch outbound requests to unexpected destinations. However, XXE can be performed client-side, or in other contexts in which the software is not acting directly as a server, so the "Server" portion of the SSRF acronym does not necessarily apply.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
664	Server Side Request Forgery

References

[REF-913]Alexander Polyakov and Dmitry Chastukhin. "SSRF vs. Business-critical applications: XXE tunneling in SAP". 2012 July 6. < https://media.blackhat.com/bh-us-12/Briefings/Polyakov/BH_US_12_Polyakov_SSRF_Business_Slides.pdf >.

[REF-914]Alexander Polyakov, Dmitry Chastukhin and Alexey Tyurin. "SSRF vs. Business-critical Applications. Part 1: XXE Tunnelling in SAP NetWeaver". < <http://erpscan.com/wp-content/uploads/2012/08/SSRF-vs-Business-critical-applications-whitepaper.pdf> >.

[REF-915]Riyaz Ahemed Walikar. "Cross Site Port Attacks - XSPA - Part 1". 2012 November 7. < <https://ibreak.software/2012/11/cross-site-port-attacks-xspa-part-1/> >.

[REF-916]Riyaz Ahemed Walikar. "Cross Site Port Attacks - XSPA - Part 2". 2012 November 3. < <https://ibreak.software/2012/11/cross-site-port-attacks-xspa-part-2/> >.

[REF-917]Riyaz Ahemed Walikar. "Cross Site Port Attacks - XSPA - Part 3". 2012 November 4. < <https://ibreak.software/2012/11/cross-site-port-attacks-xspa-part-3/> >.

[REF-918]Vladimir Vorontsov and Alexander Golovko. "SSRF attacks and sockets: smorgasbord of vulnerabilities". < <https://www.slideshare.net/DefconRussia/vorontsov-golovko-ssrf-attacks-and-sockets-smorgasbord-of-vulnerabilities> >.2023-04-07.

[REF-919]ONsec Lab. "SSRF bible. Cheatsheet". 2013 January 6. < <https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/edit?pli=1#> >.

[REF-920]Deral Heiland. "Web Portals: Gateway To Information, Or A Hole In Our Perimeter Defenses". 2008 February. < <http://www.shmoocon.org/2008/presentations/Web%20portals,%20gateway%20to%20information.ppt> >.

CWE-920: Improper Restriction of Power Consumption

Weakness ID : 920

Structure : Simple

Abstraction : Base

Description

The product operates in an environment in which power is a limited resource that cannot be automatically replenished, but the product does not properly restrict the amount of power that its operation consumes.

Extended Description

In environments such as embedded or mobile devices, power can be a limited resource such as a battery, which cannot be automatically replenished by the product itself, and the device might not always be directly attached to a reliable power source. If the product uses too much power too quickly, then this could cause the device (and subsequently, the product) to stop functioning until power is restored, or increase the financial burden on the device owner because of increased power costs.


Normal operation of an application will consume power. However, in some cases, an attacker could cause the application to consume more power than intended, using components such as:

- Display
- CPU
- Disk I/O
- GPS
- Sound
- Microphone
- USB interface


Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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 "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	971

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345

Applicable Platforms

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



Technology : Mobile (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) DoS: Crash, Exit, or Restart <i>The power source could be drained, causing the application - and the entire device - to cease functioning.</i>	

MemberOf Relationships

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

Nature	Type	ID	Name		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 6.2
ISA/IEC 62443	Part 4-2		Req CR 6.2

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

CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Weakness ID : 921

Structure : Simple

Abstraction : Base

Description

The product stores sensitive information in a file system or device that does not have built-in access control.

Extended Description

While many modern file systems or devices utilize some form of access control in order to restrict access to data, not all storage mechanisms have this capability. For example, memory cards, floppy disks, CDs, and USB devices are typically made accessible to any user within the system. This can become a problem when sensitive data is stored in these mechanisms in a multi-user environment, because anybody on the system can read or write this data.

On Android devices, external storage is typically globally readable and writable by other applications on the device. External storage may also be easily accessible through the mobile device's USB connection or physically accessible through the device's memory card port.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		922	Insecure Storage of Sensitive Information	1835

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		199	Information Management Errors	2333

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories <i>Attackers can read sensitive information by accessing the unrestricted storage mechanism.</i>	
Integrity	Modify Application Data	

Scope	Impact	Likelihood
	Modify Files or Directories	
	<i>Attackers can modify or delete sensitive information by accessing the unrestricted storage mechanism.</i>	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-921]Android Open Source Project. "Security Tips". 2013 July 6. < <https://developer.android.com/training/articles/security-tips.html#StoringData> >.2023-04-07.

CWE-922: Insecure Storage of Sensitive Information

Weakness ID : 922
Structure : Simple
Abstraction : Class

Description

The product stores sensitive information without properly limiting read or write access by unauthorized actors.

Extended Description

If read access is not properly restricted, then attackers can steal the sensitive information. If write access is not properly restricted, then attackers can modify and possibly delete the data, causing incorrect results and possibly 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	P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	B	312	Cleartext Storage of Sensitive Information	771
ParentOf	B	921	Storage of Sensitive Data in a Mechanism without Access Control	1834

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2449

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
Integrity	Read Files or Directories <i>Attackers can read sensitive information by accessing the unrestricted storage mechanism.</i>	
	Modify Application Data Modify Files or Directories <i>Attackers can overwrite sensitive information by accessing the unrestricted storage mechanism.</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-2009-2272	password and username stored in cleartext in a cookie https://www.cve.org/CVERecord?id=CVE-2009-2272

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Relationship

There is an overlapping relationship between insecure storage of sensitive information (CWE-922) and missing encryption of sensitive information (CWE-311). Encryption is often used to prevent an attacker from reading the sensitive data. However, encryption does not prevent the attacker from erasing or overwriting the data. While data tampering would be visible upon inspection, the integrity and availability of the data is compromised prior to the audit.

Maintenance

This is a high-level entry that includes children from various parts of the CWE research view (CWE-1000). Currently, most of the information is in these child entries. This entry will be made more comprehensive in later CWE versions.

CWE-923: Improper Restriction of Communication Channel to Intended Endpoints

Weakness ID : 923**Structure** : Simple**Abstraction** : Class

Description

The product establishes a communication channel to (or from) an endpoint for privileged or protected operations, but it does not properly ensure that it is communicating with the correct endpoint.

Extended Description

Attackers might be able to spoof the intended endpoint from a different system or process, thus gaining the same level of access as the intended endpoint.

While this issue frequently involves authentication between network-based clients and servers, other types of communication channels and endpoints can have this weakness.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	687
ParentOf	V	291	Reliance on IP Address for Authentication	715
ParentOf	V	297	Improper Validation of Certificate with Host Mismatch	729
ParentOf	G	300	Channel Accessible by Non-Endpoint	737
ParentOf	B	419	Unprotected Primary Channel	1024
ParentOf	B	420	Unprotected Alternate Channel	1025
ParentOf	B	940	Improper Verification of Source of a Communication Channel	1852
ParentOf	B	941	Incorrectly Specified Destination in a Communication Channel	1855
ParentOf	V	942	Permissive Cross-domain Policy with Untrusted Domains	1857
ParentOf	V	1275	Sensitive Cookie with Improper SameSite Attribute	2123
CanFollow	B	322	Key Exchange without Entity Authentication	795
CanFollow	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	870

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Gain Privileges or Assume Identity <i>If an attacker can spoof the endpoint, the attacker gains all the privileges that were intended for the original endpoint.</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:

These cross-domain policy files mean to allow Flash and Silverlight applications hosted on other domains to access its data:

Flash crossdomain.xml :

Example Language: XML

(Bad)

```
<cross-domain-policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.adobe.com/xml/schemas/PolicyFile.xsd">
<allow-access-from domain="*.example.com"/>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Silverlight clientaccesspolicy.xml :

Example Language: XML

(Bad)

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
<cross-domain-access>
<policy>
<allow-from http-request-headers="SOAPAction">
<domain uri="*" />
</allow-from>
<grant-to>
<resource path="/" include-subpaths="true"/>
</grant-to>
</policy>
</cross-domain-access>
</access-policy>
```

These entries are far too permissive, allowing any Flash or Silverlight application to send requests. A malicious application hosted on any other web site will be able to send requests on behalf of any user tricked into executing it.

Example 2:

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

Example Language: Java

(Bad)

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

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

Observed Examples

Reference	Description
CVE-2022-30319	S-bus functionality in a home automation product performs access control using an IP allowlist, which can be bypassed by a forged IP address. https://www.cve.org/CVERecord?id=CVE-2022-30319
CVE-2022-22547	A troubleshooting tool exposes a web server on a random port between 9000-65535 that could be used for information gathering https://www.cve.org/CVERecord?id=CVE-2022-22547
CVE-2022-4390	A WAN interface on a router has firewall restrictions enabled for IPv4, but it does not for IPv6, which is enabled by default https://www.cve.org/CVERecord?id=CVE-2022-4390
CVE-2012-2292	Product has a Silverlight cross-domain policy that does not restrict access to another application, which allows remote attackers to bypass the Same Origin Policy. https://www.cve.org/CVERecord?id=CVE-2012-2292
CVE-2012-5810	Mobile banking application does not verify hostname, leading to financial loss. https://www.cve.org/CVERecord?id=CVE-2012-5810
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing Adversary-in-the-Middle (AITM) attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). https://www.cve.org/CVERecord?id=CVE-2014-1266
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning https://www.cve.org/CVERecord?id=CVE-2000-1218

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
161	Infrastructure Manipulation
481	Contradictory Destinations in Traffic Routing Schemes
501	Android Activity Hijack
697	DHCP Spoofing

CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel

Weakness ID : 924

Structure : Simple

Abstraction : Base

Description

The product establishes a communication channel with an endpoint and receives a message from that endpoint, but it does not sufficiently ensure that the message was not modified during transmission.

Extended Description

Attackers might be able to modify the message and spoof the endpoint by interfering with the data as it crosses the network or by redirecting the connection to a system under their control.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		345	Insufficient Verification of Data Authenticity	858

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

Nature	Type	ID	Name	Page
ChildOf		345	Insufficient Verification of Data Authenticity	858

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

Nature	Type	ID	Name	Page
MemberOf		1020	Verify Message Integrity	2455

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

Nature	Type	ID	Name	Page
MemberOf		1214	Data Integrity Issues	2498
MemberOf		417	Communication Channel Errors	2347

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Gain Privileges or Assume Identity <i>If an attackers can spoof the endpoint, the attacker gains all the privileges that were intended for the original endpoint.</i>	

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		1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

Notes

Maintenance

This entry should be made more comprehensive in later CWE versions, as it is likely an important design flaw that underlies (or chains to) other weaknesses.

CWE-925: Improper Verification of Intent by Broadcast Receiver

Weakness ID : 925

Structure : Simple

Abstraction : Variant

Description

The Android application uses a Broadcast Receiver that receives an Intent but does not properly verify that the Intent came from an authorized source.


Extended Description

Certain types of Intents, identified by action string, can only be broadcast by the operating system itself, not by third-party applications. However, when an application registers to receive these implicit system intents, it is also registered to receive any explicit intents. While a malicious application cannot send an implicit system intent, it can send an explicit intent to the target application, which may assume that any received intent is a valid implicit system intent and not an explicit intent from another application. This may lead to unintended behavior.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		940	Improper Verification of Source of a Communication Channel	1852

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Alternate Terms

Intent Spoofing :

Common Consequences

Scope	Impact	Likelihood
Integrity	Gain Privileges or Assume Identity <i>Another application can impersonate the operating system and cause the software to perform an unintended action.</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

Before acting on the Intent, check the Intent Action to make sure it matches the expected System action.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: XML (Bad)

```
<manifest package="com.example.vulnerableApplication">
  <application>
    ...
    <receiver android:name=".ShutdownReceiver">
      <intent-filter>
        <action android:name="android.intent.action.ACTION_SHUTDOWN" />
      </intent-filter>
    </receiver>
    ...
  </application>
</manifest>
```

The ShutdownReceiver class will handle the intent:

Example Language: Java (Bad)

```
...
IntentFilter filter = new IntentFilter(Intent.ACTION_SHUTDOWN);
BroadcastReceiver sReceiver = new ShutDownReceiver();
registerReceiver(sReceiver, filter);
...
public class ShutdownReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(final Context context, final Intent intent) {
    mainActivity.saveLocalData();
    mainActivity.stopActivity();
  }
}
```

Because the method does not confirm that the intent action is the expected system intent, any received intent will trigger the shutdown procedure, as shown here:

Example Language: Java (Attack)

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

An attacker can use this behavior to cause a denial of service.

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	1396	Comprehensive Categorization: Access Control	1400	2540

Notes

Maintenance

This entry will be made more comprehensive in later CWE versions.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
499	Android Intent Intercept

References

[REF-922]Erika Chin, Adrienne Porter Felt, Kate Greenwood and David Wagner. "Analyzing Inter-Application Communication in Android". < <http://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf> >.

CWE-926: Improper Export of Android Application Components

Weakness ID : 926

Structure : Simple

Abstraction : Variant

Description

The Android application exports a component for use by other applications, but does not properly restrict which applications can launch the component or access the data it contains.

Extended Description

The attacks and consequences of improperly exporting a component may depend on the exported component:

- If access to an exported Activity is not restricted, any application will be able to launch the activity. This may allow a malicious application to gain access to sensitive information, modify the internal state of the application, or trick a user into interacting with the victim application while believing they are still interacting with the malicious application.
- If access to an exported Service is not restricted, any application may start and bind to the Service. Depending on the exposed functionality, this may allow a malicious application to perform unauthorized actions, gain access to sensitive information, or corrupt the internal state of the application.
- If access to a Content Provider is not restricted to only the expected applications, then malicious applications might be able to access the sensitive data. Note that in Android before 4.2, the Content Provider is automatically exported unless it has been explicitly declared as NOT exported.

Relationships

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		285	Improper Authorization	691

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Background Details

There are three types of components that can be exported in an Android application.

- An Activity is an application component that provides a UI for users to interact with. A typical application will have multiple Activity screens that perform different functions, such as a main Activity screen and a separate settings Activity screen.
- A Service is an application component that is started by another component to execute an operation in the background, even after the invoking component is terminated. Services do not have a UI component visible to the user.
- The Content Provider mechanism can be used to share data with other applications or internally within the same application.

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	Unexpected State DoS: Crash, Exit, or Restart DoS: Instability Varies by Context <i>Other applications, possibly untrusted, can launch the Activity.</i>	
Availability Integrity	Unexpected State Gain Privileges or Assume Identity DoS: Crash, Exit, or Restart DoS: Instability Varies by Context <i>Other applications, possibly untrusted, can bind to the Service.</i>	
Confidentiality Integrity	Read Application Data Modify Application Data <i>Other applications, possibly untrusted, can read or modify the data that is offered by the Content Provider.</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: Build and Compilation

Strategy = Attack Surface Reduction

If they do not need to be shared by other applications, explicitly mark components with `android:exported="false"` in the application manifest.

Phase: Build and Compilation

Strategy = Attack Surface Reduction

If you only intend to use exported components between related apps under your control, use `android:protectionLevel="signature"` in the xml manifest to restrict access to applications signed by you.

Phase: Build and Compilation

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Limit Content Provider permissions (read/write) as appropriate.

Phase: Build and Compilation**Phase: Architecture and Design**

Strategy = Separation of Privilege

Limit Content Provider permissions (read/write) as appropriate.

Demonstrative Examples**Example 1:**

This application is exporting an activity and a service in its manifest.xml:

Example Language: XML

(Bad)

```
<activity android:name="com.example.vulnerableApp.mainScreen">
...
  <intent-filter>
    <action android:name="com.example.vulnerableApp.OPEN_UI" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
...
</activity>
<service android:name="com.example.vulnerableApp.backgroundService">
...
  <intent-filter>
    <action android:name="com.example.vulnerableApp.START_BACKGROUND" />
  </intent-filter>
...
</service>
```

Because these components have intent filters but have not explicitly set 'android:exported=false' elsewhere in the manifest, they are automatically exported so that any other application can launch them. This may lead to unintended behavior or exploits.

Example 2:

This application has created a content provider to enable custom search suggestions within the application:

Example Language: XML

(Bad)

```
<provider>
  android:name="com.example.vulnerableApp.searchDB"
  android:authorities="com.example.vulnerableApp.searchDB">
</provider>
```

Because this content provider is only intended to be used within the application, it does not need to be exported. However, in Android before 4.2, it is automatically exported thus potentially allowing malicious applications to access sensitive information.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-923]Android Open Source Project. "Security Tips". 2013 July 6. < <https://developer.android.com/training/articles/security-tips#ContentProviders> >.2023-04-07.

CWE-927: Use of Implicit Intent for Sensitive Communication

Weakness ID : 927

Structure : Simple

Abstraction : Variant

Description

The Android application uses an implicit intent for transmitting sensitive data to other applications.

Extended Description

Since an implicit intent does not specify a particular application to receive the data, any application can process the intent by using an Intent Filter for that intent. This can allow untrusted applications to obtain sensitive data. There are two variations on the standard broadcast intent, ordered and sticky.

Ordered broadcast intents are delivered to a series of registered receivers in order of priority as declared by the Receivers. A malicious receiver can give itself a high priority and cause a denial of service by stopping the broadcast from propagating further down the chain. There is also the possibility of malicious data modification, as a receiver may also alter the data within the Intent before passing it on to the next receiver. The downstream components have no way of asserting that the data has not been altered earlier in the chain.



Sticky broadcast intents remain accessible after the initial broadcast. An old sticky intent will be broadcast again to any new receivers that register for it in the future, greatly increasing the chances of information exposure over time. Also, sticky broadcasts cannot be protected by permissions that may apply to other kinds of intents.

In addition, any broadcast intent may include a URI that references data that the receiving component does not normally have the privileges to access. The sender of the intent can include special privileges that grant the receiver read or write access to the specific URI included in the intent. A malicious receiver that intercepts this intent will also gain those privileges and be able to read or write the resource at the specified URI.

Relationships

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

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
	<i>Other applications, possibly untrusted, can read the data that is offered through the Intent.</i>	
Integrity	Varies by Context <i>The application may handle responses from untrusted applications on the device, which could cause it to perform unexpected or unauthorized actions.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

If the application only requires communication with its own components, then the destination is always known, and an explicit intent could be used.

Demonstrative Examples

Example 1:

This application wants to create a user account in several trusted applications using one broadcast intent:

Example Language: Java

(Bad)

```
Intent intent = new Intent();
intent.setAction("com.example.CreateUser");
intent.putExtra("Username", uname_string);
intent.putExtra("Password", pw_string);
sendBroadcast(intent);
```

This application assumes only the trusted applications will be listening for the action. A malicious application can register for this action and intercept the user's login information, as below:

Example Language: Java

(Attack)

```
IntentFilter filter = new IntentFilter("com.example.CreateUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
```

When a broadcast contains sensitive information, create an allowlist of applications that can receive the action using the application's manifest file, or programmatically send the intent to each individual intended receiver.

Example 2:

This application interfaces with a web service that requires a separate user login. It creates a sticky intent, so that future trusted applications that also use the web service will know who the current user is:

Example Language: Java

(Bad)

```
Intent intent = new Intent();
```

```
intent.setAction("com.example.service.UserExists");
intent.putExtra("Username", uname_string);
sendStickyBroadcast(intent);
```

Example Language: Java

(Attack)

```
IntentFilter filter = new IntentFilter("com.example.service.UserExists");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
```

Sticky broadcasts can be read by any application at any time, and so should never contain sensitive information such as a username.

Example 3:

This application is sending an ordered broadcast, asking other applications to open a URL:

Example Language: Java

(Bad)

```
Intent intent = new Intent();
intent.setAction("com.example.OpenURL");
intent.putExtra("URL_TO_OPEN", url_string);
sendOrderedBroadcastAsUser(intent);
```

Any application in the broadcast chain may alter the data within the intent. This malicious application is altering the URL to point to an attack site:

Example Language: Java

(Attack)

```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String Url = intent.getStringExtra(Intent.URL_TO_OPEN);
        attackURL = "www.example.com/attack?" + Url;
        setResultData(attackURL);
    }
}
```

The final receiving application will then open the attack URL. Where possible, send intents to specific trusted applications instead of using a broadcast chain.

Example 4:

This application sends a special intent with a flag that allows the receiving application to read a data file for backup purposes.

Example Language: Java

(Bad)

```
Intent intent = new Intent();
intent.setAction("com.example.BackupUserData");
intent.setData(file_uri);
intent.addFlags(FLAG_GRANT_READ_URI_PERMISSION);
sendBroadcast(intent);
```

Example Language: Java

(Attack)

```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Uri userData = intent.getData();
        stealUserData(userData);
    }
}
```




Any malicious application can register to receive this intent. Because of the FLAG_GRANT_READ_URI_PERMISSION included with the intent, the malicious receiver code can read the user's data.

Observed Examples

Reference	Description
CVE-2022-4903	An Android application does not use FLAG_IMMUTABLE when creating a PendingIntent. https://www.cve.org/CVERecord?id=CVE-2022-4903

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-922]Erika Chin, Adrienne Porter Felt, Kate Greenwood and David Wagner. "Analyzing Inter-Application Communication in Android". < <http://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf> >.

[REF-923]Android Open Source Project. "Security Tips". 2013 July 6. < <https://developer.android.com/training/articles/security-tips#ContentProviders> >.2023-04-07.

CWE-939: Improper Authorization in Handler for Custom URL Scheme

Weakness ID : 939

Structure : Simple

Abstraction : Base

Description

The product uses a handler for a custom URL scheme, but it does not properly restrict which actors can invoke the handler using the scheme.

Extended Description

Mobile platforms and other architectures allow the use of custom URL schemes to facilitate communication between applications. In the case of iOS, this is the only method to do inter-application communication. The implementation is at the developer's discretion which may open security flaws in the application. An example could be potentially dangerous functionality such as modifying files through a custom URL scheme.

Relationships

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


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

Nature	Type	ID	Name	Page
ChildOf		862	Missing Authorization	1789

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2497

Applicable Platforms

Technology : Mobile (Prevalence = Undetermined)

Potential Mitigations

Phase: Architecture and Design

Utilize a user prompt pop-up to authorize potentially harmful actions such as those modifying data or dealing with sensitive information. When designing functionality of actions in the URL scheme, consider whether the action should be accessible to all mobile applications, or if an allowlist of applications to interface with is appropriate.

Demonstrative Examples

Example 1:

This iOS application uses a custom URL scheme. The replaceFileText action in the URL scheme allows an external application to interface with the file incomingMessage.txt and replace the contents with the text field of the query string.

External Application

Example Language: Objective-C

(Good)

```
NSString *stringURL = @"appscheme://replaceFileText?file=incomingMessage.txt&text=hello";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Application URL Handler

Example Language:

(Bad)

```
-(BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    if (!url) {
        return NO;
    }
    NSString *action = [url host];
    if([action isEqualToString: @"replaceFileText"]) {
        NSDictionary *dict = [self parseQueryStringExampleFunction:[url query]];
        //this function will write contents to a specified file
        FileObject *objectFile = [self writeToFile:[dict objectForKey: @"file"] withText:[dict objectForKey: @"text"]];
    }
    return YES;
}
```

The handler has no restriction on who can use its functionality. The handler can be invoked using any method that invokes the URL handler such as the following malicious iframe embedded on a web page opened by Safari.

Example Language: HTML

(Attack)

```
<iframe src="appscheme://replaceFileText?file=Bookmarks.dat&text=listOfMaliciousWebsites">
```

The attacker can host a malicious website containing the iframe and trick users into going to the site via a crafted phishing email. Since Safari automatically executes iframes, the user is not prompted when the handler executes the iframe code which automatically invokes the URL

handler replacing the bookmarks file with a list of malicious websites. Since `replaceFileText` is a potentially dangerous action, an action that modifies data, there should be a sanity check before the `writeToFile:withText:` function.

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.

Observed Examples

Reference	Description
CVE-2013-5725	URL scheme has action <code>replace</code> which requires no user prompt and allows remote attackers to perform undesired actions. https://www.cve.org/CVERecord?id=CVE-2013-5725
CVE-2013-5726	URL scheme has action <code>follow</code> and <code>favorite</code> which allows remote attackers to force user to perform undesired actions. https://www.cve.org/CVERecord?id=CVE-2013-5726

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-938]Guillaume Ross. "Scheming for Privacy and Security". 2013 November 1. < https://brooksreview.net/2013/11/guest-post_scheming-for-privacy-and-security/ >.2023-04-07.

CWE-940: Improper Verification of Source of a Communication Channel

Weakness ID : 940

Structure : Simple

Abstraction : Base

Description

The product establishes a communication channel to handle an incoming request that has been initiated by an actor, but it does not properly verify that the request is coming from the expected origin.




Extended Description

When an attacker can successfully establish a communication channel from an untrusted origin, the attacker may be able to gain privileges and access unexpected functionality.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		346	Origin Validation Error	860
ChildOf		923	Improper Restriction of Communication Channel to Intended Endpoints	1836
ParentOf		925	Improper Verification of Intent by Broadcast Receiver	1841

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

Nature	Type	ID	Name	Page
MemberOf		1014	Identify Actors	2450

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

Nature	Type	ID	Name	Page
MemberOf		417	Communication Channel Errors	2347

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
Other	Varies by Context <i>An attacker can access any functionality that is inadvertently accessible to the source.</i>	

Potential Mitigations

Phase: Architecture and Design

Use a mechanism that can validate the identity of the source, such as a certificate, and validate the integrity of data to ensure that it cannot be modified in transit using an Adversary-in-the-Middle (AITM) attack. When designing functionality of actions in the URL scheme, consider whether the action should be accessible to all mobile applications, or if an allowlist of applications to interface with is appropriate.

Demonstrative Examples

Example 1:

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

Example Language: Java

(Bad)

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

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

Example 2:

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

Example Language: Java

(Bad)

```
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            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.

Observed Examples

Reference	Description
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning https://www.cve.org/CVERecord?id=CVE-2000-1218
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning https://www.cve.org/CVERecord?id=CVE-2005-0877
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers https://www.cve.org/CVERecord?id=CVE-2001-1452

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2515
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Notes

Relationship

While many access control issues involve authenticating the user, this weakness is more about authenticating the actual source of the communication channel itself; there might not be any "user" in such cases.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
500	WebView Injection
594	Traffic Injection
595	Connection Reset
596	TCP RST Injection

References

[REF-324]Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995 August 1. < <https://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf> >.2024-11-17.

CWE-941: Incorrectly Specified Destination in a Communication Channel

Weakness ID : 941

Structure : Simple

Abstraction : Base

Description

The product creates a communication channel to initiate an outgoing request to an actor, but it does not correctly specify the intended destination for that actor.

Extended Description

Attackers at the destination may be able to spoof trusted servers to steal data or cause a denial of service.



There are at least two distinct weaknesses that can cause the product to communicate with an unintended destination:

- If the product allows an attacker to control which destination is specified, then the attacker can cause it to connect to an untrusted or malicious destination. For example, because UDP is a connectionless protocol, UDP packets can be spoofed by specifying a false source address in the packet; when the server receives the packet and sends a reply, it will specify a destination by using the source of the incoming packet - i.e., the false source. The server can then be tricked into sending traffic to the wrong host, which is effective for hiding the real source of an attack and for conducting a distributed denial of service (DDoS). As another example, server-side request forgery (SSRF) and XML External Entity (XXE) can be used to trick a server into making outgoing requests to hosts that cannot be directly accessed by the attacker due to firewall restrictions.
- If the product incorrectly specifies the destination, then an attacker who can control this destination might be able to spoof trusted servers. While the most common occurrence is likely due to misconfiguration by an administrator, this can be resultant from other weaknesses. For example, the product might incorrectly parse an e-mail or IP address and send sensitive data to an unintended destination. As another example, an Android application may use a "sticky broadcast" to communicate with a receiver for a particular application, but since sticky broadcasts can be processed by *any* receiver, this can allow a malicious application to access restricted data that was only intended for a different application.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		923	Improper Restriction of Communication Channel to Intended Endpoints	1836
CanPrecede		406	Insufficient Control of Network Message Volume (Network Amplification)	997

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

Nature	Type	ID	Name	Page
MemberOf		1014	Identify Actors	2450

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

Nature	Type	ID	Name	Page
MemberOf		417	Communication Channel Errors	2347

Applicable Platforms

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

Technology : Mobile (*Prevalence = Undetermined*)

Demonstrative Examples

Example 1:

This code listens on a port for DNS requests and sends the result to the requesting address.

Example Language: Python

(Bad)

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( ( UDP_IP,UDP_PORT) )
while true:
    data = sock.recvfrom(1024)
    if not data:
        break
    (requestIP, nameToResolve) = parseUDPpacket(data)
    record = resolveName(nameToResolve)
    sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Observed Examples

Reference	Description
CVE-2013-5211	composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses. https://www.cve.org/CVERecord?id=CVE-2013-5211
CVE-1999-0513	Classic "Smurf" attack, using spoofed ICMP packets to broadcast addresses. https://www.cve.org/CVERecord?id=CVE-1999-0513
CVE-1999-1379	DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker. https://www.cve.org/CVERecord?id=CVE-1999-1379

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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

References

[REF-941]US-CERT. "UDP-based Amplification Attacks". 2014 January 7. < <https://www.us-cert.gov/ncas/alerts/TA14-017A> >.

[REF-942]Fortify. "Android Bad Practices: Sticky Broadcast". < https://www.hpe.com/us/en/solutions/infrastructure-security.html?jumpid=va_wnmstr1ug6_aid-510326901 >.2023-04-07.

CWE-942: Permissive Cross-domain Policy with Untrusted Domains

Weakness ID : 942

Structure : Simple

Abstraction : Variant

Description

The product uses a cross-domain policy file that includes domains that should not be trusted.

Extended Description

A cross-domain policy file ("crossdomain.xml" in Flash and "clientaccesspolicy.xml" in Silverlight) defines a list of domains from which a server is allowed to make cross-domain requests. When making a cross-domain request, the Flash or Silverlight client will first look for the policy file on the target server. If it is found, and the domain hosting the application is explicitly allowed to make requests, the request is made.

Therefore, if a cross-domain policy file includes domains that should not be trusted, such as when using wildcards, then the application could be attacked by these untrusted domains.


An overly permissive policy file allows many of the same attacks seen in Cross-Site Scripting (CWE-79). Once the user has executed a malicious Flash or Silverlight application, they are vulnerable to a variety of attacks. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site.

In many cases, the attack can be launched without the victim even being aware of it.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		183	Permissive List of Allowed Inputs	464
ChildOf		923	Improper Restriction of Communication Channel to Intended Endpoints	1836
ChildOf		863	Incorrect Authorization	1796
CanPrecede		668	Exposure of Resource to Wrong Sphere	1478

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

Nature	Type	ID	Name	Page
MemberOf		1011	Authorize Actors	2446

Applicable Platforms

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

Technology : Web Based (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	Bypass Protection Mechanism	
Availability	Read Application Data	
Access Control	Varies by Context	

Scope	Impact	Likelihood
	<i>An attacker may be able to bypass the web browser's same-origin policy. An attacker can exploit the weakness to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running ActiveX controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Avoid using wildcards in the cross-domain policy file. Any domain matching the wildcard expression will be implicitly trusted, and can perform two-way interaction with the target server.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

For Flash, modify crossdomain.xml to use meta-policy options such as 'master-only' or 'none' to reduce the possibility of an attacker planting extraneous cross-domain policy files on a server.

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

For Flash, modify crossdomain.xml to use meta-policy options such as 'master-only' or 'none' to reduce the possibility of an attacker planting extraneous cross-domain policy files on a server.

Demonstrative Examples

Example 1:

These cross-domain policy files mean to allow Flash and Silverlight applications hosted on other domains to access its data:

Flash crossdomain.xml :

Example Language: XML

(Bad)

```
<cross-domain-policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.adobe.com/xml/schemas/PolicyFile.xsd">
<allow-access-from domain="*.example.com"/>
```

```
<allow-access-from domain="*" />
</cross-domain-policy>
```

Silverlight clientaccesspolicy.xml :

Example Language: XML

(Bad)

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
<cross-domain-access>
<policy>
<allow-from http-request-headers="SOAPAction">
<domain uri="*" />
</allow-from>
<grant-to>
<resource path="/" include-subpaths="true"/>
</grant-to>
</policy>
</cross-domain-access>
</access-policy>
```

These entries are far too permissive, allowing any Flash or Silverlight application to send requests. A malicious application hosted on any other web site will be able to send requests on behalf of any user tricked into executing it.

Observed Examples

Reference	Description
CVE-2012-2292	Product has a Silverlight cross-domain policy that does not restrict access to another application, which allows remote attackers to bypass the Same Origin Policy. https://www.cve.org/CVERecord?id=CVE-2012-2292
CVE-2014-2049	The default Flash Cross Domain policies in a product allows remote attackers to access user files. https://www.cve.org/CVERecord?id=CVE-2014-2049
CVE-2007-6243	Chain: Adobe Flash Player does not sufficiently restrict the interpretation and usage of cross-domain policy files, which makes it easier for remote attackers to conduct cross-domain and cross-site scripting (XSS) attacks. https://www.cve.org/CVERecord?id=CVE-2007-6243
CVE-2008-4822	Chain: Adobe Flash Player and earlier does not properly interpret policy files, which allows remote attackers to bypass a non-root domain policy. https://www.cve.org/CVERecord?id=CVE-2008-4822
CVE-2010-3636	Chain: Adobe Flash Player does not properly handle unspecified encodings during the parsing of a cross-domain policy file, which allows remote web servers to bypass intended access restrictions via unknown vectors. https://www.cve.org/CVERecord?id=CVE-2010-3636

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-943]Apurva Udaykumar. "Setting a crossdomain.xml file for HTTP streaming". 2012 November 9. Adobe. < <https://web.archive.org/web/20121124184922/http://www.adobe.com/devnet/adobe-media-server/articles/cross-domain-xml-for-streaming.html> >.2023-04-07.

[REF-944]Adobe. "Cross-domain policy for Flash movies". Adobe. < http://kb2.adobe.com/cps/142/tn_14213.html >.

[REF-945]Microsoft Corporation. "HTTP Communication and Security with Silverlight". < [https://learn.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc838250\(v=vs.95\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc838250(v=vs.95)?redirectedfrom=MSDN) >.2023-04-07.

[REF-946]Microsoft Corporation. "Network Security Access Restrictions in Silverlight". < [https://learn.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645032\(v=vs.95\)](https://learn.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645032(v=vs.95)) >.2023-04-07.

[REF-947]Dongseok Jang, Aishwarya Venkataraman, G. Michael Sawka and Hovav Shacham. "Analyzing the Crossdomain Policies of Flash Applications". 2011 May. < <http://cseweb.ucsd.edu/~hovav/dist/crossdomain.pdf> >.

CWE-943: Improper Neutralization of Special Elements in Data Query Logic

Weakness ID : 943
Structure : Simple
Abstraction : Class

Description

The product generates a query intended to access or manipulate data in a data store such as a database, but it does not neutralize or incorrectly neutralizes special elements that can modify the intended logic of the query.

Extended Description

Depending on the capabilities of the query language, an attacker could inject additional logic into the query to:

- Modify the intended selection criteria, thus changing which data entities (e.g., records) are returned, modified, or otherwise manipulated
- Append additional commands to the query
- Return more entities than intended
- Return fewer entities than intended
- Cause entities to be sorted in an unexpected way






The ability to execute additional commands or change which entities are returned has obvious risks. But when the product logic depends on the order or number of entities, this can also lead to vulnerabilities. For example, if the query expects to return only one entity that specifies an administrative user, but an attacker can change which entities are returned, this could cause the logic to return information for a regular user and incorrectly assume that the user has administrative privileges.

While this weakness is most commonly associated with SQL injection, there are many other query languages that are also subject to injection attacks, including HTSQL, LDAP, DQL, XQuery, Xpath, and "NoSQL" languages.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	138
ParentOf		89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	206
ParentOf		90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	217
ParentOf		643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	1428
ParentOf		652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	1444

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

Nature	Type	ID	Name	Page
MemberOf		1019	Validate Inputs	2454

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Bypass Protection Mechanism	
Integrity	Read Application Data	
Availability	Modify Application Data	
Access Control	Varies by Context	

Detection Methods**Automated Static Analysis**

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

Effectiveness = High

Demonstrative Examples**Example 1:**

The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where owner matches the user name of the currently-authenticated user.

Example Language: C#

(Bad)

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '" + userName + "' AND itemname = '" + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

The query that this code intends to execute follows:

*Example Language:**(Informative)*

```
SELECT * FROM items WHERE owner = <userName> AND itemname = <itemName>;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string:

*Example Language:**(Attack)*

```
name' OR 'a'='a
```

for itemName, then the query becomes the following:

*Example Language:**(Attack)*

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

The addition of the:

*Example Language:**(Attack)*

```
OR 'a'='a
```

condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

*Example Language:**(Attack)*

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 2:

The code below constructs an LDAP query using user input address data:

*Example Language: Java**(Bad)*

```
context = new InitialDirContext(env);
String searchFilter = "StreetAddress=" + address;
NamingEnumeration answer = context.search(searchBase, searchFilter, searchCtrls);
```

Because the code fails to neutralize the address string used to construct the query, an attacker can supply an address that includes additional LDAP queries.

Example 3:

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

*Example Language: XML**(Informative)*

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



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

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

Example Language: Java

(Bad)

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

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

Example Language:

(Attack)

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

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

Observed Examples

Reference	Description
CVE-2014-2503	Injection using Documentum Query Language (DQL) https://www.cve.org/CVERecord?id=CVE-2014-2503
CVE-2014-2508	Injection using Documentum Query Language (DQL) https://www.cve.org/CVERecord?id=CVE-2014-2508

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2456
MemberOf		1409	Comprehensive Categorization: Injection	1400	2556

Notes

Relationship

It could be argued that data query languages are effectively a command language - albeit with a limited set of commands - and thus any query-language injection issue could be treated as a child of CWE-74. However, CWE-943 is intended to better organize query-oriented issues to separate them from fully-functioning programming languages, and also to provide a more precise identifier for the many query languages that do not have their own CWE identifier.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
676	NoSQL Injection

CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag

Weakness ID : 1004

Structure : Simple

Abstraction : Variant

Description

The product uses a cookie to store sensitive information, but the cookie is not marked with the HttpOnly flag.

Extended Description

The HttpOnly flag directs compatible browsers to prevent client-side script from accessing cookies. Including the HttpOnly flag in the Set-Cookie HTTP response header helps mitigate the risk associated with Cross-Site Scripting (XSS) where an attacker's script code might attempt to read the contents of a cookie and exfiltrate information obtained. When set, browsers that support the flag will not reveal the contents of the cookie to a third party via client-side script executed via XSS.

Relationships

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

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

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

Applicable Platforms

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

Technology : Web Based (*Prevalence = Undetermined*)

Background Details

An HTTP cookie is a small piece of data attributed to a specific website and stored on the user's computer by the user's web browser. This data can be leveraged for a variety of purposes including saving information entered into form fields, recording user activity, and for authentication purposes. Cookies used to save or record information generated by the user are accessed and modified by script code embedded in a web page. While cookies used for authentication are created by the website's server and sent to the user to be attached to future requests. These authentication cookies are often not meant to be accessed by the web page sent to the user, and are instead just supposed to be attached to future requests to verify authentication details.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended parties.</i>	
Integrity	Gain Privileges or Assume Identity <i>If the cookie in question is an authentication cookie, then not setting the HttpOnly flag may allow an adversary to steal authentication data (e.g., a session ID) and assume the identity of the user.</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

Leverage the HttpOnly flag when setting a sensitive cookie in a response.

Effectiveness = High

While this mitigation is effective for protecting cookies from a browser's own scripting engine, third-party components or plugins may have their own engines that allow access to cookies. Attackers might also be able to use XMLHttpRequest to read the headers directly and obtain the cookie.

Demonstrative Examples

Example 1:

In this example, a cookie is used to store a session ID for a client's interaction with a website. The intention is that the cookie will be sent to the website with each request made by the client.

The snippet of code below establishes a new cookie to hold the sessionID.

Example Language: Java

(Bad)

```
String sessionID = generateSessionId();
Cookie c = new Cookie("session_id", sessionID);
response.addCookie(c);
```

The HttpOnly flag is not set for the cookie. An attacker who can perform XSS could insert malicious script such as:

Example Language: JavaScript

(Attack)

```
document.write('')
```

When the client loads and executes this script, it makes a request to the attacker-controlled web site. The attacker can then log the request and steal the cookie.

To mitigate the risk, use the setHttpOnly(true) method.

Example Language: Java

(Good)

```
String sessionID = generateSessionId();
Cookie c = new Cookie("session_id", sessionID);
c.setHttpOnly(true);
response.addCookie(c);
```


Observed Examples

Reference	Description
CVE-2022-24045	Web application for a room automation system has client-side Javascript that sets a sensitive cookie without the HTTPOnly security attribute, allowing the cookie to be accessed. https://www.cve.org/CVERecord?id=CVE-2022-24045
CVE-2014-3852	CMS written in Python does not include the HTTPOnly flag in a Set-Cookie header, allowing remote attackers to obtain potentially sensitive information via script access to this cookie. https://www.cve.org/CVERecord?id=CVE-2014-3852

Reference	Description
CVE-2015-4138	Appliance for managing encrypted communications does not use HttpOnly flag. https://www.cve.org/CVERecord?id=CVE-2015-4138

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-2]OWASP. "HttpOnly". < <https://owasp.org/www-community/HttpOnly> >.2023-04-07.

[REF-3]Michael Howard. "Some Bad News and Some Good News". 2002. < [https://learn.microsoft.com/en-us/previous-versions/ms972826\(v=msdn.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/ms972826(v=msdn.10)?redirectedfrom=MSDN) >.2023-04-07.

[REF-4]Troy Hunt. "C is for cookie, H is for hacker - understanding HTTP only and Secure cookies". 2013 March 6. < <https://www.troyhunt.com/c-is-for-cookie-h-is-for-hacker/> >.

[REF-5]Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". < [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.

CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User

Weakness ID : 1007

Structure : Simple

Abstraction : Base

Description

The product displays information or identifiers to a user, but the display mechanism does not make it easy for the user to distinguish between visually similar or identical glyphs (homoglyphs), which may cause the user to misinterpret a glyph and perform an unintended, insecure action.

Extended Description

Some glyphs, pictures, or icons can be semantically distinct to a program, while appearing very similar or identical to a human user. These are referred to as homoglyphs. For example, the lowercase "l" (ell) and uppercase "I" (eye) have different character codes, but these characters can be displayed in exactly the same way to a user, depending on the font. This can also occur between different character sets. For example, the Latin capital letter "A" and the Greek capital letter "Α" (Alpha) are treated as distinct by programs, but may be displayed in exactly the same way to a user. Accent marks may also cause letters to appear very similar, such as the Latin capital letter grave mark "À" and its equivalent "Á" with the acute accent.

Adversaries can exploit this visual similarity for attacks such as phishing, e.g. by providing a link to an attacker-controlled hostname that looks like a hostname that the victim trusts. In a different use of homoglyphs, an adversary may create a back door username that is visually similar to the username of a regular user, which then makes it more difficult for a system administrator to detect the malicious username while reviewing logs.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		451	User Interface (UI) Misrepresentation of Critical Information	1087

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

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

Weakness Ordinalities

Resultant :

Applicable Platforms

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

Technology : Web Based (*Prevalence = Sometimes*)

Alternate Terms

Homograph Attack : "Homograph" is often used as a synonym of "homoglyph" by researchers, but according to Wikipedia, a homograph is a word that has multiple, distinct meanings.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Other <i>An attacker may ultimately redirect a user to a malicious website, by deceiving the user into believing the URL they are accessing is a trusted domain. However, the attack can also be used to forge log entries by using homoglyphs in usernames. Homoglyph manipulations are often the first step towards executing advanced attacks such as stealing a user's credentials, Cross-Site Scripting (XSS), or log forgery. If an attacker redirects a user to a malicious site, the attacker can mimic a trusted domain to steal account credentials and perform actions on behalf of the user, without the user's knowledge. Similarly, an attacker could create a username for a website that contains homoglyph characters, making it difficult for an admin to review logs and determine which users performed which actions.</i>	

Detection Methods

Manual Dynamic Analysis

If utilizing user accounts, attempt to submit a username that contains homoglyphs. Similarly, check to see if links containing homoglyphs can be sent via email, web browsers, or other mechanisms.

Effectiveness = Moderate

Potential Mitigations

Phase: Implementation

Use a browser that displays Punycode for IDNs in the URL and status bars, or which color code various scripts in URLs. Due to the prominence of homoglyph attacks, several browsers now help safeguard against this attack via the use of Punycode. For example, Mozilla Firefox and Google Chrome will display IDNs as Punycode if top-level domains do not restrict which characters can be used in domain names or if labels mix scripts for different languages.

Phase: Implementation

Use an email client that has strict filters and prevents messages that mix character sets to end up in a user's inbox. Certain email clients such as Google's GMail prevent the use of non-Latin characters in email addresses or in links contained within emails. This helps prevent homoglyph attacks by flagging these emails and redirecting them to a user's spam folder.

Demonstrative Examples

Example 1:

The following looks like a simple, trusted URL that a user may frequently access.

Example Language: (Attack)

http://www.#x#m#l#.#m

However, the URL above is comprised of Cyrillic characters that look identical to the expected ASCII characters. This results in most users not being able to distinguish between the two and assuming that the above URL is trusted and safe. The "e" is actually the "CYRILLIC SMALL LETTER IE" which is represented in HTML as the character е, while the "a" is actually the "CYRILLIC SMALL LETTER A" which is represented in HTML as the character а. The "p", "c", and "o" are also Cyrillic characters in this example. Viewing the source reveals a URL of "http://www.еxаmрlе.соm". An adversary can utilize this approach to perform an attack such as a phishing attack in order to drive traffic to a malicious website.

Example 2:

The following displays an example of how creating usernames containing homoglyphs can lead to log forgery.

Assume an adversary visits a legitimate, trusted domain and creates an account named "admin", except the 'a' and 'i' characters are Cyrillic characters instead of the expected ASCII. Any actions the adversary performs will be saved to the log file and look like they came from a legitimate administrator account.

Example Language: (Result)

123.123.123.123 #dm#n [17/Jul/2017:09:05:49 -0400] "GET /example/users/userlist HTTP/1.1" 401 12846
123.123.123.123 #dm#n [17/Jul/2017:09:06:51 -0400] "GET /example/users/userlist HTTP/1.1" 200 4523
123.123.123.123 admin [17/Jul/2017:09:10:02 -0400] "GET /example/users/editusers HTTP/1.1" 200 6291
123.123.123.123 #dm#n [17/Jul/2017:09:10:02 -0400] "GET /example/users/editusers HTTP/1.1" 200 6291

Upon closer inspection, the account that generated three of these log entries is "аdmіn". Only the third log entry is by the legitimate admin account. This makes it more difficult to determine which actions were performed by the adversary and which actions were executed by the legitimate "admin" account.

Observed Examples

Reference	Description
CVE-2013-7236	web forum allows impersonation of users with homoglyphs in account names https://www.cve.org/CVERecord?id=CVE-2013-7236
CVE-2012-0584	Improper character restriction in URLs in web browser https://www.cve.org/CVERecord?id=CVE-2012-0584

Reference	Description
CVE-2009-0652	Incomplete denylist does not include homoglyphs of "/" and "?" characters in URLs https://www.cve.org/CVERecord?id=CVE-2009-0652
CVE-2017-5015	web browser does not convert hyphens to punycode, allowing IDN spoofing in URLs https://www.cve.org/CVERecord?id=CVE-2017-5015
CVE-2005-0233	homoglyph spoofing using punycode in URLs and certificates https://www.cve.org/CVERecord?id=CVE-2005-0233
CVE-2005-0234	homoglyph spoofing using punycode in URLs and certificates https://www.cve.org/CVERecord?id=CVE-2005-0234
CVE-2005-0235	homoglyph spoofing using punycode in URLs and certificates https://www.cve.org/CVERecord?id=CVE-2005-0235

MemberOf Relationships

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

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
632	Homograph Attack via Homoglyphs

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-8]Gregory Baatard and Peter Hannay. "The 2011 IDN Homograph Attack Mitigation Survey". 2012. ECU Publications. < <http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1174&context=ecuworks2012> >.

CWE-1021: Improper Restriction of Rendered UI Layers or Frames

Weakness ID : 1021

Structure : Simple

Abstraction : Base

Description

The web application does not restrict or incorrectly restricts frame objects or UI layers that belong to another application or domain, which can lead to user confusion about which interface the user is interacting with.



Extended Description

A web application is expected to place restrictions on whether it is allowed to be rendered within frames, iframes, objects, embed or applet elements. Without the restrictions, users can be tricked into interacting with the application when they were not intending to.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		451	User Interface (UI) Misrepresentation of Critical Information	1087
ChildOf		441	Unintended Proxy or Intermediary ('Confused Deputy')	1072

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

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

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

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

Applicable Platforms

Technology : Web Based (*Prevalence = Undetermined*)

Alternate Terms

Clickjacking :

UI Redress Attack :

Tapjacking : "Tapjacking" is similar to clickjacking, except it is used for mobile applications in which the user "taps" the application instead of performing a mouse click.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism Read Application Data Modify Application Data <i>An attacker can trick a user into performing actions that are masked and hidden from the user's view. The impact varies widely, depending on the functionality of the underlying application. For example, in a social media application, clickjacking could be used to trick the user into changing privacy settings.</i>	

Detection Methods**Automated Static Analysis**

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

Effectiveness = High

Potential Mitigations**Phase: Implementation**

The use of X-Frame-Options allows developers of web content to restrict the usage of their application within the form of overlays, frames, or iFrames. The developer can indicate from which domains can frame the content. The concept of X-Frame-Options is well documented, but

implementation of this protection mechanism is in development to cover gaps. There is a need for allowing frames from multiple domains.

Phase: Implementation

A developer can use a "frame-breaker" script in each page that should not be framed. This is very helpful for legacy browsers that do not support X-Frame-Options security feature previously mentioned. It is also important to note that this tactic has been circumvented or bypassed. Improper usage of frames can persist in the web application through nested frames. The "frame-breaking" script does not intuitively account for multiple nested frames that can be presented to the user.

Phase: Implementation




This defense-in-depth technique can be used to prevent the improper usage of frames in web applications. It prioritizes the valid sources of data to be loaded into the application through the usage of declarative policies. Based on which implementation of Content Security Policy is in use, the developer should use the "frame-ancestors" directive or the "frame-src" directive to mitigate this weakness. Both directives allow for the placement of restrictions when it comes to allowing embedded content.

Observed Examples

Reference	Description
CVE-2017-7440	E-mail preview feature in a desktop application allows clickjacking attacks via a crafted e-mail message https://www.cve.org/CVERecord?id=CVE-2017-7440
CVE-2017-5697	Hardware/firmware product has insufficient clickjacking protection in its web user interface https://www.cve.org/CVERecord?id=CVE-2017-5697
CVE-2017-4015	Clickjacking in data-loss prevention product via HTTP response header. https://www.cve.org/CVERecord?id=CVE-2017-4015
CVE-2016-2496	Tapjacking in permission dialog for mobile OS allows access of private storage using a partially-overlapping window. https://www.cve.org/CVERecord?id=CVE-2016-2496
CVE-2015-1241	Tapjacking in web browser related to page navigation and touch/gesture events. https://www.cve.org/CVERecord?id=CVE-2015-1241
CVE-2017-0492	System UI in mobile OS allows a malicious application to create a UI overlay of the entire screen to gain privileges. https://www.cve.org/CVERecord?id=CVE-2017-0492

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design		1344 2512
MemberOf		1396	Comprehensive Categorization: Access Control		1400 2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
103	Clickjacking
181	Flash File Overlay
222	iFrame Overlay
504	Task Impersonation
506	Tapjacking

CAPEC-ID Attack Pattern Name

587	Cross Frame Scripting (XFS)
654	Credential Prompt Impersonation

References

[REF-35]Andrew Horton. "Clickjacking For Shells". < <https://www.exploit-db.com/docs/17881.pdf> >.

[REF-36]OWASP. "Clickjacking - OWASP". < <https://owasp.org/www-community/attacks/Clickjacking> >.2023-04-07.

[REF-37]Internet Security. "SecTheory". < <https://www.sectheory.com/clickjacking.htm> >.2023-04-07.

[REF-38]W3C. "Content Security Policy Level 3". < <https://w3c.github.io/webappsec-csp/> >.

CWE-1022: Use of Web Link to Untrusted Target with window.opener Access

Weakness ID : 1022

Structure : Simple

Abstraction : Variant

Description

The web application produces links to untrusted external sites outside of its sphere of control, but it does not properly prevent the external site from modifying security-critical properties of the window.opener object, such as the location property.

Extended Description

When a user clicks a link to an external site ("target"), the target="_blank" attribute causes the target site's contents to be opened in a new window or tab, which runs in the same process as the original page. The window.opener object records information about the original page that offered the link. If an attacker can run script on the target page, then they could read or modify certain properties of the window.opener object, including the location property - even if the original and target site are not the same origin. An attacker can modify the location property to automatically redirect the user to a malicious site, e.g. as part of a phishing attack. Since this redirect happens in the original window/tab - which is not necessarily visible, since the browser is focusing the display on the new target page - the user might not notice any suspicious redirection.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		266	Incorrect Privilege Assignment	645

Applicable Platforms

Language : JavaScript (*Prevalence = Often*)

Technology : Web Based (*Prevalence = Often*)

Alternate Terms

tabnabbing :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Alter Execution Logic <i>The user may be redirected to an untrusted page that contains undesired content or malicious script code.</i>	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Specify in the design that any linked external document must not be granted access to the location object of the calling page.

Phase: Implementation

When creating a link to an external document using the <a> tag with a defined target, for example "_blank" or a named frame, provide the rel attribute with a value "noopener noreferrer". If opening the external document in a new window via javascript, then reset the opener by setting it equal to null.

Phase: Implementation

Do not use "_blank" targets. However, this can affect the usability of the application.

Demonstrative Examples

Example 1:

In this example, the application opens a link in a named window/tab without taking precautions to prevent the called page from tampering with the calling page's location in the browser.

There are two ways that this weakness is commonly seen. The first is when the application generates an <a> tag is with target="_blank" to point to a target site:

Example Language: HTML

(Bad)

```
<a href="http://attacker-site.example.com/useful-page.html" target="_blank">
```

If the attacker offers a useful page on this link (or compromises a trusted, popular site), then a user may click on this link. However, the attacker could use scripting code to modify the window.opener's location property to redirect the application to a malicious, attacker-controlled page - such as one that mimics the look and feel of the original application and convinces the user to re-enter authentication credentials, i.e. phishing:

Example Language: JavaScript

(Attack)

```
window.opener.location = 'http://phishing.example.org/popular-bank-page';
```

To mitigate this type of weakness, some browsers support the "rel" attribute with a value of "noopener", which sets the window.opener object equal to null. Another option is to use the "rel" attribute with a value of "noreferrer", which in essence does the same thing.

*Example Language: HTML**(Good)*

```
<a href="http://attacker-site.example.com/useful-page.html" target="_blank" rel="noopener noreferrer">
```

A second way that this weakness is commonly seen is when opening a new site directly within JavaScript. In this case, a new site is opened using the `window.open()` function.

*Example Language: JavaScript**(Bad)*

```
var newWindow = window.open("http://attacker-site.example.com/useful-page.html", "_blank");
```

To mitigate this, set the `window.opener` object to null.

*Example Language: JavaScript**(Good)*

```
var newWindow = window.open("http://attacker-site.example.com/useful-page.html", "_blank");  
newWindow.opener = null;
```

Observed Examples

Reference	Description
CVE-2022-4927	Library software does not use <code>rel="noopener noreferrer"</code> setting, allowing tabnabbing attacks to redirect to a malicious page https://www.cve.org/CVERecord?id=CVE-2022-4927

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control		1400 2540

References

[REF-39]Alex Yumashev. "Target="_blank" - the most underestimated vulnerability ever". 2016 May 4. < <https://medium.com/@jitbit/target-blank-the-most-underestimated-vulnerability-ever-96e328301f4c> >.

[REF-40]Ben Halpern. "The target="_blank" vulnerability by example". 2016 September 1. < <https://dev.to/ben/the-targetblank-vulnerability-by-example> >.

[REF-958]Mathias Bynens. "About rel=noopener". 2016 March 5. < <https://mathiasbynens.github.io/rel-noopener/> >.

CWE-1023: Incomplete Comparison with Missing Factors

Weakness ID : 1023**Structure** : Simple**Abstraction** : Class

Description

The product performs a comparison between entities that must consider multiple factors or characteristics of each entity, but the comparison does not include one or more of these factors.

Extended Description

An incomplete comparison can lead to resultant weaknesses, e.g., by operating on the wrong object or making a security decision without considering a required 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
ChildOf	P	697	Incorrect Comparison	1538
ParentOf	B	184	Incomplete List of Disallowed Inputs	466
ParentOf	V	187	Partial String Comparison	474
ParentOf	B	478	Missing Default Case in Multiple Condition Expression	1149
ParentOf	B	839	Numeric Range Comparison Without Minimum Check	1776

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

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-2005-2782	PHP remote file inclusion in web application that filters "http" and "https" URLs, but not "ftp". https://www.cve.org/CVERecord?id=CVE-2005-2782
CVE-2014-6394	Product does not prevent access to restricted directories due to partial string comparison with a public directory

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2014-6394

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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-1024: Comparison of Incompatible Types

Weakness ID : 1024

Structure : Simple

Abstraction : Base

Description

The product performs a comparison between two entities, but the entities are of different, incompatible types that cannot be guaranteed to provide correct results when they are directly compared.

Extended Description

In languages that are strictly typed but support casting/conversion, such as C or C++, the programmer might assume that casting one entity to the same type as another entity will ensure that the comparison will be performed correctly, but this cannot be guaranteed. In languages that are not strictly typed, such as PHP or JavaScript, there may be implicit casting/conversion to a type that the programmer is unaware of, causing unexpected results; for example, the string "123" might be converted to a number type. See examples.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		697	Incorrect Comparison	1538

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

Nature	Type	ID	Name	Page
MemberOf		19	Data Processing Errors	2330

Weakness Ordinalities

Primary :

Applicable Platforms

Language : JavaScript (*Prevalence = Undetermined*)

Language : PHP (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

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

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1397	Comprehensive Categorization: Comparison	1400	2544

CWE-1025: Comparison Using Wrong Factors

Weakness ID : 1025

Structure : Simple

Abstraction : Base

Description

The code performs a comparison between two entities, but the comparison examines the wrong factors or characteristics of the entities, which can lead to incorrect results and resultant weaknesses.

Extended Description

This can lead to incorrect results and resultant weaknesses. For example, the code might inadvertently compare references to objects, instead of the relevant contents of those objects, causing two "equal" objects to be considered unequal.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	697	Incorrect Comparison	1538
ParentOf	V	486	Comparison of Classes by Name	1172
ParentOf	V	595	Comparison of Object References Instead of Object Contents	1342

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2348

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

Demonstrative Examples

Example 1:

In the example below, two Java String objects are declared and initialized with the same string values. An if statement is used to determine if the strings are equivalent.

Example Language: Java

(Bad)

```
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
    System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

Example Language:

(Good)

```
if (str1.equals(str2)) {
    System.out.println("str1 equals str2");
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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-1037: Processor Optimization Removal or Modification of Security-critical Code

Weakness ID : 1037

Structure : Simple

Abstraction : Base

Description

The developer builds a security-critical protection mechanism into the software, but the processor optimizes the execution of 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
PeerOf		1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	2098

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

Nature	Type	ID	Name	Page
MemberOf		438	Behavioral Problems	2348

Weakness Ordinalities

Primary : This weakness does not depend on other weaknesses and is the result of choices made by the processor in executing the specified application.

Applicable Platforms

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Bypass Protection Mechanism <i>A successful exploitation of this weakness will change the order of an application's execution and will likely be used to bypass specific protection mechanisms. This bypass can be exploited further to potentially read data that should otherwise be inaccessible.</i>	High

Detection Methods

White Box

In theory this weakness can be detected through the use of white box testing techniques where specifically crafted test cases are used in conjunction with debuggers to verify the order of statements being executed.

Effectiveness = Opportunistic

Although the mentioned detection method is theoretically possible, the use of speculative execution is a preferred way of increasing processor performance. The reality is that a large number of statements are executed out of order, and determining if any of them break an access control property would be extremely opportunistic.

Observed Examples

Reference	Description
CVE-2017-5715	Intel, ARM, and AMD processor optimizations related to speculative execution and branch prediction cause access control checks to be bypassed when placing data into the cache. Often known as "Spectre". https://www.cve.org/CVERecord?id=CVE-2017-5715
CVE-2017-5753	Intel, ARM, and AMD processor optimizations related to speculative execution and branch prediction cause access control checks to be bypassed when placing data into the cache. Often known as "Spectre". https://www.cve.org/CVERecord?id=CVE-2017-5753

Reference	Description
CVE-2017-5754	Intel processor optimizations related to speculative execution cause access control checks to be bypassed when placing data into the cache. Often known as "Meltdown". https://www.cve.org/CVERecord?id=CVE-2017-5754

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1398	Comprehensive Categorization: Component Interaction	1400	2545

Notes

Maintenance

As of CWE 4.9, members of the CWE Hardware SIG are closely analyzing this entry and others to improve CWE's coverage of transient execution weaknesses, which include issues related to Spectre, Meltdown, and other attacks. Additional investigation may include other weaknesses related to microarchitectural state. As a result, this entry might change significantly in CWE 4.10.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
663	Exploitation of Transient Instruction Execution

References

[REF-11]Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz and Yuval Yarom. "Spectre Attacks: Exploiting Speculative Execution". 2018 January 3. < <https://arxiv.org/abs/1801.01203> >.

[REF-12]Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom and Mike Hamburg. "Meltdown". 2018 January 3. < <https://arxiv.org/abs/1801.01207> >.

CWE-1038: Insecure Automated Optimizations

Weakness ID : 1038

Structure : Simple

Abstraction : Class


Description

The product uses a mechanism that automatically optimizes code, e.g. to improve a characteristic such as performance, but the optimizations can have an unintended side effect that might violate an intended security assumption.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591

Nature	Type	ID	Name	Page
ChildOf	I P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	1063
ParentOf	B	733	Compiler Optimization Removal or Modification of Security-critical Code	1570
ParentOf	B	1037	Processor Optimization Removal or Modification of Security-critical Code	1879

Weakness Ordinalities

Primary : This weakness does not depend on other weaknesses and is the result of choices made during optimization.

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic <i>The optimizations alter the order of execution resulting in side effects that were not intended by the original developer.</i>	

Observed Examples

Reference	Description
CVE-2017-5715	Intel, ARM, and AMD processor optimizations related to speculative execution and branch prediction cause access control checks to be bypassed when placing data into the cache. Often known as "Spectre". https://www.cve.org/CVERecord?id=CVE-2017-5715
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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1398	Comprehensive Categorization: Component Interaction	1400	2545

CWE-1039: Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations

Weakness ID : 1039

Structure : Simple

Abstraction : Class

Description

The product uses an automated mechanism such as machine learning to recognize complex data inputs (e.g. image or audio) as a particular concept or category, but it does not properly detect or handle inputs that have been modified or constructed in a way that causes the mechanism to detect a different, incorrect concept.

Extended Description

When techniques such as machine learning are used to automatically classify input streams, and those classifications are used for security-critical decisions, then any mistake in classification can introduce a vulnerability that allows attackers to cause the product to make the wrong security decision. If the automated mechanism is not developed or "trained" with enough input data, then attackers may be able to craft malicious input that intentionally triggers the incorrect classification.

Targeted technologies include, but are not necessarily limited to:

- automated speech recognition
- automated image recognition

For example, an attacker might modify road signs or road surface markings to trick autonomous vehicles into misreading the sign/markings and performing a dangerous action.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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]	697	Incorrect Comparison	1538
ChildOf	[P]	693	Protection Mechanism Failure	1529

Weakness Ordinalities

Primary : This weakness does not depend on other weaknesses and is the result of choices made during optimization.

Applicable Platforms

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


Technology : AI/ML (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Bypass Protection Mechanism <i>When the automated recognition is used in a protection mechanism, an attacker may be able to craft inputs that are misinterpreted in a way that grants excess privileges.</i>	

MemberOf Relationships

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

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

Notes

Relationship

Further investigation is needed to determine if better relationships exist or if additional organizational entries need to be created. For example, this issue might be better related to

"recognition of input as an incorrect type," which might place it as a sibling of CWE-704 (incorrect type conversion).

References

- [REF-16]Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow and Rob Fergus. "Intriguing properties of neural networks". 2014 February 9. < <https://arxiv.org/abs/1312.6199> >.
- [REF-17]OpenAI. "Attacking Machine Learning with Adversarial Examples". 2017 February 4. < <https://openai.com/research/attacking-machine-learning-with-adversarial-examples> >.2023-04-07.
- [REF-15]James Vincent. "Magic AI: These are the Optical Illusions that Trick, Fool, and Flummox Computers". 2017 April 2. The Verge. < <https://www.theverge.com/2017/4/12/15271874/ai-adversarial-images-fooling-attacks-artificial-intelligence> >.
- [REF-13]Xuejing Yuan, Yuxuan Chen, Yue Zhao, Yunhui Long, Xiaokang Liu, Kai Chen, Shengzhi Zhang, Heqing Huang, Xiaofeng Wang and Carl A. Gunter. "CommanderSong: A Systematic Approach for Practical Adversarial Voice Recognition". 2018 January 4. < <https://arxiv.org/pdf/1801.08535.pdf> >.
- [REF-14]Nicholas Carlini and David Wagner. "Audio Adversarial Examples: Targeted Attacks on Speech-to-Text". 2018 January 5. < <https://arxiv.org/abs/1801.01944> >.

CWE-1041: Use of Redundant Code

Weakness ID : 1041
Structure : Simple
Abstraction : Base

Description

The product has multiple functions, methods, procedures, macros, etc. that contain the same code.


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. For example, if there are two copies of the same code, the programmer might fix a weakness in one copy while forgetting to fix the same weakness in another copy.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Potential Mitigations

Phase: Implementation

Merge common functionality into a single function and then call that function from across the entire code base.

Demonstrative Examples

Example 1:

In the following Java example the code performs some complex math when specific test conditions are met. The math is the same in each case and the equations are repeated within the code. Unfortunately if a future change needs to be made then that change needs to be made in all locations. This opens the door to mistakes being made and the changes not being made in the same way in each instance.

Example Language: Java

(Bad)

```
public class Main {
    public static void main(String[] args) {
        double s = 10.0;
        double r = 1.0;
        double pi = 3.14159;
        double surface_area;
        if(r > 0.0) {
            // complex math equations
            surface_area = pi * r * s + pi * Math.pow(r, 2);
        }
        if(r > 1.0) {
            // a complex set of math
            surface_area = pi * r * s + pi * Math.pow(r, 2);
        }
    }
}
```

It is recommended to place the complex math into its own function and then call that function whenever necessary.

Example Language: Java

(Good)

```
public class Main {
    private double ComplexMath(double r, double s) {
        //complex math equations
        double pi = Math.PI;
        double surface_area = pi * r * s + pi * Math.pow(r, 2);
        return surface_area;
    }
    public static void main(String[] args) {
        double s = 10.0;
        double r = 1.0;
        double surface_area;
        if(r > 0.0) {
            surface_area = ComplexMath(r, s);
        }
        if(r > 1.0) {
            surface_area = ComplexMath(r, s);
        }
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-19		

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-1042: Static Member Data Element outside of a Singleton Class Element

Weakness ID : 1042

Structure : Simple

Abstraction : Variant

Description

The code contains a member element that is declared as static (but not final), in which its parent class element is not a singleton class - that is, a class element that can be used only once in the 'to' association of a Create action.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	1176	Inefficient CPU Computation	1980

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

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

Nature	Type	ID	Name	View	Page
MemberOf		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-3		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements

Weakness ID : 1043

Structure : Simple

Abstraction : Base

Description

The product uses a data element that has an excessively large number of sub-elements with non-primitive data types such as structures or aggregated objects.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "excessively large" may vary for each product or developer, CISQ recommends a default of 5 sub-elements.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1093	Excessively Complex Data Representation	1942

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-12		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1044: Architecture with Number of Horizontal Layers Outside of Expected Range

Weakness ID : 1044

Structure : Simple

Abstraction : Base

Description

The product's architecture contains too many - or too few - horizontal layers.

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.

While the interpretation of "expected range" may vary for each product or developer, CISQ recommends a default minimum of 4 layers and maximum of 8 layers.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities



Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-9		

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-1045: Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor

Weakness ID : 1045

Structure : Simple

Abstraction : Base

Description

A parent class has a virtual destructor method, but the parent has a child class that does not have a virtual destructor.


Extended Description

This issue can prevent the product from running reliably, since the child might not perform essential destruction operations. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability, such as a memory leak (CWE-401).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities






Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-17		

References

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

[REF-977]QuantStart. "C++ Virtual Destructors: How to Avoid Memory Leaks". < <https://www.quantstart.com/articles/C-Virtual-Destructors-How-to-Avoid-Memory-Leaks/> >.2023-04-07.

[REF-978]GeeksforGeeks. "Virtual Destructor". < <https://www.geeksforgeeks.org/virtual-destructor/> >.

CWE-1046: Creation of Immutable Text Using String Concatenation

Weakness ID : 1046

Structure : Simple

Abstraction : Base

Description

The product creates an immutable text string using string concatenation operations.

Extended Description

When building a string via a looping feature (e.g., a FOR or WHILE loop), the use of += to append to the existing string will result in the creation of a new object with each iteration. This programming pattern can be inefficient in comparison with use of text buffer data elements. This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this could be influenced to create performance problem.

Relationships


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

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

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

Nature	Type	ID	Name	Page
ChildOf		1176	Inefficient CPU Computation	1980

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-2		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1047: Modules with Circular Dependencies

Weakness ID : 1047
Structure : Simple
Abstraction : Base

Description

The product contains modules in which one module has references that cycle back to itself, i.e., there are circular dependencies.

Extended Description

As an example, with Java, this weakness might indicate cycles between packages.

This issue makes it more difficult to maintain the product due to insufficient modularity, 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.

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities






Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-7		
OMG ASCRM	ASCRM-RLB-13		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < <https://www.omg.org/spec/ASCMM/> >.2023-04-07.

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

CWE-1048: Invokable Control Element with Large Number of Outward Calls

Weakness ID : 1048

Structure : Simple

Abstraction : Base

Description

The code contains callable control elements that contain an excessively large number of references to other application objects external to the context of the callable, i.e. a Fan-Out value that is excessively large.

Extended Description

While the interpretation of "excessively large Fan-Out value" may vary for each product or developer, CISQ recommends a default of 5 referenced objects.

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	[P]	710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf	[C]	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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]	1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf	[C]	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	[C]	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-4		

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-1049: Excessive Data Query Operations in a Large Data Table

Weakness ID : 1049

Structure : Simple

Abstraction : Base

Description

The product performs a data query with a large number of joins and sub-queries on a large data table.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "large number of joins or sub-queries" may vary for each product or developer, CISQ recommends a default of 1 million rows for a "large" data table, a default minimum of 5 joins, and a default minimum of 3 sub-queries.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1176	Inefficient CPU Computation	1980

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-4		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPem)". 2016 January. < <https://www.omg.org/spec/ASCPem/> >.2023-04-07.

CWE-1050: Excessive Platform Resource Consumption within a Loop

Weakness ID : 1050

Structure : Simple

Abstraction : Base

Description

The product has a loop body or loop condition that contains a control element that directly or indirectly consumes platform resources, e.g. messaging, sessions, locks, or file descriptors.


Extended Description

This issue can make the product perform more slowly. If an attacker can influence the number of iterations in the loop, then this performance problem might allow a denial of service by consuming more platform resources than intended.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPem-PRF-8		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1051: Initialization with Hard-Coded Network Resource Configuration Data

Weakness ID : 1051
Structure : Simple
Abstraction : Base

Description

The product initializes data using hard-coded values that act as network resource identifiers.

Extended Description

This issue can prevent the product from running reliably, e.g. if it runs in an environment does not use the hard-coded network resource identifiers. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1419	Incorrect Initialization of Resource	2292

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

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

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1340	CISQ Data Protection Measures	1340	2611
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-18		

References

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

CWE-1052: Excessive Use of Hard-Coded Literals in Initialization

Weakness ID : 1052

Structure : Simple

Abstraction : Base

Description

The product initializes a data element using a hard-coded literal that is not a simple integer or static constant element.

Extended Description

This issue makes it more difficult to modify or 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		1419	Incorrect Initialization of Resource	2292

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

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

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-3		

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-1053: Missing Documentation for Design

Weakness ID : 1053

Structure : Simple

Abstraction : Base

Description

The product does not have documentation that represents how it is designed.

Extended Description

This issue can make it more difficult to understand and maintain the product. It can make it more difficult and time-consuming to detect and/or fix 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		1059	Insufficient Technical Documentation	1904

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





Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2501

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1208	Cross-Cutting Problems	1194	2495
MemberOf		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2532
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < <https://www.researchgate.net/>

publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENTS-2023-04-07.

CWE-1054: Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer

Weakness ID : 1054

Structure : Simple

Abstraction : Base

Description

The code at one architectural layer invokes code that resides at a deeper layer than the adjacent layer, i.e., the invocation skips at least one layer, and the invoked code is not part of a vertical utility layer that can be referenced from any horizontal layer.


Extended Description

This issue makes it more difficult to understand and 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

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities


Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-12		

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-1055: Multiple Inheritance from Concrete Classes

Weakness ID : 1055

Structure : Simple

Abstraction : Base

Description

The product contains a class with inheritance from more than one concrete class.

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, 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		1093	Excessively Complex Data Representation	1942

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-2		

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-1056: Invokable Control Element with Variadic Parameters

Weakness ID : 1056

Structure : Simple

Abstraction : Base

Description

A named-callable or method control element has a signature that supports a variable (variadic) number of parameters or arguments.

Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

With variadic arguments, it can be difficult or inefficient for manual analysis to be certain of which function/method is being invoked.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-8		

References

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

CWE-1057: Data Access Operations Outside of Expected Data Manager Component

Weakness ID : 1057

Structure : Simple

Abstraction : Base

Description

The product uses a dedicated, central data manager component as required by design, but it contains code that performs data-access operations that do not use this data manager.


Extended Description

This issue can make the product perform more slowly than intended, since the intended central data manager may have been explicitly optimized for performance or other quality characteristics. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities



Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464

Nature	Type	ID	Name	V	Page
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-11		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element

Weakness ID : 1058

Structure : Simple

Abstraction : Base

Description

The code contains a function or method that operates in a multi-threaded environment but owns an unsafe non-final static storable or member data element.


Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-11		

References

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

CWE-1059: Insufficient Technical Documentation

Weakness ID : 1059

Structure : Simple

Abstraction : Class

Description

The product does not contain sufficient technical or engineering documentation (whether on paper or in electronic form) that contains descriptions of all the relevant software/hardware elements of the product, such as its usage, structure, architectural components, interfaces, design, implementation, configuration, operation, etc.

Extended Description

When technical documentation is limited or lacking, products are more difficult to maintain. This indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities.

When using time-limited or labor-limited third-party/in-house security consulting services (such as threat modeling, vulnerability discovery, or pentesting), insufficient documentation can force those consultants to invest unnecessary time in learning how the product is organized, instead of focusing their expertise on finding the flaws or suggesting effective mitigations.

With respect to hardware design, the lack of a formal, final manufacturer reference can make it difficult or impossible to evaluate the final product, including post-manufacture verification. One cannot ensure that design functionality or operation is within acceptable tolerances, conforms to specifications, and is free from unexpected behavior. Hardware-related documentation may include engineering artifacts such as hardware description language (HDLs), netlists, Gerber files, Bills of Materials, EDA (Electronic Design Automation) tool files, etc.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	B	1053	Missing Documentation for Design	1898
ParentOf	B	1110	Incomplete Design Documentation	1959
ParentOf	B	1111	Incomplete I/O Documentation	1960
ParentOf	B	1112	Incomplete Documentation of Program Execution	1961
ParentOf	B	1118	Insufficient Documentation of Error Handling Techniques	1967

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context Hide Activities Reduce Reliability Quality Degradation Reduce Maintainability <i>Without a method of verification, one cannot be sure that everything only functions as expected.</i>	

Potential Mitigations

Phase: Documentation

Phase: Architecture and Design

Ensure that design documentation is detailed enough to allow for post-manufacturing verification.

Observed Examples

Reference	Description
CVE-2022-3203	A wireless access point manual specifies that the only method of configuration is via web interface (CWE-1059), but there is an undisclosed telnet server that was activated by default (CWE-912). https://www.cve.org/CVERecord?id=CVE-2022-3203

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1195	Manufacturing and Life Cycle Management Concerns	1194	2490
MemberOf	C	1208	Cross-Cutting Problems	1194	2495

Nature	Type	ID	Name	V	Page
MemberOf		1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf		1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2529
MemberOf		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2532
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 2-4		Req SP.02.03 BR
ISA/IEC 62443	Part 2-4		Req SP.02.03 RE(1)
ISA/IEC 62443	Part 2-4		Req SP.03.03 RE(1)
ISA/IEC 62443	Part 4-1		Req SG-1
ISA/IEC 62443	Part 4-1		Req SG-2
ISA/IEC 62443	Part 4-1		Req SG-3
ISA/IEC 62443	Part 4-1		Req SG-4
ISA/IEC 62443	Part 4-1		Req SG-5
ISA/IEC 62443	Part 4-1		Req SG-6
ISA/IEC 62443	Part 4-1		Req SG-7

References

[REF-1248]Securing Energy Infrastructure Executive Task Force (SEI ETF). "Categories of Security Vulnerabilities in ICS". 2022 March 9. < https://inl.gov/wp-content/uploads/2022/03/SEI-ETF-NCSV-TPT-Categories-of-Security-Vulnerabilities-ICS-v1_03-09-22.pdf >.

[REF-1254]FDA. "Cybersecurity in Medical Devices: Quality System Considerations and Content of Premarket Submissions Draft Guidance for Industry and Food and Drug Administration Staff (DRAFT GUIDANCE)". 2022 April 8. < <https://www.fda.gov/media/119933/download> >.

CWE-1060: Excessive Number of Inefficient Server-Side Data Accesses

Weakness ID : 1060

Structure : Simple

Abstraction : Base

Description

The product performs too many data queries without using efficient data processing functionality such as stored procedures.

Extended Description

This issue can make the product perform more slowly due to computational expense. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "too many data queries" may vary for each product or developer, CISQ recommends a default maximum of 5 data queries for an inefficient function/procedure.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-9		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1061: Insufficient Encapsulation

Weakness ID : 1061

Structure : Simple

Abstraction : Class

Description

The product does not sufficiently hide the internal representation and implementation details of data or methods, which might allow external components or modules to modify data unexpectedly, invoke unexpected functionality, or introduce dependencies that the programmer did not intend.

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	P	710	Improper Adherence to Coding Standards	1558
ParentOf	B	766	Critical Data Element Declared Public	1615
ParentOf	B	1054	Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer	1899
ParentOf	B	1057	Data Access Operations Outside of Expected Data Manager Component	1902
ParentOf	B	1062	Parent Class with References to Child Class	1909
ParentOf	B	1083	Data Access from Outside Expected Data Manager Component	1932
ParentOf	B	1090	Method Containing Access of a Member Element from Another Class	1939
ParentOf	B	1100	Insufficient Isolation of System-Dependent Functions	1949
ParentOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1954

Weakness Ordinalities

Indirect :

Demonstrative Examples

Example 1:

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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-969]Wikipedia. "Encapsulation (computer programming)". < [https://en.wikipedia.org/wiki/Encapsulation_\(computer_programming\)](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)) >.

CWE-1062: Parent Class with References to Child Class

Weakness ID : 1062

Structure : Simple

Abstraction : Base

Description

The code has a parent class that contains references to a child class, its methods, or its members.


Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-14		

References

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

CWE-1063: Creation of Class Instance within a Static Code Block

Weakness ID : 1063

Structure : Simple

Abstraction : Base

Description

A static code block creates an instance of a class.

Extended Description

This pattern identifies situations where a storable data element or member data element is initialized with a value in a block of code which is declared as static.

This issue can make the product perform more slowly by performing initialization before it is needed. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1176	Inefficient CPU Computation	1980

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-1		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1064: Invokable Control Element with Signature Containing an Excessive Number of Parameters

Weakness ID : 1064

Structure : Simple

Abstraction : Base

Description

The product contains a function, subroutine, or method whose signature has an unnecessarily large number of parameters/arguments.

Extended Description

This issue makes it more difficult to understand and/or 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.

While the interpretation of "large number of parameters." may vary for each product or developer, CISQ recommends a default maximum of 7 parameters/arguments.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-13		

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-1065: Runtime Resource Management Control Element in a Component Built to Run on Application Servers

Weakness ID : 1065

Structure : Simple

Abstraction : Base

Description

The product uses deployed components from application servers, but it also uses low-level functions/methods for management of resources, instead of the API provided by the application server.


Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-5		

References

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

CWE-1066: Missing Serialization Control Element

Weakness ID : 1066

Structure : Simple

Abstraction : Base

Description

The product contains a serializable data element that does not have an associated serialization method.

Extended Description

This issue can prevent the product from running reliably, e.g. by triggering an exception. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

As examples, the serializable nature of a data element comes from a serializable `SerializableAttribute` attribute in .NET and the inheritance from the `java.io.Serializable` interface in Java.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This `MemberOf` relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-2		

References

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

CWE-1067: Excessive Execution of Sequential Searches of Data Resource

Weakness ID : 1067

Structure : Simple

Abstraction : Base

Description

The product contains a data query against an SQL table or view that is configured in a way that does not utilize an index and may cause sequential searches to be performed.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1176	Inefficient CPU Computation	1980

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-5		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1068: Inconsistency Between Implementation and Documented Design

Weakness ID : 1068

Structure : Simple

Abstraction : Base

Description

The implementation of the product is not consistent with the design as described within the relevant documentation.

Extended Description

This issue makes it more difficult to maintain the product due to inconsistencies, 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	P	710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2501

Weakness Ordinalities

Indirect :

Applicable Platforms

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

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1069: Empty Exception Block

Weakness ID : 1069

Structure : Simple

Abstraction : Variant

Description

An invokable code block contains an exception handling block that does not contain any code, i.e. is empty.

Extended Description

When an exception handling block (such as a Catch and Finally block) is used, but that block is empty, this can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1071	Empty Code Block	1919

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

Potential Mitigations

Phase: Implementation

For every exception block add code that handles the specific exception in the way intended by the application.

Demonstrative Examples

Example 1:

In the following Java example, the code catches an ArithmeticException.

Example Language: Java

(Bad)

```
public class Main {
    public static void main(String[] args) {
        int a = 1;
        int b = 0;
        int c = 0;
        try {
            c = a / b;
        } catch(ArithmeticException ae) {
        }
    }
}
```

Since the exception block is empty, no action is taken.

In the code below the exception has been logged and the bad execution has been handled in the desired way allowing the program to continue in an expected way.

Example Language: Java

(Good)

```
public class Main {
    public static void main(String[] args) {
        int a = 1;
        int b = 0;
        int c = 0;
```


```

try {
    c = a / b;
} catch(ArithmeticException ae) {
    log.error("Divided by zero detected, setting to -1.");
    c = -1;
}
}
}

```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-1		

References

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

CWE-1070: Serializable Data Element Containing non-Serializable Item Elements

Weakness ID : 1070

Structure : Simple

Abstraction : Base

Description

The product contains a serializable, storable data element such as a field or member, but the data element contains member elements that are not serializable.

Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

As examples, the serializable nature of a data element comes from a serializable SerializableAttribute attribute in .NET and the inheritance from the java.io.Serializable interface in Java.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
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
OMG ASCRM	ASCRM-RLB-3		

References

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

CWE-1071: Empty Code Block

Weakness ID : 1071

Structure : Simple

Abstraction : Base

Description

The source code contains a block that does not contain any code, i.e., the block is empty.




Extended Description

Empty code blocks can occur in the bodies of conditionals, function or method definitions, exception handlers, etc. While an empty code block might be intentional, it might also indicate incomplete implementation, accidental code deletion, unexpected macro expansion, etc. For some programming languages and constructs, an empty block might be allowed by the syntax, but the lack of any behavior within the block might violate a convention or API in such a way that it is an error.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1164	Irrelevant Code	1976
ParentOf		585	Empty Synchronized Block	1327
ParentOf		1069	Empty Exception Block	1916

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

Demonstrative Examples

Example 1:

In the following Java example, the code catches an `ArithmeticException`.

Example Language: Java

(Bad)

```
public class Main {
    public static void main(String[] args) {
        int a = 1;
        int b = 0;
        int c = 0;
        try {
            c = a / b;
        } catch (ArithmeticException ae) {
        }
    }
}
```

Since the exception block is empty, no action is taken.

In the code below the exception has been logged and the bad execution has been handled in the desired way allowing the program to continue in an expected way.

Example Language: Java

(Good)

```
public class Main {
    public static void main(String[] args) {
        int a = 1;
        int b = 0;
        int c = 0;
        try {
            c = a / b;
        } catch (ArithmeticException ae) {
            log.error("Divided by zero detected, setting to -1.");
            c = -1;
        }
    }
}
```

Example 2:

The following code attempts to synchronize on an object, but does not execute anything in the synchronized block. This does not actually accomplish anything and may be a sign that a programmer is wrestling with synchronization but has not yet achieved the result they intend.

Example Language: Java

(Bad)

```
synchronized(this) { }
```

Instead, in a correct usage, the synchronized statement should contain procedures that access or modify data that is exposed to multiple threads. For example, consider a scenario in which several threads are accessing student records at the same time. The method which sets the student ID to a new value will need to make sure that nobody else is accessing this data at the same time and will require synchronization.

Example Language:

(Good)

```
public void setID(int ID){
    synchronized(this){
        this.ID = ID;
    }
}
```

MemberOf Relationships

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

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

CWE-1072: Data Resource Access without Use of Connection Pooling

Weakness ID : 1072

Structure : Simple

Abstraction : Base

Description

The product accesses a data resource through a database without using a connection pooling capability.

Extended Description

This issue can make the product perform more slowly, as connection pools allow connections to be reused without the overhead and time consumption of opening and closing a new connection. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-13		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

[REF-974]Wikipedia. "Connection pool". < https://en.wikipedia.org/wiki/Connection_pool >.

CWE-1073: Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses

Weakness ID : 1073

Structure : Simple

Abstraction : Base

Description

The product contains a client with a function or method that contains a large number of data accesses/queries that are sent through a data manager, i.e., does not use efficient database capabilities.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large number of data accesses/queries" may vary for each product or developer, CISQ recommends a default maximum of 2 data accesses per function/method.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-10		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1074: Class with Excessively Deep Inheritance

Weakness ID : 1074

Structure : Simple

Abstraction : Base

Description

A class has an inheritance level that is too high, i.e., it has a large number of parent classes.

Extended Description

This issue makes it more difficult to understand and 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.


While the interpretation of "large number of parent classes" may vary for each product or developer, CISQ recommends a default maximum of 7 parent classes.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf		1093	Excessively Complex Data Representation	1942

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-17		

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-1075: Unconditional Control Flow Transfer outside of Switch Block

Weakness ID : 1075

Structure : Simple

Abstraction : Base

Description

The product performs unconditional control transfer (such as a "goto") in code outside of a branching structure such as a switch block.

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		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities


Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-1		

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-1076: Insufficient Adherence to Expected Conventions

Weakness ID : 1076

Structure : Simple

Abstraction : Class

Description

The product's architecture, source code, design, documentation, or other artifact does not follow required conventions.














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		710	Improper Adherence to Coding Standards	1558
ParentOf		586	Explicit Call to Finalize()	1329
ParentOf		594	J2EE Framework: Saving Unserializable Objects to Disk	1341
ParentOf		1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	1889
ParentOf		1070	Serializable Data Element Containing non-Serializable Item Elements	1918
ParentOf		1078	Inappropriate Source Code Style or Formatting	1927
ParentOf		1079	Parent Class without Virtual Destructor Method	1929
ParentOf		1082	Class Instance Self Destruction Control Element	1931
ParentOf		1087	Class with Virtual Method without a Virtual Destructor	1936
ParentOf		1091	Use of Object without Invoking Destructor Method	1940
ParentOf		1097	Persistent Storable Data Element without Associated Comparison Control Element	1946
ParentOf		1098	Data Element containing Pointer Item without Proper Copy Control Element	1947
ParentOf		1108	Excessive Reliance on Global Variables	1957

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-1077: Floating Point Comparison with Incorrect Operator

Weakness ID : 1077

Structure : Simple

Abstraction : Variant

Description

The code performs a comparison such as an equality test between two float (floating point) values, but it uses comparison operators that do not account for the possibility of loss of precision.

Extended Description

Numeric calculation using floating point values can generate imprecise results because of rounding errors. As a result, two different calculations might generate numbers that are

mathematically equal, but have slightly different bit representations that do not translate to the same mathematically-equal values. As a result, an equality test or other comparison might produce unexpected results.

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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]	697	Incorrect Comparison	1538

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1397	Comprehensive Categorization: Comparison	1400	2544

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-9		

References

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

[REF-975]Bruce Dawson. "Comparing Floating Point Numbers, 2012 Edition". 2012 February 5. < <https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/> >.

CWE-1078: Inappropriate Source Code Style or Formatting

Weakness ID : 1078

Structure : Simple

Abstraction : Class














Description

The source code does not follow desired style or formatting for indentation, white space, comments, 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		1076	Insufficient Adherence to Expected Conventions	1925
ParentOf		546	Suspicious Comment	1266
ParentOf		547	Use of Hard-coded, Security-relevant Constants	1267
ParentOf		1085	Invokable Control Element with Excessive Volume of Commented-out Code	1934
ParentOf		1099	Inconsistent Naming Conventions for Identifiers	1948
ParentOf		1106	Insufficient Use of Symbolic Constants	1955
ParentOf		1107	Insufficient Isolation of Symbolic Constant Definitions	1956
ParentOf		1109	Use of Same Variable for Multiple Purposes	1958
ParentOf		1113	Inappropriate Comment Style	1962
ParentOf		1114	Inappropriate Whitespace Style	1963
ParentOf		1115	Source Code Element without Standard Prologue	1963
ParentOf		1116	Inaccurate Comments	1964
ParentOf		1117	Callable with Insufficient Behavioral Summary	1966

Weakness Ordinalities

Indirect :

Demonstrative Examples

Example 1:

The usage of symbolic names instead of hard-coded constants is preferred.

The following is an example of using a hard-coded constant instead of a symbolic name.

Example Language: C

(Bad)

```
char buffer[1024];
...
fgets(buffer, 1024, stdin);
```

If the buffer value needs to be changed, then it has to be altered in more than one place. If the developer forgets or does not find all occurrences, in this example it could lead to a buffer overflow.

Example Language: C

(Good)

```
enum { MAX_BUFFER_SIZE = 1024 };
...
char buffer[MAX_BUFFER_SIZE];
...
fgets(buffer, MAX_BUFFER_SIZE, stdin);
```

In this example the developer will only need to change one value and all references to the buffer size are updated, as a symbolic name is used instead of a hard-coded constant.

MemberOf Relationships

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

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

CWE-1079: Parent Class without Virtual Destructor Method

Weakness ID : 1079

Structure : Simple

Abstraction : Base

Description

A parent class contains one or more child classes, but the parent class does not have a virtual destructor method.


Extended Description

This issue can prevent the product from running reliably due to undefined or unexpected behaviors. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-16		

References

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

CWE-1080: Source Code File with Excessive Number of Lines of Code

Weakness ID : 1080

Structure : Simple

Abstraction : Base

Description

A source code file has too many lines of code.

Extended Description

This issue makes it more difficult to understand and/or 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.

While the interpretation of "too many lines of code" may vary for each product or developer, CISQ recommends a default threshold value of 1000.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-8		

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-1082: Class Instance Self Destruction Control Element

Weakness ID : 1082

Structure : Simple

Abstraction : Base

Description

The code contains a class instance that calls the method or function to delete or destroy itself.

Extended Description

For example, in C++, "delete this" will cause the object to delete itself.

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
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
OMG ASCRM	ASCRM-RLB-7		

References

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

[REF-976]Standard C++ Foundation. "Memory Management". < <https://isocpp.org/wiki/faq/freestore-mgmt#delete-this> >.

CWE-1083: Data Access from Outside Expected Data Manager Component

Weakness ID : 1083

Structure : Simple

Abstraction : Base

Description

The product is intended to manage data access through a particular data manager component such as a relational or non-SQL database, but it contains code that performs data access operations without using that component.

Extended Description


When the product has a data access component, the design may be intended to handle all data access operations through that component. If a data access operation is performed outside of that component, then this may indicate a violation of the intended design.

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
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
OMG ASCRM	ASCRM-RLB-10		

References

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

CWE-1084: Invokable Control Element with Excessive File or Data Access Operations

Weakness ID : 1084

Structure : Simple

Abstraction : Base

Description

A function or method contains too many operations that utilize a data manager or file resource.

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.

While the interpretation of "too many operations" may vary for each product or developer, CISQ recommends a default maximum of 7 operations for the same data manager or file.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-14		

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-1085: Invokable Control Element with Excessive Volume of Commented-out Code

Weakness ID : 1085

Structure : Simple

Abstraction : Base

Description

A function, method, procedure, etc. contains an excessive amount of code that has been commented out within its body.

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.

While the interpretation of "excessive volume" may vary for each product or developer, CISQ recommends a default threshold of 2% of commented 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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-6		

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-1086: Class with Excessive Number of Child Classes

Weakness ID : 1086

Structure : Simple

Abstraction : Base

Description

A class contains an unnecessarily large number of children.

Extended Description

This issue makes it more difficult to understand and maintain the software, 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.

While the interpretation of "large number of children" may vary for each product or developer, CISQ recommends a default maximum of 10 child classes.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1093	Excessively Complex Data Representation	1942

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-18		

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-1087: Class with Virtual Method without a Virtual Destructor**Weakness ID** : 1087**Structure** : Simple**Abstraction** : Base**Description**

A class contains a virtual method, but the method does not have an associated virtual destructor.


Extended Description

This issue can prevent the product from running reliably, e.g. due to undefined behavior. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-15		

References

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

CWE-1088: Synchronous Access of Remote Resource without Timeout

Weakness ID : 1088

Structure : Simple

Abstraction : Base

Description

The code has a synchronous call to a remote resource, but there is no timeout for the call, or the timeout is set to infinite.

Extended Description

This issue can prevent the product from running reliably, since an outage for the remote resource can cause the product to hang. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	821	Incorrect Synchronization	1731

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-19		

References

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

CWE-1089: Large Data Table with Excessive Number of Indices

Weakness ID : 1089

Structure : Simple

Abstraction : Base

Description

The product uses a large data table that contains an excessively large number of indices.

Extended Description


This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "excessively large number of indices" may vary for each product or developer, CISQ recommends a default threshold of 1000000 rows for a "large" table and a default threshold of 3 indices.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-6		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1090: Method Containing Access of a Member Element from Another Class

Weakness ID : 1090

Structure : Simple

Abstraction : Base

Description

A method for a class performs an operation that directly accesses a member element from another class.


Extended Description

This issue suggests poor encapsulation and makes it more difficult to understand and 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

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-16		

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-1091: Use of Object without Invoking Destructor Method

Weakness ID : 1091

Structure : Simple

Abstraction : Base

Description

The product contains a method that accesses an object but does not later invoke the element's associated finalize/destructor method.



Extended Description

This issue can make the product perform more slowly by retaining memory and/or other resources longer than necessary. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

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		1076	Insufficient Adherence to Expected Conventions	1925
ChildOf		772	Missing Release of Resource after Effective Lifetime	1632

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-15		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1092: Use of Same Invokable Control Element in Multiple Architectural Layers

Weakness ID : 1092

Structure : Simple

Abstraction : Base

Description

The product uses the same control element across multiple architectural layers.

Extended Description

This issue makes it more difficult to understand and 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	P	710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-10		

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-1093: Excessively Complex Data Representation

Weakness ID : 1093

Structure : Simple

Abstraction : Class

Description

The product uses an unnecessarily complex internal representation for its data structures or interrelationships between those structures.



Extended Description




This issue makes it more difficult to understand or 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		710	Improper Adherence to Coding Standards	1558
ParentOf		1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements	1887

Nature	Type	ID	Name	Page
ParentOf		1055	Multiple Inheritance from Concrete Classes	1900
ParentOf		1074	Class with Excessively Deep Inheritance	1923
ParentOf		1086	Class with Excessive Number of Child Classes	1935

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-1094: Excessive Index Range Scan for a Data Resource

Weakness ID : 1094

Structure : Simple

Abstraction : Base

Description

The product contains an index range scan for a large data table, but the scan can cover a large number of rows.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "excessive index range" may vary for each product or developer, CISQ recommends a threshold of 1000000 table rows and a threshold of 10 for the index range.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2464
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2507
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-7		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1095: Loop Condition Value Update within the Loop

Weakness ID : 1095

Structure : Simple

Abstraction : Base

Description

The product uses a loop with a control flow condition based on a value that is updated within the body of the loop.

Extended Description

This issue makes it more difficult to understand and/or 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		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities


Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-5		

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-1096: Singleton Class Instance Creation without Proper Locking or Synchronization

Weakness ID : 1096

Structure : Simple

Abstraction : Variant

Description

The product implements a Singleton design pattern but does not use appropriate locking or other synchronization mechanism to ensure that the singleton class is only instantiated once.

Extended Description

This issue can prevent the product from running reliably, e.g. by making the instantiation process non-thread-safe and introducing deadlock (CWE-833) or livelock conditions. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		820	Missing Synchronization	1729

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-12		

References

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

CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element

Weakness ID : 1097

Structure : Simple

Abstraction : Base

Description

The product uses a storable data element that does not have all of the associated functions or methods that are necessary to support comparison.

Extended Description

For example, with Java, a class that is made persistent requires both hashCode() and equals() methods to be defined.

This issue can prevent the product from running reliably, due to incorrect or unexpected comparison results. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
ChildOf		595	Comparison of Object References Instead of Object Contents	1342

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-4		

References

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

CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element

Weakness ID : 1098
Structure : Simple
Abstraction : Base

Description

The code contains a data element with a pointer that does not have an associated copy or constructor method.


Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1129	CISQ Quality Measures (2016) - Reliability	1128	2461
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
OMG ASCRM	ASCRM-RLB-6		

References

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

CWE-1099: Inconsistent Naming Conventions for Identifiers

Weakness ID : 1099

Structure : Simple

Abstraction : Base

Description

The product's code, documentation, or other artifacts do not consistently use the same naming conventions for variables, callables, groups of related callables, I/O capabilities, data types, file names, or similar types of elements.

Extended Description

This issue makes it more difficult to understand and/or maintain the product due to inconsistencies, 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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1100: Insufficient Isolation of System-Dependent Functions

Weakness ID : 1100

Structure : Simple

Abstraction : Base

Description

The product or code does not isolate system-dependent functionality into separate standalone modules.

Extended Description


This issue makes it more difficult to maintain and/or port 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

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1101: Reliance on Runtime Component in Generated Code

Weakness ID : 1101
Structure : Simple
Abstraction : Base

Description

The product uses automatically-generated code that cannot be executed without a specific runtime support component.


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, 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		710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1102: Reliance on Machine-Dependent Data Representation

Weakness ID : 1102

Structure : Simple

Abstraction : Base

Description

The code uses a data representation that relies on low-level data representation or constructs that may vary across different processors, physical machines, OSes, or other physical components.

Extended Description


This issue makes it more difficult to maintain and/or port 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		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591

Nature	Type	ID	Name	Page
PeerOf		1105	Insufficient Encapsulation of Machine-Dependent Functionality	1954

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1103: Use of Platform-Dependent Third Party Components

Weakness ID : 1103

Structure : Simple

Abstraction : Base

Description

The product relies on third-party components that do not provide equivalent functionality across all desirable platforms.

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		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1104: Use of Unmaintained Third Party Components

Weakness ID : 1104

Structure : Simple

Abstraction : Base

Description

The product relies on third-party components that are not actively supported or maintained by the original developer or a trusted proxy for the original developer.

Extended Description


Reliance on components that are no longer maintained can make it difficult or impossible to fix significant bugs, vulnerabilities, or quality issues. In effect, unmaintained code can become obsolete.

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		1357	Reliance on Insufficiently Trustworthy Component	2266

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Applicable Platforms

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


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

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1352	OWASP Top Ten 2021 Category A06:2021 - Vulnerable and Outdated Components	1344	2515
MemberOf		1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

References

[REF-1212]"A06:2021 - Vulnerable and Outdated Components". 2021 September 4. OWASP. <
https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ >.

CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality

Weakness ID : 1105

Structure : Simple

Abstraction : Base

Description

The product or code uses machine-dependent functionality, but it does not sufficiently encapsulate or isolate this functionality from the rest of the code.


Extended Description




This issue makes it more difficult to port or 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

Nature	Type	ID	Name	Page
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591
ParentOf		188	Reliance on Data/Memory Layout	476
PeerOf		1102	Reliance on Machine-Dependent Data Representation	1951

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Demonstrative Examples

Example 1:

In this example function, the memory address of variable b is derived by adding 1 to the address of variable a. This derived address is then used to assign the value 0 to b.

Example Language: C

(Bad)

```
void example() {  
    char a;  
    char b;  
    *(&a + 1) = 0;  
}
```

Here, b may not be one byte past a. It may be one byte in front of a. Or, they may have three bytes between them because they are aligned on 32-bit boundaries.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1106: Insufficient Use of Symbolic Constants

Weakness ID : 1106

Structure : Simple

Abstraction : Base

Description

The source code uses literal constants that may need to change or evolve over time, instead of using symbolic constants.

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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1107: Insufficient Isolation of Symbolic Constant Definitions

Weakness ID : 1107

Structure : Simple

Abstraction : Base

Description

The source code uses symbolic constants, but it does not sufficiently place the definitions of these constants into a more centralized or isolated location.

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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1108: Excessive Reliance on Global Variables

Weakness ID : 1108

Structure : Simple

Abstraction : Base

Description

The code is structured in a way that relies too much on using or setting global variables throughout various points in the code, instead of preserving the associated information in a narrower, more local context.

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		1076	Insufficient Adherence to Expected Conventions	1925

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1109: Use of Same Variable for Multiple Purposes

Weakness ID : 1109

Structure : Simple

Abstraction : Base

Description

The code contains a callable, block, or other code element in which the same variable is used to control more than one unique task or store more than one instance of data.

Extended Description


Use of the same variable for multiple purposes can make it more difficult for a person to read or understand the code, potentially hiding other quality issues.

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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1110: Incomplete Design Documentation

Weakness ID : 1110

Structure : Simple

Abstraction : Base

Description

The product's design documentation does not adequately describe control flow, data flow, system initialization, relationships between tasks, components, rationales, or other important aspects of the design.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1904

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

Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2501

Weakness Ordinalities

Indirect :




Applicable Platforms

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

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

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		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2532
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1111: Incomplete I/O Documentation

Weakness ID : 1111

Structure : Simple

Abstraction : Base

Description

The product's documentation does not adequately define inputs, outputs, or system/software interfaces.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1904

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


Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2501

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2532
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENTS >.2023-04-07.

CWE-1112: Incomplete Documentation of Program Execution

Weakness ID : 1112

Structure : Simple

Abstraction : Base

Description

The document does not fully define all mechanisms that are used to control or influence how product-specific programs are executed.

Extended Description

This includes environmental variables, configuration files, registry keys, command-line switches or options, or system settings.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1904

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

Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2501

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1113: Inappropriate Comment Style

Weakness ID : 1113

Structure : Simple

Abstraction : Base

Description

The source code uses comment styles or formats that are inconsistent or do not follow expected standards for the product.

Extended Description

This issue makes it more difficult to maintain the product due to insufficient legibility, 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	G	1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1114: Inappropriate Whitespace Style

Weakness ID : 1114

Structure : Simple

Abstraction : Base

Description

The source code contains whitespace that is inconsistent across the code or does not follow expected standards for the product.

Extended Description

This issue makes it more difficult to understand and 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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1115: Source Code Element without Standard Prologue

Weakness ID : 1115

Structure : Simple
Abstraction : Base

Description

The source code contains elements such as source files that do not consistently provide a prologue or header that has been standardized for the project.

Extended Description

The lack of a prologue can make it more difficult to accurately and quickly understand the associated code. Standard prologues or headers may contain information such as module name, version number, author, date, purpose, function, assumptions, limitations, accuracy considerations, etc.

This issue makes it more difficult to maintain the product due to insufficient analyzability, 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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1116: Inaccurate Comments

Weakness ID : 1116
Structure : Simple
Abstraction : Base

Description

The source code contains comments that do not accurately describe or explain aspects of the portion of the code with which the comment is associated.

Extended Description

When a comment does not accurately reflect the associated code elements, this can introduce confusion to a reviewer (due to inconsistencies) or make it more difficult and less efficient to validate that the code is implementing the intended behavior correctly.

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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Potential Mitigations

Phase: Implementation

Verify that each comment accurately reflects what is intended to happen during execution of the code.

Demonstrative Examples

Example 1:

In the following Java example the code performs a calculation to determine how much medicine to administer. A comment is provided to give insight into what the calculation should be doing. Unfortunately the comment does not match the actual code and thus leaves the reader to wonder which is correct.

Example Language: Java

(Bad)

```
public class Main {
    public static void main(String[] args) {
        int pt_weight = 83;
        int mg_per_kg = 3;
        int daily_dose = 0;
        // Add the patient weight and Mg/Kg to calculate the correct daily dose
    }
}
```

```
    daily_dose = pt_weight * mg_per_kg;
    return dosage;
  }
}
```

In the correction below, the code functionality has been verified, and the comment has been corrected to reflect the proper calculation.

Example Language: Java (Good)

```
public class Main {
    public static void main(String[] args) {
        int pt_weight = 83;
        int mg_per_kg = 3;
        int daily_dose = 0;
        // Multiply the patient weight and Mg/Kg to calculate the correct daily dose
        daily_dose = pt_weight * mg_per_kg;
        return dosage;
    }
}
```

Note that in real-world code, these values should be validated to disallow negative numbers, prevent integer overflow, etc.

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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1117: Callable with Insufficient Behavioral Summary

Weakness ID : 1117

Structure : Simple

Abstraction : Base

Description

The code contains a function or method whose signature and/or associated inline documentation does not sufficiently describe the callable's inputs, outputs, side effects, assumptions, or return codes.

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, 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		1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1118: Insufficient Documentation of Error Handling Techniques

Weakness ID : 1118

Structure : Simple

Abstraction : Base

Description

The documentation does not sufficiently describe the techniques that are used for error handling, exception processing, or similar mechanisms.

Extended Description

Documentation may need to cover error handling techniques at multiple layers, such as module, executable, compilable code unit, or callable.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1904

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

Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2501

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1119: Excessive Use of Unconditional Branching

Weakness ID : 1119

Structure : Simple

Abstraction : Base

Description

The code uses too many unconditional branches (such as "goto").

Extended Description

This issue makes it more difficult to understand and/or 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		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1120: Excessive Code Complexity

Weakness ID : 1120

Structure : Simple

Abstraction : Class

Description

The code is too complex, as calculated using a well-defined, quantitative measure.

Extended Description

This issue makes it more difficult to understand and/or 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.

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	B	1047	Modules with Circular Dependencies	1891
ParentOf	B	1056	Invokable Control Element with Variadic Parameters	1901
ParentOf	B	1060	Excessive Number of Inefficient Server-Side Data Accesses	1906
ParentOf	B	1064	Invokable Control Element with Signature Containing an Excessive Number of Parameters	1911
ParentOf	B	1075	Unconditional Control Flow Transfer outside of Switch Block	1924
ParentOf	B	1080	Source Code File with Excessive Number of Lines of Code	1930
ParentOf	B	1095	Loop Condition Value Update within the Loop	1944
ParentOf	B	1119	Excessive Use of Unconditional Branching	1968

Nature	Type	ID	Name	Page
ParentOf	B	1121	Excessive McCabe Cyclomatic Complexity	1970
ParentOf	B	1122	Excessive Halstead Complexity	1971
ParentOf	B	1123	Excessive Use of Self-Modifying Code	1972
ParentOf	B	1124	Excessively Deep Nesting	1973
ParentOf	B	1125	Excessive Attack Surface	1974

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-1121: Excessive McCabe Cyclomatic Complexity

Weakness ID : 1121

Structure : Simple

Abstraction : Base

Description

The code contains McCabe cyclomatic complexity that exceeds a desirable maximum.

Extended Description

This issue makes it more difficult to understand and/or 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	G	1120	Excessive Code Complexity	1969

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



Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-11		

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

[REF-964]Wikipedia. "Cyclomatic Complexity". 2018 April 3. < https://en.wikipedia.org/wiki/Cyclomatic_complexity >.

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < <https://www.omg.org/spec/ASCMM/> >.2023-04-07.

CWE-1122: Excessive Halstead Complexity

Weakness ID : 1122

Structure : Simple

Abstraction : Base

Description

The code is structured in a way that a Halstead complexity measure exceeds a desirable maximum.

Extended Description

A variety of Halstead complexity measures exist, such as program vocabulary size or volume.

This issue makes it more difficult to understand and/or 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		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

[REF-965]Wikipedia. "Halstead complexity measures". 2017 November 2. < https://en.wikipedia.org/wiki/Halstead_complexity_measures >.

CWE-1123: Excessive Use of Self-Modifying Code

Weakness ID : 1123

Structure : Simple

Abstraction : Base

Description

The product uses too much self-modifying code.

Extended Description

This issue makes it more difficult to understand or 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, 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		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

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

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1124: Excessively Deep Nesting

Weakness ID : 1124

Structure : Simple

Abstraction : Base

Description

The code contains a callable or other code grouping in which the nesting / branching is too deep.

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		1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1125: Excessive Attack Surface

Weakness ID : 1125

Structure : Simple

Abstraction : Base

Description

The product has an attack surface whose quantitative measurement exceeds a desirable maximum.

Extended Description

Originating from software security, an "attack surface" measure typically reflects the number of input points and output points that can be utilized by an untrusted party, i.e. a potential attacker. A larger attack surface provides more places to attack, and more opportunities for developers to introduce weaknesses. In some cases, this measure may reflect other aspects of quality besides security; e.g., a product with many inputs and outputs may require a large number of tests in order to improve code coverage.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	1120	Excessive Code Complexity	1969

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

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2502

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

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

References

[REF-966]Pratyusa Manadhata. "An Attack Surface Metric". 2008 November. < <http://reports-archive.adm.cs.cmu.edu/anon/2008/CMU-CS-08-152.pdf> >.

[REF-967]Pratyusa Manadhata and Jeannette M. Wing. "Measuring a System's Attack Surface". 2004. < <http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/ManadhataWing04.pdf> >.

CWE-1126: Declaration of Variable with Unnecessarily Wide Scope

Weakness ID : 1126

Structure : Simple

Abstraction : Base

Description

The source code declares a variable in one scope, but the variable is only used within a narrower scope.


Extended Description

This issue makes it more difficult to understand and/or 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		710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-1127: Compilation with Insufficient Warnings or Errors

Weakness ID : 1127

Structure : Simple

Abstraction : Base

Description

The code is compiled without sufficient warnings enabled, which may prevent the detection of subtle bugs or quality issues.

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	P	710	Improper Adherence to Coding Standards	1558

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-1164: Irrelevant Code

Weakness ID : 1164

Structure : Simple

Abstraction : Class

Description

1976

The product contains code that is not essential for execution, i.e. makes no state changes and has no side effects that alter data or control flow, such that removal of the code would have no impact to functionality or correctness.

Extended Description

Irrelevant code could include dead code, initialization that is not used, empty blocks, code that could be entirely removed due to optimization, etc.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	ⓧ	107	Struts: Unused Validation Form	265
ParentOf	ⓧ	110	Struts: Validator Without Form Field	270
ParentOf	ⓑ	561	Dead Code	1283
ParentOf	ⓑ	563	Assignment to Variable without Use	1289
ParentOf	ⓑ	1071	Empty Code Block	1919

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	
Other	Reduce Performance	

Demonstrative Examples

Example 1:

The condition for the second if statement is impossible to satisfy. It requires that the variables be non-null. However, on the only path where s can be assigned a non-null value, there is a return statement.

Example Language: C++

(Bad)

```
String s = null;
if (b) {
    s = "Yes";
    return;
}
if (s != null) {
    Dead();
}
```

Example 2:

The following code excerpt assigns to the variable r and then overwrites the value without using it.

Example Language: C

(Bad)

```
r = getName();
r = getNewBuffer(buf);
```

Observed Examples

Reference	Description
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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-1173: Improper Use of Validation Framework

Weakness ID : 1173

Structure : Simple

Abstraction : Base

Description

The product does not use, or incorrectly uses, an input validation framework that is provided by the source language or an independent library.









Extended Description

Many modern coding languages provide developers with input validation frameworks to make the task of input validation easier and less error-prone. These frameworks will automatically check all input against specified criteria and direct execution to error handlers when invalid input is received. The improper use (i.e., an incorrect implementation or missing altogether) of these frameworks is not directly exploitable, but can lead to an exploitable condition if proper input validation is not performed later in the product. Not using provided input validation frameworks can also hurt the maintainability of code as future developers may not recognize the downstream input validation being used in the place of the validation framework.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
ParentOf		102	Struts: Duplicate Validation Forms	252
ParentOf		105	Struts: Form Field Without Validator	259
ParentOf		106	Struts: Plug-in Framework not in Use	262
ParentOf		108	Struts: Unvalidated Action Form	267
ParentOf		109	Struts: Validator Turned Off	269
ParentOf		554	ASP.NET Misconfiguration: Not Using Input Validation Framework	1278
ParentOf		1174	ASP.NET Misconfiguration: Improper Model Validation	1979

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2499

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State <i>Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.</i>	

Detection Methods

Automated Static Analysis

Some instances of improper input validation can be detected using automated static analysis. A static analysis tool might allow the user to specify which application-specific methods or functions perform input validation; the tool might also have built-in knowledge of validation frameworks such as Struts. The tool may then suppress or de-prioritize any associated warnings. This allows the analyst to focus on areas of the software in which input validation does not appear to be present. Except in the cases described in the previous paragraph, automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.




Potential Mitigations

Phase: Implementation

Properly use provided input validation frameworks.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2552

CWE-1174: ASP.NET Misconfiguration: Improper Model Validation

Weakness ID : 1174

Structure : Simple

Abstraction : Variant

Description

The ASP.NET application does not use, or incorrectly uses, the model validation framework.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1173	Improper Use of Validation Framework	1978

Weakness Ordinalities

Indirect :

Applicable Platforms




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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State <i>Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.</i>	

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		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2552

CWE-1176: Inefficient CPU Computation

Weakness ID : 1176

Structure : Simple

Abstraction : Class

Description

The product performs CPU computations using algorithms that are not as efficient as they could be for the needs of the developer, i.e., the computations can be optimized further.







Extended Description

This issue can make the product perform more slowly, possibly in ways that are noticeable to the users. If an attacker can influence the amount of computation that must be performed, e.g. by triggering worst-case complexity, then this performance problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	993
ParentOf		1042	Static Member Data Element outside of a Singleton Class Element	1886
ParentOf		1046	Creation of Immutable Text Using String Concatenation	1890
ParentOf		1049	Excessive Data Query Operations in a Large Data Table	1894
ParentOf		1063	Creation of Class Instance within a Static Code Block	1910
ParentOf		1067	Excessive Execution of Sequential Searches of Data Resource	1914

Weakness Ordinalities

Indirect :

Primary :

Common Consequences


Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	
Other	Reduce Performance	

Observed Examples

Reference	Description
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		Page
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

References

[REF-1008]Wikipedia. "Computational complexity theory". < https://en.wikipedia.org/wiki/Computational_complexity_theory >.

CWE-1177: Use of Prohibited Code

Weakness ID : 1177

Structure : Simple

Abstraction : Class

Description

The product uses a function, library, or third party component that has been explicitly prohibited, whether by the developer or the customer.

Extended Description

The developer - or customers - may wish to restrict or eliminate use of a function, library, or third party component for any number of reasons, including real or suspected vulnerabilities; difficulty to

use securely; export controls or license requirements; obsolete or poorly-maintained code; internal code being scheduled for deprecation; etc.

To reduce risk of vulnerabilities, the developer might maintain a list of "banned" functions that programmers must avoid using because the functions are difficult or impossible to use securely. This issue can also make the product more costly and difficult to maintain.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	Ⓔ	242	Use of Inherently Dangerous Function	593
ParentOf	Ⓔ	676	Use of Potentially Dangerous Function	1498

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Demonstrative Examples

Example 1:

The code below calls the gets() function to read in data from the command line.

Example Language: C

(Bad)

```
char buff[24];
printf("Please enter your name and press <Enter>\n");
gets(buff);
...
}
```

However, gets() is inherently unsafe, because it copies all input from STDIN to the buffer without checking size. This allows the user to provide a string that is larger than the buffer size, resulting in an overflow condition.

Example 2:

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 buff[24];
    strcpy(buff, 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-2007-4004	FTP client uses inherently insecure gets() function and is setuid root on some systems, allowing buffer overflow https://www.cve.org/CVERecord?id=CVE-2007-4004

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

References

[REF-1009]Tim Rains. "Microsoft's Free Security Tools - banned.h". 2012 August 0. < <https://www.microsoft.com/en-us/security/blog/2012/08/30/microsofts-free-security-tools-banned-h/> >.2023-04-07.

[REF-1010]Michael Howard. "Microsoft's Free Security Tools - banned.h". 2011 June. < <https://www.microsoft.com/en-us/security/blog/2012/08/30/microsofts-free-security-tools-banned-h/> >.2023-04-07.

CWE-1188: Initialization of a Resource with an Insecure Default

Weakness ID : 1188

Structure : Simple

Abstraction : Base

Description

The product initializes or sets a resource with a default that is intended to be changed by the administrator, but the default is not secure.



Extended Description

Developers often choose default values that leave the product as open and easy to use as possible out-of-the-box, under the assumption that the administrator can (or should) change the default value. However, this ease-of-use comes at a cost when the default is insecure and the administrator does not change it.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1419	Incorrect Initialization of Resource	2292
ParentOf		453	Insecure Default Variable Initialization	1091

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1465

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2345
MemberOf		452	Initialization and Cleanup Errors	2348

Weakness Ordinalities

Primary :

Demonstrative Examples

Example 1:

This code attempts to login a user using credentials from a POST request:

Example Language: PHP

(Bad)

```
// $user and $pass automatically set from POST request
if (login_user($user,$pass)) {
    $authorized = true;
}
...
if ($authorized) {
    generatePage();
}
```

Because the \$authorized variable is never initialized, PHP will automatically set \$authorized to any value included in the POST request if register_globals is enabled. An attacker can send a POST request with an unexpected third value 'authorized' set to 'true' and gain authorized status without supplying valid credentials.

Here is a fixed version:

Example Language: PHP

(Good)

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$authorized = false;
if (login_user($user,$pass)) {
    $authorized = true;
}
...
```

This code avoids the issue by initializing the \$authorized variable to false and explicitly retrieving the login credentials from the \$_POST variable. Regardless, register_globals should never be enabled and is disabled by default in current versions of PHP.

Observed Examples

Reference	Description
CVE-2022-36349	insecure default variable initialization in BIOS firmware for a hardware board allows DoS https://www.cve.org/CVERecord?id=CVE-2022-36349
CVE-2022-42467	A generic database browser interface has a default mode that exposes a web server to the network, allowing queries to the database. https://www.cve.org/CVERecord?id=CVE-2022-42467

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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

Maintenance

This entry improves organization of concepts under initialization. The typical CWE model is to cover "Missing" and "Incorrect" behaviors. Arguably, this entry could be named as "Incorrect" instead of "Insecure." This might be changed in the near future.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
665	Exploitation of Thunderbolt Protection Flaws

CWE-1189: Improper Isolation of Shared Resources on System-on-a-Chip (SoC)

Weakness ID : 1189

Structure : Simple

Abstraction : Base

Description

The System-On-a-Chip (SoC) does not properly isolate shared resources between trusted and untrusted agents.





Extended Description

A System-On-a-Chip (SoC) has a lot of functionality, but it may have a limited number of pins or pads. A pin can only perform one function at a time. However, it can be configured to perform multiple different functions. This technique is called pin multiplexing. Similarly, several resources on the chip may be shared to multiplex and support different features or functions. When such resources are shared between trusted and untrusted agents, untrusted agents may be able to access the assets intended to be accessed only by the trusted agents.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1478
ChildOf		653	Improper Isolation or Compartmentalization	1445
ParentOf		1303	Non-Transparent Sharing of Microarchitectural Resources	2186
PeerOf		1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2237

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2237

Weakness Ordinalities

Primary :

Applicable Platforms

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>If resources being used by a trusted user are shared with an untrusted user, the untrusted user may be able to modify the functionality of the shared resource of the trusted user.</i>	
Integrity	Quality Degradation <i>The functionality of the shared resource may be intentionally degraded.</i>	

Detection Methods

Automated Dynamic Analysis

Pre-silicon / post-silicon: Test access to shared systems resources (memory ranges, control registers, etc.) from untrusted software to verify that the assets are not incorrectly exposed to untrusted agents. Note that access to shared resources can be dynamically allowed or revoked based on system flows. Security testing should cover such dynamic shared resource allocation and access control modification flows.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

When sharing resources, avoid mixing agents of varying trust levels. Untrusted agents should not share resources with trusted agents.

Demonstrative Examples

Example 1:

Consider the following SoC design. The Hardware Root of Trust (HROt) local SRAM is memory mapped in the core{0-N} address space. The HROt allows or disallows access to private memory ranges, thus allowing the sram to function as a mailbox for communication between untrusted and trusted HROt partitions.

We assume that the threat is from malicious software in the untrusted domain. We assume this software has access to the core{0-N} memory map and can be running at any privilege level on the untrusted cores. The capability of this threat in this example is communication to and from the mailbox region of SRAM modulated by the hrot_iface. To address this threat, information must not enter or exit the shared region of SRAM through hrot_iface when in secure or privileged mode.

Observed Examples

Reference	Description
CVE-2020-8698	Processor has improper isolation of shared resources allowing for information disclosure. https://www.cve.org/CVERecord?id=CVE-2020-8698
CVE-2019-6260	Baseboard Management Controller (BMC) device implements Advanced High-performance Bus (AHB) bridges that do not require authentication for arbitrary read and write access to the BMC's physical address space from the host, and possibly the network [REF-1138]. https://www.cve.org/CVERecord?id=CVE-2019-6260

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2491
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2613
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
124	Shared Resource Manipulation

References

[REF-1036]Ali Abbasi and Majid Hashemi. "Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack". 2016. < <https://www.blackhat.com/docs/eu-16/materials/eu-16-Abbasi-Ghost-In-The-PLC-Designing-An-Undetectable-Programmable-Logic-Controller-Rootkit-wp.pdf> >.

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

CWE-1190: DMA Device Enabled Too Early in Boot Phase

Weakness ID : 1190

Structure : Simple

Abstraction : Base

Description

The product enables a Direct Memory Access (DMA) capable device before the security configuration settings are established, which allows an attacker to extract data from or gain privileges on the product.

Extended Description

DMA is included in a number of devices because it allows data transfer between the computer and the connected device, using direct hardware access to read or write directly to main memory without any OS interaction. An attacker could exploit this to access secrets. Several virtualization-based mitigations have been introduced to thwart DMA attacks. These are usually configured/setup during boot time. However, certain IPs that are powered up before boot is complete (known as

early boot IPs) may be DMA capable. Such IPs, if not trusted, could launch DMA attacks and gain access to assets that should otherwise be protected.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		696	Incorrect Behavior Order	1535

Applicable Platforms

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Modify Memory <i>DMA devices have direct write access to main memory and due to time of attack will be able to bypass OS or Bootloader access control.</i>	High

Potential Mitigations

Phase: Architecture and Design

Utilize an IOMMU to orchestrate IO access from the start of the boot process.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1196	Security Flow Issues	1194	2490
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1038]"DMA attack". 2019 October 9. < https://en.wikipedia.org/wiki/DMA_attack >.

[REF-1039]A. Theodore Markettos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore and Robert N. M. Watson. "Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals". 2019 February 5. < https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_05A-1_Markettos_paper.pdf >.

[REF-1040]Maximillian Dornseif, Michael Becher and Christian N. Klein. "FireWire all your memory are belong to us". 2005. < <http://www.orkspace.net/secdocs/Conferences/CanSecWest/2005/0wn3d%20by%20an%20iPod%20-%20Firewire1394%20Issues.pdf> >.2023-04-07.

[REF-1041]Rory Breuk, Albert Spruyt and Adam Boileau. "Integrating DMA attacks in exploitation frameworks". 2012 February 0. < https://www.os3.nl/_media/2011-2012/courses/rp1/p14_report.pdf >.

[REF-1042]Maximillian Dornseif. "Owned by an iPod". 2004. < <https://web.archive.org/web/20060505224959/https://pacsec.jp/psj04/psj04-dornseif-e.ppt> >.2023-04-07.

[REF-1044]Dmytro Oleksiuk. "My aimful life". 2015 September 2. < <http://blog.cr4.sh/2015/09/breaking-uefi-security-with-software.html> >.

[REF-1046]A. Theodore Marketos and Adam Boileau. "Hit by a Bus:Physical Access Attacks with Firewire". 2006. < https://security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf >.

CWE-1191: On-Chip Debug and Test Interface With Improper Access Control

Weakness ID : 1191

Structure : Simple

Abstraction : Base

Description

The chip does not implement or does not correctly perform access control to check whether users are authorized to access internal registers and test modes through the physical debug/test interface.

Extended Description

A device's internal information may be accessed through a scan chain of interconnected internal registers, usually through a JTAG interface. The JTAG interface provides access to these registers in a serial fashion in the form of a scan chain for the purposes of debugging programs running on a device. Since almost all information contained within a device may be accessed over this interface, device manufacturers typically insert some form of authentication and authorization to prevent unintended use of this sensitive information. This mechanism is implemented in addition to on-chip protections that are already present.

If authorization, authentication, or some other form of access control is not implemented or not implemented correctly, a user may be able to bypass on-chip protection mechanisms through the debug interface.

Sometimes, designers choose not to expose the debug pins on the motherboard. Instead, they choose to hide these pins in the intermediate layers of the board. This is primarily done to work around the lack of debug authorization inside the chip. In such a scenario (without debug authorization), when the debug interface is exposed, chip internals are accessible to an attacker.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	687
PeerOf		1263	Improper Physical Access Control	2097

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1299	Missing Protection Mechanism for Alternate Hardware Interface	2174

Weakness Ordinalities

Primary :

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
Confidentiality	Read Application Data	High
Confidentiality	Read Memory	High
Authorization	Execute Unauthorized Code or Commands	High
Integrity	Modify Memory	High
Integrity	Modify Application Data	High
Access Control	Bypass Protection Mechanism	High

Detection Methods

Dynamic Analysis with Manual Results Interpretation

Authentication and authorization of debug and test interfaces should be part of the architecture and design review process. Withholding of private register documentation from the debug and test interface public specification ("Security by obscurity") should not be considered as sufficient security.

Dynamic Analysis with Manual Results Interpretation

Dynamic tests should be done in the pre-silicon and post-silicon stages to verify that the debug and test interfaces are not open by default.

Fuzzing

Tests that fuzz Debug and Test Interfaces should ensure that no access without appropriate authentication and authorization is possible.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

If feasible, the manufacturer should disable the JTAG interface or implement authentication and authorization for the JTAG interface. If authentication logic is added, it should be resistant to timing attacks. Security-sensitive data stored in registers, such as keys, etc. should be cleared when entering debug mode.

Effectiveness = High

Demonstrative Examples

Example 1:

A home, WiFi-router device implements a login prompt which prevents an unauthorized user from issuing any commands on the device until appropriate credentials are provided. The credentials are protected on the device and are checked for strength against attack.

Example Language: Other

(Bad)

If the JTAG interface on this device is not hidden by the manufacturer, the interface may be identified using tools such as JTAGulator. If it is hidden but not disabled, it can be exposed by physically wiring to the board.

By issuing a "halt" command before the OS starts, the unauthorized user pauses the watchdog timer and prevents the router from restarting (once the watchdog timer would have expired). Having paused the router, an unauthorized user is able to execute code and inspect and modify data in the device, even extracting all of the router's firmware. This allows the user to examine the router and potentially exploit it.

JTAG is useful to chip and device manufacturers during design, testing, and production and is included in nearly every product. Without proper authentication and authorization, the interface may allow tampering with a product.

Example Language: Other

(Good)

In order to prevent exposing the debugging interface, manufacturers might try to obfuscate the JTAG interface or blow device internal fuses to disable the JTAG interface. Adding authentication and authorization to this interface makes use by unauthorized individuals much more difficult.

Example 2:

The following example code is a snippet from the JTAG wrapper module in the RISC-V debug module of the HACK@DAC'21 Openpiton SoC [REF-1355]. To make sure that the JTAG is accessed securely, the developers have included a primary authentication mechanism based on a password.

The developers employed a Finite State Machine (FSM) to implement this authentication. When a user intends to read from or write to the JTAG module, they must input a password.

In the subsequent state of the FSM module, the entered password undergoes Hash-based Message Authentication Code (HMAC) calculation using an internal HMAC submodule. Once the HMAC for the entered password is computed by the HMAC submodule, the FSM transitions to the next state, where it compares the computed HMAC with the expected HMAC for the password.

If the computed HMAC matches the expected HMAC, the FSM grants the user permission to perform read or write operations on the JTAG module. [REF-1352]

Example Language: Verilog

(Bad)

```
...
    PassChkValid: begin
        if(hashValid) begin
            if(exp_hash == pass_hash) begin
                pass_check = 1'b1;
            end else begin
                pass_check = 1'b0;
            end
            state_d = Idle;
        end else begin
            state_d = PassChkValid;
        end
    end
...

```

However, in the given vulnerable part of the code, the JTAG module has not defined a limitation for several continuous wrong password attempts. This omission poses a significant security risk, allowing attackers to carry out brute-force attacks without restrictions.

Without a limitation on wrong password attempts, an attacker can repeatedly guess different passwords until they gain unauthorized access to the JTAG module. This leads to various malicious activities, such as unauthorized read from or write to debug module interface.

To mitigate the mentioned vulnerability, developers need to implement a restriction on the number of consecutive incorrect password attempts allowed by the JTAG module, which can achieve by incorporating a mechanism that temporarily locks the module after a certain number of failed attempts.[REF-1353][REF-1354]

Example Language: Verilog

(Good)

```
...
case (state_q)
  Idle: begin
    ...
    else if ( (dm::dtm_op_e'(dmi.op) == dm::DTM_PASS) && (miss_pass_check_cnt_q != 2'b11) )
    begin
      state_d = Write;
      pass_mode = 1'b1;
    end
    ...
  end
  ...
  PassChkValid: begin
    if(hashValid) begin
      if(exp_hash == pass_hash) begin
        pass_check = 1'b1;
      end else begin
        pass_check = 1'b0;
        miss_pass_check_cnt_d = miss_pass_check_cnt_q + 1
      end
      state_d = Idle;
    end else begin
      state_d = PassChkValid;
    end
  end
end
...
```

Example 3:

The example code below is taken from the JTAG access control mechanism of the HACK@DAC'21 buggy OpenPiton SoC [REF-1364]. Access to JTAG allows users to access sensitive information in the system. Hence, access to JTAG is controlled using cryptographic authentication of the users. In this example (see the vulnerable code source), the password checker uses HMAC-SHA256 for authentication. It takes a 512-bit secret message from the user, hashes it using HMAC, and compares its output with the expected output to determine the authenticity of the user.

Example Language: Verilog

(Bad)

```
...
logic [31-1:0] data_d, data_q;
...
logic [512-1:0] pass_data;
...
Write: begin
  ...
  if (pass_mode) begin
    pass_data = { {60{8'h00}}, data_d};
    state_d = PassChk;
    pass_mode = 1'b0;
  }
  ...
end
...
```

The vulnerable code shows an incorrect implementation of the HMAC authentication where it only uses the least significant 32 bits of the secret message for the authentication (the remaining 480 bits are hard coded as zeros). As a result, the system is susceptible to brute-force attacks on the access control mechanism of JTAG, where the attacker only needs to determine 32 bits of the secret message instead of 512 bits.

To mitigate this issue, remove the zero padding and use all 512 bits of the secret message for HMAC authentication [REF-1365].

Example Language: Verilog

(Good)







```
...
logic [512-1:0] data_d, data_q;
...
logic [512-1:0] pass_data;
...
Write: begin
    ...
    if (pass_mode) begin
        pass_data = data_d;
        state_d = PassChk;
        pass_mode = 1'b0;
    ...
end
...
```

Observed Examples

Reference	Description
CVE-2019-18827	chain: JTAG interface is not disabled (CWE-1191) during ROM code execution, introducing a race condition (CWE-362) to extract encryption keys https://www.cve.org/CVERecord?id=CVE-2019-18827

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems		2495
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List		2613
MemberOf		1396	Comprehensive Categorization: Access Control		2540

Notes

Relationship

CWE-1191 and CWE-1244 both involve physical debug access, but the weaknesses are different. CWE-1191 is effectively about missing authorization for a debug interface, i.e. JTAG. CWE-1244 is about providing internal assets with the wrong debug access level, exposing the asset to untrusted debug agents.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1037]Kurt Rosenfeld and Ramesh Karri. "Attacks and Defenses for JTAG". 2010 February. <
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5406671> >.

[REF-1043]Gopal Vishwakarma and Wonjun Lee. "Exploiting JTAG and Its Mitigation in IOT: A Survey". 2018 December 3. < <https://www.mdpi.com/1999-5903/10/12/121/pdf> >.2023-04-07.

[REF-1084]Gopal Vishwakarma and Wonjun Lee. "JTAG Explained (finally!): Why "IoT", Software Security Engineers, and Manufacturers Should Care". < <https://www.mdpi.com/1999-5903/10/12/121/pdf> >.2023-04-07.

[REF-1085]Bob Molyneaux, Mark McDermott and Anil Sabbavarapu. "Design for Testability & Design for Debug". < http://users.ece.utexas.edu/~mcdermot/vlsi-2/Lecture_17.pdf >.

[REF-1355]Florian Zaruba. "dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L192:L204 >.2023-09-18.

[REF-1354]Florian Zaruba. "Fix CWE-1191 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/58f984d492fdb0369c82ef10fcbbaa4b9850f9fb/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L200 >.2023-09-18.

[REF-1353]Florian Zaruba. "Fix CWE-1191 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/58f984d492fdb0369c82ef10fcbbaa4b9850f9fb/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L131 >.2023-09-18.

[REF-1352]Florian Zaruba. "dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L118:L204 >.2023-09-18.

[REF-1364]"dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L82 >.2023-07-15.

[REF-1365]"fix cwe_1205 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/c4f4b832218b50c406dbf9f425d3b654117c1355/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L158 >.2023-07-22.

CWE-1192: Improper Identifier for IP Block used in System-On-Chip (SOC)

Weakness ID : 1192

Structure : Simple

Abstraction : Base

Description

The System-on-Chip (SoC) does not have unique, immutable identifiers for each of its components.

Extended Description

A System-on-Chip (SoC) comprises several components (IP) with varied trust requirements. It is required that each IP is identified uniquely and should distinguish itself from other entities in the SoC without any ambiguity. The unique secured identity is required for various purposes. Most of the time the identity is used to route a transaction or perform certain actions, including resetting, retrieving a sensitive information, and acting upon or on behalf of something else.

There are several variants of this weakness:

- A "missing" identifier is when the SoC does not define any mechanism to uniquely identify the IP.
- An "insufficient" identifier might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended.
- A "misconfigured" mechanism occurs when a mechanism is available but not implemented correctly.

- An "ignored" identifier occurs when the SoC/IP has not applied any policies or does not act upon the identifier securely.

Relationships

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

Applicable Platforms

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	High

Potential Mitigations




Phase: Architecture and Design

Strategy = Separation of Privilege

Every identity generated in the SoC should be unique and immutable in hardware. The actions that an IP is trusted or not trusted should be clearly defined, implemented, configured, and tested. If the definition is implemented via a policy, then the policy should be immutable or protected with clear authentication and authorization.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1198	Privilege Separation and Access Control Issues	1194	2491
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
113	Interface Manipulation

CWE-1193: Power-On of Untrusted Execution Core Before Enabling Fabric Access Control

Weakness ID : 1193

Structure : Simple

Abstraction : Base

Description

The product enables components that contain untrusted firmware before memory and fabric access controls have been enabled.

Extended Description

After initial reset, System-on-Chip (SoC) fabric access controls and other security features need to be programmed by trusted firmware as part of the boot sequence. If untrusted IPs or peripheral microcontrollers are enabled first, then the untrusted component can master transactions on the hardware bus and target memory or other assets to compromise the SoC boot firmware.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		696	Incorrect Behavior Order	1535

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>An untrusted component can master transactions on the HW bus and target memory or other assets to compromise the SoC boot firmware.</i>	High




Potential Mitigations

Phase: Architecture and Design

The boot sequence should enable fabric access controls and memory protections before enabling third-party hardware IPs and peripheral microcontrollers that use untrusted firmware.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1196	Security Flow Issues	1194	2490
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1130]Mark Ermolov, Positive Technologies. "Intel x86 Root of Trust: loss of trust". 2020 March 5. < <https://blog.ptsecurity.com/2020/03/intelx86-root-of-trust-loss-of-trust.html> >.

[REF-1042]Maximillian Dornseif. "Owned by an iPod". 2004. < <https://web.archive.org/web/20060505224959/https://pacsec.jp/psj04/psj04-dornseif-e.ppt> >.2023-04-07.

CWE-1204: Generation of Weak Initialization Vector (IV)

Weakness ID : 1204

Structure : Simple

1996

Abstraction : Base**Description**

The product uses a cryptographic primitive that uses an Initialization Vector (IV), but the product does not generate IVs that are sufficiently unpredictable or unique according to the expected cryptographic requirements for that primitive.



Extended Description

By design, some cryptographic primitives (such as block ciphers) require that IVs must have certain properties for the uniqueness and/or unpredictability of an IV. Primitives may vary in how important these properties are. If these properties are not maintained, e.g. by a bug in the code, then the cryptography may be weakened or broken by attacking the IVs themselves.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	821
ParentOf		329	Generation of Predictable IV with CBC Mode	818

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

Nature	Type	ID	Name	Page
MemberOf		310	Cryptographic Issues	2339

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>If the IV is not properly initialized, data that is encrypted can be compromised and information about the data can be leaked. See [REF-1179].</i>	

Potential Mitigations**Phase: Implementation**

Different cipher modes have different requirements for their IVs. When choosing and implementing a mode, it is important to understand those requirements in order to keep security guarantees intact. Generally, it is safest to generate a random IV, since it will be both unpredictable and have a very low chance of being non-unique. IVs do not have to be kept secret, so if generating duplicate IVs is a concern, a list of already-used IVs can be kept and checked against. NIST offers recommendations on generation of IVs for modes of which they have approved. These include options for when random IVs are not practical. For CBC, CFB, and OFB, see [REF-1175]; for GCM, see [REF-1178].

Demonstrative Examples**Example 1:**

In the following examples, CBC mode is used when encrypting data:

Example Language: C

(Bad)

```
EVP_CIPHER_CTX ctx;
```

```
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv,0,EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx,EVP_bf_cbc(), key,iv);
```

Example Language: Java

(Bad)

```
public class SymmetricCipherTest {
    public static void main() {
        byte[] text = "Secret".getBytes();
        byte[] iv = {
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
        };
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        kg.init(56);
        SecretKey key = kg.generateKey();
        Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
        IvParameterSpec ips = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, key, ips);
        return cipher.doFinal(inpBytes);
    }
}
```

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

Example 2:

The Wired Equivalent Privacy (WEP) protocol used in the 802.11 wireless standard only supported 40-bit keys, and the IVs were only 24 bits, increasing the chances that the same IV would be reused for multiple messages. The IV was included in plaintext as part of the packet, making it directly observable to attackers. Only 5000 messages are needed before a collision occurs due to the "birthday paradox" [REF-1176]. Some implementations would reuse the same IV for each packet. This IV reuse made it much easier for attackers to recover plaintext from two packets with the same IV, using well-understood attacks, especially if the plaintext was known for one of the packets [REF-1175].

Observed Examples

Reference	Description
CVE-2020-1472	ZeroLogon vulnerability - use of a static IV of all zeroes in AES-CFB8 mode https://www.cve.org/CVERecord?id=CVE-2020-1472
CVE-2011-3389	BEAST attack in SSL 3.0 / TLS 1.0. In CBC mode, chained initialization vectors are non-random, allowing decryption of HTTPS traffic using a chosen plaintext attack. https://www.cve.org/CVERecord?id=CVE-2011-3389
CVE-2001-0161	wireless router does not use 6 of the 24 bits for WEP encryption, making it easier for attackers to decrypt traffic https://www.cve.org/CVERecord?id=CVE-2001-0161
CVE-2001-0160	WEP card generates predictable IV values, making it easier for attackers to decrypt traffic https://www.cve.org/CVERecord?id=CVE-2001-0160
CVE-2017-3225	device bootloader uses a zero initialization vector during AES-CBC https://www.cve.org/CVERecord?id=CVE-2017-3225
CVE-2016-6485	crypto framework uses PHP rand function - which is not cryptographically secure - for an initialization vector https://www.cve.org/CVERecord?id=CVE-2016-6485
CVE-2014-5386	encryption routine does not seed the random number generator, causing the same initialization vector to be generated repeatedly https://www.cve.org/CVERecord?id=CVE-2014-5386

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

Functional Areas

- Cryptography

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1414	Comprehensive Categorization: Randomness	1400	2564

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.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
20	Encryption Brute Forcing
97	Cryptanalysis

References

[REF-1175]Nikita Borisov, Ian Goldberg and David Wagner. "Intercepting Mobile Communications: The Insecurity of 802.11". Proceedings of the Seventh Annual International Conference on Mobile Computing And Networking. 2001 July. ACM. < <http://www.isaac.cs.berkeley.edu/isaac/mobicom.pdf> >.

[REF-1175]Nikita Borisov, Ian Goldberg and David Wagner. "Intercepting Mobile Communications: The Insecurity of 802.11". Proceedings of the Seventh Annual International Conference on Mobile Computing And Networking. 2001 July. ACM. < <http://www.isaac.cs.berkeley.edu/isaac/mobicom.pdf> >.

[REF-1176]Wikipedia. "Birthday problem". 2021 March 6. < https://en.wikipedia.org/wiki/Birthday_problem >.

[REF-1177]Wikipedia. "Initialization Vector". 2021 March 8. < https://en.wikipedia.org/wiki/Initialization_vector >.

[REF-1178]NIST. "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC". 2007 November. < <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf> >.2023-04-07.

[REF-1179]Arxum Path Security. "CBC Mode is Malleable. Don't trust it for Authentication". 2019 October 6. < <https://arxumpathsecurity.com/blog/2019/10/16/cbc-mode-is-malleable-dont-trust-it-for-authentication> >.2023-04-07.

CWE-1209: Failure to Disable Reserved Bits

Weakness ID : 1209

Structure : Simple

Abstraction : Base

Description

The reserved bits in a hardware design are not disabled prior to production. Typically, reserved bits are used for future capabilities and should not support any functional logic in the design. However, designers might covertly use these bits to debug or further develop new capabilities in production hardware. Adversaries with access to these bits will write to them in hopes of compromising hardware state.

Extended Description

Reserved bits are labeled as such so they can be allocated for a later purpose. They are not to do anything in the current design. However, designers might want to use these bits to debug or control/configure a future capability to help minimize time to market (TTM). If the logic being controlled by these bits is still enabled in production, an adversary could use the logic to induce unwanted/unsupported behavior in the hardware.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	710	Improper Adherence to Coding Standards	1558

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability	Varies by Context <i>This type of weakness all depends on the capabilities of the logic being controlled or configured by the reserved bits.</i>	

Scope	Impact	Likelihood
Authentication		
Authorization		
Non-Repudiation		

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Include a feature to disable reserved bits.

Phase: Integration

Any writes to these reserve bits are blocked (e.g., ignored, access-protected, etc.), or an exception can be asserted.

Demonstrative Examples

Example 1:

Assume a hardware Intellectual Property (IP) has address space 0x0-0x0F for its configuration registers, with the last one labeled reserved (i.e. 0x0F). Therefore inside the Finite State Machine (FSM), the code is as follows:

Example Language: Verilog

(Bad)

```
reg gpio_out = 0; //gpio should remain low for normal operation
case (register_address)
  4'b1111 : //0x0F
    begin
      gpio_out = 1;
    end
```

An adversary may perform writes to reserved address space in hopes of changing the behavior of the hardware. In the code above, the GPIO pin should remain low for normal operation. However, it can be asserted by accessing the reserved address space (0x0F). This may be a concern if the GPIO state is being used as an indicator of health (e.g. if asserted the hardware may respond by shutting down or resetting the system, which may not be the correct action the system should perform).

In the code below, the condition "register_address = 0X0F" is commented out, and a default is provided that will catch any values of register_address not explicitly accounted for and take no action with regards to gpio_out. This means that an attacker who is able to write 0X0F to register_address will not enable any undocumented "features" in the process.



Example Language: Verilog

(Good)

```
reg gpio_out = 0; //gpio should remain low for normal operation
case (register_address)
  //4'b1111 : //0x0F
  default: gpio_out = gpio_out;
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces

CWE-1220: Insufficient Granularity of Access Control

Weakness ID : 1220

Structure : Simple

Abstraction : Base

Description

The product implements access controls via a policy or other feature with the intention to disable or restrict accesses (reads and/or writes) to assets in a system from untrusted agents. However, implemented access controls lack required granularity, which renders the control policy too broad because it allows accesses from unauthorized agents to the security-sensitive assets.

Extended Description

Integrated circuits and hardware engines can expose accesses to assets (device configuration, keys, etc.) to trusted firmware or a software module (commonly set by BIOS/bootloader). This access is typically access-controlled. Upon a power reset, the hardware or system usually starts with default values in registers, and the trusted firmware (Boot firmware) configures the necessary access-control protection.

A common weakness that can exist in such protection schemes is that access controls or policies are not granular enough. This condition allows agents beyond trusted agents to access assets and could lead to a loss of functionality or the ability to set up the device securely. This further results in security risks from leaked, sensitive, key material to modification of device configuration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	⚡	1222	Insufficient Granularity of Address Regions Protected by Register Locks	2010

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

Nature	Type	ID	Name	Page
MemberOf	⚠	1212	Authorization Errors	2497

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
Confidentiality	Modify Memory	High

2002

Scope	Impact	Likelihood
Integrity	Read Memory	
Availability	Execute Unauthorized Code or Commands	
Access Control	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	
	Other	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Access-control-policy protections must be reviewed for design inconsistency and common weaknesses. Access-control-policy definition and programming flow must be tested in pre-silicon, post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider a system with a register for storing AES key for encryption or decryption. The key is 128 bits, implemented as a set of four 32-bit registers. The key registers are assets and registers, AES_KEY_READ_POLICY and AES_KEY_WRITE_POLICY, and are defined to provide necessary access controls.

The read-policy register defines which agents can read the AES-key registers, and write-policy register defines which agents can program or write to those registers. Each register is a 32-bit register, and it can support access control for a maximum of 32 agents. The number of the bit when set (i.e., "1") allows respective action from an agent whose identity matches the number of the bit and, if "0" (i.e., Clear), disallows the respective action to that corresponding agent.

Example Language: Other

(Bad)

In the above example, there is only one policy register that controls access to both read and write accesses to the AES-key registers, and thus the design is not granular enough to separate read and writes access for different agents. Here, agent with identities "1" and "2" can both read and write.

A good design should be granular enough to provide separate access controls to separate actions. Access control for reads should be separate from writes. Below is an example of such implementation where two policy registers are defined for each of these actions. The policy is defined such that: the AES-key registers can only be read or used by a crypto agent with identity "1" when bit #1 is set. The AES-key registers can only be programmed by a trusted firmware with identity "2" when bit #2 is set.

Example Language:

(Good)

Example 2:

Within the AXI node interface wrapper module in the RISC-V AXI module of the HACK@DAC'19 CVA6 SoC [REF-1346], an access control mechanism is employed to regulate the access of different privileged users to peripherals.

The AXI ensures that only users with appropriate privileges can access specific peripherals. For instance, a ROM module is accessible exclusively with Machine privilege, and AXI enforces that users attempting to read data from the ROM must possess machine privilege; otherwise, access to the ROM is denied. The access control information and configurations are stored in a ROM.

Example Language: Verilog

(Bad)

```
...
for (i=0; i<NB_SUBORDINATE; i++)
begin
  for (j=0; j<NB_MANAGER; j++)
  begin
    assign connectivity_map_o[i][j] = access_ctrl_i[i][j][priv_lvl_i] || ((j==6) && access_ctrl_i[i][7][priv_lvl_i]);
  end
end
...
```

However, in the example code above, while assigning distinct privileges to AXI manager and subordinates, both the Platform-Level Interrupt Controller Specification (PLIC) and the Core-local Interrupt Controller (CLINT) (which are peripheral numbers 6 and 7 respectively) utilize the same access control configuration. This common configuration diminishes the granularity of the AXI access control mechanism.

In certain situations, it might be necessary to grant higher privileges for accessing the PLIC than those required for accessing the CLINT. Unfortunately, this differentiation is overlooked, allowing an attacker to access the PLIC with lower privileges than intended.

As a consequence, unprivileged code can read and write to the PLIC even when it was not intended to do so. In the worst-case scenario, the attacker could manipulate interrupt priorities, potentially modifying the system's behavior or availability.

To address the aforementioned vulnerability, developers must enhance the AXI access control granularity by implementing distinct access control entries for the Platform-Level Interrupt Controller (PLIC) and the Core-local Interrupt Controller (CLINT). By doing so, different privilege levels can be defined for accessing PLIC and CLINT, effectively thwarting the potential attacks previously highlighted. This approach ensures a more robust and secure system, safeguarding against unauthorized access and manipulation of interrupt priorities. [REF-1347]

Example Language: Verilog

(Good)

```
...
for (i=0; i<NB_SUBORDINATE; i++)
begin
  for (j=0; j<NB_MANAGER; j++)
  begin
    assign connectivity_map_o[i][j] = access_ctrl_i[i][j][priv_lvl_i];
  end
end
...
```

Example 3:

Consider the following SoC design. The sram in HRoT has an address range that is readable and writable by unprivileged software and it has an area that is only readable by unprivileged software. The tbus interconnect enforces access control for subordinates on the bus but uses only one bit to control both read and write access. Address 0xA0000000 - 0xA000FFFF is readable and writable by the untrusted cores core{0-N} and address 0xA0010000 - 0xA001FFFF is only readable by the untrusted cores core{0-N}.

The security policy access control is not granular enough, as it uses one bit to enable both read and write access. This gives write access to an area that should only be readable by unprivileged agents.



Access control logic should differentiate between read and write access and to have sufficient address granularity.

Observed Examples

Reference	Description
CVE-2022-24985	A form hosting website only checks the session authentication status for a single form, making it possible to bypass authentication when there are multiple forms https://www.cve.org/CVERecord?id=CVE-2022-24985
CVE-2021-36934	An operating system has an overly permission Access Control List on some system files, including those related to user passwords https://www.cve.org/CVERecord?id=CVE-2021-36934

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2491
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1346]"axi_node_intf_wrap.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/619e9fb0ef32ee1e01ad76b8732a156572c65700/src/axi_node/src/axi_node_intf_wrap.sv#L430 >.2023-09-18.

[REF-1347]"axi_node_intf_wrap.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/2078f2552194eda37ba87e54cbfef10f1aa41fa5/src/axi_node/src/axi_node_intf_wrap.sv#L430 >.2023-09-18.

CWE-1221: Incorrect Register Defaults or Module Parameters

Weakness ID : 1221

Structure : Simple

Abstraction : Base

Description

Hardware description language code incorrectly defines register defaults or hardware Intellectual Property (IP) parameters to insecure values.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to defined default values that are hard coded in the hardware description language (HDL) code of the hardware unit. Hardware descriptive languages also support definition of parameter variables,

which can be defined in code during instantiation of the hardware IP module. Such parameters are generally used to configure a specific instance of a hardware IP in the design.

The system security settings of a hardware design can be affected by incorrectly defined default values or IP parameters. The hardware IP would be in an insecure state at power reset, and this can be exposed or exploited by untrusted software running on the system. Both register defaults and parameters are hardcoded values, which cannot be changed using software or firmware patches but must be changed in hardware silicon. Thus, such security issues are considerably more difficult to address later in the lifecycle. Hardware designs can have a large number of such parameters and register defaults settings, and it is important to have design tool support to check these settings in an automated way and be able to identify which settings are security sensitive.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1419	Incorrect Initialization of Resource	2292

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context <i>Degradation of system functionality, or loss of access control enforcement can occur.</i>	
Integrity		
Availability		
Access Control		

Potential Mitigations

Phase: Architecture and Design

During hardware design, all the system parameters and register defaults must be reviewed to identify security sensitive settings.

Phase: Implementation

The default values of these security sensitive settings need to be defined as part of the design review phase.

Phase: Testing

Testing phase should use automated tools to test that values are configured per design specifications.

Demonstrative Examples

Example 1:

Consider example design module system verilog code shown below. The register_example module is an example parameterized module that defines two parameters, REGISTER_WIDTH and REGISTER_DEFAULT. Register_example module defines a Secure_mode setting, which when set makes the register content read-only and not modifiable by software writes. register_top module instantiates two registers, Insecure_Device_ID_1 and Insecure_Device_ID_2. Generally, registers

containing device identifier values are required to be read only to prevent any possibility of software modifying these values.

Example Language: Verilog

(Bad)

```
// Parameterized Register module example
// Secure_mode : REGISTER_DEFAULT[0] : When set to 1 register is read only and not writable//
module register_example
#(
    parameter REGISTER_WIDTH = 8, // Parameter defines width of register, default 8 bits
    parameter [REGISTER_WIDTH-1:0] REGISTER_DEFAULT = 2**REGISTER_WIDTH - 2 // Default value of register
    computed from Width. Sets all bits to 1s except bit 0 (Secure_mode)
)
(
    input [REGISTER_WIDTH-1:0] Data_in,
    input Clk,
    input resetn,
    input write,
    output reg [REGISTER_WIDTH-1:0] Data_out
);
reg Secure_mode;
always @(posedge Clk or negedge resetn)
    if (~resetn)
        begin
            Data_out <= REGISTER_DEFAULT; // Register content set to Default at reset
            Secure_mode <= REGISTER_DEFAULT[0]; // Register Secure_mode set at reset
        end
    else if (write & ~Secure_mode)
        begin
            Data_out <= Data_in;
        end
endmodule

module register_top
(
    input Clk,
    input resetn,
    input write,
    input [31:0] Data_in,
    output reg [31:0] Secure_reg,
    output reg [31:0] Insecure_reg
);
    register_example #(
        .REGISTER_WIDTH (32),
        .REGISTER_DEFAULT (1224) // Incorrect Default value used bit 0 is 0.
    ) Insecure_Device_ID_1 (
        .Data_in (Data_in),
        .Data_out (Secure_reg),
        .Clk (Clk),
        .resetn (resetn),
        .write (write)
    );
    register_example #(
        .REGISTER_WIDTH (32) // Default not defined 2^32-2 value will be used as default.
    ) Insecure_Device_ID_2 (
        .Data_in (Data_in),
        .Data_out (Insecure_reg),
        .Clk (Clk),
        .resetn (resetn),
        .write (write)
    );
endmodule
```

These example instantiations show how, in a hardware design, it would be possible to instantiate the register module with insecure defaults and parameters.

In the example design, both registers will be software writable since Secure_mode is defined as zero.

*Example Language: Verilog**(Good)*

```

register_example #(
    .REGISTER_WIDTH (32),
    .REGISTER_DEFAULT (1225) // Correct default value set, to enable Secure_mode
) Secure_Device_ID_example (
    .Data_in (Data_in),
    .Data_out (Secure_reg),
    .Clk (Clk),
    .resetn (resetn),
    .write (write)
);

```

Example 2:

The example code is taken from the fuse memory inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1356]. Fuse memory can be used to store key hashes, password hashes, and configuration information. For example, the password hashes of JTAG and HMAC are stored in the fuse memory in the OpenPiton design.

During the firmware setup phase, data in the Fuse memory are transferred into the registers of the corresponding SoC peripherals for initialization. However, if the offset to access the password hash is set incorrectly, programs cannot access the correct password hash from the fuse memory, breaking the functionalities of the peripherals and even exposing sensitive information through other peripherals.

*Example Language: Verilog**(Bad)*

```

parameter MEM_SIZE = 100;
localparam JTAG_OFFSET = 81;
const logic [MEM_SIZE-1:0][31:0] mem = {
    // JTAG expected hamc hash
    32'h49ac13af, 32'h1276f1b8, 32'h6703193a, 32'h65eb531b,
    32'h3025ccca, 32'h3e8861f4, 32'h329edfe5, 32'h98f763b4,
    ...
assign jtag_hash_o = {mem[JTAG_OFFSET-1],mem[JTAG_OFFSET-2],mem[JTAG_OFFSET-3],
mem[JTAG_OFFSET-4],mem[JTAG_OFFSET-5],mem[JTAG_OFFSET-6],mem[JTAG_OFFSET-7],mem[JTAG_OFFSET-8]};
...

```

The following vulnerable code accesses the JTAG password hash from the fuse memory. However, the JTAG_OFFSET is incorrect, and the fuse memory outputs the wrong values to jtag_hash_o. Moreover, setting incorrect offset gives the ability to attackers to access JTAG by knowing other low-privileged peripherals' passwords.

To mitigate this, change JTAG_OFFSET to the correct address of the JTAG key [REF-1357].

*Example Language: Verilog**(Good)*

```

parameter MEM_SIZE = 100;
localparam JTAG_OFFSET = 100;

```

Example 3:

The following example code is excerpted from the Access Control module, acct_wrapper, in the Hack@DAC'21 buggy OpenPiton System-on-Chip (SoC). Within this module, a set of memory-mapped I/O registers, referred to as acct_mem, each 32-bit wide, is utilized to store access control permissions for peripherals [REF-1437]. Access control registers are typically used to define and enforce permissions and access rights for various system resources.

However, in the buggy SoC, these registers are all enabled at reset, i.e., essentially granting unrestricted access to all system resources [REF-1438]. This will introduce security vulnerabilities and risks to the system, such as privilege escalation or exposing sensitive information to unauthorized users or processes.

Example Language: Verilog

(Bad)

```
module acct_wrapper #(
...
  always @(posedge clk_i)
  begin
    if(~(rst_ni && ~rst_6))
    begin
      for (j=0; j < AcCt_MEM_SIZE; j=j+1)
      begin
        acct_mem[j] <= 32'hffffff;
      end
    end
  end
...
)
```

To fix this issue, the access control registers must be properly initialized during the reset phase of the SoC. Correct initialization values should be established to maintain the system's integrity, security, predictable behavior, and allow proper control of peripherals. The specifics of what values should be set depend on the SoC's design and the requirements of the system. To address the problem depicted in the bad code example [REF-1438], the default value for "acct_mem" should be set to 32'h00000000 (see good code example [REF-1439]). This ensures that during startup or after any reset, access to protected data is restricted until the system setup is complete and security procedures properly configure the access control settings.

Example Language: Verilog

(Good)

```
module acct_wrapper #(
...
  always @(posedge clk_i)
  begin
    if(~(rst_ni && ~rst_6))
    begin
      for (j=0; j < AcCt_MEM_SIZE; j=j+1)
      begin
        acct_mem[j] <= 32'h00000000;
      end
    end
  end
...
)
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
166	Force the System to Reset Values

References

[REF-1356]"fuse_mem.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/main/piton/design/chip/tile/ariane/src/fuse_mem/fuse_mem.sv#L14-L15 >.2023-07-15.

[REF-1357]"fix CWE 1221 in fuse_mem.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/compare/main...cwe_1221_in_fuse_mem#diff-d7275edeac22f76691a31c83f005d0177359ad710ad6549ece3d069ed043ef21 >.2023-07-24.

[REF-1437]"acct_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/65d0ffdab7426da4509c98d62e163bcce642f651/piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv#L39 >.

[REF-1438]"Bad Code acct_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/65d0ffdab7426da4509c98d62e163bcce642f651/piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv#L79C1-L86C16 >.

[REF-1439]"Good Code acct_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/062de4f25002d2dcdbdb0a82af36b80a517592612/piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv#L84 >.

CWE-1222: Insufficient Granularity of Address Regions Protected by Register Locks

Weakness ID : 1222

Structure : Simple

Abstraction : Variant

Description

The product defines a large address region protected from modification by the same register lock control bit. This results in a conflict between the functional requirement that some addresses need to be writable by software during operation and the security requirement that the system configuration lock bit must be set during the boot process.

Extended Description


Integrated circuits and hardware IPs can expose the device configuration controls that need to be programmed after device power reset by a trusted firmware or software module (commonly set by BIOS/bootloader) and then locked from any further modification. In hardware design, this is commonly implemented using a programmable lock bit which enables/disables writing to a protected set of registers or address regions. When the programmable lock bit is set, the relevant address region can be implemented as a hardcoded value in hardware logic that cannot be changed later.

A problem can arise wherein the protected region definition is not granular enough. After the programmable lock bit has been set, then this new functionality cannot be implemented without change to the hardware design.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1220	Insufficient Granularity of Access Control	2002

Applicable Platforms

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

Operating System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Other <i>System security configuration cannot be defined in a way that does not conflict with functional requirements of device.</i>	

Potential Mitigations

Phase: Architecture and Design

The defining of protected locked registers should be reviewed or tested early in the design phase with software teams to ensure software flows are not blocked by the security locks. As an alternative to using register lock control bits and fixed access control regions, the hardware design could use programmable security access control configuration so that device trusted firmware can configure and change the protected regions based on software usage and security models.

Demonstrative Examples

Example 1:

For example, consider a hardware unit with a 32 kilobyte configuration address space where the first 8 kilobyte address contains security sensitive controls that must only be writable by device bootloader. One way to protect the security configuration could be to define a 32 bit system configuration locking register (SYS_LOCK) where each bit lock locks the corresponding 1 kilobyte region.

Example Language: Other

(Bad)

If a register exists within the first kilobyte address range (e.g. SW_MODE, address 0x310) and needs to be software writable at runtime, then this register cannot be written in a securely configured system since SYS_LOCK register lock bit 0 must be set to protect other security settings (e.g. SECURITY_FEATURE_ENABLE, address 0x0004). The only fix would be to change the hardware logic or not set the security lock bit.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1198	Privilege Separation and Access Control Issues	1194	2491
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
679	Exploitation of Improperly Configured or Implemented Memory Protections

CWE-1223: Race Condition for Write-Once Attributes

Weakness ID : 1223
Structure : Simple
Abstraction : Base

Description

A write-once register in hardware design is programmable by an untrusted software component earlier than the trusted software component, resulting in a race condition issue.

Extended Description


Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to defined default values that are hard coded in the hardware description language (HDL) code of the hardware unit. A common security protection method used to protect register settings from modification by software is to make them write-once. This means the hardware implementation only allows writing to such registers once, and they become read-only after having been written once by software. This is useful to allow initial boot software to configure systems settings to secure values while blocking runtime software from modifying such hardware settings.

Implementation issues in hardware design of such controls can expose such registers to a race condition security flaw. For example, consider a hardware design that has two different software/firmware modules executing in parallel. One module is trusted (module A) and another is untrusted (module B). In this design it could be possible for Module B to send write cycles to the write-once register before Module A. Since the field is write-once the programmed value from Module A will be ignored and the pre-empted value programmed by Module B will be used by hardware.

Relationships

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

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>System configuration cannot be programmed in a secure way.</i>	

Potential Mitigations

Phase: Architecture and Design

During hardware design all register write-once or sticky fields must be evaluated for proper configuration.

Phase: Testing

The testing phase should use automated tools to test that values are not reprogrammable and that write-once fields lock on writing zeros.

Demonstrative Examples

Example 1:

consider the example design module system verilog code shown below.
register_write_once_example module is an example of register that has a write-once field defined.
Bit 0 field captures the write_once_status value.

Example Language: Verilog (Bad)

```
module register_write_once_example
(
    input [15:0] Data_in,
    input Clk,
    input ip_resetrn,
    input global_resetrn,
    input write,
    output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetrn)
if (~ip_resetrn)
begin
    Data_out <= 16'h0000;
    Write_once_status <= 1'b0;
end
else if (write & ~Write_once_status)
begin
    Data_out <= Data_in & 16'hFFFE; // Input data written to register after masking bit 0
    Write_once_status <= 1'b1; // Write once status set after first write.
end
else if (~write)
begin
    Data_out[15:1] <= Data_out[15:1];
    Data_out[0] <= Write_once_status;
end
endmodule
```

The first system component that sends a write cycle to this register can program the value. This could result in a race condition security issue in the SoC design, if an untrusted agent is running in the system in parallel with the trusted component that is expected to program the register.

Example Language: (Good)

Trusted firmware or software trying to set the write-once field:

- Must confirm the Write_once_status (bit 0) value is zero, before programming register. If another agent has programmed the register before, then Write_once_status value will be one.
- After writing to the register, the trusted software can issue a read to confirm that the valid setting has been programmed.

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	1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions

CWE-1224: Improper Restriction of Write-Once Bit Fields

Weakness ID : 1224

Structure : Simple

Abstraction : Base

Description

The hardware design control register "sticky bits" or write-once bit fields are improperly implemented, such that they can be reprogrammed by software.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to define default values that are hard coded in the hardware description language (HDL) code of the hardware unit. A common security protection method used to protect register settings from modification by software is to make the settings write-once or "sticky." This allows writing to such registers only once, whereupon they become read-only. This is useful to allow initial boot software to configure systems settings to secure values while blocking runtime software from modifying such hardware settings.

Failure to implement write-once restrictions in hardware design can expose such registers to being re-programmed by software and written multiple times. For example, write-once fields could be implemented to only be write-protected if they have been set to value "1", wherein they would work as "write-1-once" and not "write-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	P	284	Improper Access Control	687

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context <i>System configuration cannot be programmed in a secure way.</i>	
Integrity		
Availability		
Access Control		

Potential Mitigations

Phase: Architecture and Design

During hardware design all register write-once or sticky fields must be evaluated for proper configuration.

Phase: Testing

The testing phase should use automated tools to test that values are not reprogrammable and that write-once fields lock on writing zeros.

Demonstrative Examples

Example 1:

Consider the example design module system verilog code shown below. register_write_once_example module is an example of register that has a write-once field defined. Bit 0 field captures the write_once_status value. This implementation can be for a register that is defined by specification to be a write-once register, since the write_once_status field gets written by input data bit 0 on first write.

Example Language: Verilog

(Bad)

```
module register_write_once_example
(
  input [15:0] Data_in,
  input Clk,
  input ip_resetrn,
  input global_resetrn,
  input write,
  output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetrn)
  if (~ip_resetrn)
    begin
      Data_out <= 16'h0000;
      Write_once_status <= 1'b0;
    end
  else if (write & ~Write_once_status)
    begin
      Data_out <= Data_in & 16'hFFFE;
      Write_once_status <= Data_in[0]; // Input bit 0 sets Write_once_status
    end
  else if (~write)
    begin
      Data_out[15:1] <= Data_out[15:1];
      Data_out[0] <= Write_once_status;
    end
end
endmodule
```

The above example only locks further writes if write_once_status bit is written to one. So it acts as write_1-Once instead of the write-once attribute.

Example Language: Verilog

(Good)

```
module register_write_once_example
(
  input [15:0] Data_in,
  input Clk,
  input ip_resetrn,
  input global_resetrn,
  input write,
  output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetrn)
  if (~ip_resetrn)
    begin
      Data_out <= 16'h0000;
      Write_once_status <= 1'b0;
    end
  else if (write & ~Write_once_status)
    begin
      Data_out <= Data_in & 16'hFFFE;
      Write_once_status <= 1'b1; // Write once status set on first write, independent of input
    end
  else if (~write)
    begin
      Data_out[15:1] <= Data_out[15:1];
      Data_out[0] <= Write_once_status;
    end
end
endmodule
```



```

begin
  Data_out[15:1] <= Data_out[15:1];
  Data_out[0] <= Write_once_status;
end
endmodule

```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
680	Exploitation of Improperly Controlled Registers

CWE-1229: Creation of Emergent Resource

Weakness ID : 1229

Structure : Simple

Abstraction : Class

Description

The product manages resources or behaves in a way that indirectly creates a new, distinct resource that can be used by attackers in violation of the intended policy.

Extended Description

A product is only expected to behave in a way that was specifically intended by the developer. Resource allocation and management is expected to be performed explicitly by the associated code. However, in systems with complex behavior, the product might indirectly produce new kinds of resources that were never intended in the original design. For example, a covert channel is a resource that was never explicitly intended by the developer, but it is useful to attackers. "Parasitic computing," while not necessarily malicious in nature, effectively tricks a product into performing unintended computations on behalf of another 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		664	Improper Control of a Resource Through its Lifetime	1463
ParentOf		514	Covert Channel	1227

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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-1049]Wikipedia. "Parasitic computing". < https://en.wikipedia.org/wiki/Parasitic_computing >.

CWE-1230: Exposure of Sensitive Information Through Metadata

Weakness ID : 1230

Structure : Simple

Abstraction : Base

Description

The product prevents direct access to a resource containing sensitive information, but it does not sufficiently limit access to metadata that is derived from the original, sensitive information.


Extended Description

Developers might correctly prevent unauthorized access to a database or other resource containing sensitive information, but they might not consider that portions of the original information might also be recorded in metadata, search indices, statistical reports, or other resources. If these resources are not also restricted, then attackers might be able to extract some or all of the original information, or otherwise infer some details. For example, an attacker could specify search terms that are known to be unique to a particular person, or view metadata such as activity or creation dates in order to identify usage patterns.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		285	Improper Authorization	691
ParentOf		202	Exposure of Sensitive Information Through Data Queries	523
ParentOf		612	Improper Authorization of Index Containing Sensitive Information	1379

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2497
MemberOf		199	Information Management Errors	2333

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1396	Comprehensive Categorization: Access Control	1400	2540

CWE-1231: Improper Prevention of Lock Bit Modification

Weakness ID : 1231

Structure : Simple

Abstraction : Base

Description

The product uses a trusted lock bit for restricting access to registers, address regions, or other resources, but the product does not prevent the value of the lock bit from being modified after it has been set.

Extended Description

In integrated circuits and hardware intellectual property (IP) cores, device configuration controls are commonly programmed after a device power reset by a trusted firmware or software module (e.g., BIOS/bootloader) and then locked from any further modification.

This behavior is commonly implemented using a trusted lock bit. When set, the lock bit disables writes to a protected set of registers or address regions. Design or coding errors in the implementation of the lock bit protection feature may allow the lock bit to be modified or cleared by software after it has been set. Attackers might be able to unlock the system and features that the bit is intended to protect.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	IP	284	Improper Access Control	687

Weakness Ordinalities

Primary :

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
Access Control	Modify Memory	High

Scope	Impact	Likelihood
	Registers protected by lock bit can be modified even when lock is set.	

Detection Methods

Manual Analysis

Set the lock bit. Power cycle the device. Attempt to clear the lock bit. If the information is changed, implement a design fix. Retest. Also, attempt to indirectly clear the lock bit or bypass it.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security lock bit protections must be reviewed for design inconsistency and common weaknesses. Security lock programming flow and lock properties must be tested in pre-silicon and post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider the example design below for a digital thermal sensor that detects overheating of the silicon and triggers system shutdown. The system critical temperature limit (CRITICAL_TEMP_LIMIT) and thermal sensor calibration (TEMP_SENSOR_CALIB) data have to be programmed by firmware, and then the register needs to be locked (TEMP_SENSOR_LOCK).

Example Language: Other

(Bad)

In this example, note that if the system heats to critical temperature, the response of the system is controlled by the TEMP_HW_SHUTDOWN bit [1], which is not lockable. Thus, the intended security property of the critical temperature sensor cannot be fully protected, since software can misconfigure the TEMP_HW_SHUTDOWN register even after the lock bit is set to disable the shutdown response.

Example Language:

(Good)

To fix this weakness, one could change the TEMP_HW_SHUTDOWN field to be locked by TEMP_SENSOR_LOCK.

Example 2:

The following example code is a snippet from the register locks inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1350]. Register locks help prevent SoC peripherals' registers from malicious use of resources. The registers that can potentially leak secret data are locked by register locks.

In the vulnerable code, the reglk_mem is used for locking information. If one of its bits toggle to 1, the corresponding peripheral's registers will be locked. In the context of the HACK@DAC System-on-Chip (SoC), it is pertinent to note the existence of two distinct categories of reset signals.

First, there is a global reset signal denoted as "rst_ni," which possesses the capability to simultaneously reset all peripherals to their respective initial states.

Second, we have peripheral-specific reset signals, such as "rst_9," which exclusively reset individual peripherals back to their initial states. The administration of these reset signals is the responsibility of the reset controller module.

Example Language: Verilog

(Bad)

```
always @(posedge clk_i)
begin
    if(~(rst_ni && ~jtag_unlock && ~rst_9))
        begin
            for (j=0; j < 6; j=j+1) begin
                reglk_mem[j] <= 'h0;
            end
        end
end
...
```

In the buggy SoC architecture during HACK@DAC'21, a critical issue arises within the reset controller module. Specifically, the reset controller can inadvertently transmit a peripheral reset signal to the register lock within the user privilege domain.

This unintentional action can result in the reset of the register locks, potentially exposing private data from all other peripherals, rendering them accessible and readable.

To mitigate the issue, remove the extra reset signal rst_9 from the register lock if condition. [REF-1351]

Example Language: Verilog

(Good)





```
always @(posedge clk_i)
begin
    if(~(rst_ni && ~jtag_unlock))
        begin
            for (j=0; j < 6; j=j+1) begin
                reglk_mem[j] <= 'h0;
            end
        end
end
...
```

Observed Examples

Reference	Description
CVE-2017-6283	chip reset clears critical read/write lock permissions for RSA function https://www.cve.org/CVERecord?id=CVE-2017-6283

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2613
MemberOf		1372	ICS Supply Chain: OT Counterfeit and Malicious Corruption	1358	2530
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
680	Exploitation of Improperly Controlled Registers

References

[REF-1350]"reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L80C1-L80C48 >.2023-09-18.

[REF-1351]"fix cwe 1199 in reglk". 2023. < <https://github.com/HACK-EVENT/hackatdac21/commit/5928add42895b57341ae8fc1f9b8351c35aed865#diff-1c2b09dd092a56e5fb2be431a3849e72ff489d2ae> >.2023-09-18.

CWE-1232: Improper Lock Behavior After Power State Transition

Weakness ID : 1232

Structure : Simple

Abstraction : Base

Description

Register lock bit protection disables changes to system configuration once the bit is set. Some of the protected registers or lock bits become programmable after power state transitions (e.g., Entry and wake from low power sleep modes) causing the system configuration to be changeable.

Extended Description

Devices may allow device configuration controls which need to be programmed after device power reset via a trusted firmware or software module (commonly set by BIOS/bootloader) and then locked from any further modification. This action is commonly implemented using a programmable lock bit, which, when set, disables writes to a protected set of registers or address regions.

After a power state transition, the lock bit is set to unlocked. Some common weaknesses that can exist in such a protection scheme are that the lock gets cleared, the values of the protected registers get reset, or the lock become programmable.

Relationships

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

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
Access Control	Modify Memory	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security Lock bit protections should be reviewed for behavior across supported power state transitions. Security lock programming flow and lock properties should be tested in pre-silicon and post-silicon testing including testing across power transitions.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider the memory configuration settings of a system that uses DDR3 DRAM memory. Protecting the DRAM memory configuration from modification by software is required to ensure that system memory access control protections cannot be bypassed. This can be done by using lock bit protection that locks all of the memory configuration registers. The memory configuration lock can be set by the BIOS during the boot process.

If such a system also supports a rapid power on mode like hibernate, the DRAM data must be saved to a disk before power is removed and restored back to the DRAM once the system powers back up and before the OS resumes operation after returning from hibernate.

To support the hibernate transition back to the operating state, the DRAM memory configuration must be reprogrammed even though it was locked previously. As the hibernate resume does a partial reboot, the memory configuration could be altered before the memory lock is set. Functionally the hibernate resume flow requires a bypass of the lock-based protection. The memory configuration must be securely stored and restored by trusted system firmware. Lock settings and system configuration must be restored to the same state it was in before the device entered into the hibernate mode.

Example 2:

The example code below is taken from the register lock module (reglk_wrapper) of the Hack@DAC'21 buggy OpenPiton System-on-Chip (SoC). Upon powering on, most of the silicon registers are initially unlocked. However, critical resources must be configured and locked by setting the lock bit in a register.

In this module, a set of six memory-mapped I/O registers (reglk_mem) is defined and maintained to control the access control of registers inside different peripherals in the SoC [REF-1432]. Each bit represents a register's read/write ability or sets of registers inside a peripheral. Setting improper lock values after system power transition or system rest would make a temporary window for the attackers to read unauthorized data, e.g., secret keys from the crypto engine, and write illegitimate data to critical registers, e.g., framework data. Furthermore, improper register lock values can also result in DoS attacks.

In this faulty implementation, the locks are disabled, i.e., initialized to zero, at reset instead of setting them to their appropriate values [REF-1433]. Improperly initialized locks might allow unauthorized access to sensitive registers, compromising the system's security.

Example Language: Verilog

(Bad)

```
module reglk_wrapper #(
...
  always @(posedge clk_i)
    begin
      if(~(rst_ni && ~jtag_unlock && ~rst_9))
        begin
          for (j=0; j < 6; j=j+1) begin
            reglk_mem[j] <= 'h0;
          end
        end
      end
    ...
endmodule
```

To resolve this issue, it is crucial to ensure that register locks are correctly initialized during the reset phase of the SoC. Correct initialization values should be established to maintain the system's integrity, security, and predictable behavior and allow for proper control of peripherals. The specifics of initializing register locks and their values depend on the SoC's design and the system's requirements; for example, access to all registers through the user privilege level should be locked at reset. To address the problem depicted in the bad code example [REF-1433], the default value for "reglk_mem" should be set to 32'hFFFFFFF. This ensures that access to protected data is restricted during power state transition or after reset until the system state transition is complete and security procedures have properly configured the register locks.

Example Language: Verilog

(Good)

```
module reglk_wrapper #(
...
  always @(posedge clk_i)
  begin
    if(~(rst_ni && ~jtag_unlock && ~rst_9))
    begin
      for (j=0; j < 6; j=j+1) begin
        reglk_mem[j] <= 'hfffffff;
      end
    end
  end
...
)
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2494
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
166	Force the System to Reset Values

References

[REF-1432]"reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/65d0ffdab7426da4509c98d62e163bcce642f651/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L39C1-L39C1 >.

[REF-1433]"Bad Code reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/65d0ffdab7426da4509c98d62e163bcce642f651/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L78C1-L85C16 >.

[REF-1434]"Good Code reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/5e2031fd3854bcc0b2ca11d13442542dd5ea98e0/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L83 >.

CWE-1233: Security-Sensitive Hardware Controls with Missing Lock Bit Protection

Weakness ID : 1233
Structure : Simple
Abstraction : Base

Description

The product uses a register lock bit protection mechanism, but it does not ensure that the lock bit prevents modification of system registers or controls that perform changes to important hardware system configuration.

Extended Description

Integrated circuits and hardware intellectual properties (IPs) might provide device configuration controls that need to be programmed after device power reset by a trusted firmware or software module, commonly set by BIOS/bootloader. After reset, there can be an expectation that the controls cannot be used to perform any further modification. This behavior is commonly implemented using a trusted lock bit, which can be set to disable writes to a protected set of registers or address regions. The lock protection is intended to prevent modification of certain system configuration (e.g., memory/memory protection unit configuration).

However, if the lock bit does not effectively write-protect all system registers or controls that could modify the protected system configuration, then an adversary may be able to use software to access the registers/controls and modify the protected hardware configuration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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
ChildOf		284	Improper Access Control	687

Weakness Ordinalities

Primary :

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
Access Control	Modify Memory <i>System Configuration protected by the lock bit can be modified even when the lock is set.</i>	

Detection Methods

Manual Analysis

Set the lock bit. Attempt to modify the information protected by the lock bit. If the information is changed, implement a design fix. Retest. Also, attempt to indirectly clear the lock bit or bypass it.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation**Phase: Testing**

Security lock bit protections must be reviewed for design inconsistency and common weaknesses. Security lock programming flow and lock properties must be tested in pre-silicon and post-silicon testing.

Demonstrative Examples**Example 1:**

Consider the example design below for a digital thermal sensor that detects overheating of the silicon and triggers system shutdown. The system critical temperature limit (CRITICAL_TEMP_LIMIT) and thermal sensor calibration (TEMP_SENSOR_CALIB) data have to be programmed by the firmware.

Example Language: Other

(Bad)

In this example note that only the CRITICAL_TEMP_LIMIT register is protected by the TEMP_SENSOR_LOCK bit, while the security design intent is to protect any modification of the critical temperature detection and response.

The response of the system, if the system heats to a critical temperature, is controlled by TEMP_HW_SHUTDOWN bit [1], which is not lockable. Also, the TEMP_SENSOR_CALIB register is not protected by the lock bit.

By modifying the temperature sensor calibration, the conversion of the sensor data to a degree centigrade can be changed, such that the current temperature will never be detected to exceed critical temperature value programmed by the protected lock.

Similarly, by modifying the TEMP_HW_SHUTDOWN.Enable bit, the system response detection of the current temperature exceeding critical temperature can be disabled.

Example Language:

(Good)

Change TEMP_HW_SHUTDOWN and TEMP_SENSOR_CALIB controls to be locked by TEMP_SENSOR_LOCK.

Observed Examples

Reference	Description
CVE-2018-9085	Certain servers leave a write protection lock bit unset after boot, potentially allowing modification of parts of flash memory. https://www.cve.org/CVERecord?id=CVE-2018-9085
CVE-2014-8273	Chain: chipset has a race condition (CWE-362) between when an interrupt handler detects an attempt to write-enable the BIOS (in violation of the lock bit), and when the handler resets the write-enable bit back to 0, allowing attackers to issue BIOS writes during the timing window [REF-1237]. https://www.cve.org/CVERecord?id=CVE-2014-8273

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2492

Nature	Type	ID	Name	V	Page
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2613
MemberOf	C	1372	ICS Supply Chain: OT Counterfeit and Malicious Corruption	1358	2530
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
176	Configuration/Environment Manipulation
680	Exploitation of Improperly Controlled Registers

References

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

CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks

Weakness ID : 1234

Structure : Simple

Abstraction : Base

Description

System configuration protection may be bypassed during debug mode.

Extended Description

Device configuration controls are commonly programmed after a device power reset by a trusted firmware or software module (e.g., BIOS/bootloader) and then locked from any further modification. This is commonly implemented using a trusted lock bit, which when set, disables writes to a protected set of registers or address regions. The lock protection is intended to prevent modification of certain system configuration (e.g., memory/memory protection unit configuration). If debug features supported by hardware or internal modes/system states are supported in the hardware design, modification of the lock protection may be allowed allowing access and modification of configuration 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	667	Improper Locking	1472

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
Access Control	Bypass Protection Mechanism <i>Bypass of lock bit allows access and modification of system configuration even when the lock bit is set.</i>	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security Lock bit protections should be reviewed for any bypass/override modes supported. Any supported override modes either should be removed or protected using authenticated debug modes. Security lock programming flow and lock properties should be tested in pre-silicon and post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider the example Locked_override_register example. This register module supports a lock mode that blocks any writes after lock is set to 1. However, it also allows override of the lock protection when scan_mode or debug_unlocked modes are active.

Example Language: Verilog

(Bad)

```

module Locked_register_example
(
  input [15:0] Data_in,
  input Clk,
  input resetn,
  input write,
  input Lock,
  input scan_mode,
  input debug_unlocked,
  output reg [15:0] Data_out
);
reg lock_status;
always @(posedge Clk or negedge resetn)
  if (~resetn) // Register is reset resetn
    begin
      lock_status <= 1'b0;
    end
  else if (Lock)
    begin
      lock_status <= 1'b1;
    end
  else if (~Lock)
    begin
      lock_status <= lock_status
    end
always @(posedge Clk or negedge resetn)
  if (~resetn) // Register is reset resetn
    begin
      Data_out <= 16'h0000;
    end
  else if (write & (~lock_status | scan_mode | debug_unlocked) ) // Register protected by Lock bit input, overrides supported for scan_mode & debug_unlocked
    begin
      Data_out <= Data_in;
    end
  else if (~write)
    begin

```

```
Data_out <= Data_out;
end
endmodule
```

If either the scan_mode or the debug_unlocked modes can be triggered by software, then the lock protection may be bypassed.

Example Language:

(Good)

Either remove the debug and scan mode overrides or protect enabling of these modes so that only trusted and authorized users may enable these modes.

Example 2:

The following example code [REF-1375] is taken from the register lock security peripheral of the HACK@DAC'21 buggy OpenPiton SoC. It demonstrates how to lock read or write access to security-critical hardware registers (e.g., crypto keys, system integrity code, etc.). The configuration to lock all the sensitive registers in the SoC is managed through the reglk_mem registers. These reglk_mem registers are reset when the hardware powers up and configured during boot up. Malicious users, even with kernel-level software privilege, do not get access to the sensitive contents that are locked down. Hence, the security of the entire system can potentially be compromised if the register lock configurations are corrupted or if the register locks are disabled.

Example Language: Verilog

(Bad)

```
...
always @(posedge clk_i)
begin
    if(~(rst_ni && ~jtag_unlock && ~rst_9))
    begin
        for (j=0; j < 6; j=j+1) begin
            reglk_mem[j] <= 'h0;
        end
    end
end
...
```

The example code [REF-1375] illustrates an instance of a vulnerable implementation of register locks in the SoC. In this flawed implementation [REF-1375], the reglk_mem registers are also being reset when the system enters debug mode (indicated by the jtag_unlock signal). Consequently, users can simply put the processor in debug mode to access sensitive contents that are supposed to be protected by the register lock feature.

This can be mitigated by excluding debug mode signals from the reset logic of security-critical register locks as demonstrated in the following code snippet [REF-1376].




Example Language: Verilog

(Good)

```
...
always @(posedge clk_i)
begin
    if(~(rst_ni && ~rst_9))
    begin
        for (j=0; j < 6; j=j+1) begin
            reglk_mem[j] <= 'h0;
        end
    end
end
...
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf		1207	Debug and Test Problems	1194	2495
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2547

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
176	Configuration/Environment Manipulation

References

[REF-1375]"reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/cde1d9d6888bffb21d4b405ccef61b19c58dd3c/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L80C1-L80C48 >.2023-12-13.

[REF-1376]"Fix for reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/20238068b385d7ab704cabfb95ff95dd6e56e1c2/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L80 >.2023-12-13.

CWE-1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations

Weakness ID : 1235

Structure : Simple

Abstraction : Base

Description

The code uses boxed primitives, which may introduce inefficiencies into performance-critical operations.

Extended Description

Languages such as Java and C# support automatic conversion through their respective compilers from primitive types into objects of the corresponding wrapper classes, and vice versa. For example, a compiler might convert an int to Integer (called autoboxing) or an Integer to int (called unboxing). This eliminates forcing the programmer to perform these conversions manually, which makes the code cleaner.

However, this feature comes at a cost of performance and can lead to resource exhaustion and impact availability when used with generic collections. Therefore, they should not be used for scientific computing or other performance critical operations. They are only suited to support "impedance mismatch" between reference types and primitives.


Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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 "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2443

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Language : C# (*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
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) Reduce Performance <i>Incorrect autoboxing/unboxing would result in reduced performance, which sometimes can lead to resource consumption issues.</i>	Low

Potential Mitigations

Phase: Implementation

Use of boxed primitives should be limited to certain situations such as when calling methods with typed parameters. Examine the use of boxed primitives prior to use. Use SparseArrays or ArrayMap instead of HashMap to avoid performance overhead.

Demonstrative Examples

Example 1:

Java has a boxed primitive for each primitive type. A long can be represented with the boxed primitive Long. Issues arise where boxed primitives are used when not strictly necessary.

Example Language: Java

(Bad)

```
Long count = 0L;
for (long i = 0; i < Integer.MAX_VALUE; i++) {
    count += i;
}
```

In the above loop, we see that the count variable is declared as a boxed primitive. This causes autoboxing on the line that increments. This causes execution to be magnitudes less performant (time and possibly space) than if the "long" primitive was used to declare the count variable, which can impact availability of a resource.

Example 2:

This code uses primitive long which fixes the issue.

Example Language: Java

(Good)

```
long count = 0L;
for (long i = 0; i < Integer.MAX_VALUE; i++) {
    count += i;
}
```

MemberOf Relationships

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

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Oracle Coding Standard for Java	EXP04-J		Do not pass arguments to certain Java Collections Framework methods that are a different type than the collection parameter type
ISA/IEC 62443	Part 4-1		Req SI-2

References

[REF-1051]"Oracle Java Documentation". < <https://docs.oracle.com/javase/1.5.0/docs/guide/language/autoboxing.html> >.

[REF-1052]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 02. Expressions (EXP)". < <https://wiki.sei.cmu.edu/confluence/display/java/EXP04-J.+Do+not+pass+arguments+to+certain+Java+Collections+Framework+methods+that+are+a+different+type+than+the+collection+parameter+type> >.

CWE-1236: Improper Neutralization of Formula Elements in a CSV File

Weakness ID : 1236

Structure : Simple

Abstraction : Base

Description

The product saves user-provided information into a Comma-Separated Value (CSV) file, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as a command when the file is opened by a spreadsheet product.

Extended Description

User-provided data is often saved to traditional databases. This data can be exported to a CSV file, which allows users to read the data using spreadsheet software such as Excel, Numbers, or Calc. This software interprets entries beginning with '=' as formulas, which are then executed by the spreadsheet software. The software's formula language often allows methods to access hyperlinks or the local command line, and frequently allows enough characters to invoke an entire script. Attackers can populate data fields which, when saved to a CSV file, may attempt information exfiltration or other malicious activity when automatically executed by the spreadsheet software.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	138

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

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	138

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2332

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Other (*Prevalence = Undetermined*)

Alternate Terms

CSV Injection :

Formula Injection :

Excel Macro Injection :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Execute Unauthorized Code or Commands <i>Current versions of Excel warn users of untrusted content.</i>	Low

Potential Mitigations

Phase: Implementation

When generating CSV output, ensure that formula-sensitive metacharacters are effectively escaped or removed from all data before storage in the resultant CSV. Risky characters include '=' (equal), '+' (plus), '-' (minus), and '@' (at).

Effectiveness = Moderate

Unfortunately, there is no perfect solution, since different spreadsheet products act differently.

Phase: Implementation

If a field starts with a formula character, prepend it with a ' (single apostrophe), which prevents Excel from executing the formula.

Effectiveness = Moderate

It is not clear how effective this mitigation is with other spreadsheet software.

Phase: Architecture and Design

Certain implementations of spreadsheet software might disallow formulas from executing if the file is untrusted, or if the file is not authored by the current user.

Effectiveness = Limited

This mitigation has limited effectiveness because it often depends on end users opening spreadsheet software safely.

Demonstrative Examples

Example 1:

Hyperlinks or other commands can be executed when a cell begins with the formula identifier, '='

Example Language: Other

(Attack)

```
=HYPERLINK(link_location, [friendly_name])
```

Stripping the leading equals sign, or simply not executing formulas from untrusted sources, impedes malicious activity.

Example Language:

(Good)

```
HYPERLINK(link_location, [friendly_name])
```

Observed Examples

Reference	Description
CVE-2019-12134	Low privileged user can trigger CSV injection through a contact form field value https://www.cve.org/CVERecord?id=CVE-2019-12134
CVE-2019-4521	Cloud management product allows arbitrary command execution via CSV injection https://www.cve.org/CVERecord?id=CVE-2019-4521
CVE-2019-17661	CSV injection in content management system via formula code in a first or last name https://www.cve.org/CVERecord?id=CVE-2019-17661

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1409	Comprehensive Categorization: Injection	1400	2556

References

- [REF-21]OWASP. "CSV Injection". 2020 February 2. < https://owasp.org/www-community/attacks/CSV_Injection >.
- [REF-22]Jamie Rougvie. "Data Extraction to Command Execution CSV Injection". 2019 September 6. < <https://www.veracode.com/blog/secure-development/data-extraction-command-execution-csv-injection> >.
- [REF-23]George Mauer. "The Absurdly Underestimated Dangers of CSV Injection". 2017 October 7. < <http://georgemauer.net/2017/10/07/csv-injection.html> >.
- [REF-24]James Kettle. "Comma Separated Vulnerabilities". 2014 August 9. < <https://rstforums.com/forum/topic/82690-comma-separated-vulnerabilities/> >.2023-04-07.

CWE-1239: Improper Zeroization of Hardware Register

Weakness ID : 1239
Structure : Simple
Abstraction : Variant

Description

The hardware product does not properly clear sensitive information from built-in registers when the user of the hardware block changes.


Extended Description

Hardware logic operates on data stored in registers local to the hardware block. Most hardware IPs, including cryptographic accelerators, rely on registers to buffer I/O, store intermediate values, and interface with software. The result of this is that sensitive information, such as passwords or encryption keys, can exist in locations not transparent to the user of the hardware logic. When a different entity obtains access to the IP due to a change in operating mode or conditions, the new entity can extract information belonging to the previous user if no mechanisms are in place to clear register contents. It is important to clear information stored in the hardware if a physical attack on the product is detected, or if the user of the hardware block changes. The process of clearing register contents in a hardware IP is referred to as zeroization in standards for cryptographic hardware modules such as FIPS-140-2 [REF-267].


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		226	Sensitive Information in Resource Not Removed Before Reuse	569

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf		226	Sensitive Information in Resource Not Removed Before Reuse	569

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context <i>The consequences will depend on the information disclosed due to the vulnerability.</i>	

Potential Mitigations

Phase: Architecture and Design

Every register potentially containing sensitive information must have a policy specifying how and when information is cleared, in addition to clarifying if it is the responsibility of the hardware logic or IP user to initiate the zeroization procedure at the appropriate time.

Demonstrative Examples

Example 1:

Suppose a hardware IP for implementing an encryption routine works as expected, but it leaves the intermediate results in some registers that can be accessed. Exactly why this access happens is immaterial - it might be unintentional or intentional, where the designer wanted a "quick fix" for something.

Example 2:

The example code below [REF-1379] is taken from the SHA256 Interface/wrapper controller module of the HACK@DAC'21 buggy OpenPiton SoC. Within the wrapper module there are a set of 16 memory-mapped registers referenced data[0] to data[15]. These registers are 32 bits in size and are used to store the data received on the AXI Lite interface for hashing. Once both the message to be hashed and a request to start the hash computation are received, the values of these registers will be forwarded to the underlying SHA256 module for processing. Once forwarded, the values in these registers no longer need to be retained. In fact, if not cleared or overwritten, these sensitive values can be read over the AXI Lite interface, potentially compromising any previously confidential data stored therein.

Example Language: Verilog

(Bad)

```
...
// Implement SHA256 I/O memory map interface
// Write side
always @(posedge clk_i)
begin
    if(~(rst_ni && ~rst_3))
    begin
        startHash <= 0;
        newMessage <= 0;
        data[0] <= 0;
        data[1] <= 0;
        data[2] <= 0;
        ...
        data[14] <= 0;
        data[15] <= 0;
    end
end
...
```

In the previous code snippet [REF-1379] there is the lack of a data clearance mechanism for the memory-mapped I/O registers after their utilization. These registers get cleared only when a reset condition is met. This condition is met when either the global negative-edge reset input signal (rst_ni) or the dedicated reset input signal for SHA256 peripheral (rst_3) is active. In other words, if either of these reset signals is true, the registers will be cleared. However, in cases where there is not a reset condition these registers retain their values until the next hash operation. It is during the time between an old hash operation and a new hash operation that that data is open to unauthorized disclosure.

To correct the issue of data persisting between hash operations, the memory mapped I/O registers need to be cleared once the values written in these registers are propagated to the SHA256 module. This could be done for example by adding a new condition to zeroize the memory mapped I/O registers once the hash value is computed, i.e., hashValid signal asserted, as shown in the good code example below [REF-1380]. This fix will clear the memory-mapped I/O registers after the data has been provided as input to the SHA engine.

Example Language: Verilog

(Good)

```
...
// Implement SHA256 I/O memory map interface
// Write side
always @(posedge clk_i)
begin
    if(~(rst_ni && ~rst_3))
    begin
        startHash <= 0;
        newMessage <= 0;
        data[0] <= 0;
        data[1] <= 0;
        data[2] <= 0;
        ...
        data[14] <= 0;
        data[15] <= 0;
    end
end
end
```

```
else if(hashValid && ~hashValid_r)
begin
    data[0] <= 0;
    data[1] <= 0;
    data[2] <= 0;
    ...
    data[14] <= 0;
    data[15] <= 0;
end
...
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data
150	Collect Data from Common Resource Locations
204	Lifting Sensitive Data Embedded in Cache
545	Pull Data from System Resources

References

- [REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf> >.2023-04-07.
- [REF-1055]Peter Gutmann. "Data Remanence in Semiconductor Devices". 10th USENIX Security Symposium. 2001 August. < https://www.usenix.org/legacy/events/sec01/full_papers/gutmann/gutmann.pdf >.
- [REF-1379]"sha256_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/sha256/sha256_wrapper.sv#L94-L116 >.2023-12-13.
- [REF-1380]"Fix for sha256_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/e8ba396b5c7cec9031e0e0e18ac547f32cd0ed50/piton/design/chip/tile/ariane/src/sha256/sha256_wrapper.sv#L98C1-L139C18 >.2023-12-13.

CWE-1240: Use of a Cryptographic Primitive with a Risky Implementation

Weakness ID : 1240

Structure : Simple

Abstraction : Base

Description

To fulfill the need for a cryptographic primitive, the product implements a cryptographic algorithm using a non-standard, unproven, or disallowed/non-compliant cryptographic implementation.

Extended Description

Cryptographic protocols and systems depend on cryptographic primitives (and associated algorithms) as their basic building blocks. Some common examples of primitives are digital signatures, one-way hash functions, ciphers, and public key cryptography; however, the notion of

"primitive" can vary depending on point of view. See "Terminology Notes" for further explanation of some concepts.

Cryptographic primitives are defined to accomplish one very specific task in a precisely defined and mathematically reliable fashion. For example, suppose that for a specific cryptographic primitive (such as an encryption routine), the consensus is that the primitive can only be broken after trying out N different inputs (where the larger the value of N, the stronger the cryptography). For an encryption scheme like AES-256, one would expect N to be so large as to be infeasible to execute in a reasonable amount of time.

If a vulnerability is ever found that shows that one can break a cryptographic primitive in significantly less than the expected number of attempts, then that primitive is considered weakened (or sometimes in extreme cases, colloquially it is "broken"). As a result, anything using this cryptographic primitive would now be considered insecure or risky. Thus, even breaking or weakening a seemingly small cryptographic primitive has the potential to render the whole system vulnerable, due to its reliance on the primitive. A historical example can be found in TLS when using DES. One would colloquially call DES the cryptographic primitive for transport encryption in this version of TLS. In the past, DES was considered strong, because no weaknesses were found in it; importantly, DES has a key length of 56 bits. Trying $N=2^{56}$ keys was considered impractical for most actors. Unfortunately, attacking a system with 56-bit keys is now practical via brute force, which makes defeating DES encryption practical. It is now practical for an adversary to read any information sent under this version of TLS and use this information to attack the system. As a result, it can be claimed that this use of TLS is weak, and that any system depending on TLS with DES could potentially render the entire system vulnerable to attack.

Cryptographic primitives and associated algorithms are only considered safe after extensive research and review from experienced cryptographers from academia, industry, and government entities looking for any possible flaws. Furthermore, cryptographic primitives and associated algorithms are frequently reevaluated for safety when new mathematical and attack techniques are discovered. As a result and over time, even well-known cryptographic primitives can lose their compliance status with the discovery of novel attacks that might either defeat the algorithm or reduce its robustness significantly.


If ad-hoc cryptographic primitives are implemented, it is almost certain that the implementation will be vulnerable to attacks that are well understood by cryptographers, resulting in the exposure of sensitive information and other consequences.

This weakness is even more difficult to manage for hardware-implemented deployment of cryptographic algorithms. First, because hardware is not patchable as easily as software, any flaw discovered after release and production typically cannot be fixed without a recall of the product. Secondly, the hardware product is often expected to work for years, during which time computation power available to the attacker only increases. Therefore, for hardware implementations of cryptographic primitives, it is absolutely essential that only strong, proven cryptographic primitives are 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		327	Use of a Broken or Risky Cryptographic Algorithm	806

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

Nature	Type	ID	Name	Page
MemberOf		310	Cryptographic Issues	2339

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>Incorrect usage of crypto primitives could render the supposedly encrypted data as unencrypted plaintext in the worst case.</i>	High

Detection Methods

Architecture or Design Review

Review requirements, documentation, and product design to ensure that primitives are consistent with the strongest-available recommendations from trusted parties. If the product appears to be using custom or proprietary implementations that have not had sufficient public review and approval, then this is a significant concern.

Effectiveness = High

Manual Analysis

Analyze the product to ensure that implementations for each primitive do not contain any known vulnerabilities and are not using any known-weak algorithms, including MD4, MD5, SHA1, DES, etc.

Effectiveness = Moderate

Dynamic Analysis with Manual Results Interpretation

For hardware, during the implementation (pre-Silicon / post-Silicon) phase, dynamic tests should be done to ensure that outputs from cryptographic routines are indeed working properly, such as test vectors provided by NIST [REF-1236].

Effectiveness = Moderate

Dynamic Analysis with Manual Results Interpretation

It needs to be determined if the output of a cryptographic primitive is lacking entropy, which is one clear sign that something went wrong with the crypto implementation. There exist many methods of measuring the entropy of a bytestream, from sophisticated ones (like calculating Shannon's entropy of a sequence of characters) to crude ones (by compressing it and comparing the size of the original bytestream vs. the compressed - a truly random byte stream should not be compressible and hence the uncompressed and compressed bytestreams should be nearly identical in size).

Effectiveness = Moderate

Potential Mitigations

Phase: Requirements

Require compliance with the strongest-available recommendations from trusted parties, and require that compliance must be kept up-to-date, since recommendations evolve over time. For example, US government systems require FIPS 140-3 certification, which supersedes FIPS 140-2 [REF-1192] [REF-1226].

Effectiveness = High

Phase: Architecture and Design

Ensure that the architecture/design uses the strongest-available primitives and algorithms from trusted parties. For example, US government systems require FIPS 140-3 certification, which supersedes FIPS 140-2 [REF-1192] [REF-1226].

Effectiveness = High

Phase: Architecture and Design

Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. As with all cryptographic mechanisms, the source code should be available for analysis. If the algorithm may be compromised when attackers find out how it works, then it is especially weak.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Try not to use cryptographic algorithms in novel ways or with new modes of operation even when you "know" it is secure. For example, using SHA-2 chaining to create a 1-time pad for encryption might sound like a good idea, but one should not do this.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Ensure that the design can replace one cryptographic primitive or algorithm with another in the next generation ("cryptographic agility"). Where possible, use wrappers to make the interfaces uniform. This will make it easier to upgrade to stronger algorithms. This is especially important for hardware, which can be more difficult to upgrade quickly than software; design the hardware at a replaceable block level.

Effectiveness = Defense in Depth

Phase: Architecture and Design

Do not use outdated or non-compliant cryptography algorithms. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong [REF-267].

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Phase: Implementation

Do not use a linear-feedback shift register (LFSR) or other legacy methods as a substitute for an accepted and standard Random Number Generator.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Phase: Implementation

Do not use a checksum as a substitute for a cryptographically generated hash.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted cryptographic library or framework. Industry-standard implementations will save development time and are more likely to avoid errors that can occur during implementation of cryptographic algorithms. However, the library/framework could be used incorrectly during implementation.

Effectiveness = High

Phase: Architecture and Design

Phase: Implementation

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 the prevention of common attacks.

Effectiveness = Moderate

Phase: Architecture and Design

Phase: Implementation

Do not store keys in areas accessible to untrusted agents. Carefully manage and protect the cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography algorithm is irrelevant.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

Re-using random values may compromise security.

Example Language:

(Bad)

Suppose an Encryption algorithm needs a random value for a key. Instead of using a DRNG (Deterministic Random Number Generator), the designer uses a linear-feedback shift register (LFSR) to generate the value.

While an LFSR may provide pseudo-random number generation service, the entropy (measure of randomness) of the resulting output may be less than that of an accepted DRNG (like that used in dev/urandom). Thus, using an LFSR weakens the strength of the cryptographic system, because it may be possible for an attacker to guess the LFSR output and subsequently the encryption key.

Example Language:

(Good)

If a cryptographic algorithm expects a random number as its input, provide one. Do not provide a pseudo-random value.

Observed Examples

Reference	Description
CVE-2020-4778	software uses MD5, which is less safe than the default SHA-256 used by related products https://www.cve.org/CVERecord?id=CVE-2020-4778
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates. https://www.cve.org/CVERecord?id=CVE-2005-2946
CVE-2019-3907	identity card uses MD5 hash of a salt and password https://www.cve.org/CVERecord?id=CVE-2019-3907
CVE-2021-34687	personal key is transmitted over the network using a substitution cipher https://www.cve.org/CVERecord?id=CVE-2021-34687
CVE-2020-14254	product does not disable TLS-RSA cipher suites, allowing decryption of traffic if TLS 2.0 and secure ciphers are not enabled.

Reference	Description
CVE-2019-1543	https://www.cve.org/CVERecord?id=CVE-2020-14254 SSL/TLS library generates 16-byte nonces but reduces them to 12 byte nonces for the ChaCha20-Poly1305 cipher, converting them in a way that violates the cipher's requirements for unique nonces.
CVE-2017-9267	https://www.cve.org/CVERecord?id=CVE-2019-1543 LDAP interface allows use of weak ciphers
CVE-2017-7971	https://www.cve.org/CVERecord?id=CVE-2017-9267 SCADA product allows "use of outdated cipher suites"
CVE-2020-6616	https://www.cve.org/CVERecord?id=CVE-2017-7971 Chip implementing Bluetooth uses a low-entropy PRNG instead of a hardware RNG, allowing spoofing.
CVE-2019-1715	https://www.cve.org/CVERecord?id=CVE-2020-6616 security product has insufficient entropy in the DRBG, allowing collisions and private key discovery
CVE-2014-4192	https://www.cve.org/CVERecord?id=CVE-2019-1715 Dual_EC_DRBG implementation in RSA toolkit does not correctly handle certain byte requests, simplifying plaintext recovery
CVE-2007-6755	https://www.cve.org/CVERecord?id=CVE-2014-4192 Recommendation for Dual_EC_DRBG algorithm contains point Q constants that could simplify decryption
	https://www.cve.org/CVERecord?id=CVE-2007-6755

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1205	Security Primitives and Cryptography Issues	1194	2494
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2613
MemberOf	C	1402	Comprehensive Categorization: Encryption	1400	2548

Notes

Terminology

Terminology for cryptography varies widely, from informal and colloquial to mathematically-defined, with different precision and formalism depending on whether the stakeholder is a developer, cryptologist, etc. Yet there is a need for CWE to be self-consistent while remaining understandable and acceptable to multiple audiences. As of CWE 4.6, CWE terminology around "primitives" and "algorithms" is emerging as shown by the following example, subject to future consultation and agreement within the CWE and cryptography communities. Suppose one wishes to send encrypted data using a CLI tool such as OpenSSL. One might choose to use AES with a 256-bit key and require tamper protection (GCM mode, for instance). For compatibility's sake, one might also choose the ciphertext to be formatted to the PKCS#5 standard. In this case, the "cryptographic system" would be AES-256-GCM with PKCS#5 formatting. The "cryptographic function" would be AES-256 in the GCM mode of operation, and the "algorithm" would be AES. Colloquially, one would say that AES (and sometimes AES-256) is the "cryptographic primitive," because it is the algorithm that realizes the concept of symmetric encryption (without modes of operation or other protocol related modifications). In practice, developers and architects typically refer to base cryptographic algorithms (AES, SHA, etc.) as cryptographic primitives.

Maintenance

Since CWE 4.4, various cryptography-related entries, including CWE-327 and CWE-1240, have been slated for extensive research, analysis, and community consultation to define consistent terminology, improve relationships, and reduce overlap or duplication. As of CWE 4.6, this work is still ongoing.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
97	Cryptanalysis

References

- [REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf> >.2023-04-07.
- [REF-1227]Wikipedia. "Cryptographic primitive". < https://en.wikipedia.org/wiki/Cryptographic_primitive >.
- [REF-1226]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-2: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < <https://csrc.nist.gov/publications/detail/fips/140/2/final> >.
- [REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < <https://csrc.nist.gov/publications/detail/fips/140/3/final> >.
- [REF-1236]NIST. "CAVP Testing: Individual Component Testing". < <https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/component-testing> >.

CWE-1241: Use of Predictable Algorithm in Random Number Generator

Weakness ID : 1241

Structure : Simple

Abstraction : Base

Description

The device uses an algorithm that is predictable and generates a pseudo-random number.

Extended Description

Pseudo-random number generator algorithms are predictable because their registers have a finite number of possible states, which eventually lead to repeating patterns. As a result, pseudo-random number generators (PRNGs) can compromise their randomness or expose their internal state to various attacks, such as reverse engineering or tampering. It is highly recommended to use hardware-based true random number generators (TRNGs) to ensure the security of encryption schemes. TRNGs generate unpredictable, unbiased, and independent random numbers because they employ physical phenomena, e.g., electrical noise, as sources to generate random numbers.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	821

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

Nature	Type	ID	Name	Page
MemberOf		1213	Random Number Issues	2498

Applicable Platforms

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	High

Potential Mitigations

Phase: Architecture and Design

A true random number generator should be specified for cryptographic algorithms.

Phase: Implementation

A true random number generator should be implemented for cryptographic algorithms.

Demonstrative Examples

Example 1:

Suppose a cryptographic function expects random value to be supplied for the crypto algorithm.

During the implementation phase, due to space constraint, a cryptographically secure random-number-generator could not be used, and instead of using a TRNG (True Random Number Generator), a LFSR (Linear Feedback Shift Register) is used to generate a random value. While an LFSR will provide a pseudo-random number, its entropy (measure of randomness) is insufficient for a cryptographic algorithm.

Example 2:

The example code is taken from the PRNG inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1370]. The SoC implements a pseudo-random number generator using a Linear Feedback Shift Register (LFSR).

An example of LFSR with the polynomial function $P(x) = x^6 + x^4 + x^3 + 1$ is shown in the figure.

Example Language: Verilog

(Bad)

```
reg in_sr, entropy16_valid;
reg [15:0] entropy16;
assign entropy16_o = entropy16;
assign entropy16_valid_o = entropy16_valid;
always @ (*)
begin
    in_sr = ^ (poly_i [15:0] & entropy16 [15:0]);
end
```

A LFSR's input bit is determined by the output of a linear function of two or more of its previous states. Therefore, given a long cycle, a LFSR-based PRNG will enter a repeating cycle, which is predictable.

Observed Examples

Reference	Description
CVE-2021-3692	PHP framework uses mt_rand() function (Marsenne Twister) when generating tokens https://www.cve.org/CVERecord?id=CVE-2021-3692

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	1205	Security Primitives and Cryptography Issues	1194	2494
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2564

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.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
97	Cryptanalysis

References

[REF-1370]"rng_16.v". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/main/piton/design/chip/tile/ariane/src/rand_num/rng_16.v#L12-L22 >.2023-07-15.

CWE-1242: Inclusion of Undocumented Features or Chicken Bits

Weakness ID : 1242

Structure : Simple

Abstraction : Base

Description

The device includes chicken bits or undocumented features that can create entry points for unauthorized actors.

Extended Description

A common design practice is to use undocumented bits on a device that can be used to disable certain functional security features. These bits are commonly referred to as "chicken bits". They can facilitate quick identification and isolation of faulty components, features that negatively affect performance, or features that do not provide the required controllability for debug and test. Another way to achieve this is through implementation of undocumented features. An attacker might exploit these interfaces for unauthorized access.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	687

Applicable Platforms

2044

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

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

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

The implementation of chicken bits in a released product is highly discouraged. If implemented at all, ensure that they are disabled in production devices. All interfaces to a device should be documented.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider a device that comes with various security measures, such as secure boot. The secure-boot process performs firmware-integrity verification at boot time, and this code is stored in a separate SPI-flash device. However, this code contains undocumented "special access features" intended to be used only for performing failure analysis and intended to only be unlocked by the device designer.

Example Language: Other

(Bad)

Attackers dump the code from the device and then perform reverse engineering to analyze the code. The undocumented, special-access features are identified, and attackers can activate them by sending specific commands via UART before secure-boot phase completes. Using these hidden features, attackers can perform reads and writes to memory via the UART interface. At runtime, the attackers can also execute arbitrary code and dump the entire memory contents.

Remove all chicken bits and hidden features that are exposed to attackers. Add authorization schemes that rely on cryptographic primitives to access any features that the manufacturer does not want to expose. Clearly document all interfaces.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1198	Privilege Separation and Access Control Issues	1194	2491
MemberOf		1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2529
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-1		Req SVV-3
ISA/IEC 62443	Part 4-2		Req CR 2.12

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
36	Using Unpublished Interfaces or Functionality
212	Functionality Misuse

References

[REF-1071]Ali Abbasi, Tobias Scharnowski and Thorsten Holz. "Doors of Durin: The Veiled Gate to Siemens S7 Silicon". < <https://i.blackhat.com/eu-19/Wednesday/eu-19-Abbasi-Doors-Of-Durin-The-Veiled-Gate-To-Siemens-S7-Silicon.pdf> >.

[REF-1072]Sergei Skorobogatov and Christopher Woods. "Breakthrough Silicon Scanning Discovers Backdoor in Military Chip". < https://www.cl.cam.ac.uk/~sps32/Silicon_scan_draft.pdf >.

[REF-1073]Chris Domas. "God Mode Unlocked: Hardware Backdoors in x86 CPUs". < <https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPU.pdf> >.

[REF-1074]Jonathan Brossard. "Hardware Backdooring is Practical". < https://media.blackhat.com/bh-us-12/Briefings/Brossard/BH_US_12_Brossard_Backdoor_Hacking_Slides.pdf >.

[REF-1075]Sergei Skorobogatov. "Security, Reliability, and Backdoors". < https://www.cl.cam.ac.uk/~sps32/SG_talk_SRB.pdf >.

CWE-1243: Sensitive Non-Volatile Information Not Protected During Debug

Weakness ID : 1243

Structure : Simple

Abstraction : Base

Description

Access to security-sensitive information stored in fuses is not limited during debug.

Extended Description

Several security-sensitive values are programmed into fuses to be used during early-boot flows or later at runtime. Examples of these security-sensitive values include root keys, encryption keys, manufacturing-specific information, chip-manufacturer-specific information, and original-equipment-manufacturer (OEM) data. After the chip is powered on, these values are sensed from fuses and stored in temporary locations such as registers and local memories. These locations are typically access-control protected from untrusted agents capable of accessing them. Even to trusted agents, only read-access is provided. However, these locations are not blocked during debug operations, allowing a user to access this 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		1263	Improper Physical Access Control	2097

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
Confidentiality	Modify Memory	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Disable access to security-sensitive information stored in fuses directly and also reflected from temporary storage locations when in debug mode.

Demonstrative Examples

Example 1:

Sensitive manufacturing data (such as die information) are stored in fuses. When the chip powers on, these values are read from the fuses and stored in microarchitectural registers. These registers are only given read access to trusted software running on the core. Untrusted software running on the core is not allowed to access these registers.

Example Language: Other

(Bad)

All microarchitectural registers in this chip can be accessed through the debug interface. As a result, even an untrusted debugger can access this data and retrieve sensitive manufacturing data.

Example Language:

(Good)

Registers used to store sensitive values read from fuses should be blocked during debug. These registers should be disconnected from the debug interface.

Example 2:

The example code below is taken from one of the AES cryptographic accelerators of the HACK@DAC'21 buggy OpenPiton SoC [REF-1366]. The operating system (OS) uses three AES keys to encrypt and decrypt sensitive data using this accelerator. These keys are sensitive data stored in fuses. The security of the OS will be compromised if any of these AES keys are leaked. During system bootup, these AES keys are sensed from fuses and stored in temporary hardware registers of the AES peripheral. Access to these temporary registers is disconnected during the debug state to prevent them from leaking through debug access. In this example (see the vulnerable code source), the registers key0, key1, and key2 are used to store the three AES keys (which are accessed through key_big0, key_big1, and key_big2 signals). The OS selects one of these three keys through the key_big signal, which is used by the AES engine.

Example Language: Verilog

(Bad)

```
...
assign key_big0 = debug_mode_i ? 192'b0 : {key0[0],
key0[1], key0[2], key0[3], key0[4], key0[5]};
assign key_big1 = debug_mode_i ? 192'b0 : {key1[0],
key1[1], key1[2], key1[3], key1[4], key1[5]};
assign key_big2 = {key2[0], key2[1], key2[2],
key2[3], key2[4], key2[5]};
```

```
...
assign key_big = key_sel[1] ? key_big2 : ( key_sel[0] ?
key_big1 : key_big0 );
...
```

The above code illustrates an instance of a vulnerable implementation for blocking AES key mechanism when the system is in debug mode (i.e., when `debug_mode_i` is asserted). During debug mode, key accesses through `key_big0` and `key_big1` are effectively disconnected, as their values are set to zero. However, the key accessed via the `key_big2` signal remains accessible, creating a potential pathway for sensitive fuse data leakage, specifically AES key2, during debug mode. Furthermore, even though it is not strictly necessary to disconnect the `key_big` signal when entering debug mode (since disconnecting `key_big0`, `key_big1`, and `key_big2` will inherently disconnect `key_big`), it is advisable, in line with the defense-in-depth strategy, to also sever the connection to `key_big`. This additional security measure adds an extra layer of protection and safeguards the AES keys against potential future modifications to the `key_big` logic.

To mitigate this, disconnect access through `key_big2` and `key_big` during debug mode [REF-1367].

Example Language: Verilog

(Good)

```
...
assign key_big0 = debug_mode_i ? 192'b0 : {key0[0],
key0[1], key0[2], key0[3], key0[4], key0[5]};
assign key_big1 = debug_mode_i ? 192'b0 : {key1[0],
key1[1], key1[2], key1[3], key1[4], key1[5]};
assign key_big2 = debug_mode_i ? 192'b0 : {key2[0],
key2[1], key2[2], key2[3], key2[4], key2[5]};
...
assign key_big = debug_mode_i ? 192'b0 : ( key_sel[1] ?
key_big2 : ( key_sel[0] ? key_big1 : key_big0 ) );
...
```

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	1207	Debug and Test Problems	1194	2495
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
116	Excavation
545	Pull Data from System Resources

References

[REF-1366]"aes0_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L56C1-L57C1 >.2023-07-15.

[REF-1367]"fix cwe_1243 in aes0_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/cde1d9d6888bffb21d4b405cce61b19c58dd3c/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L56 >.2023-09-28.

CWE-1244: Internal Asset Exposed to Unsafe Debug Access Level or State

Weakness ID : 1244

Structure : Simple

2048

Abstraction : Base**Description**

The product uses physical debug or test interfaces with support for multiple access levels, but it assigns the wrong debug access level to an internal asset, providing unintended access to the asset from untrusted debug agents.

Extended Description

Debug authorization can have multiple levels of access, defined such that different system internal assets are accessible based on the current authorized debug level. Other than debugger authentication (e.g., using passwords or challenges), the authorization can also be based on the system state or boot stage. For example, full system debug access might only be allowed early in boot after a system reset to ensure that previous session data is not accessible to the authenticated debugger.

If this protection mechanism does not ensure that internal assets have the correct debug access level during each boot stage or change in system state, an attacker could obtain sensitive information from the internal asset using a debugger.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		863	Incorrect Authorization	1796

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Integrity	Modify Memory	
Authorization	Gain Privileges or Assume Identity	
Access Control	Bypass Protection Mechanism	

Detection Methods**Manual Analysis**

Check 2 devices for their passcode to authenticate access to JTAG/debugging ports. If the passcodes are missing or the same, update the design to fix and retest. Check communications over JTAG/debugging ports for encryption. If the communications are not encrypted, fix the design and retest.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

For security-sensitive assets accessible over debug/test interfaces, only allow trusted agents.

Effectiveness = High

Phase: Architecture and Design

Apply blinding [REF-1219] or masking techniques in strategic areas.

Effectiveness = Limited

Phase: Implementation

Add shielding or tamper-resistant protections to the device, which increases the difficulty and cost for accessing debug/test interfaces.

Effectiveness = Limited

Demonstrative Examples

Example 1:

The JTAG interface is used to perform debugging and provide CPU core access for developers. JTAG-access protection is implemented as part of the JTAG_SHIELD bit in the hw_digctl_ctrl register. This register has no default value at power up and is set only after the system boots from ROM and control is transferred to the user software.

Example Language: Other

(Bad)

This means that since the end user has access to JTAG at system reset and during ROM code execution before control is transferred to user software, a JTAG user can modify the boot flow and subsequently disclose all CPU information, including data-encryption keys.

Example Language:

(Informative)

The default value of this register bit should be set to 1 to prevent the JTAG from being enabled at system reset.

Example 2:

The example code below is taken from the CVA6 processor core of the HACK@DAC'21 buggy OpenPiton SoC. Debug access allows users to access internal hardware registers that are otherwise not exposed for user access or restricted access through access control protocols. Hence, requests to enter debug mode are checked and authorized only if the processor has sufficient privileges. In addition, debug accesses are also locked behind password checkers. Thus, the processor enters debug mode only when the privilege level requirement is met, and the correct debug password is provided.

The following code [REF-1377] illustrates an instance of a vulnerable implementation of debug mode. The core correctly checks if the debug requests have sufficient privileges and enables the debug_mode_d and debug_mode_q signals. It also correctly checks for debug password and enables umode_i signal.

Example Language: Verilog

(Bad)

```
module csr_regfile #(
...
    // check that we actually want to enter debug depending on the privilege level we are currently in
    unique case (priv_lvl_o)
        riscv::PRIV_LVL_M: begin
            debug_mode_d = dcsr_q.ebreakm;
        end
    ...
endmodule
```

```

riscv::PRIV_LVL_U: begin
    debug_mode_d = dcsr_q.ebreaku;
...
    assign priv_lvl_o = (debug_mode_q || umode_i) ? riscv::PRIV_LVL_M : priv_lvl_q;
...
    debug_mode_q <= debug_mode_d;
...

```

However, it grants debug access and changes the privilege level, `priv_lvl_o`, even when one of the two checks is satisfied and the other is not. Because of this, debug access can be granted by simply requesting with sufficient privileges (i.e., `debug_mode_q` is enabled) and failing the password check (i.e., `umode_i` is disabled). This allows an attacker to bypass the debug password checking and gain debug access to the core, compromising the security of the processor.

A fix to this issue is to only change the privilege level of the processor when both checks are satisfied, i.e., the request has enough privileges (i.e., `debug_mode_q` is enabled) and the password checking is successful (i.e., `umode_i` is enabled) [REF-1378].

Example Language: Verilog

(Good)

```

module csr_regfile #(
...
    // check that we actually want to enter debug depending on the privilege level we are currently in
    unique case (priv_lvl_o)
        riscv::PRIV_LVL_M: begin
            debug_mode_d = dcsr_q.ebreakm;
...
        riscv::PRIV_LVL_U: begin
            debug_mode_d = dcsr_q.ebreaku;
...
    assign priv_lvl_o = (debug_mode_q && umode_i) ? riscv::PRIV_LVL_M : priv_lvl_q;
...
    debug_mode_q <= debug_mode_d;
...




```

Observed Examples

Reference	Description
CVE-2019-18827	After ROM code execution, JTAG access is disabled. But before the ROM code is executed, JTAG access is possible, allowing a user full system access. This allows a user to modify the boot flow and successfully bypass the secure-boot process. https://www.cve.org/CVERecord?id=CVE-2019-18827

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2495
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2613
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2540

Notes

Relationship

CWE-1191 and CWE-1244 both involve physical debug access, but the weaknesses are different. CWE-1191 is effectively about missing authorization for a debug interface, i.e. JTAG.

CWE-1244 is about providing internal assets with the wrong debug access level, exposing the asset to untrusted debug agents.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
114	Authentication Abuse

References

[REF-1056]F-Secure Labs. "Multiple Vulnerabilities in Barco Clickshare: JTAG access is not permanently disabled". < <https://labs.f-secure.com/advisories/multiple-vulnerabilities-in-barco-clickshare/> >.

[REF-1057]Kurt Rosenfeld and Ramesh Karri. "Attacks and Defenses for JTAG". < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5406671> >.

[REF-1219]Monodeep Kar, Arvind Singh, Santosh Ghosh, Sanu Mathew, Anand Rajan, Vivek De, Raheem Beyah and Saibal Mukhopadhyay. "Blindsight: Blinding EM Side-Channel Leakage using Built-In Fully Integrated Inductive Voltage Regulator". 2018 February. < <https://arxiv.org/pdf/1802.09096.pdf> >.2023-04-07.

[REF-1377]"csr_regfile.sv line 938". 2021. < https://github.com/HACK-EVENT/hackatdac19/blob/57e7b2109c1ea2451914878df2e6ca740c2dcf34/src/csr_regfile.sv#L938 >.2023-12-13.

[REF-1378]"Fix for csr_regfile.sv line 938". 2021. < https://github.com/HACK-EVENT/hackatdac19/blob/a7b61209e56c48eec585eeede8413997ec71e4a/src/csr_regfile.sv#L938C31-L938C56 >.2023-12-13.

CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic

Weakness ID : 1245

Structure : Simple

Abstraction : Base

Description

Faulty finite state machines (FSMs) in the hardware logic allow an attacker to put the system in an undefined state, to cause a denial of service (DoS) or gain privileges on the victim's system.

Extended Description

The functionality and security of the system heavily depend on the implementation of FSMs. FSMs can be used to indicate the current security state of the system. Lots of secure data operations and data transfers rely on the state reported by the FSM. Faulty FSM designs that do not account for all states, either through undefined states (left as don't cares) or through incorrect implementation, might lead an attacker to drive the system into an unstable state from which the system cannot recover without a reset, thus causing a DoS. Depending on what the FSM is used for, an attacker might also gain additional privileges to launch further attacks and compromise the security guarantees.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		684	Incorrect Provision of Specified Functionality	1514

Applicable Platforms

- Language : Not Language-Specific (Prevalence = Undetermined)
- Operating_System : Not OS-Specific (Prevalence = Undetermined)
- Architecture : Not Architecture-Specific (Prevalence = Undetermined)
- Technology : System on Chip (Prevalence = Undetermined)

Common Consequences

Scope	Impact	Likelihood
Availability	Unexpected State	
Access Control	DoS: Crash, Exit, or Restart	
	DoS: Instability	
	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Define all possible states and handle all unused states through default statements. Ensure that system defaults to a secure state.

Effectiveness = High

Demonstrative Examples

Example 1:

The Finite State Machine (FSM) shown in the "bad" code snippet below assigns the output ("out") based on the value of state, which is determined based on the user provided input ("user_input").

Example Language: Verilog

(Bad)

```
module fsm_1(out, user_input, clk, rst_n);
input [2:0] user_input;
input clk, rst_n;
output reg [2:0] out;
reg [1:0] state;
always @ (posedge clk or negedge rst_n )
begin
    if (!rst_n)
        state = 3'h0;
    else
        case (user_input)
            3'h0:
            3'h1:
            3'h2:
            3'h3: state = 2'h3;
            3'h4: state = 2'h2;
            3'h5: state = 2'h1;
        endcase
    end
    out <= {1'h1, state};
endmodule
```

The case statement does not include a default to handle the scenario when the user provides inputs of 3'h6 and 3'h7. Those inputs push the system to an undefined state and might cause a crash (denial of service) or any other unanticipated outcome.

Adding a default statement to handle undefined inputs mitigates this issue. This is shown in the "Good" code snippet below. The default statement is in bold.

Example Language: Verilog

(Good)

```
case (user_input)
  3'h0:
  3'h1:
  3'h2:
  3'h3: state = 2'h3;
  3'h4: state = 2'h2;
  3'h5: state = 2'h1;
  default: state = 2'h0;
endcase
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State

References

[REF-1060]Farimah Farahmandi and Prabhat Mishra. "FSM Anomaly Detection using Formal Analysis". < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8119228&tag=1> >.

CWE-1246: Improper Write Handling in Limited-write Non-Volatile Memories

Weakness ID : 1246**Structure :** Simple**Abstraction :** Base

Description

The product does not implement or incorrectly implements wear leveling operations in limited-write non-volatile memories.


Extended Description

Non-volatile memories such as NAND Flash, EEPROM, etc. have individually erasable segments, each of which can be put through a limited number of program/erase or write cycles. For example, the device can only endure a limited number of writes, after which the device becomes unreliable. In order to wear out the cells in a uniform manner, non-volatile memory and storage products based on the above-mentioned technologies implement a technique called wear leveling. Once a set threshold is reached, wear leveling maps writes of a logical block to a different physical block. This prevents a single physical block from prematurely failing due to a high concentration of writes. If wear leveling is improperly implemented, attackers may be able to programmatically cause the storage to become unreliable within a much shorter time than would normally be expected.

Relationships

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

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Storage Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Instability	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Include secure wear leveling algorithms and ensure they may not be bypassed.

Effectiveness = High

Demonstrative Examples

Example 1:

An attacker can render a memory line unusable by repeatedly causing a write to the memory line.

Below is example code from [REF-1058] that the user can execute repeatedly to cause line failure. W is the maximum associativity of any cache in the system; S is the size of the largest cache in the system.

Example Language: C++

(Attack)

```
// Do aligned alloc of (W+1) arrays each of size S
while(1) {
    for (ii = 0; ii < W + 1; ii++)
        array[ii].element[0]++;
}
```

Without wear leveling, the above attack will be successful. Simple randomization of blocks will not suffice as instead of the original physical block, the randomized physical block will be worn out.



Example Language:

(Good)

Wear leveling must be used to even out writes to the device.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1202	Memory and Storage Issues	1194	2493

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
212	Functionality Misuse

References

[REF-1058]Moinuddin Qureshi, Michele Franchescini, Vijayalakshmi Srinivasan, Luis Lastras, Bulent Abali and John Karidis. "Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling". < <https://researcher.watson.ibm.com/researcher/files/us-moinqureshi/papers-sgap.pdf> >.2023-04-07.

[REF-1059]Micron. "Bad Block Management in NAND Flash Memory". < https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2959_bbm_in_nand_flash.pdf >.

CWE-1247: Improper Protection Against Voltage and Clock Glitches

Weakness ID : 1247

Structure : Simple

Abstraction : Base

Description

The device does not contain or contains incorrectly implemented circuitry or sensors to detect and mitigate voltage and clock glitches and protect sensitive information or software contained on the device.


Extended Description

A device might support features such as secure boot which are supplemented with hardware and firmware support. This involves establishing a chain of trust, starting with an immutable root of trust by checking the signature of the next stage (culminating with the OS and runtime software) against a golden value before transferring control. The intermediate stages typically set up the system in a secure state by configuring several access control settings. Similarly, security logic for exercising a debug or testing interface may be implemented in hardware, firmware, or both. A device needs to guard against fault attacks such as voltage glitches and clock glitches that an attacker may employ in an attempt to compromise the system.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1384	Improper Handling of Physical or Environmental Conditions	2269

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1332	Improper Handling of Faults that Lead to Instruction Skips	2240

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Technology : System on Chip (*Prevalence = Undetermined*)

Technology : Power Management Hardware (*Prevalence = Undetermined*)

Technology : Clock/Counter Hardware (*Prevalence = Undetermined*)

Technology : Sensor Hardware (*Prevalence = Undetermined*)

Common Consequences

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

Detection Methods

Manual Analysis

Put the processor in an infinite loop, which is then followed by instructions that should not ever be executed, since the loop is not expected to exit. After the loop, toggle an I/O bit (for oscilloscope monitoring purposes), print a console message, and reenter the loop. Note that to ensure that the loop exit is actually captured, many NOP instructions should be coded after the loop branch instruction and before the I/O bit toggle and the print statement. Margining the clock consists of varying the clock frequency until an anomaly occurs. This could be a continuous frequency change or it could be a single cycle. The single cycle method is described here. For every 1000th clock pulse, the clock cycle is shortened by 10 percent. If no effect is observed, the width is shortened by 20%. This process is continued in 10% increments up to and including 50%. Note that the cycle time may be increased as well, down to seconds per cycle. Separately, the voltage is margined. Note that the voltage could be increased or decreased. Increasing the voltage has limits, as the circuitry may not be able to withstand a drastically increased voltage. This process starts with a 5% reduction of the DC supply to the CPU chip for 5 millisecond repeated at 1KHz. If this has no effect, the process is repeated, but a 10% reduction is used. This process is repeated at 10% increments down to a 50% reduction. If no effects are observed at 5 millisecond, the whole process is repeated using a 10 millisecond pulse. If no effects are observed, the process is repeated in 10 millisecond increments out to 100 millisecond pulses. While these are suggested starting points for testing circuitry for weaknesses, the limits may need to be pushed further at the risk of device damage. See [REF-1217] for descriptions of Smart Card attacks against a clock (section 14.6.2) and using a voltage glitch (section 15.5.3).

Effectiveness = Moderate

Dynamic Analysis with Manual Results Interpretation

During the implementation phase where actual hardware is available, specialized hardware tools and apparatus such as ChipWhisperer may be used to check if the platform is indeed susceptible to voltage and clock glitching attacks.

Architecture or Design Review

Review if the protections against glitching merely transfer the attack target. For example, suppose a critical authentication routine that an attacker would want to bypass is given the protection of modifying certain artifacts from within that specific routine (so that if the routine is bypassed, one can examine the artifacts and figure out that an attack must have happened). However, if the attacker has the ability to bypass the critical authentication routine, they might also have the ability to bypass the other protection routine that checks the artifacts. Basically, depending on these kind of protections is akin to resorting to "Security by Obscurity".

Architecture or Design Review

Many SoCs come equipped with a built-in Dynamic Voltage and Frequency Scaling (DVFS) that can control the voltage and clocks via software alone. However, there have been demonstrated attacks (like Plundervolt and CLKSCREW) that target this DVFS [REF-1081] [REF-1082]. During the design and implementation phases, one needs to check if the interface to this power management feature is available from unprivileged SW (CWE-1256), which would make the attack very easy.

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

At the circuit-level, using Tunable Replica Circuits (TRCs) or special flip-flops such as Razor flip-flops helps mitigate glitch attacks. Working at the SoC or platform base, level sensors may be implemented to detect glitches. Implementing redundancy in security-sensitive code (e.g., where checks are performed) also can help with mitigation of glitch attacks.

Demonstrative Examples

Example 1:

Below is a representative snippet of C code that is part of the secure-boot flow. A signature of the runtime-firmware image is calculated and compared against a golden value. If the signatures match, the bootloader loads runtime firmware. If there is no match, an error halt occurs. If the underlying hardware executing this code does not contain any circuitry or sensors to detect voltage or clock glitches, an attacker might launch a fault-injection attack right when the signature check is happening (at the location marked with the comment), causing a bypass of the signature-checking process.

Example Language: C

(Bad)

```
...
if (signature_matches) // <-Glitch Here
{
    load_runtime_firmware();
}
else
{
    do_not_load_runtime_firmware();
}
...
```

After bypassing secure boot, an attacker can gain access to system assets to which the attacker should not have access.

Example Language:

(Good)

If the underlying hardware detects a voltage or clock glitch, the information can be used to prevent the glitch from being successful.

Observed Examples






Reference	Description
CVE-2019-17391	Lack of anti-glitch protections allows an attacker to launch a physical attack to bypass the secure boot and read protected eFuses. https://www.cve.org/CVERecord?id=CVE-2019-17391
CVE-2021-33478	IP communication firmware allows access to a boot shell via certain impulses https://www.cve.org/CVERecord?id=CVE-2021-33478

Functional Areas

- Power
- Clock

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1206	Power, Clock, Thermal, and Reset Concerns	1194	2494
MemberOf		1365	ICS Communications: Unreliability	1358	2523
MemberOf		1367	ICS Dependencies (& Architecture): External Physical Systems	1358	2525
MemberOf		1388	Physical Access Issues and Concerns	1194	2539
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
624	Hardware Fault Injection
625	Mobile Device Fault Injection

References

[REF-1061]Keith Bowman, James Tschanz, Chris Wilkerson, Shih-Lien Lu, Tanay Karnik, Vivek De and Shekhar Borkar. "Circuit Techniques for Dynamic Variation Tolerance". < <https://dl.acm.org/doi/10.1145/1629911.1629915> >.2023-04-07.

[REF-1062]Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner and Trevor Mudge. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation". < <https://web.eecs.umich.edu/~taustin/papers/MICRO36-Razor.pdf> >.

[REF-1063]James Tschanz, Keith Bowman, Steve Walstra, Marty Agostinelli, Tanay Karnik and Vivek De. "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance". < <https://ieeexplore.ieee.org/document/5205410> >.

[REF-1064]Bilgiday Yuce, Nahid F. Ghalaty, Chinmay Deshpande, Conor Patrick, Leyla Nazhandali and Patrick Schaumont. "FAME: Fault-attack Aware Microprocessor Extensions for Hardware Fault Detection and Software Fault Response". < <https://dl.acm.org/doi/10.1145/2948618.2948626> >.2023-04-07.

[REF-1065]Keith A. Bowman, James W. Tschanz, Shih-Lien L. Lu, Paolo A. Aseron, Muhammad M. Khellah, Arijit Raychowdhury, Bibiche M. Geuskens, Carlos Tokunaga, Chris B. Wilkerson,

Tanay Karnik and Vivek De. "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance". < <https://ieeexplore.ieee.org/document/5654663> >.

[REF-1066]Niek Timmers and Albert Spruyt. "Bypassing Secure Boot Using Fault Injection". < <https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf> >.

[REF-1217]Ross Anderson. "Security Engineering". 2001. < <https://www.cl.cam.ac.uk/~rja14/musicfiles/manuscripts/SEv1.pdf> >.

[REF-1217]Ross Anderson. "Security Engineering". 2001. < <https://www.cl.cam.ac.uk/~rja14/musicfiles/manuscripts/SEv1.pdf> >.

[REF-1081]Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Frank Piessens and Daniel Gruss. "Plundervolt". < <https://plundervolt.com/> >.

[REF-1082]Adrian Tang, Simha Sethumadhavan and Salvatore Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". < <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-tang.pdf> >.

[REF-1285]Texas Instruments. "Physical Security Attacks Against Silicon Devices". 2022 January 1. < <https://www.ti.com/lit/an/swra739/swra739.pdf?ts=1644234570420> >.

[REF-1286]Lennert Wouters, Benedikt Gierlichs and Bart Preneel. "On The Susceptibility of Texas Instruments SimpleLink Platform Microcontrollers to Non-Invasive Physical Attacks". 2022 March 4. < <https://eprint.iacr.org/2022/328.pdf> >.

CWE-1248: Semiconductor Defects in Hardware Logic with Security-Sensitive Implications

Weakness ID : 1248

Structure : Simple

Abstraction : Base

Description

The security-sensitive hardware module contains semiconductor defects.

Extended Description

A semiconductor device can fail for various reasons. While some are manufacturing and packaging defects, the rest are due to prolonged use or usage under extreme conditions. Some mechanisms that lead to semiconductor defects include encapsulation failure, die-attach failure, wire-bond failure, bulk-silicon defects, oxide-layer faults, aluminum-metal faults (including electromigration, corrosion of aluminum, etc.), and thermal/electrical stress. These defects manifest as faults on chip-internal signals or registers, have the effect of inputs, outputs, or intermediate signals being always 0 or always 1, and do not switch as expected. If such faults occur in security-sensitive hardware modules, the security objectives of the hardware module may be compromised.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1529

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
Availability	DoS: Instability	
Access Control		

Potential Mitigations

Phase: Testing

While semiconductor-manufacturing companies implement several mechanisms to continuously improve the semiconductor manufacturing process to ensure reduction of defects, some defects can only be fixed after manufacturing. Post-manufacturing testing of silicon die is critical. Fault models such as stuck-at-0 or stuck-at-1 must be used to develop post-manufacturing test cases and achieve good coverage. Once the silicon packaging is done, extensive post-silicon testing must be performed to ensure that hardware logic implementing security functionalities is defect-free.

Phase: Operation

Operating the hardware outside device specification, such as at extremely high temperatures, voltage, etc., accelerates semiconductor degradation and results in defects. When these defects manifest as faults in security-critical, hardware modules, it results in compromise of security guarantees. Thus, operating the device within the specification is important.

Demonstrative Examples

Example 1:

The network-on-chip implements a firewall for access control to peripherals from all IP cores capable of mastering transactions.

Example Language: Other






(Bad)

A manufacturing defect in this logic manifests itself as a logical fault, which always sets the output of the filter to "allow" access.

Post-manufacture testing must be performed to ensure that hardware logic implementing security functionalities is defect-free.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1195	Manufacturing and Life Cycle Management Concerns	1194	2490
MemberOf		1206	Power, Clock, Thermal, and Reset Concerns	1194	2494
MemberOf		1388	Physical Access Issues and Concerns	1194	2539
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
624	Hardware Fault Injection
625	Mobile Device Fault Injection

References

[REF-1067]Brian Bailey. "Why Chips Die". < <https://semiengineering.com/why-chips-die/> >.

[REF-1068]V. Lakshminarayan. "What causes semiconductor devices to fail". < Original >.2023-04-07.

CWE-1249: Application-Level Admin Tool with Inconsistent View of Underlying Operating System

Weakness ID : 1249

Structure : Simple

Abstraction : Base

Description

The product provides an application for administrators to manage parts of the underlying operating system, but the application does not accurately identify all of the relevant entities or resources that exist in the OS; that is, the application's model of the OS's state is inconsistent with the OS's actual state.

Extended Description

Many products provide web-based applications or other interfaces for managing the underlying operating system. This is common with cloud, network access devices, home networking, and other systems. When the management tool does not accurately represent what is in the OS - such as user accounts - then the administrator might not see suspicious activities that would be noticed otherwise.

For example, numerous systems utilize a web front-end for administrative control. They also offer the ability to add, alter, and drop users with various privileges as it relates to the functionality of the system. A potential architectural weakness may exist where the user information reflected in the web interface does not mirror the users in the underlying operating system. Many web UI or REST APIs use the underlying operating system for authentication; the system's logic may also track an additional set of user capabilities within configuration files and datasets for authorization capabilities. When there is a discrepancy between the user information in the UI or REST API's interface system and the underlying operating system's user listing, this may introduce a weakness into the system. For example, if an attacker compromises the OS and adds a new user account - a "ghost" account - then the attacker could escape detection if the management tool does not list the newly-added account.

This discrepancy could be exploited in several ways:


- A rogue admin could insert a new account into a system that will persist if they are terminated or wish to take action on a system that cannot be directly associated with them.
- An attacker can leverage a separate command injection attack available through the web interface to insert a ghost account with shell privileges such as ssh.
- An attacker can leverage existing web interface APIs, manipulated in such a way that a new user is inserted into the operating system, and the user web account is either partially created or not at all.
- An attacker could create an admin account which is viewable by an administrator, use this account to create the ghost account, delete logs and delete the first created admin account.

Many of these attacker scenarios can be realized by leveraging separate vulnerabilities related to XSS, command injection, authentication bypass, or logic flaws on the various systems.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1250	Improper Preservation of Consistency Between Independent Representations of Shared State	2064

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

Technology : Web Based (*Prevalence = Undetermined*)

Alternate Terms

Ghost in the Shell :

Common Consequences

Scope	Impact	Likelihood
Access Control	Varies by Context	
Accountability	Hide Activities	
Other	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Ensure that the admin tool refreshes its model of the underlying OS on a regular basis, and note any inconsistencies with configuration files or other data sources that are expected to have the same data.

Demonstrative Examples

Example 1:

Suppose that an attacker successfully gains root privileges on a Linux system and adds a new 'user2' account:

Example Language: Other

(Attack)

```
echo "user2:x:0:0::/root:/" >> /etc/passwd;
echo "user2:{$6$IdvyrM6VJnG8Su5U$1gmW3Nm.IO4vxTQDQ1C8urm72JCadOHZQwqiH/
nRtL8dPY80xS4Ovs5bPCMWnXKKWwmsocSWXupUf17LB3oS.:17256:0:99999:7::" >> /etc/shadow;
```

This new user2 account would not be noticed on the web interface, if the interface does not refresh its data of available users.

It could be argued that for this specific example, an attacker with root privileges would be likely to compromise the admin tool or otherwise feed it with false data. However, this example shows how the discrepancy in critical data can help attackers to escape detection.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2565

References

[REF-1070]Tony Martin. "Ghost in the Shell Weakness". 2020 February 3. < <https://friendsglobal.com/ghost-in-the-shell/ghost-in-the-shell-weakness/> >.2023-04-07.

CWE-1250: Improper Preservation of Consistency Between Independent Representations of Shared State

Weakness ID : 1250

Structure : Simple

Abstraction : Base

Description

The product has or supports multiple distributed components or sub-systems that are each required to keep their own local copy of shared data - such as state or cache - but the product does not ensure that all local copies remain consistent with each other.

Extended Description

In highly distributed environments, or on systems with distinct physical components that operate independently, there is often a need for each component to store and update its own local copy of key data such as state or cache, so that all components have the same "view" of the overall system and operate in a coordinated fashion. For example, users of a social media service or a massively multiplayer online game might be using their own personal computers while also interacting with different physical hosts in a globally distributed service, but all participants must be able to have the same "view" of the world. Alternately, a processor's Memory Management Unit (MMU) might have "shadow" MMUs to distribute its workload, and all shadow MMUs are expected to have the same accessible ranges of memory.

In such environments, it becomes critical for the product to ensure that this "shared state" is consistently modified across all distributed systems. If state is not consistently maintained across all systems, then critical transactions might take place out of order, or some users might not get the same data as other users. When this inconsistency affects correctness of operations, it can introduce vulnerabilities in mechanisms that depend on consistent 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	[P]	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	[B]	1249	Application-Level Admin Tool with Inconsistent View of Underlying Operating System	2062
ParentOf	[B]	1251	Mirrored Regions with Different Values	2065

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Cloud Computing (*Prevalence = Undetermined*)

Technology : Security Hardware (*Prevalence = Undetermined*)

Demonstrative Examples

Example 1:

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

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Research Gap

Issues related to state and cache - creation, preservation, and update - are a significant gap in CWE that is expected to be addressed in future versions. It likely has relationships to concurrency and synchronization, incorrect behavior order, and other areas that already have some coverage in CWE, although the focus has typically been on independent processes on the same operating system - not on independent systems that are all a part of a larger system-of-systems.

References

[REF-1069]Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu and Haryadi S. Gunawi. "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems". 2016. < <https://ucare.cs.uchicago.edu/pdf/asplos16-TaxDC.pdf> >.

CWE-1251: Mirrored Regions with Different Values

Weakness ID : 1251

Structure : Simple

Abstraction : Base

Description

The product's architecture mirrors regions without ensuring that their contents always stay in sync.

Extended Description

Having mirrored regions with different values might result in the exposure of sensitive information or possibly system compromise.

In the interest of increased performance, one might need to duplicate a resource. A cache memory is a common example of this concept, which keeps a "local" copy of a data element in the high speed cache memory. Unfortunately, this speed improvement comes with a downside, since the product needs to ensure that the local copy always mirrors the original copy truthfully. If they get out of sync, the computational result is no longer true.

During hardware design, memory is not the only item which gets mirrored. There are many other entities that get mirrored, as well: registers, memory regions, and, in some cases, even whole computational units. For example, within a multi-core processor, if all memory accesses for each and every core goes through a single Memory-Management Unit (MMU) then the MMU will become a performance bottleneck. In such cases, duplicating local MMUs that will serve only a subset of the cores rather than all of them may resolve the performance issue. These local copies are also called "shadow copies" or "mirrored copies."

If the original resource never changed, local duplicate copies getting out of sync would never be an issue. However, the values of the original copy will sometimes change. When the original copy changes, the mirrored copies must also change, and change fast.


This situation of shadow-copy-possibly-out-of-sync-with-original-copy might occur as a result of multiple scenarios, including the following:

- After the values in the original copy change, due to some reason the original copy does not send the "update" request to its shadow copies.
- After the values in the original copy change, the original copy dutifully sends the "update" request to its shadow copies, but due to some reason the shadow copy does not "execute" this update request.
- After the values in the original copy change, the original copy sends the "update" request to its shadow copies, and the shadow copy executes this update request faithfully. However, during the small time period when the original copy has "new" values and the shadow copy is still holding the "old" values, an attacker can exploit the old values. Then it becomes a race condition between the attacker and the update process of who can reach the target, shadow copy first, and, if the attacker reaches first, the attacker wins.
- The attacker might send a "spoofed" update request to the target shadow copy, pretending that this update request is coming from the original copy. This spoofed request might cause the targeted shadow copy to update its values to some attacker-friendly values, while the original copies remain unchanged by the attacker.
- Suppose a situation where the original copy has a system of reverting back to its original value if it does not hear back from all the shadow copies that such copies have successfully completed the update request. In such a case, an attack might occur as follows: (1) the original copy might send an update request; (2) the shadow copy updates it; (3) the shadow copy sends back the successful completion message; (4) through a separate issue, the attacker is able to intercept the shadow copy's completion message. In this case, the original copy thinks that the update did not succeed, hence it reverts to its original value. Now there is a situation where the original copy has the "old" value, and the shadow copy has the "new" 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		1250	Improper Preservation of Consistency Between Independent Representations of Shared State	2064

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1312	Missing Protection for Mirrored Regions in On-Chip Fabric Firewall	2196

Applicable Platforms

Language : VHDL (*Prevalence = Undetermined*)

Language : Verilog (*Prevalence = Undetermined*)

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context	
Integrity		
Availability		
Access Control		
Accountability		
Authentication		
Authorization		
Non-Repudiation		

Potential Mitigations

Phase: Architecture and Design

Whenever there are multiple, physically different copies of the same value that might change and the process to update them is not instantaneous and atomic, it is impossible to assert that the original and shadow copies will always be in sync - there will always be a time period when they are out of sync. To mitigate the consequential risk, the recommendations essentially are: Make this out-of-sync time period as small as possible, and Make the update process as robust as possible.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

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

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1202	Memory and Storage Issues	1194	2493
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2565

Notes

Research Gap

Issues related to state and cache - creation, preservation, and update - are a significant gap in CWE that is expected to be addressed in future versions. It has relationships to concurrency and

synchronization, incorrect behavior order, and other areas that already have some coverage in CWE, although the focus has typically been on independent processes on the same operating system - not on independent systems that are all a part of a larger system-of-systems.

CWE-1252: CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations

Weakness ID : 1252

Structure : Simple

Abstraction : Base

Description

The CPU is not configured to provide hardware support for exclusivity of write and execute operations on memory. This allows an attacker to execute data from all of memory.

Extended Description

CPUs provide a special bit that supports exclusivity of write and execute operations. This bit is used to segregate areas of memory to either mark them as code (instructions, which can be executed) or data (which should not be executed). In this way, if a user can write to a region of memory, the user cannot execute from that region and vice versa. This exclusivity provided by special hardware bit is leveraged by the operating system to protect executable space. While this bit is available in most modern processors by default, in some CPUs the exclusivity is implemented via a memory-protection unit (MPU) and memory-management unit (MMU) in which memory regions can be carved out with exact read, write, and execute permissions. However, if the CPU does not have an MMU/MPU, then there is no write exclusivity. Without configuring exclusivity of operations via segregated areas of memory, an attacker may be able to inject malicious code onto memory and later execute it.

Relationships

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

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Microcontroller Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Implement a dedicated bit that can be leveraged by the Operating System to mark data areas as non-executable. If such a bit is not available in the CPU, implement MMU/MPU (memory management unit / memory protection unit).

Phase: Integration

If MMU/MPU are not available, then the firewalls need to be implemented in the SoC interconnect to mimic the write-exclusivity operation.

Demonstrative Examples

Example 1:

MCS51 Microcontroller (based on 8051) does not have a special bit to support write exclusivity. It also does not have an MMU/MPU support. The Cortex-M CPU has an optional MPU that supports up to 8 regions.

Example Language: Other

(Bad)

The optional MPU is not configured.

If the MPU is not configured, then an attacker will be able to inject malicious data into memory and execute it.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as a member. This 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	1201	Core and Compute Issues	1194	2492
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
679	Exploitation of Improperly Configured or Implemented Memory Protections

References

[REF-1076]ARM. "Cortex-R4 Manual". < <https://developer.arm.com/Processors/Cortex-M4> >.2023-04-07.

[REF-1077]Intel. "MCS 51 Microcontroller Family User's Manual". < <http://web.mit.edu/6.115/www/document/8051.pdf> >.

[REF-1078]ARM. "Memory Protection Unit (MPU)". < https://web.archive.org/web/20200630034848/https://static.docs.arm.com/100699/0100/armv8m_architecture_memory_protection_unit_100699_0100_00_en.pdf >.2023-04-07.

CWE-1253: Incorrect Selection of Fuse Values

Weakness ID : 1253

Structure : Simple

Abstraction : Base

Description

The logic level used to set a system to a secure state relies on a fuse being unblown. An attacker can set the system to an insecure state merely by blowing the fuse.

Extended Description

Fuses are often used to store secret data, including security configuration data. When not blown, a fuse is considered to store a logic 0, and, when blown, it indicates a logic 1. Fuses are generally considered to be one-directional, i.e., once blown to logic 1, it cannot be reset to logic 0. However, if the logic used to determine system-security state (by leveraging the values sensed from the fuses) uses negative logic, an attacker might blow the fuse and drive the system to an insecure 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	[P]	693	Protection Mechanism Failure	1529

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
Access Control	Bypass Protection Mechanism	
Authorization	Gain Privileges or Assume Identity	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Read Memory	
Integrity	Modify Memory Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Logic should be designed in a way that blown fuses do not put the product into an insecure state that can be leveraged by an attacker.

Demonstrative Examples

Example 1:

A chip implements a secure boot and uses the sensed value of a fuse "do_secure_boot" to determine whether to perform a secure boot or not. If this fuse value is "0", the system performs secure boot. Otherwise, it does not perform secure boot.



An attacker blows the "do_secure_boot" fuse to "1". After reset, the attacker loads a custom bootloader, and, since the fuse value is now "1", the system does not perform secure boot, and the attacker can execute their custom firmware image.

Since by default, a fuse-configuration value is a "0", an attacker can blow it to a "1" with inexpensive hardware.

If the logic is reversed, an attacker cannot easily reset the fuse. Note that, with specialized and expensive equipment, an attacker with full physical access might be able to "unblow" the fuse value to a "0".

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State

References

[REF-1080]Christopher Tarnovsky. "Security Failures in Secure Devices". < <https://www.blackhat.com/presentations/bh-europe-08/Tarnovsky/Presentation/bh-eu-08-tarnovsky.pdf> >.

CWE-1254: Incorrect Comparison Logic Granularity

Weakness ID : 1254

Structure : Simple

Abstraction : Base

Description

The product's comparison logic is performed over a series of steps rather than across the entire string in one operation. If there is a comparison logic failure on one of these steps, the operation may be vulnerable to a timing attack that can result in the interception of the process for nefarious purposes.

Extended Description

Comparison logic is used to compare a variety of objects including passwords, Message Authentication Codes (MACs), and responses to verification challenges. When comparison logic is implemented at a finer granularity (e.g., byte-by-byte comparison) and breaks in the case of a comparison failure, an attacker can exploit this implementation to identify when exactly the failure occurred. With multiple attempts, the attacker may be able to guess the correct password/response to challenge and elevate their privileges.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		697	Incorrect Comparison	1538
ChildOf		208	Observable Timing Discrepancy	537
PeerOf		1261	Improper Handling of Single Event Upsets	2091

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
Confidentiality Authorization	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation




The hardware designer should ensure that comparison logic is implemented so as to compare in one operation instead in smaller chunks.

Observed Examples

Reference	Description
CVE-2019-10482	Smartphone OS uses comparison functions that are not in constant time, allowing side channels https://www.cve.org/CVERecord?id=CVE-2019-10482
CVE-2019-10071	Java-oriented framework compares HMAC signatures using String.equals() instead of a constant-time algorithm, causing timing discrepancies https://www.cve.org/CVERecord?id=CVE-2019-10071
CVE-2014-0984	Password-checking function in router terminates validation of a password entry when it encounters the first incorrect character, which allows remote attackers to obtain passwords via a brute-force attack that relies on timing differences in responses to incorrect password guesses, aka a timing side-channel attack. https://www.cve.org/CVERecord?id=CVE-2014-0984

MemberOf Relationships

This MemberOf relationships table shows additional CWE Categories and Views that reference this weakness as 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		1199	General Circuit and Logic Design Concerns	1194	2492
MemberOf		1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Notes

Maintenance

CWE 4.16 removed a demonstrative example for a hardware module because it was inaccurate and unable to be adapted. The CWE team is developing an alternative.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions