```
        endcase
    end // always @ (posedge wb_clk_i)
  endmodule
```

The above code snippet [REF-1402] illustrates an instance of a vulnerable implementation of the AES wrapper module, where p_c[i] registers are cleared at reset. Otherwise, p_c[i]registers either maintain their old values (if reglk_ctrl_i[3]is true) or get filled through the AXI signal wdata. Note that p_c[i]registers can be read through the AXI Lite interface (not shown in snippet). However, p_c[i] registers are never cleared after their usage once the AES engine has completed the encryption process of the message. In a multi-user or multi-process environment, not clearing registers may result in the attacker process accessing data left by the victim, leading to data leakage or unintentional information disclosure.

To fix this issue, it is essential to ensure that these internal registers are cleared in a timely manner after their usage, i.e., the encryption process is complete. This is illustrated below by monitoring the assertion of the cipher text valid signal, ct_valid [REF-1403].

*Example Language: Verilog* *(Good)*

```
module aes0_wrapper #(...)(...);
...
always @(posedge clk_i)
  begin
    if(~(rst_ni && ~rst_1)) //clear p_c[i] at reset
      ...
    else if(ct_valid) //encryption process complete, clear p_c[i]
      begin
        p_c[0] <= 0;
        p_c[1] <= 0;
        p_c[2] <= 0;
        p_c[3] <= 0;
      end
    else if(en && we)
      case(address[8:3])
      ...
      endcase
    end // always @ (posedge wb_clk_i)
  endmodule
```

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2019-3733** | Cryptography library does not clear heap memory before release<br>*https://www.cve.org/CVERecord?id=CVE-2019-3733* |
| **CVE-2003-0001** | Ethernet NIC drivers do not pad frames with null bytes, leading to infoleak from malformed packets.<br>*https://www.cve.org/CVERecord?id=CVE-2003-0001* |
| **CVE-2003-0291** | router does not clear information from DHCP packets that have been previously used<br>*https://www.cve.org/CVERecord?id=CVE-2003-0291* |
| **CVE-2005-1406** | Products do not fully clear memory buffers when less data is stored into the buffer than previous.<br>*https://www.cve.org/CVERecord?id=CVE-2005-1406* |
| **CVE-2005-1858** | Products do not fully clear memory buffers when less data is stored into the buffer than previous.<br>*https://www.cve.org/CVERecord?id=CVE-2005-1858* |
| **CVE-2005-3180** | Products do not fully clear memory buffers when less data is stored into the buffer than previous.<br>*https://www.cve.org/CVERecord?id=CVE-2005-3180* |

| Reference | Description |
|---|---|
| **CVE-2005-3276** | Product does not clear a data structure before writing to part of it, yielding information leak of previously used memory. *https://www.cve.org/CVERecord?id=CVE-2005-3276* |
| **CVE-2002-2077** | Memory not properly cleared before reuse. *https://www.cve.org/CVERecord?id=CVE-2002-2077* |

## Functional Areas

- Memory Management
- Networking

## 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 | 729 | OWASP Top Ten 2004 Category A8 - Insecure Storage | 711 | 2359 |
| MemberOf | C | 742 | CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM) | 734 | 2367 |
| MemberOf | C | 876 | CERT C++ Secure Coding Section 08 - Memory Management (MEM) | 868 | 2398 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1202 | Memory and Storage Issues | 1194 | 2493 |
| MemberOf | C | 1416 | Comprehensive Categorization: Resource Lifecycle Management | 1400 | 2566 |

## Notes

### Relationship

There is a close association between CWE-226 and CWE-212. The difference is partially that of perspective. CWE-226 is geared towards the final stage of the resource lifecycle, in which the resource is deleted, eliminated, expired, or otherwise released for reuse. Technically, this involves a transfer to a different control sphere, in which the original contents of the resource are no longer relevant. CWE-212, however, is intended for sensitive data in resources that are intentionally shared with others, so they are still active. This distinction is useful from the perspective of the CWE research view (CWE-1000).

### Maintenance

This entry needs modification to clarify the differences with CWE-212. The description also combines two problems that are distinct from the CWE research perspective: the inadvertent transfer of information to another sphere, and improper initialization/shutdown. Some of the associated taxonomy mappings reflect these different uses.

### Research Gap

This is frequently found for network packets, but it can also exist in local memory allocation, files, etc.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Sensitive Information Uncleared Before Use |
| CERT C Secure Coding | MEM03-C | | Clear sensitive information stored in reusable resources returned for reuse |

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

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 37 | Retrieve Embedded Sensitive Data |

### References

[REF-1402]"aes0_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/65d0ffdab7426da4509c98d62e163bcce642f651/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L84C2-L90C29 >.2024-02-14.

[REF-1403]"Fix for aes0_wrapper". 2023 November 9. < https://github.com/HACK-EVENT/hackatdac21/blob/0034dff6852365a8c4e36590a47ea8b088d725ae/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L96C1-L102C16 >.2024-02-14.

## CWE-228: Improper Handling of Syntactically Invalid Structure

**Weakness ID :** 228
**Structure :** Simple
**Abstraction :** Class

### Description

The product does not handle or incorrectly handles input that is not syntactically well-formed with respect to the associated specification.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 703 | Improper Check or Handling of Exceptional Conditions | 1544 |
| ChildOf | P | 707 | Improper Neutralization | 1554 |
| ParentOf | B | 166 | Improper Handling of Missing Special Element | 429 |
| ParentOf | B | 167 | Improper Handling of Additional Special Element | 431 |
| ParentOf | B | 168 | Improper Handling of Inconsistent Special Elements | 433 |
| ParentOf | B | 229 | Improper Handling of Values | 577 |
| ParentOf | B | 233 | Improper Handling of Parameters | 581 |
| ParentOf | B | 237 | Improper Handling of Structural Elements | 587 |
| ParentOf | B | 241 | Improper Handling of Unexpected Data Type | 591 |

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Availability | Unexpected State<br>DoS: Crash, Exit, or Restart<br>DoS: Resource Consumption (CPU)<br>*If an input is syntactically invalid, then processing the input could place the system in an unexpected state that could lead to a crash, consume available system resources or other unintended behaviors.* | |

### 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:

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

*Example Language: Java* *(Bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    if("com.example.URLHandler.openURL".equals(intent.getAction())) {
      String URL = intent.getStringExtra("URLToOpen");
      int length = URL.length();
    ...
    }
  }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2004-0270** | Anti-virus product has assert error when line length is non-numeric.<br>*https://www.cve.org/CVERecord?id=CVE-2004-0270* |

## 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 | 728 | OWASP Top Ten 2004 Category A7 - Improper Error Handling | 711 | 2359 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

## Notes

### Maintenance

This entry needs more investigation. Public vulnerability research generally focuses on the manipulations that generate invalid structure, instead of the weaknesses that are exploited by those manipulations. For example, a common attack involves making a request that omits a required field, which can trigger a crash in some cases. The crash could be due to a named chain such as CWE-690 (Unchecked Return Value to NULL Pointer Dereference), but public reports rarely cover this aspect of a vulnerability.

**Theoretical**

The validity of input could be roughly classified along "syntactic", "semantic", and "lexical" dimensions. If the specification requires that an input value should be delimited with the "[" and "]" square brackets, then any input that does not follow this specification would be syntactically invalid. If the input between the brackets is expected to be a number, but the letters "aaa" are provided, then the input is syntactically invalid. If the input is a number and enclosed in brackets, but the number is outside of the allowable range, then it is semantically invalid. The inter-relationships between these properties - and their associated weaknesses- need further exploration.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Structure and Validity Problems |
| OWASP Top Ten 2004 | A7 | CWE More Specific | Improper Error Handling |

## CWE-229: Improper Handling of Values

**Weakness ID :** 229
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not properly handle when the expected number of values for parameters, fields, or arguments is not provided in input, or if those values are undefined.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 228 | Improper Handling of Syntactically Invalid Structure | 575 |
| ParentOf | V | 230 | Improper Handling of Missing Values | 578 |
| ParentOf | V | 231 | Improper Handling of Extra Values | 579 |
| ParentOf | V | 232 | Improper Handling of Undefined Values | 580 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 19 | Data Processing Errors | 2330 |

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

## CWE-230: Improper Handling of Missing Values

**Weakness ID :** 230
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not handle or incorrectly handles when a parameter, field, or argument name is specified, but the associated value is missing, i.e. it is empty, blank, or null.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 229 | Improper Handling of Values | 577 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### Demonstrative Examples

**Example 1:**

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

*Example Language: Java* *(Bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    if("com.example.URLHandler.openURL".equals(intent.getAction())) {
      String URL = intent.getStringExtra("URLToOpen");
      int length = URL.length();
    ...
    }
  }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

### Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2002-0422** | Blank Host header triggers resultant infoleak. *https://www.cve.org/CVERecord?id=CVE-2002-0422* |
| **CVE-2000-1006** | Blank "charset" attribute in MIME header triggers crash. *https://www.cve.org/CVERecord?id=CVE-2000-1006* |
| **CVE-2004-1504** | Blank parameter causes external error infoleak. *https://www.cve.org/CVERecord?id=CVE-2004-1504* |
| **CVE-2005-2053** | Blank parameter causes external error infoleak. *https://www.cve.org/CVERecord?id=CVE-2005-2053* |

### 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 | 851 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR) | 844 | 2386 |
| MemberOf | C | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Notes

**Research Gap**

Some "crash by port scan" bugs are probably due to this, but lack of diagnosis makes it difficult to be certain.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
| --- | --- | --- | --- |
| PLOVER | | | Missing Value Error |
| The CERT Oracle Secure Coding Standard for Java (2011) | ERR08-J | | Do not catch NullPointerException or any of its ancestors |

## CWE-231: Improper Handling of Extra Values

**Weakness ID :** 231
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not handle or incorrectly handles when more values are provided than expected.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 229 | Improper Handling of Values | 577 |
| CanPrecede | B | 120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') | 310 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Notes

**Relationship**

This can overlap buffer overflows.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Extra Value Error |

## CWE-232: Improper Handling of Undefined Values

**Weakness ID :** 232
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not handle or incorrectly handles when a value is not defined or supported for the associated parameter, field, or argument name.

### Relationships

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 | 229 | Improper Handling of Values | 577 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### Demonstrative Examples

**Example 1:**

In this example, an address parameter is read and trimmed of whitespace.

*Example Language: Java* *(Bad)*

```
String address = request.getParameter("address");
address = address.trim();
String updateString = "UPDATE shippingInfo SET address='?' WHERE email='cwe@example.com'";
emailAddress = con.prepareStatement(updateString);
emailAddress.setString(1, address);
```

If the value of the address parameter is null (undefined), the servlet will throw a NullPointerException when the trim() is attempted.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2000-1003** | Client crash when server returns unknown driver type. *https://www.cve.org/CVERecord?id=CVE-2000-1003* |

### 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 | 851 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR) | 844 | 2386 |
| MemberOf | C | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Undefined Value Error |
| The CERT Oracle Secure Coding Standard for Java (2011) | ERR08-J | | Do not catch NullPointerException or any of its ancestors |

## CWE-233: Improper Handling of Parameters

**Weakness ID :** 233
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not properly handle when the expected number of parameters, fields, or arguments is not provided in input, or if those parameters are undefined.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 228 | Improper Handling of Syntactically Invalid Structure | 575 |
| ParentOf | V | 234 | Failure to Handle Missing Parameter | 583 |
| ParentOf | V | 235 | Improper Handling of Extra Parameters | 585 |
| ParentOf | V | 236 | Improper Handling of Undefined Parameters | 586 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 19 | Data Processing Errors | 2330 |

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

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

*Example Language: Java* *(Bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    if("com.example.URLHandler.openURL".equals(intent.getAction())) {
      String URL = intent.getStringExtra("URLToOpen");
      int length = URL.length();
    ...
    }
  }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

## 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Parameter Problems |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 39 | Manipulating Opaque Client-based Data Tokens |

## CWE-234: Failure to Handle Missing Parameter

**Weakness ID :** 234
**Structure :** Simple
**Abstraction :** Variant

### Description

If too few arguments are sent to a function, the function will still pop the expected number of arguments from the stack. Potentially, a variable number of arguments could be exhausted in a function as well.

### Relationships

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 | 233 | Improper Handling of Parameters | 581 |

### Applicable Platforms

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

### Likelihood Of Exploit

High

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Confidentiality<br>Availability<br>Access Control | Execute Unauthorized Code or Commands<br>Gain Privileges or Assume Identity<br><br>*There is the potential for arbitrary code execution with privileges of the vulnerable program if function parameter list is exhausted.* | |
| Availability | DoS: Crash, Exit, or Restart<br><br>*Potentially a program could fail if it needs more arguments then are available.* | |

### Potential Mitigations

**Phase: Build and Compilation**

This issue can be simply combated with the use of proper build process.

### Phase: Implementation

Forward declare all functions. This is the recommended solution. Properly forward declaration of all used functions will result in a compiler error if too few arguments are sent to a function.

## Demonstrative Examples

### Example 1:

The following example demonstrates the weakness.

*Example Language: C* *(Bad)*

```
foo_funct(one, two);
void foo_funct(int one, int two, int three) {
  printf("1) %d\n2) %d\n3) %d\n", one, two, three);
}
```

*Example Language: C* *(Bad)*

```
void some_function(int foo, ...) {
  int a[3], i;
  va_list ap;
  va_start(ap, foo);
  for (i = 0; i < sizeof(a) / sizeof(int); i++) a[i] = va_arg(ap, int);
  va_end(ap);
}
int main(int argc, char *argv[]) {
  some_function(17, 42);
}
```

This can be exploited to disclose information with no work whatsoever. In fact, each time this function is run, it will print out the next 4 bytes on the stack after the two numbers sent to it.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2004-0276** | Server earlier allows remote attackers to cause a denial of service (crash) via an HTTP request with a sequence of "%" characters and a missing Host field. *https://www.cve.org/CVERecord?id=CVE-2004-0276* |
| **CVE-2002-1488** | Chat client allows remote malicious IRC servers to cause a denial of service (crash) via a PART message with (1) a missing channel or (2) a channel that the user is not in. *https://www.cve.org/CVERecord?id=CVE-2002-1488* |
| **CVE-2002-1169** | Proxy allows remote attackers to cause a denial of service (crash) via an HTTP request to helpout.exe with a missing HTTP version numbers. *https://www.cve.org/CVERecord?id=CVE-2002-1169* |
| **CVE-2000-0521** | Web server allows disclosure of CGI source code via an HTTP request without the version number. *https://www.cve.org/CVERecord?id=CVE-2000-0521* |
| **CVE-2001-0590** | Application server allows a remote attacker to read the source code to arbitrary 'jsp' files via a malformed URL request which does not end with an HTTP protocol specification. *https://www.cve.org/CVERecord?id=CVE-2001-0590* |
| **CVE-2003-0239** | Chat software allows remote attackers to cause a denial of service via malformed GIF89a headers that do not contain a GCT (Global Color Table) or an LCT (Local Color Table) after an Image Descriptor. *https://www.cve.org/CVERecord?id=CVE-2003-0239* |
| **CVE-2002-1023** | Server allows remote attackers to cause a denial of service (crash) via an HTTP GET request without a URI. *https://www.cve.org/CVERecord?id=CVE-2002-1023* |

| Reference | Description |
|---|---|
| **CVE-2002-1236** | CGI crashes when called without any arguments. *https://www.cve.org/CVERecord?id=CVE-2002-1236* |
| **CVE-2003-0422** | CGI crashes when called without any arguments. *https://www.cve.org/CVERecord?id=CVE-2003-0422* |
| **CVE-2002-1531** | Crash in HTTP request without a Content-Length field. *https://www.cve.org/CVERecord?id=CVE-2002-1531* |
| **CVE-2002-1077** | Crash in HTTP request without a Content-Length field. *https://www.cve.org/CVERecord?id=CVE-2002-1077* |
| **CVE-2002-1358** | Empty elements/strings in protocol test suite affect many SSH2 servers/clients. *https://www.cve.org/CVERecord?id=CVE-2002-1358* |
| **CVE-2003-0477** | FTP server crashes in PORT command without an argument. *https://www.cve.org/CVERecord?id=CVE-2003-0477* |
| **CVE-2002-0107** | Resultant infoleak in web server via GET requests without HTTP/1.0 version string. *https://www.cve.org/CVERecord?id=CVE-2002-0107* |
| **CVE-2002-0596** | GET request with empty parameter leads to error message infoleak (path disclosure). *https://www.cve.org/CVERecord?id=CVE-2002-0596* |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Notes

#### Maintenance

This entry will be deprecated in a future version of CWE. The term "missing parameter" was used in both PLOVER and CLASP, with completely different meanings. However, data from both taxonomies was merged into this entry. In PLOVER, it was meant to cover malformed inputs that do not contain required parameters, such as a missing parameter in a CGI request. This entry's observed examples and classification came from PLOVER. However, the description, demonstrative example, and other information are derived from CLASP. They are related to an incorrect number of function arguments, which is already covered by CWE-685.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Missing Parameter Error |
| CLASP | | | Missing parameter |

## CWE-235: Improper Handling of Extra Parameters

**Weakness ID :** 235
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not handle or incorrectly handles when the number of parameters, fields, or arguments with the same name exceeds the expected amount.

### Relationships

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 | 233 | Improper Handling of Parameters | 581 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2003-1014** | MIE. multiple gateway/security products allow restriction bypass using multiple MIME fields with the same name, which are interpreted differently by clients. *https://www.cve.org/CVERecord?id=CVE-2003-1014* |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Notes

**Relationship**

This type of problem has a big role in multiple interpretation vulnerabilities and various HTTP attacks.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Extra Parameter Error |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 460 | HTTP Parameter Pollution (HPP) |

## CWE-236: Improper Handling of Undefined Parameters

**Weakness ID :** 236
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not handle or incorrectly handles when a particular parameter, field, or argument name is not defined or supported by the product.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 233 | Improper Handling of Parameters | 581 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1488** | Crash in IRC client via PART message from a channel the user is not in. *https://www.cve.org/CVERecord?id=CVE-2002-1488* |
| **CVE-2001-0650** | Router crash or bad route modification using BGP updates with invalid transitive attribute. *https://www.cve.org/CVERecord?id=CVE-2001-0650* |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Undefined Parameter Error |

## CWE-237: Improper Handling of Structural Elements

**Weakness ID :** 237
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not handle or incorrectly handles inputs that are related to complex structures.

### Relationships

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 | 228 | Improper Handling of Syntactically Invalid Structure | 575 |
| ParentOf | V | 238 | Improper Handling of Incomplete Structural Elements | 588 |
| ParentOf | V | 239 | Failure to Handle Incomplete Element | 589 |
| ParentOf | B | 240 | Improper Handling of Inconsistent Structural Elements | 590 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 19 | Data Processing Errors | 2330 |

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Element Problems |

## CWE-238: Improper Handling of Incomplete Structural Elements

**Weakness ID :** 238
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not handle or incorrectly handles when a particular structural element is not completely specified.

### Relationships

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 | 237 | Improper Handling of Structural Elements | 587 |

### Weakness Ordinalities

**Resultant :**

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity | Unexpected State | |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Notes

**Relationship**

Can be primary to other problems.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Missing Element Error |

## CWE-239: Failure to Handle Incomplete Element

**Weakness ID :** 239
**Structure :** Simple
**Abstraction :** Variant

### Description

The product does not properly handle when a particular element is not completely specified.

### Relationships

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 | 237 | Improper Handling of Structural Elements | 587 |
| PeerOf | C | 404 | Improper Resource Shutdown or Release | 987 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Other | Varies by Context<br>Unexpected State | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1532** | HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it.<br>*https://www.cve.org/CVERecord?id=CVE-2002-1532* |
| **CVE-2003-0195** | Partial request is not timed out. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2003-0195* |
| **CVE-2005-2526** | MFV. CPU exhaustion in printer via partial printing request then early termination of connection. *https://www.cve.org/CVERecord?id=CVE-2005-2526* |
| **CVE-2002-1906** | CPU consumption by sending incomplete HTTP requests and leaving the connections open. *https://www.cve.org/CVERecord?id=CVE-2002-1906* |

### 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Incomplete Element |

## CWE-240: Improper Handling of Inconsistent Structural Elements

**Weakness ID :** 240
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not handle or incorrectly handles when two or more structural elements should be consistent, but are not.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 707 | Improper Neutralization | 1554 |
| ChildOf | B | 237 | Improper Handling of Structural Elements | 587 |
| ParentOf | B | 130 | Improper Handling of Length Parameter Inconsistency | 357 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Other | Varies by Context<br>Unexpected State | |

### Demonstrative Examples

**Example 1:**

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-2014-0160** | Chain: "Heartbleed" bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data. *https://www.cve.org/CVERecord?id=CVE-2014-0160* |
| **CVE-2009-2299** | Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data. *https://www.cve.org/CVERecord?id=CVE-2009-2299* |

## 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 | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Inconsistent Elements |

# CWE-241: Improper Handling of Unexpected Data Type

**Weakness ID :** 241
**Structure :** Simple
**Abstraction :** Base

## Description

The product does not handle or incorrectly handles when a particular element is not the expected type, e.g. it expects a digit (0-9) but is provided with a letter (A-Z).

## Relationships

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 | 228 | Improper Handling of Syntactically Invalid Structure | 575 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 19 | Data Processing Errors | 2330 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Other | Varies by Context<br>Unexpected State | |

## Potential Mitigations

**Phase: Implementation**

*Strategy = Input Validation*

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

**Phase: Implementation**

*Strategy = Input Validation*

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-1999-1156** | FTP server crash via PORT command with non-numeric character. *https://www.cve.org/CVERecord?id=CVE-1999-1156* |
| **CVE-2004-0270** | Anti-virus product has assert error when line length is non-numeric. *https://www.cve.org/CVERecord?id=CVE-2004-0270* |

### MemberOf Relationships

This MemberOf relationships table shows additional CWE 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 | 743 | CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO) | 734 | 2368 |
| MemberOf | C | 877 | CERT C++ Secure Coding Section 09 - Input Output (FIO) | 868 | 2398 |
| MemberOf | C | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 2438 |
| MemberOf | C | 1163 | SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO) | 1154 | 2480 |
| MemberOf | C | 1407 | Comprehensive Categorization: Improper Neutralization | 1400 | 2553 |

### Notes

**Research Gap**

Probably under-studied.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Wrong Data Type |
| CERT C Secure Coding | FIO37-C | CWE More Abstract | Do not assume that fgets() or fgetws() returns a nonempty string when successful |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 48 | Passing Local Filenames to Functions That Expect a URL |

## CWE-242: Use of Inherently Dangerous Function

**Weakness ID :** 242
**Structure :** Simple
**Abstraction :** Base

### Description

The product calls a function that can never be guaranteed to work safely.

### Extended Description

Certain functions behave in dangerous ways regardless of how they are used. Functions in this category were often implemented without taking security concerns into account. The gets() function is unsafe because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to gets() and overflow the destination buffer. Similarly, the >> operator is unsafe to use when reading into a statically-allocated character array because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to the >> operator and overflow the destination 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 | 1177 | Use of Prohibited Code | 1981 |

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

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

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Detection Methods

**Automated Static Analysis**

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

*Effectiveness = High*

## Potential Mitigations

**Phase: Implementation**

**Phase: Requirements**

Ban the use of dangerous functions. Use their safe equivalent.

**Phase: Testing**

Use grep or static analysis tools to spot usage of dangerous functions.

## Demonstrative Examples

**Example 1:**

The code below calls gets() to read information into a buffer.

*Example Language: C* *(Bad)*

```
char buf[BUFSIZE];
gets(buf);
```

The gets() function in C is inherently unsafe.

**Example 2:**

The code below calls the gets() function to read in data from the command line.

*Example Language: C* *(Bad)*

```
  char buf[24];
  printf("Please enter your name and press <Enter>\n");
  gets(buf);
  ...
}
```

However, gets() is inherently unsafe, because it copies all input from STDIN to the buffer without checking size. This allows the user to provide a string that is larger than the buffer size, resulting in an overflow condition.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2007-4004** | FTP client uses inherently insecure gets() function and is setuid root on some systems, allowing buffer overflow *https://www.cve.org/CVERecord?id=CVE-2007-4004* |

## MemberOf Relationships

This MemberOf relationships table shows additional CWE 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 748 | CERT C Secure Coding Standard (2008) Appendix - POSIX (POS) | 734 | 2372 |
| MemberOf | C | 1001 | SFP Secondary Cluster: Use of an Improper API | 888 | 2441 |
| MemberOf | C | 1171 | SEI CERT C Coding Standard - Guidelines 50. POSIX (POS) | 1154 | 2484 |
| MemberOf | C | 1412 | Comprehensive Categorization: Poor Coding Practices | 1400 | 2559 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Dangerous Functions |
| CERT C Secure Coding | POS33-C | CWE More Abstract | Do not use vfork() |
| Software Fault Patterns | SFP3 | | Use of an improper API |

## References

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

[REF-194]Herbert Schildt. "Herb Schildt's C++ Programming Cookbook". 2008 April 8. McGraw-Hill Osborne Media.

[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-243: Creation of chroot Jail Without Changing Working Directory

**Weakness ID :** 243
**Structure :** Simple
**Abstraction :** Variant

## Description

The product uses the chroot() system call to create a jail, but does not change the working directory afterward. This does not prevent access to files outside of the jail.

## Extended Description

Improper use of chroot() may allow attackers to escape from the chroot jail. The chroot() function call does not change the process's current working directory, so relative paths may still refer to file system resources outside of the chroot jail after chroot() has been called.

## Relationships

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 | 573 | Improper Following of Specification by Caller | 1307 |
| ChildOf | C | 669 | Incorrect Resource Transfer Between Spheres | 1480 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1015 | Limit Access | 2451 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Weakness Ordinalities

**Resultant :**

## Applicable Platforms

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

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

**Operating_System** : Unix *(Prevalence = Undetermined)*

## Background Details

The chroot() system call allows a process to change its perception of the root directory of the file system. After properly invoking chroot(), a process cannot access any files outside the directory tree defined by the new root directory. Such an environment is called a chroot jail and is commonly used to prevent the possibility that a processes could be subverted and used to access unauthorized files. For instance, many FTP servers run in chroot jails to prevent an attacker who discovers a new vulnerability in the server from being able to download the password file or other sensitive files on the system.

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Files or Directories | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Demonstrative Examples

### Example 1:

Consider the following source code from a (hypothetical) FTP server:

*Example Language: C* *(Bad)*

```
chroot("/var/ftproot");
...
fgets(filename, sizeof(filename), network);
localfile = fopen(filename, "r");
while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF) {
  fwrite(buf, 1, sizeof(buf), network);
}
fclose(localfile);
```

This code is responsible for reading a filename from the network, opening the corresponding file on the local machine, and sending the contents over the network. This code could be used to implement the FTP GET command. The FTP server calls chroot() in its initialization routines in an attempt to prevent access to files outside of /var/ftproot. But because the server does not change the current working directory by calling chdir("/"), an attacker could request the file "../../../../../etc/passwd" and obtain a copy of the system password file.

## Affected Resources

- 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 979 | SFP Secondary Cluster: Failed Chroot Jail | 888 | 2429 |
| MemberOf | C | 1416 | Comprehensive Categorization: Resource Lifecycle Management | 1400 | 2566 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Directory Restriction |
| Software Fault Patterns | SFP17 | | Failed chroot jail |

## 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-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection')

**Weakness ID :** 244
**Structure :** Simple
**Abstraction :** Variant

## Description

Using realloc() to resize buffers that store sensitive information can leave the sensitive information exposed to attack, because it is not removed from memory.

## Extended Description

When sensitive data such as a password or an encryption key is not removed from memory, it could be exposed to an attacker using a "heap inspection" attack that reads the sensitive data using memory dumps or other methods. The realloc() function is commonly used to increase the size of a block of allocated memory. This operation often requires copying the contents of the old memory block into a new and larger block. This operation leaves the contents of the original block intact but inaccessible to the program, preventing the program from being able to scrub sensitive data from memory. If an attacker can later examine the contents of a memory dump, the sensitive data could be exposed.

## Relationships

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 | 226 | Sensitive Information in Resource Not Removed Before Reuse | 569 |
| CanPrecede | C | 669 | Incorrect Resource Transfer Between Spheres | 1480 |

## Applicable Platforms

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

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Other | Read Memory<br>Other<br><br>*Be careful using vfork() and fork() in security sensitive code. The process state will not be cleaned up and will contain traces of data from past use.* | |

## Demonstrative Examples

**Example 1:**

The following code calls realloc() on a buffer containing sensitive data:

*Example Language: C* *(Bad)*

```
cleartext_buffer = get_secret();...
cleartext_buffer = realloc(cleartext_buffer, 1024);
...
scrub_memory(cleartext_buffer, 1024);
```

There is an attempt to scrub the sensitive data from memory, but realloc() is used, so it could return a pointer to a different part of memory. The memory that was originally allocated for cleartext_buffer could still contain an uncleared copy of the data.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2019-3733** | Cryptography library does not clear heap memory before release *https://www.cve.org/CVERecord?id=CVE-2019-3733* |

### 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 742 | CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM) | 734 | 2367 |
| MemberOf | C | 876 | CERT C++ Secure Coding Section 08 - Memory Management (MEM) | 868 | 2398 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1399 | Comprehensive Categorization: Memory Safety | 1400 | 2546 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Heap Inspection |
| CERT C Secure Coding | MEM03-C | | Clear sensitive information stored in reusable resources returned for reuse |
| Software Fault Patterns | SFP23 | | Exposed Data |

### 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-245: J2EE Bad Practices: Direct Management of Connections

**Weakness ID :** 245
**Structure :** Simple
**Abstraction :** Variant

### Description

The J2EE application directly manages connections, instead of using the container's connection management facilities.

## Extended Description

The J2EE standard forbids the direct management of connections. It requires that applications use the container's resource management facilities to obtain connections to resources. Every major web application container provides pooled database connection management as part of its resource management framework. Duplicating this functionality in an application is difficult and error prone, which is part of the reason it is forbidden under the J2EE standard.

## Relationships

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 | 695 | Use of Low-Level Functionality | 1533 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

**Language** : Java *(Prevalence = Undetermined)*

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Quality Degradation | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Demonstrative Examples

### Example 1:

In the following example, the class DatabaseConnection opens and manages a connection to a database for a J2EE application. The method openDatabaseConnection opens a connection to the database using a DriverManager to create the Connection object conn to the database specified in the string constant CONNECT_STRING.

*Example Language: Java* *(Bad)*

```
public class DatabaseConnection {
  private static final String CONNECT_STRING = "jdbc:mysql://localhost:3306/mysqldb";
  private Connection conn = null;
  public DatabaseConnection() {
  }
  public void openDatabaseConnection() {
    try {
      conn = DriverManager.getConnection(CONNECT_STRING);
```

```
    } catch (SQLException ex) {...}
  }
  // Member functions for retrieving database connection and accessing database
  ...
}
```

The use of the DriverManager class to directly manage the connection to the database violates the J2EE restriction against the direct management of connections. The J2EE application should use the web application container's resource management facilities to obtain a connection to the database as shown in the following example.

*Example Language:* *(Good)*

```
public class DatabaseConnection {
  private static final String DB_DATASRC_REF = "jdbc:mysql://localhost:3306/mysqldb";
  private Connection conn = null;
  public DatabaseConnection() {
  }
  public void openDatabaseConnection() {
    try {
      InitialContext ctx = new InitialContext();
      DataSource datasource = (DataSource) ctx.lookup(DB_DATASRC_REF);
      conn = datasource.getConnection();
    } catch (NamingException ex) {...}
    } catch (SQLException ex) {...}
  }
  // Member functions for retrieving database connection and accessing database
  ...
}
```

## 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 1001 | SFP Secondary Cluster: Use of an Improper API | 888 | 2441 |
| MemberOf | C | 1412 | Comprehensive Categorization: Poor Coding Practices | 1400 | 2559 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | J2EE Bad Practices: getConnection() |
| Software Fault Patterns | SFP3 | | Use of an improper API |

## References

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

# CWE-246: J2EE Bad Practices: Direct Use of Sockets

**Weakness ID :** 246
**Structure :** Simple
**Abstraction :** Variant

## Description

The J2EE application directly uses sockets instead of using framework method calls.

### Extended Description

The J2EE standard permits the use of sockets only for the purpose of communication with legacy systems when no higher-level protocol is available. Authoring your own communication protocol requires wrestling with difficult security issues.

Without significant scrutiny by a security expert, chances are good that a custom communication protocol will suffer from security problems. Many of the same issues apply to a custom implementation of a standard protocol. While there are usually more resources available that address security concerns related to implementing a standard protocol, these resources are also available to attackers.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 695 | Use of Low-Level Functionality | 1533 |

### Weakness Ordinalities

**Resultant :**

### Applicable Platforms

**Language** : Java *(Prevalence = Undetermined)*

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Quality Degradation | |

### Detection Methods

**Automated Static Analysis**

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

*Effectiveness = High*

### Potential Mitigations

**Phase: Architecture and Design**

Use framework method calls instead of using sockets directly.

### Demonstrative Examples

**Example 1:**

The following example opens a socket to connect to a remote server.

*Example Language: Java* *(Bad)*

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  // Perform servlet tasks.
```

```
  ...
  // Open a socket to a remote server (bad).
  Socket sock = null;
  try {
    sock = new Socket(remoteHostname, 3000);
    // Do something with the socket.
    ...
  } catch (Exception e) {
    ...
  }
}
```

A Socket object is created directly within the Java servlet, which is a dangerous way to manage remote connections.

### MemberOf Relationships

This MemberOf relationships table shows additional CWE 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 1001 | SFP Secondary Cluster: Use of an Improper API | 888 | 2441 |
| MemberOf | C | 1412 | Comprehensive Categorization: Poor Coding Practices | 1400 | 2559 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | J2EE Bad Practices: Sockets |
| Software Fault Patterns | SFP3 | | Use of an improper API |

### References

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

## CWE-248: Uncaught Exception

**Weakness ID :** 248
**Structure :** Simple
**Abstraction :** Base

### Description

An exception is thrown from a function, but it is not caught.

### Extended Description

When an exception is not caught, it may cause the program to crash or expose sensitive information.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 755 | Improper Handling of Exceptional Conditions | 1585 |
| ChildOf | C | 705 | Incorrect Control Flow Scoping | 1550 |
| ParentOf | V | 600 | Uncaught Exception in Servlet | 1352 |

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

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

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

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

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

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

## Applicable Platforms

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

**Language** : Java *(Prevalence = Undetermined)*

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Availability<br>Confidentiality | DoS: Crash, Exit, or Restart<br>Read Application Data<br>*An uncaught exception could cause the system to be placed in a state that could lead to a crash, exposure of sensitive information or other unintended behaviors.* | |

## 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 example attempts to resolve a hostname.

*Example Language: Java* *(Bad)*

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
  String ip = req.getRemoteAddr();
  InetAddress addr = InetAddress.getByName(ip);
  ...
  out.println("hello " + addr.getHostName());
}
```

A DNS lookup failure will cause the Servlet to throw an exception.

**Example 2:**

The _alloca() function allocates memory on the stack. If an allocation request is too large for the available stack space, _alloca() throws an exception. If the exception is not caught, the program will crash, potentially enabling a denial of service attack. _alloca() has been deprecated as of Microsoft Visual Studio 2005(R). It has been replaced with the more secure _alloca_s().

**Example 3:**

EnterCriticalSection() can raise an exception, potentially causing the program to crash. Under operating systems prior to Windows 2000, the EnterCriticalSection() function can raise an exception in low memory situations. If the exception is not caught, the program will crash, potentially enabling a denial of service attack.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2023-41151** | SDK for OPC Unified Architecture (OPC UA) server has uncaught exception when a socket is blocked for writing but the server tries to send an error *https://www.cve.org/CVERecord?id=CVE-2023-41151* |
| **CVE-2023-21087** | Java code in a smartphone OS can encounter a "boot loop" due to an uncaught exception *https://www.cve.org/CVERecord?id=CVE-2023-21087* |

## 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 730 | OWASP Top Ten 2004 Category A9 - Denial of Service | 711 | 2360 |
| MemberOf | C | 851 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR) | 844 | 2386 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 962 | SFP Secondary Cluster: Unchecked Status Condition | 888 | 2421 |
| MemberOf | C | 1141 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR) | 1133 | 2469 |
| MemberOf | C | 1181 | SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP) | 1178 | 2487 |
| MemberOf | C | 1410 | Comprehensive Categorization: Insufficient Control Flow Management | 1400 | 2557 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Often Misused: Exception Handling |
| The CERT Oracle Secure Coding Standard for Java (2011) | ERR05-J | | Do not let checked exceptions escape from a finally block |
| The CERT Oracle Secure Coding Standard for Java (2011) | ERR06-J | | Do not throw undeclared checked exceptions |
| SEI CERT Perl Coding Standard | EXP31-PL | Exact | Do not suppress or ignore exceptions |
| Software Fault Patterns | SFP4 | | Unchecked Status Condition |

## 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-250: Execution with Unnecessary Privileges

**Weakness ID :** 250
**Structure :** Simple
**Abstraction :** Base

## Description

The product performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.

## Extended Description

New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.

Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 657 | Violation of Secure Design Principles | 1454 |
| ChildOf | C | 269 | Improper Privilege Management | 653 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1015 | Limit Access | 2451 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Applicable Platforms

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

**Technology** : Mobile *(Prevalence = Undetermined)*

## Likelihood Of Exploit

Medium

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity<br>Availability<br>Access Control | Gain Privileges or Assume Identity<br>Execute Unauthorized Code or Commands<br>Read Application Data<br>DoS: Crash, Exit, or Restart<br>*An attacker will be able to gain access to any resources that are allowed by the extra privileges. Common results include executing code, disabling services, and reading restricted data.* | |

### Detection Methods

**Manual Analysis**

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

**Black Box**

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and perform a login. Look for library functions and system calls that indicate when privileges are being raised or dropped. Look for accesses of resources that are restricted to normal users.

**Automated Static Analysis - Binary or Bytecode**

According to SOAR, the following detection techniques may be useful: Highly cost effective: Compare binary / bytecode to application permission manifest Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

*Effectiveness = High*

**Manual Static Analysis - Binary or Bytecode**

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

*Effectiveness = SOAR Partial*

**Dynamic Analysis with Automated Results Interpretation**

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

*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

*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*

**Automated Static Analysis**

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

*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**

**Phase: Operation**

*Strategy = Environment Hardening*

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

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [REF-76]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

**Phase: Architecture and Design**

*Strategy = Attack Surface Reduction*

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [REF-76]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

**Phase: Implementation**

Perform extensive input validation for any privileged code that must be exposed to the user and reject anything that does not fit your strict requirements.

**Phase: Implementation**

When dropping privileges, ensure that they have been dropped successfully to avoid CWE-273. As protection mechanisms in the environment get stronger, privilege-dropping calls may fail even if it seems like they would always succeed.

**Phase: Implementation**

If circumstances force you to run with extra privileges, then determine the minimum access level necessary. First identify the different permissions that the software and its users will need to perform their actions, such as file read and write permissions, network socket permissions, and so forth. Then explicitly allow those actions while denying all else [REF-76]. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability. This mitigation is much more prone to error than dropping the privileges in the first place.

**Phase: Operation**

**Phase: System Configuration**

*Strategy = Environment Hardening*

Ensure that the software runs properly under the United States Government Configuration Baseline (USGCB) [REF-199] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

## Demonstrative Examples

**Example 1:**

This code temporarily raises the program's privileges to allow creation of a new user folder.

*Example Language: Python* *(Bad)*

```
def makeNewUserDir(username):
  if invalidUsername(username):
    #avoid CWE-22 and CWE-78
    print('Usernames cannot contain invalid characters')
    return False
  try:
    raisePrivileges()
    os.mkdir('/home/' + username)
    lowerPrivileges()
  except OSError:
    print('Unable to create new user directory for user:' + username)
    return False
  return True
```

While the program only raises its privilege level to create the folder and immediately lowers it again, if the call to os.mkdir() throws an exception, the call to lowerPrivileges() will not occur. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

**Example 2:**

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

*Example Language: C* *(Bad)*

```
chroot(APP_HOME);
chdir("/");
```

```
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

**Example 3:**

This application intends to use a user's location to determine the timezone the user is in:

*Example Language: Java* *(Bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
setTimeZone(userCurrLocation);
```

This is unnecessary use of the location API, as this information is already available using the Android Time API. Always be sure there is not another way to obtain needed information before resorting to using the location API.

**Example 4:**

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

*Example Language: XML* *(Bad)*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

*Example Language: Java* *(Bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2007-4217** | FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients. *https://www.cve.org/CVERecord?id=CVE-2007-4217* |

| Reference | Description |
|---|---|
| **CVE-2008-1877** | Program runs with privileges and calls another program with the same privileges, which allows read of arbitrary files. *https://www.cve.org/CVERecord?id=CVE-2008-1877* |
| **CVE-2007-5159** | OS incorrectly installs a program with setuid privileges, allowing users to gain privileges. *https://www.cve.org/CVERecord?id=CVE-2007-5159* |
| **CVE-2008-4638** | Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209). *https://www.cve.org/CVERecord?id=CVE-2008-4638* |
| **CVE-2008-0162** | Program does not drop privileges before calling another program, allowing code execution. *https://www.cve.org/CVERecord?id=CVE-2008-0162* |
| **CVE-2008-0368** | setuid root program allows creation of arbitrary files through command line argument. *https://www.cve.org/CVERecord?id=CVE-2008-0368* |
| **CVE-2007-3931** | Installation script installs some programs as setuid when they shouldn't be. *https://www.cve.org/CVERecord?id=CVE-2007-3931* |
| **CVE-2020-3812** | mail program runs as root but does not drop its privileges before attempting to access a file. Attacker can use a symlink from their home directory to a directory only readable by root, then determine whether the file exists based on the response. *https://www.cve.org/CVERecord?id=CVE-2020-3812* |
| **CVE-2003-0908** | Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access "open file" dialog. *https://www.cve.org/CVERecord?id=CVE-2003-0908* |

## 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 | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 753 | 2009 Top 25 - Porous Defenses | 750 | 2374 |
| MemberOf | C | 815 | OWASP Top Ten 2010 Category A6 - Security Misconfiguration | 809 | 2379 |
| MemberOf | C | 858 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER) | 844 | 2390 |
| MemberOf | C | 866 | 2011 Top 25 - Porous Defenses | 900 | 2393 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1418 | Comprehensive Categorization: Violation of Secure Design Principles | 1400 | 2570 |

## Notes

### Relationship

There is a close association with CWE-653 (Insufficient Separation of Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible.

### Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category. Both CWE-272 and CWE-250 are in active use by the community. The "least privilege" phrase has multiple interpretations.

**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 |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Often Misused: Privilege Management |
| The CERT Oracle Secure Coding Standard for Java (2011) | SER09-J | | Minimize privileges before deserializing from a privilege context |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.05 BR |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.08 BR |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.08 RE(1) |
| ISA/IEC 62443 | Part 2-4 | | Req SP.05.07 BR |
| ISA/IEC 62443 | Part 2-4 | | Req SP.09.02 RE(4) |
| ISA/IEC 62443 | Part 2-4 | | Req SP.09.03 BR |
| ISA/IEC 62443 | Part 2-4 | | Req SP.09.04 BR |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.1 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.2 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 2.1 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 2.1 RE 1 |
| ISA/IEC 62443 | Part 4-1 | | Req SD-4 |
| ISA/IEC 62443 | Part 4-2 | | Req CCSC 3 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.1 |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 69 | Target Programs with Elevated Privileges |
| 104 | Cross Zone Scripting |
| 470 | Expanding Control over the Operating System from the Database |

## References

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

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

[REF-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-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-199]NIST. "United States Government Configuration Baseline (USGCB)". < https://csrc.nist.gov/Projects/United-States-Government-Configuration-Baseline >.2023-03-28.

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

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

# CWE-252: Unchecked Return Value

**Weakness ID :** 252
**Structure :** Simple
**Abstraction :** Base

## Description

The product does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.

## Extended Description

Two common programmer assumptions are "this function call can never fail" and "it doesn't matter if this function call fails". If an attacker can force the function to fail or otherwise return a value that is not expected, then the subsequent program logic could lead to a vulnerability, because the product is not in a state that the programmer assumes. For example, if the program calls a function to drop privileges but does not check the return code to ensure that privileges were successfully dropped, then the program will continue to operate with the higher privileges.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 754 | Improper Check for Unusual or Exceptional Conditions | 1577 |
| ParentOf | | 690 | Unchecked Return Value to NULL Pointer Dereference | 1523 |
| PeerOf | B | 273 | Improper Check for Dropped Privileges | 667 |
| CanPrecede | B | 476 | NULL Pointer Dereference | 1139 |

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

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

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

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

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Background Details

Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Availability<br>Integrity | Unexpected State<br>DoS: Crash, Exit, or Restart<br>*An unexpected return value could place the system in a state that could lead to a crash or other unintended behaviors.* | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Implementation

Check the results of all functions that return a value and verify that the value is expected.

*Effectiveness = High*

*Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.*

### Phase: Implementation

Ensure that you account for all possible return values from the function.

### Phase: Implementation

When designing a function, make sure you return a value or throw an exception in case of an error.

## Demonstrative Examples

### Example 1:

Consider the following code segment:

*Example Language: C* *(Bad)*

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when fgets() returns, buf will contain a null-terminated string of length 9 or less. But if an I/O error occurs, fgets() will not null-terminate buf. Furthermore, if the end of the file is reached before any characters are read, fgets() returns without writing anything to buf. In both of these situations, fgets() signals that something unusual has happened by returning NULL, but in this code, the warning will not be noticed. The lack of a null terminator in buf can result in a buffer overflow in the subsequent call to strcpy().

**Example 2:**

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:**

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by malloc().

*Example Language: C* *(Bad)*

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or allow the program to die with a segmentation fault when it tries to dereference the null pointer." This argument ignores three important considerations:

- Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.
- It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.
- The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

**Example 4:**

The following examples read a file into a byte array.

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

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current()) {
  String userName = (String) i.Current();
  String pFileName = PFILE_ROOT + "/" + userName;
  StreamReader sr = new StreamReader(pFileName);
  sr.Read(byteArray,0,1024);//the file is always 1k bytes
  sr.Close();
  processPFile(userName, byteArray);
}
```

*Example Language: Java* *(Bad)*

```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
  String userName = (String) i.next();
  String pFileName = PFILE_ROOT + "/" + userName;
  FileInputStream fis = new FileInputStream(pFileName);
  fis.read(byteArray); // the file is always 1k bytes
  fis.close();
  processPFile(userName, byteArray);
```

The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

**Example 5:**

The following code does not check to see if the string returned by getParameter() is null before calling the member function compareTo(), potentially causing a NULL dereference.

*Example Language: Java* *(Bad)*

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM) == 0) {
  ...
}
...
```

The following code does not check to see if the string returned by the Item property is null before calling the member function Equals(), potentially causing a NULL dereference.

*Example Language: Java* *(Bad)*

```
String itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM)) {
  ...
}
...
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

**Example 6:**

The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

*Example Language: Java* *(Bad)*

```
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

**Example 7:**

The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

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

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
Dim MyArray(50) As Byte
MyFile.Read(MyArray, 0, 50)
DoDangerousOperation(MyArray(20))
```

In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

**Example 8:**

It is not uncommon for Java programmers to misunderstand read() and related methods that are part of many java.io classes. Most errors and unusual events in Java result in an exception being thrown. But the stream and reader classes do not consider it unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested. This behavior makes it important for programmers to examine the return value from read() and other IO methods to ensure that they receive the amount of data they expect.

**Example 9:**

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

*Example Language: C* *(Bad)*

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

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

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

**Example 10:**

The following function attempts to acquire a lock in order to perform operations on a shared resource.

*Example Language: C* *(Bad)*

```
void f(pthread_mutex_t *mutex) {
  pthread_mutex_lock(mutex);
  /* access shared resource */
```

```
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason, the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races, correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting them to higher levels.

*Example Language: C* *(Good)*

```
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

## Observed Examples

| Reference | Description |
|---|---|
| **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* |
| **CVE-2020-6078** | Chain: The return value of a function returning a pointer is not checked for success (CWE-252) resulting in the later use of an uninitialized variable (CWE-456) and a null pointer dereference (CWE-476) *https://www.cve.org/CVERecord?id=CVE-2020-6078* |
| **CVE-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-2007-3798** | Unchecked return value leads to resultant integer overflow and code execution. *https://www.cve.org/CVERecord?id=CVE-2007-3798* |
| **CVE-2006-4447** | Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://www.cve.org/CVERecord?id=CVE-2006-4447* |
| **CVE-2006-2916** | Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://www.cve.org/CVERecord?id=CVE-2006-2916* |
| **CVE-2008-5183** | chain: unchecked return value can lead to NULL dereference *https://www.cve.org/CVERecord?id=CVE-2008-5183* |
| **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-2017-6964** | Linux-based device mapper encryption program does not check the return value of setuid and setgid allowing attackers to execute code with unintended privileges. *https://www.cve.org/CVERecord?id=CVE-2017-6964* |

| Reference | Description |
|---|---|
| **CVE-2002-1372** | Chain: Return values of file/socket operations are not checked (CWE-252), allowing resultant consumption of file descriptors (CWE-772). *https://www.cve.org/CVERecord?id=CVE-2002-1372* |

### MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 227 | 7PK - API Abuse | 700 | 2334 |
| MemberOf | C | 728 | OWASP Top Ten 2004 Category A7 - Improper Error Handling | 711 | 2359 |
| MemberOf | C | 742 | CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM) | 734 | 2367 |
| MemberOf | C | 847 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP) | 844 | 2384 |
| MemberOf | C | 876 | CERT C++ Secure Coding Section 08 - Memory Management (MEM) | 868 | 2398 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 962 | SFP Secondary Cluster: Unchecked Status Condition | 888 | 2421 |
| MemberOf | C | 1129 | CISQ Quality Measures (2016) - Reliability | 1128 | 2461 |
| MemberOf | C | 1131 | CISQ Quality Measures (2016) - Security | 1128 | 2463 |
| MemberOf | C | 1136 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP) | 1133 | 2466 |
| MemberOf | C | 1167 | SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR) | 1154 | 2482 |
| MemberOf | C | 1171 | SEI CERT C Coding Standard - Guidelines 50. POSIX (POS) | 1154 | 2484 |
| MemberOf | C | 1181 | SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP) | 1178 | 2487 |
| MemberOf | C | 1306 | CISQ Quality Measures - Reliability | 1305 | 2504 |
| MemberOf | C | 1308 | CISQ Quality Measures - Security | 1305 | 2506 |
| MemberOf | C | 1405 | Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions | 1400 | 2552 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Unchecked Return Value |
| CLASP | | | Ignored function return value |
| OWASP Top Ten 2004 | A7 | CWE More Specific | Improper Error Handling |
| CERT C Secure Coding | ERR33-C | Imprecise | Detect and handle standard library errors |
| CERT C Secure Coding | POS54-C | Imprecise | Detect and handle POSIX library errors |
| The CERT Oracle Secure Coding Standard for Java (2011) | EXP00-J | | Do not ignore values returned by methods |
| SEI CERT Perl Coding Standard | EXP32-PL | Exact | Do not ignore function return values |
| Software Fault Patterns | SFP4 | | Unchecked Status Condition |

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| OMG ASCSM | ASCSM-CWE-252-resource | | |
| OMG ASCRM | ASCRM-CWE-252-data | | |
| OMG ASCRM | ASCRM-CWE-252-resource | | |

### References

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

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

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

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

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

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

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

## CWE-253: Incorrect Check of Function Return Value

**Weakness ID :** 253
**Structure :** Simple
**Abstraction :** Base

### Description

The product incorrectly checks a return value from a function, which prevents it from detecting errors or exceptional conditions.

### Extended Description

Important and common functions will return some value about the success of its actions. This will alert the program whether or not to handle any errors caused by that function.

### Relationships

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 | 754 | Improper Check for Unusual or Exceptional Conditions | 1577 |
| ChildOf | C | 573 | Improper Following of Specification by Caller | 1307 |

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

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

## Applicable Platforms

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

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Availability<br>Integrity | Unexpected State<br>DoS: Crash, Exit, or Restart<br>*An unexpected return value could place the system in a state that could lead to a crash or other unintended behaviors.* | |

## Potential Mitigations

**Phase: Architecture and Design**

*Strategy = Language Selection*

Use a language or compiler that uses exceptions and requires the catching of those exceptions.

**Phase: Implementation**

Properly check all functions which return a value.

**Phase: Implementation**

When designing any function make sure you return a value or throw an exception in case of an error.

## Demonstrative Examples

**Example 1:**

This code attempts to allocate memory for 4 integers and checks if the allocation succeeds.

*Example Language: C* *(Bad)*

```
tmp = malloc(sizeof(int) * 4);
if (tmp < 0 ) {
  perror("Failure");
  //should have checked if the call returned 0
}
```

The code assumes that only a negative return value would indicate an error, but malloc() may return a null pointer when there is an error. The value of tmp could then be equal to 0, and the error would be missed.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2023-49286** | Chain: function in web caching proxy does not correctly check a return value (CWE-253) leading to a reachable assertion (CWE-617)<br>*https://www.cve.org/CVERecord?id=CVE-2023-49286* |

### MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 962 | SFP Secondary Cluster: Unchecked Status Condition | 888 | 2421 |
| MemberOf | C | 1167 | SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR) | 1154 | 2482 |
| MemberOf | C | 1171 | SEI CERT C Coding Standard - Guidelines 50. POSIX (POS) | 1154 | 2484 |
| MemberOf | C | 1412 | Comprehensive Categorization: Poor Coding Practices | 1400 | 2559 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Misinterpreted function return value |
| Software Fault Patterns | SFP4 | | Unchecked Status Condition |
| CERT C Secure Coding | ERR33-C | Imprecise | Detect and handle standard library errors |
| CERT C Secure Coding | POS54-C | Imprecise | Detect and handle POSIX library errors |

### References

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

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

## CWE-256: Plaintext Storage of a Password

**Weakness ID :** 256
**Structure :** Simple
**Abstraction :** Base

### Description

Storing a password in plaintext may result in a system compromise.

### Extended Description

Password management issues occur when a password is stored in plaintext in an application's properties, configuration file, or memory. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource. In some contexts, even storage of a plaintext password in memory is considered a security risk if the password is not cleared immediately after it 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 | | 522 | Insufficiently Protected Credentials | 1234 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 255 | Credentials Management Errors | 2336 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

## 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 storing passwords in easily accessible locations.

**Phase: Architecture and Design**

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

A programmer might attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password because the encoding can be detected and decoded easily.

*Effectiveness = None*

## Demonstrative Examples

**Example 1:**

The following code reads a password from a properties file and uses the password to connect to a database.

*Example Language: Java* *(Bad)*

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
```

```
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

**Example 2:**

The following code reads a password from the registry and uses the password to create a new network credential.

*Example Language: Java* *(Bad)*

```
...
String password = regKey.GetValue(passKey).toString();
NetworkCredential netCred = new NetworkCredential(username,password,domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system

**Example 3:**

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 4:**

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.

At least one OT product stored a password in plaintext.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-30275** | Remote Terminal Unit (RTU) uses a driver that relies on a password stored in plaintext. *https://www.cve.org/CVERecord?id=CVE-2022-30275* |

### 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 | 2335 |
| MemberOf | C | 930 | OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management | 928 | 2410 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1028 | OWASP Top Ten 2017 Category A2 - Broken Authentication | 1026 | 2457 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Password Management |
| Software Fault Patterns | SFP23 | | Exposed Data |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.5 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.5 |

### References

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

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

[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/ >.

## CWE-257: Storing Passwords in a Recoverable Format

**Weakness ID :** 257
**Structure :** Simple
**Abstraction :** Base

### Description

The storage of passwords in a recoverable format makes them subject to password reuse attacks by malicious users. In fact, it should be noted that recoverable encrypted passwords provide no significant benefit over plaintext passwords since they are subject not only to reuse by malicious attackers but also by malicious insiders. If a system administrator can recover a password directly, or use a brute force search on the available information, the administrator can use the password on other accounts.

## Relationships

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 | 522 | Insufficiently Protected Credentials | 1234 |
| PeerOf | V | 259 | Use of Hard-coded Password | 630 |
| PeerOf | V | 259 | Use of Hard-coded Password | 630 |
| PeerOf | B | 798 | Use of Hard-coded Credentials | 1699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 255 | Credentials Management Errors | 2336 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Access Control | Gain Privileges or Assume Identity<br>*User's passwords may be revealed.* | |
| Access Control | Gain Privileges or Assume Identity<br>*Revealed passwords may be reused elsewhere to impersonate the users in question.* | |

## Detection Methods

**Automated Static Analysis**

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

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

Use strong, non-reversible encryption to protect stored passwords.

## Demonstrative Examples

### Example 1:

Both of these examples verify a password by comparing it to a stored compressed version.

*Example Language: C* *(Bad)*

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

*Example Language: Java* *(Bad)*

```
int VerifyAdmin(String password) {
  if (passwd.Equals(compress(password), compressed_password)) {
    return(0);
  }
  //Diagnostic Mode
  return(1);
}
```

Because a compression algorithm is used instead of a one way hashing algorithm, an attacker can recover compressed passwords stored in the database.

### Example 2:

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.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-30018** | A messaging platform serializes all elements of User/Group objects, making private information available to adversaries *https://www.cve.org/CVERecord?id=CVE-2022-30018* |

## 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

**Maintenance**

The meaning of this entry needs to be investigated more closely, especially with respect to what is meant by "recoverable."

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Storing passwords in a recoverable format |
| Software Fault Patterns | SFP23 | | Exposed Data |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 49 | Password Brute Forcing |

## References

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

# CWE-258: Empty Password in Configuration File

**Weakness ID :** 258
**Structure :** Simple
**Abstraction :** Variant

## Description

Using an empty string as a password is insecure.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 521 | Weak Password Requirements | 1231 |
| ChildOf | B | 260 | Password in Configuration File | 636 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

## Potential Mitigations

**Phase: System Configuration**

Passwords should be at least eight characters long -- the longer the better. Avoid passwords that are in any way similar to other passwords you have. Avoid using words that may be found in a dictionary, names book, on a map, etc. Consider incorporating numbers and/or punctuation into your password. If you do use common words, consider replacing letters in that word with numbers and punctuation. However, do not use "similar-looking" punctuation. For example, it is not a good idea to change cat to c@t, ca+, (@+, or anything similar. Finally, it is never appropriate to use an empty string as a password.

## Demonstrative Examples

**Example 1:**

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 the password is provided as an empty string.

This Java example shows a properties file with an empty password string.

*Example Language: Java* *(Bad)*

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

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 and the password is provided as an empty string.

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

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

An empty string should never be used as a password as this can allow unauthorized access to the application. Username and password information should not be included in a configuration file or a properties file in clear text. If possible, encrypt this information and avoid CWE-260 and CWE-13.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-26117** | Network access control (NAC) product has a configuration file with an empty password *https://www.cve.org/CVERecord?id=CVE-2022-26117* |

### 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 | 2335 |
| MemberOf | C | 950 | SFP Secondary Cluster: Hardcoded Sensitive Data | 888 | 2417 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Password Management: Empty Password in Configuration File |

### References

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

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

## CWE-259: Use of Hard-coded Password

**Weakness ID :** 259
**Structure :** Simple
**Abstraction :** Variant

### Description

The product contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.

### Extended Description

A hard-coded password typically leads to a significant authentication failure that can be difficult for the system administrator to detect. Once 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 for a hard-coded password.
Outbound: the product connects to another system or component, and it contains hard-coded password 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 | 798 | Use of Hard-coded Credentials | 1699 |
| PeerOf | B | 257 | Storing Passwords in a Recoverable Format | 625 |
| PeerOf | V | 321 | Use of Hard-coded Cryptographic Key | 792 |
| PeerOf | B | 257 | Storing Passwords in a Recoverable Format | 625 |
| CanFollow | C | 656 | Reliance on Security Through Obscurity | 1452 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | B | 798 | Use of Hard-coded Credentials | 1699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | B | 798 | Use of Hard-coded Credentials | 1699 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

| Scope | Impact | Likelihood |
| --- | --- | --- |
| | *If hard-coded passwords are used, it is almost certain that malicious users will gain access through the account in question.* | |

## Detection Methods

### Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

### Black Box

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

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Architecture and Design

For outbound authentication: store passwords 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.

### Phase: Architecture and Design

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

### Phase: Architecture and Design

Perform access control checks and limit which entities can access the feature that requires the hard-coded password. 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: apply strong one-way hashes to your 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 receiving an incoming password during authentication, take the hash of the password and compare it to the hash that you have saved. Use randomly assigned salts for each separate hash that you generate. 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 which 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 used 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 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 4:**

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-2022-29964** | Distributed Control System (DCS) has hard-coded passwords for local shell access<br>*https://www.cve.org/CVERecord?id=CVE-2022-29964* |
| **CVE-2021-37555** | Telnet service for IoT feeder for dogs and cats has hard-coded password [REF-1288]<br>*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<br>*https://www.cve.org/CVERecord?id=CVE-2021-35033* |

## 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 | 2335 |
| MemberOf | C | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 753 | 2009 Top 25 - Porous Defenses | 750 | 2374 |
| MemberOf | C | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 950 | SFP Secondary Cluster: Hardcoded Sensitive Data | 888 | 2417 |
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

This entry could be split into multiple variants: an inbound variant (as seen in the second demonstrative example) and an outbound variant (as seen in the first demonstrative example). These variants are likely to have different consequences, detectability, etc. More importantly, from a vulnerability theory perspective, they could be characterized as different behaviors.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Password Management: Hard-Coded Password |
| CLASP | | | Use of hard-coded password |
| OWASP Top Ten 2004 | A3 | CWE More Specific | Broken Authentication and Session Management |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC03-J | | Never hard code sensitive information |
| Software Fault Patterns | SFP33 | | Hardcoded sensitive data |

## References

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

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

[REF-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-260: Password in Configuration File

**Weakness ID :** 260
**Structure :** Simple
**Abstraction :** Base

## Description

The product stores a password in a configuration file that might be accessible to actors who do not know the password.

## Extended Description

This can result in compromise of the system for which the password is used. An attacker could gain access to this file and learn the stored password or worse yet, change the password to one of their choosing.

## Relationships

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 | 522 | Insufficiently Protected Credentials | 1234 |
| ParentOf | V | 13 | ASP.NET Misconfiguration: Password in Configuration File | 13 |
| ParentOf | V | 258 | Empty Password in Configuration File | 628 |
| ParentOf | V | 555 | J2EE Misconfiguration: Plaintext Password in Configuration File | 1279 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 255 | Credentials Management Errors | 2336 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

## 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 storing passwords in easily accessible locations.

**Phase: Architecture and Design**

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

## Demonstrative Examples

**Example 1:**

Below is a snippet from a Java properties file.

*Example Language: Java* *(Bad)*

```
webapp.ldap.username = secretUsername
webapp.ldap.password = secretPassword
```

Because the LDAP credentials are stored in plaintext, anyone with access to the file can gain access to the resource.

**Example 2:**

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.

## Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2022-38665** | A continuous delivery pipeline management tool stores an unencypted password in a configuration file.<br>*https://www.cve.org/CVERecord?id=CVE-2022-38665* |

## Affected Resources

- 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 | 254 | 7PK - Security Features | 700 | 2335 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1349 | OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration | 1344 | 2514 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Password Management: Password in Configuration File |

### References

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

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

## CWE-261: Weak Encoding for Password

**Weakness ID :** 261
**Structure :** Simple
**Abstraction :** Base

### Description

Obscuring a password with a trivial encoding does not protect the password.

### Extended Description

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password.

### Relationships

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 | | 522 | Insufficiently Protected Credentials | 1234 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

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

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

## 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

Passwords should be encrypted with keys that are at least 128 bits in length for adequate security.

## Demonstrative Examples

### Example 1:

The following code reads a password from a properties file and uses the password to connect to a database.

*Example Language: Java* *(Bad)*

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = Base64.decode(prop.getProperty("password"));
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone with access to config.properties can read the value of password and easily determine that the value has been base 64 encoded. If a devious employee has access to this information, they can use it to break into the system.

### Example 2:

The following code reads a password from the registry and uses the password to create a new network credential.

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

```
...
string value = regKey.GetValue(passKey).ToString();
byte[] decVal = Convert.FromBase64String(value);
NetworkCredential netCred = newNetworkCredential(username,decVal.toString(),domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

### 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 | 2335 |
| MemberOf | C | 729 | OWASP Top Ten 2004 Category A8 - Insecure Storage | 711 | 2359 |
| MemberOf | C | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Notes

#### Other

The "crypt" family of functions uses weak cryptographic algorithms and should be avoided. It may be present in some projects for compatibility.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Password Management: Weak Cryptography |
| OWASP Top Ten 2004 | A8 | CWE More Specific | Insecure Storage |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 55 | Rainbow Table Password Cracking |

### References

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

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

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

## CWE-262: Not Using Password Aging

**Weakness ID :** 262
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not have a mechanism in place for managing password aging.

### Extended Description

Password aging (or password rotation) is a policy that forces users to change their passwords after a defined time period passes, such as every 30 or 90 days. Without mechanisms such as aging, users might not change their passwords in a timely manner.

Note that while password aging was once considered an important security feature, it has since fallen out of favor by many, because it is not as effective against modern threats compared to other mechanisms such as slow hashes. In addition, forcing frequent changes can unintentionally encourage users to select less-secure passwords. However, password aging is still in use due to factors such as compliance requirements, e.g., Payment Card Industry Data Security Standard (PCI DSS).

## Relationships

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 | 2279 |
| PeerOf | B | 309 | Use of Password System for Primary Authentication | 761 |
| PeerOf | B | 324 | Use of a Key Past its Expiration Date | 799 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 255 | Credentials Management Errors | 2336 |

## Applicable Platforms

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

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*As passwords age, the probability that they are compromised grows.* | |

## Potential Mitigations

**Phase: Architecture and Design**

As part of a product's design, require users to change their passwords regularly and avoid reusing previous passwords.

**Phase: Implementation**

Developers might disable clipboard paste operations into password fields as a way to discourage users from pasting a password into a clipboard. However, this might encourage users to choose less-secure passwords that are easier to type, and it can reduce the usability of password managers [REF-1294].

*Effectiveness = Discouraged Common Practice*

## Demonstrative Examples

**Example 1:**

A system does not enforce the changing of passwords every certain period.

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 951 | SFP Secondary Cluster: Insecure Authentication Policy | 888 | 2417 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Not allowing password aging |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 16 | Dictionary-based Password Attack |
| 49 | Password Brute Forcing |
| 55 | Rainbow Table Password Cracking |
| 70 | Try Common or Default Usernames and Passwords |
| 509 | Kerberoasting |
| 555 | Remote Services with Stolen Credentials |
| 560 | Use of Known Domain Credentials |
| 561 | Windows Admin Shares with Stolen Credentials |
| 565 | Password Spraying |
| 600 | Credential Stuffing |
| 652 | Use of Known Kerberos Credentials |
| 653 | Use of Known Operating System Credentials |

## References

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

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

[REF-1305]Kurt Seifried and other members of the CWE-Research mailing list. "Discussion Thread: Time to retire CWE-262 and CWE-263". 2021 December 3. < https://www.mail-archive.com/cwe-research-list@mitre.org/msg00018.html >.2022-10-11.

[REF-1289]Lance Spitzner. "Time for Password Expiration to Die". 2021 June 7. < https://www.sans.org/blog/time-for-password-expiration-to-die/ >.

[REF-1290]Lorrie Cranor. "Time to rethink mandatory password changes". 2016 March 2. < https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2016/03/time-rethink-mandatory-password-changes >.

[REF-1291]Eugene Spafford. "Security Myths and Passwords". 2006 April 9. < https://www.cerias.purdue.edu/site/blog/post/password-change-myths/ >.

[REF-1292]National Cyber Security Centre. "Password administration for system owners". 2018 November 9. < https://www.ncsc.gov.uk/collection/passwords >.2023-04-07.

[REF-1293]NIST. "Digital Identity Guidelines: Authentication and Lifecycle Management(SP 800-63B)". 2017 June. < https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf >.2023-04-07.

[REF-1294]National Cyber Security Centre. "Let them paste passwords". 2017 January 2. < https://www.ncsc.gov.uk/blog-post/let-them-paste-passwords >.2023-04-07.

## CWE-263: Password Aging with Long Expiration

**Weakness ID :** 263
**Structure :** Simple
**Abstraction :** Base

### Description

The product supports password aging, but the expiration period is too long.

### Extended Description

Password aging (or password rotation) is a policy that forces users to change their passwords after a defined time period passes, such as every 30 or 90 days. A long expiration provides more time for attackers to conduct password cracking before users are forced to change to a new password.

Note that while password aging was once considered an important security feature, it has since fallen out of favor by many, because it is not as effective against modern threats compared to other mechanisms such as slow hashes. In addition, forcing frequent changes can unintentionally encourage users to select less-secure passwords. However, password aging is still in use due to factors such as compliance requirements, e.g., Payment Card Industry Data Security Standard (PCI DSS).

### Relationships

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 | 2279 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 255 | Credentials Management Errors | 2336 |

### Applicable Platforms

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

### Likelihood Of Exploit

Low

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

| Scope | Impact | Likelihood |
|---|---|---|
| | *As passwords age, the probability that they are compromised grows.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Ensure that password aging is limited so that there is a defined maximum age for passwords. Note that if the expiration window is too short, it can cause users to generate poor or predictable passwords.

**Phase: Architecture and Design**

Ensure that the user is notified several times leading up to the password expiration.

**Phase: Architecture and Design**

Create mechanisms to prevent users from reusing passwords or creating similar passwords.

**Phase: Implementation**

Developers might disable clipboard paste operations into password fields as a way to discourage users from pasting a password into a clipboard. However, this might encourage users to choose less-secure passwords that are easier to type, and it can reduce the usability of password managers [REF-1294].

*Effectiveness = Discouraged Common Practice*

## Demonstrative Examples

**Example 1:**

A system requires the changing of passwords every five years.

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 951 | SFP Secondary Cluster: Insecure Authentication Policy | 888 | 2417 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Allowing password aging |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 16 | Dictionary-based Password Attack |
| 49 | Password Brute Forcing |
| 55 | Rainbow Table Password Cracking |
| 70 | Try Common or Default Usernames and Passwords |
| 509 | Kerberoasting |
| 555 | Remote Services with Stolen Credentials |
| 560 | Use of Known Domain Credentials |
| 561 | Windows Admin Shares with Stolen Credentials |
| 565 | Password Spraying |
| 600 | Credential Stuffing |
| 652 | Use of Known Kerberos Credentials |
| 653 | Use of Known Operating System Credentials |

### References

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

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

[REF-1305]Kurt Seifried and other members of the CWE-Research mailing list. "Discussion Thread: Time to retire CWE-262 and CWE-263". 2021 December 3. < https://www.mail-archive.com/cwe-research-list@mitre.org/msg00018.html >.2022-10-11.

[REF-1289]Lance Spitzner. "Time for Password Expiration to Die". 2021 June 7. < https://www.sans.org/blog/time-for-password-expiration-to-die/ >.

[REF-1290]Lorrie Cranor. "Time to rethink mandatory password changes". 2016 March 2. < https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2016/03/time-rethink-mandatory-password-changes >.

[REF-1291]Eugene Spafford. "Security Myths and Passwords". 2006 April 9. < https://www.cerias.purdue.edu/site/blog/post/password-change-myths/ >.

[REF-1292]National Cyber Security Centre. "Password administration for system owners". 2018 November 9. < https://www.ncsc.gov.uk/collection/passwords >.2023-04-07.

[REF-1293]NIST. "Digital Identity Guidelines: Authentication and Lifecycle Management(SP 800-63B)". 2017 June. < https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf >.2023-04-07.

[REF-1294]National Cyber Security Centre. "Let them paste passwords". 2017 January 2. < https://www.ncsc.gov.uk/blog-post/let-them-paste-passwords >.2023-04-07.

## CWE-266: Incorrect Privilege Assignment

**Weakness ID :** 266
**Structure :** Simple
**Abstraction :** Base

### Description

A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor.

### Relationships

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 | 269 | Improper Privilege Management | 653 |
| ParentOf | V | 9 | J2EE Misconfiguration: Weak Access Permissions for EJB Methods | 8 |
| ParentOf | V | 520 | .NET Misconfiguration: Use of Impersonation | 1230 |
| ParentOf | V | 556 | ASP.NET Misconfiguration: Use of Identity Impersonation | 1280 |
| ParentOf | V | 1022 | Use of Web Link to Untrusted Target with window.opener Access | 1872 |
| CanAlsoBe | C | 286 | Incorrect User Management | 698 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Weakness Ordinalities

**Resultant :**

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br><br>*A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.* | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

**Phase: Operation**

*Strategy = Environment Hardening*

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

## Demonstrative Examples

**Example 1:**

The following example demonstrates the weakness.

*Example Language: C* *(Bad)*

```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

**Example 2:**

The following example demonstrates the weakness.

*Example Language: Java* *(Bad)*

```
AccessController.doPrivileged(new PrivilegedAction() {
  public Object run() {
    // privileged code goes here, for example:
    System.loadLibrary("awt");
    return null;
    // nothing to return
```

```
}
```

**Example 3:**

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-1999-1193** | untrusted user placed in unix "wheel" group *https://www.cve.org/CVERecord?id=CVE-1999-1193* |
| **CVE-2005-2741** | Product allows users to grant themselves certain rights that can be used to escalate privileges. *https://www.cve.org/CVERecord?id=CVE-2005-2741* |
| **CVE-2005-2496** | Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue. *https://www.cve.org/CVERecord?id=CVE-2005-2496* |
| **CVE-2004-0274** | Product mistakenly assigns a particular status to an entity, leading to increased privileges. *https://www.cve.org/CVERecord?id=CVE-2004-0274* |

## Affected Resources

- System Process

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control | 711 | 2356 |
| MemberOf | C | 859 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) | 844 | 2390 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 1149 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC) | 1133 | 2473 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Incorrect Privilege Assignment |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC00-J | | Do not allow privileged blocks to leak sensitive information across a trust boundary |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC01-J | | Do not allow tainted variables in privileged blocks |

### References

[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-267: Privilege Defined With Unsafe Actions

**Weakness ID :** 267
**Structure :** Simple
**Abstraction :** Base

### Description

A particular privilege, role, capability, or right can be used to perform unsafe actions that were not intended, even when it is assigned to the correct entity.

### Relationships

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 | 269 | Improper Privilege Management | 653 |
| ParentOf | V | 623 | Unsafe ActiveX Control Marked Safe For Scripting | 1397 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
| --- | --- | --- |
| Access Control | Gain Privileges or Assume Identity<br>*A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.* | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

**Phase: Operation**

*Strategy = Environment Hardening*

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

## Demonstrative Examples

**Example 1:**

This code intends to allow only Administrators to print debug information about a system.

*Example Language: Java* *(Bad)*

```
public enum Roles {
  ADMIN,USER,GUEST
}
public void printDebugInfo(User requestingUser){
  if(isAuthenticated(requestingUser)){
    switch(requestingUser.role){
      case GUEST:
        System.out.println("You are not authorized to perform this command");
        break;
      default:
        System.out.println(currentDebugState());
        break;
    }
  }
  else{
    System.out.println("You must be logged in to perform this command");
  }
}
```

While the intention was to only allow Administrators to print the debug information, the code as written only excludes those with the role of "GUEST". Someone with the role of "ADMIN" or "USER" will be allowed access, which goes against the original intent. An attacker may be able to use this debug information to craft an attack on the system.

## Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2002-1981** | Roles have access to dangerous procedures (Accessible entities).<br>*https://www.cve.org/CVERecord?id=CVE-2002-1981* |
| **CVE-2002-1671** | Untrusted object/method gets access to clipboard (Accessible entities).<br>*https://www.cve.org/CVERecord?id=CVE-2002-1671* |

| Reference | Description |
|---|---|
| **CVE-2004-2204** | Gain privileges using functions/tags that should be restricted (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2004-2204* |
| **CVE-2000-0315** | Traceroute program allows unprivileged users to modify source address of packet (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2000-0315* |
| **CVE-2004-0380** | Bypass domain restrictions using a particular file that references unsafe URI schemes (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2004-0380* |
| **CVE-2002-1154** | Script does not restrict access to an update command, leading to resultant disk consumption and filled error logs (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2002-1154* |
| **CVE-2002-1145** | "public" database user can use stored procedure to modify data controlled by the database owner (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2002-1145* |
| **CVE-2000-0506** | User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2000-0506* |
| **CVE-2002-2042** | Allows attachment to and modification of privileged processes (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2002-2042* |
| **CVE-2000-1212** | User with privilege can edit raw underlying object using unprotected method (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2000-1212* |
| **CVE-2005-1742** | Inappropriate actions allowed by a particular role(Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2005-1742* |
| **CVE-2001-1480** | Untrusted entity allowed to access the system clipboard (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2001-1480* |
| **CVE-2001-1551** | Extra Linux capability allows bypass of system-specified restriction (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2001-1551* |
| **CVE-2001-1166** | User with debugging rights can read entire process (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2001-1166* |
| **CVE-2005-1816** | Non-root admins can add themselves or others to the root admin group (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2005-1816* |
| **CVE-2005-2173** | Users can change certain properties of objects to perform otherwise unauthorized actions (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2005-2173* |
| **CVE-2005-2027** | Certain debugging commands not restricted to just the administrator, allowing registry modification and infoleak (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2005-2027* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

Note: there are 2 separate sub-categories here: - privilege incorrectly allows entities to perform certain actions - object is incorrectly accessible to entities with a given privilege

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Unsafe Privilege |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 58 | Restful Privilege Elevation |
| 634 | Probe Audio and Video Peripherals |
| 637 | Collect Data from Clipboard |
| 643 | Identify Shared Files/Directories on System |
| 648 | Collect Data from Screen Capture |

## References

[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-268: Privilege Chaining

**Weakness ID :** 268
**Structure :** Simple
**Abstraction :** Base

## Description

Two distinct privileges, roles, capabilities, or rights can be combined in a way that allows an entity to perform unsafe actions that would not be allowed without that combination.

## Relationships

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 | 269 | Improper Privilege Management | 653 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

### Likelihood Of Exploit

High

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*A user can be given or gain access rights of another user. This can give the user unauthorized access to sensitive information including the access information of another user.* | |

### Potential Mitigations

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

**Phase: Operation**

*Strategy = Environment Hardening*

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

### Demonstrative Examples

**Example 1:**

This code allows someone with the role of "ADMIN" or "OPERATOR" to reset a user's password. The role of "OPERATOR" is intended to have less privileges than an "ADMIN", but still be able to help users with small issues such as forgotten passwords.

*Example Language: Java* *(Bad)*

```
public enum Roles {
  ADMIN,OPERATOR,USER,GUEST
}
public void resetPassword(User requestingUser, User user, String password ){
  if(isAuthenticated(requestingUser)){
    switch(requestingUser.role){
      case GUEST:
        System.out.println("You are not authorized to perform this command");
        break;
      case USER:
        System.out.println("You are not authorized to perform this command");
        break;
      default:
        setPassword(user,password);
        break;
    }
```

```
    }
  else{
    System.out.println("You must be logged in to perform this command");
  }
}
```

This code does not check the role of the user whose password is being reset. It is possible for an Operator to gain Admin privileges by resetting the password of an Admin account and taking control of that account.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-1736** | Chaining of user rights. *https://www.cve.org/CVERecord?id=CVE-2005-1736* |
| **CVE-2002-1772** | Gain certain rights via privilege chaining in alternate channel. *https://www.cve.org/CVERecord?id=CVE-2002-1772* |
| **CVE-2005-1973** | Application is allowed to assign extra permissions to itself. *https://www.cve.org/CVERecord?id=CVE-2005-1973* |
| **CVE-2003-0640** | "operator" user can overwrite usernames and passwords to gain admin privileges. *https://www.cve.org/CVERecord?id=CVE-2003-0640* |

### 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 | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control | 711 | 2356 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1364 | ICS Communications: Zone Boundary Failures | 1358 | 2522 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Notes

**Relationship**

There is some conceptual overlap with Unsafe Privilege.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Privilege Chaining |

### References

[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-269: Improper Privilege Management

**Weakness ID :** 269
**Structure :** Simple
**Abstraction :** Class

## Description

The product does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.

## Relationships

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |
| ParentOf | B | 250 | Execution with Unnecessary Privileges | 606 |
| ParentOf | B | 266 | Incorrect Privilege Assignment | 645 |
| ParentOf | B | 267 | Privilege Defined With Unsafe Actions | 648 |
| ParentOf | B | 268 | Privilege Chaining | 651 |
| ParentOf | B | 270 | Privilege Context Switching Error | 659 |
| ParentOf | C | 271 | Privilege Dropping / Lowering Errors | 660 |
| ParentOf | B | 274 | Improper Handling of Insufficient Privileges | 670 |
| ParentOf | B | 648 | Incorrect Use of Privileged APIs | 1437 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Likelihood Of Exploit

Medium

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

## Detection Methods

**Automated Static Analysis**

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

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Follow the principle of least privilege when assigning access rights to entities in a software system.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

## Demonstrative Examples

**Example 1:**

This code temporarily raises the program's privileges to allow creation of a new user folder.

*Example Language: Python* *(Bad)*

```
def makeNewUserDir(username):
  if invalidUsername(username):
    #avoid CWE-22 and CWE-78
    print('Usernames cannot contain invalid characters')
    return False
  try:
    raisePrivileges()
    os.mkdir('/home/' + username)
    lowerPrivileges()
  except OSError:
    print('Unable to create new user directory for user:' + username)
    return False
  return True
```

While the program only raises its privilege level to create the folder and immediately lowers it again, if the call to os.mkdir() throws an exception, the call to lowerPrivileges() will not occur. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

**Example 2:**

The following example demonstrates the weakness.

*Example Language: C* *(Bad)*

```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

**Example 3:**

The following example demonstrates the weakness.

*Example Language: Java* *(Bad)*

```
AccessController.doPrivileged(new PrivilegedAction() {
  public Object run() {
    // privileged code goes here, for example:
    System.loadLibrary("awt");
    return null;
    // nothing to return
  }
```

**Example 4:**

This code intends to allow only Administrators to print debug information about a system.

*Example Language: Java* *(Bad)*

```
public enum Roles {
  ADMIN,USER,GUEST
}
public void printDebugInfo(User requestingUser){
  if(isAuthenticated(requestingUser)){
    switch(requestingUser.role){
      case GUEST:
        System.out.println("You are not authorized to perform this command");
        break;
      default:
        System.out.println(currentDebugState());
        break;
    }
  }
  else{
    System.out.println("You must be logged in to perform this command");
  }
}
```

While the intention was to only allow Administrators to print the debug information, the code as written only excludes those with the role of "GUEST". Someone with the role of "ADMIN" or "USER" will be allowed access, which goes against the original intent. An attacker may be able to use this debug information to craft an attack on the system.

**Example 5:**

This code allows someone with the role of "ADMIN" or "OPERATOR" to reset a user's password. The role of "OPERATOR" is intended to have less privileges than an "ADMIN", but still be able to help users with small issues such as forgotten passwords.

*Example Language: Java* *(Bad)*

```
public enum Roles {
  ADMIN,OPERATOR,USER,GUEST
}
public void resetPassword(User requestingUser, User user, String password ){
  if(isAuthenticated(requestingUser)){
    switch(requestingUser.role){
      case GUEST:
        System.out.println("You are not authorized to perform this command");
        break;
      case USER:
        System.out.println("You are not authorized to perform this command");
        break;
      default:
        setPassword(user,password);
        break;
      }
    }
  else{
    System.out.println("You must be logged in to perform this command");
  }
}
```

This code does not check the role of the user whose password is being reset. It is possible for an Operator to gain Admin privileges by resetting the password of an Admin account and taking control of that account.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1555** | Terminal privileges are not reset when a user logs out. *https://www.cve.org/CVERecord?id=CVE-2001-1555* |
| **CVE-2001-1514** | Does not properly pass security context to child processes in certain cases, allows privilege escalation. *https://www.cve.org/CVERecord?id=CVE-2001-1514* |
| **CVE-2001-0128** | Does not properly compute roles. *https://www.cve.org/CVERecord?id=CVE-2001-0128* |
| **CVE-1999-1193** | untrusted user placed in unix "wheel" group *https://www.cve.org/CVERecord?id=CVE-1999-1193* |
| **CVE-2005-2741** | Product allows users to grant themselves certain rights that can be used to escalate privileges. *https://www.cve.org/CVERecord?id=CVE-2005-2741* |
| **CVE-2005-2496** | Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue. *https://www.cve.org/CVERecord?id=CVE-2005-2496* |
| **CVE-2004-0274** | Product mistakenly assigns a particular status to an entity, leading to increased privileges. *https://www.cve.org/CVERecord?id=CVE-2004-0274* |
| **CVE-2007-4217** | FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients. *https://www.cve.org/CVERecord?id=CVE-2007-4217* |
| **CVE-2007-5159** | OS incorrectly installs a program with setuid privileges, allowing users to gain privileges. *https://www.cve.org/CVERecord?id=CVE-2007-5159* |
| **CVE-2008-4638** | Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209). *https://www.cve.org/CVERecord?id=CVE-2008-4638* |
| **CVE-2007-3931** | Installation script installs some programs as setuid when they shouldn't be. *https://www.cve.org/CVERecord?id=CVE-2007-3931* |
| **CVE-2002-1981** | Roles have access to dangerous procedures (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2002-1981* |
| **CVE-2002-1671** | Untrusted object/method gets access to clipboard (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2002-1671* |
| **CVE-2000-0315** | Traceroute program allows unprivileged users to modify source address of packet (Accessible entities). *https://www.cve.org/CVERecord?id=CVE-2000-0315* |
| **CVE-2000-0506** | User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions). *https://www.cve.org/CVERecord?id=CVE-2000-0506* |

## 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 | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | V | 1200 | Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors | 1200 | 2608 |

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | V | 1350 | Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses | 1350 | 2615 |
| MemberOf | C | 1364 | ICS Communications: Zone Boundary Failures | 1358 | 2522 |
| MemberOf | C | 1365 | ICS Communications: Unreliability | 1358 | 2523 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1373 | ICS Engineering (Construction/Deployment): Trust Model Problems | 1358 | 2531 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |
| MemberOf | V | 1425 | Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses | 1425 | 2621 |
| MemberOf | V | 1430 | Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses | 1430 | 2622 |

## Notes

### Maintenance

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Privilege Management Error |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.08 BR |
| ISA/IEC 62443 | Part 3-2 | | Req CR 3.1 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.2 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 2.1 |
| ISA/IEC 62443 | Part 4-1 | | Req SD-3 |
| ISA/IEC 62443 | Part 4-1 | | Req SD-4 |
| ISA/IEC 62443 | Part 4-1 | | Req SI-1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 2.1 |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 58 | Restful Privilege Elevation |
| 122 | Privilege Abuse |
| 233 | Privilege Escalation |

## References

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

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

[REF-1287]MITRE. "Supplemental Details - 2022 CWE Top 25". 2022 June 8. < https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25_supplemental.html#problematicMappingDetails >.2024-11-17.

## CWE-270: Privilege Context Switching Error

**Weakness ID :** 270
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not properly manage privileges while it is switching between different contexts that have different privileges or spheres of 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 | 269 | Improper Privilege Management | 653 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*A user can assume the identity of another user with separate privileges in another context. This will give the user unauthorized access that may allow them to acquire the access information of other users.* | |

### Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

**Phase: Operation**

*Strategy = Environment Hardening*

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

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1688** | Web browser cross domain problem when user hits "back" button. *https://www.cve.org/CVERecord?id=CVE-2002-1688* |
| **CVE-2003-1026** | Web browser cross domain problem when user hits "back" button. *https://www.cve.org/CVERecord?id=CVE-2003-1026* |
| **CVE-2002-1770** | Cross-domain issue - third party product passes code to web browser, which executes it in unsafe zone. *https://www.cve.org/CVERecord?id=CVE-2002-1770* |
| **CVE-2005-2263** | Run callback in different security context after it has been changed from untrusted to trusted. * note that "context switch before actions are completed" is one type of problem that happens frequently, espec. in browsers. *https://www.cve.org/CVERecord?id=CVE-2005-2263* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Research Gap

This concept needs more study.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Privilege Context Switching Error |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 17 | Using Malicious Files |
| 30 | Hijacking a Privileged Thread of Execution |
| 35 | Leverage Executable Code in Non-Executable Files |

## 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-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-271: Privilege Dropping / Lowering Errors

**Weakness ID :** 271
**Structure :** Simple

**Abstraction :** Class

## Description

The product does not drop privileges before passing control of a resource to an actor that does not have those privileges.

## Extended Description

In some contexts, a system executing with elevated permissions will hand off a process/file/etc. to another process or user. If the privileges of an entity are not reduced, then elevated privileges are spread throughout a system and possibly to an attacker.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 269 | Improper Privilege Management | 653 |
| ParentOf | B | 272 | Least Privilege Violation | 663 |
| ParentOf | B | 273 | Improper Check for Dropped Privileges | 667 |
| PeerOf | B | 274 | Improper Handling of Insufficient Privileges | 670 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.* | |
| Access Control<br>Non-Repudiation | Gain Privileges or Assume Identity<br>Hide Activities<br>*If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.* | |

## Potential Mitigations

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

## Demonstrative Examples

**Example 1:**

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

*Example Language: C* *(Bad)*

```
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2000-1213** | Program does not drop privileges after acquiring the raw socket.<br>*https://www.cve.org/CVERecord?id=CVE-2000-1213* |
| **CVE-2001-0559** | Setuid program does not drop privileges after a parsing error occurs, then calls another program to handle the error.<br>*https://www.cve.org/CVERecord?id=CVE-2001-0559* |
| **CVE-2001-0787** | Does not drop privileges in related groups when lowering privileges.<br>*https://www.cve.org/CVERecord?id=CVE-2001-0787* |
| **CVE-2002-0080** | Does not drop privileges in related groups when lowering privileges.<br>*https://www.cve.org/CVERecord?id=CVE-2002-0080* |
| **CVE-2001-1029** | Does not drop privileges before determining access to certain files.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1029* |
| **CVE-1999-0813** | Finger daemon does not drop privileges when executing programs on behalf of the user being fingered.<br>*https://www.cve.org/CVERecord?id=CVE-1999-0813* |
| **CVE-1999-1326** | FTP server does not drop privileges if a connection is aborted during file transfer. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-1999-1326* |
| **CVE-2000-0172** | Program only uses seteuid to drop privileges. *https://www.cve.org/CVERecord?id=CVE-2000-0172* |
| **CVE-2004-2504** | Windows program running as SYSTEM does not drop privileges before executing other programs (many others like this, especially involving the Help facility). *https://www.cve.org/CVERecord?id=CVE-2004-2504* |
| **CVE-2004-0213** | Utility Manager launches winhlp32.exe while running with raised privileges, which allows local users to gain system privileges. *https://www.cve.org/CVERecord?id=CVE-2004-0213* |
| **CVE-2004-0806** | Setuid program does not drop privileges before executing program specified in an environment variable. *https://www.cve.org/CVERecord?id=CVE-2004-0806* |
| **CVE-2004-0828** | Setuid program does not drop privileges before processing file specified on command line. *https://www.cve.org/CVERecord?id=CVE-2004-0828* |
| **CVE-2004-2070** | Service on Windows does not drop privileges before using "view file" option, allowing code execution. *https://www.cve.org/CVERecord?id=CVE-2004-2070* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Privilege Dropping / Lowering Errors |

## References

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

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

# CWE-272: Least Privilege Violation

**Weakness ID :** 272
**Structure :** Simple
**Abstraction :** Base

## Description

The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.

## Relationships

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 | | 271 | Privilege Dropping / Lowering Errors | 660 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Confidentiality | Gain Privileges or Assume Identity<br>Read Application Data<br>Read Files or Directories<br>*An attacker may be able to access resources with the elevated privilege that could not be accessed with the attacker's original privileges. This is particularly likely in conjunction with another flaw, such as a buffer overflow.* | |

## Detection Methods

**Automated Static Analysis - Binary or Bytecode**

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

*Effectiveness = SOAR Partial*

**Dynamic Analysis with Automated Results Interpretation**

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

*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*

### Automated Static Analysis

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

*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**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Follow the principle of least privilege when assigning access rights to entities in a software system.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

## Demonstrative Examples

### Example 1:

The following example demonstrates the weakness.

*Example Language: C* *(Bad)*

```c
setuid(0);
// Do some important stuff
setuid(old_uid);
// Do some non privileged stuff.
```

### Example 2:

The following example demonstrates the weakness.

*Example Language: Java* *(Bad)*

```java
AccessController.doPrivileged(new PrivilegedAction() {
```

```
public Object run() {
  // privileged code goes here, for example:
  System.loadLibrary("awt");
  return null;
  // nothing to return
}
```

**Example 3:**

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

*Example Language: C* *(Bad)*

```
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

## 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 | 2335 |
| MemberOf | C | 748 | CERT C Secure Coding Standard (2008) Appendix - POSIX (POS) | 734 | 2372 |
| MemberOf | C | 859 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) | 844 | 2390 |
| MemberOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1149 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC) | 1133 | 2473 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

### Other

If system privileges are not dropped when it is reasonable to do so, this is not a vulnerability by itself. According to the principle of least privilege, access should be allowed only when it is absolutely necessary to the function of a given system, and only for the minimal necessary amount of time. Any further allowance of privilege widens the window of time during which a successful exploitation of the system will provide an attacker with that same privilege. If at all possible, limit the allowance of system privilege to small, simple sections of code that may be called atomically. When a program calls a privileged function, such as chroot(), it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Least Privilege Violation |
| CLASP | | | Failure to drop privileges when reasonable |
| CERT C Secure Coding | POS02-C | | Follow the principle of least privilege |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC00-J | | Do not allow privileged blocks to leak sensitive information across a trust boundary |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC01-J | | Do not allow tainted variables in privileged blocks |
| Software Fault Patterns | SFP36 | | Privilege |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 17 | Using Malicious Files |
| 35 | Leverage Executable Code in Non-Executable Files |
| 76 | Manipulating Web Input to File System Calls |

### 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-273: Improper Check for Dropped Privileges

**Weakness ID :** 273
**Structure :** Simple
**Abstraction :** Base

### Description

The product attempts to drop privileges but does not check or incorrectly checks to see if the drop succeeded.

### Extended Description

If the drop fails, the product will continue to run with the raised privileges, which might provide additional access to unprivileged users.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 271 | Privilege Dropping / Lowering Errors | 660 |
| ChildOf | C | 754 | Improper Check for Unusual or Exceptional Conditions | 1577 |

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| PeerOf | B | 252 | Unchecked Return Value | 613 |

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

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Background Details

In Windows based environments that have access control, impersonation is used so that access checks can be performed on a client identity by a server with higher privileges. By impersonating the client, the server is restricted to client-level security -- although in different threads it may have much higher privileges.

## Likelihood Of Exploit

Medium

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.* | |
| Access Control<br>Non-Repudiation | Gain Privileges or Assume Identity<br>Hide Activities<br>*If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.* | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

**Phase: Implementation**

Check the results of all functions that return a value and verify that the value is expected.

*Effectiveness = High*

*Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.*

**Phase: Implementation**

In Windows, make sure that the process token has the SeImpersonatePrivilege(Microsoft Server 2003). Code that relies on impersonation for security must ensure that the impersonation succeeded, i.e., that a proper privilege demotion happened.

## Demonstrative Examples

**Example 1:**

This code attempts to take on the privileges of a user before creating a file, thus avoiding performing the action with unnecessarily high privileges:

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

```
bool DoSecureStuff(HANDLE hPipe) {
  bool fDataWritten = false;
  ImpersonateNamedPipeClient(hPipe);
  HANDLE hFile = CreateFile(...);
  /../
  RevertToSelf()
  /../
}
```

The call to ImpersonateNamedPipeClient may fail, but the return value is not checked. If the call fails, the code may execute with higher privileges than intended. In this case, an attacker could exploit this behavior to write a file to a location that the attacker does not have access to.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2006-4447** | Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://www.cve.org/CVERecord?id=CVE-2006-4447* |
| **CVE-2006-2916** | Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://www.cve.org/CVERecord?id=CVE-2006-2916* |

## Affected Resources

- System Process

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 748 | CERT C Secure Coding Standard (2008) Appendix - POSIX (POS) | 734 | 2372 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 962 | SFP Secondary Cluster: Unchecked Status Condition | 888 | 2421 |
| MemberOf | C | 1171 | SEI CERT C Coding Standard - Guidelines 50. POSIX (POS) | 1154 | 2484 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure to check whether privileges were dropped successfully |
| CERT C Secure Coding | POS37-C | Exact | Ensure that privilege relinquishment is successful |
| Software Fault Patterns | SFP4 | | Unchecked Status Condition |

### References

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

## CWE-274: Improper Handling of Insufficient Privileges

**Weakness ID :** 274
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not handle or incorrectly handles when it has insufficient privileges to perform an operation, leading to resultant weaknesses.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 269 | Improper Privilege Management | 653 |
| ChildOf | C | 755 | Improper Handling of Exceptional Conditions | 1585 |
| PeerOf | C | 271 | Privilege Dropping / Lowering Errors | 660 |
| CanAlsoBe | B | 280 | Improper Handling of Insufficient Permissions or Privileges | 679 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Other<br>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*

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1564** | System limits are not properly enforced after privileges are dropped.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1564* |
| **CVE-2005-3286** | Firewall crashes when it can't read a critical memory block that was protected by a malicious process.<br>*https://www.cve.org/CVERecord?id=CVE-2005-3286* |
| **CVE-2005-1641** | Does not give admin sufficient privileges to overcome otherwise legitimate user actions.<br>*https://www.cve.org/CVERecord?id=CVE-2005-1641* |

## 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 | 901 | SFP Primary Cluster: Privilege | 888 | 2407 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

CWE-280 and CWE-274 are too similar. It is likely that CWE-274 will be deprecated in the future.

### Relationship

Overlaps dropped privileges, insufficient permissions.

### Theoretical

This has a layering relationship with Unchecked Error Condition and Unchecked Return Value.

### Theoretical

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the product makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Insufficient privileges |

## CWE-276: Incorrect Default Permissions

**Weakness ID :** 276
**Structure :** Simple
**Abstraction :** Base

### Description

During installation, installed file permissions are set to allow anyone to modify those files.

### Relationships

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 | 1559 |

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

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 275 | Permission Issues | 2339 |

### Weakness Ordinalities

**Primary :**

### Applicable Platforms

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

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

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

### Likelihood Of Exploit

Medium

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity | Read Application Data<br>Modify Application Data | |

## Detection Methods

**Automated Static Analysis - Binary or Bytecode**

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inter-application Flow Analysis

*Effectiveness = SOAR Partial*

**Manual Static Analysis - Binary or Bytecode**

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

*Effectiveness = SOAR Partial*

**Dynamic Analysis with Automated Results Interpretation**

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria Web Application Scanner Web Services Scanner Database Scanners

*Effectiveness = SOAR Partial*

**Dynamic Analysis with Manual Results Interpretation**

According to SOAR, the following detection techniques may be useful: Highly cost effective: Host Application Interface Scanner Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Forced Path Execution

*Effectiveness = High*

**Manual Static Analysis - Source Code**

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

*Effectiveness = High*

**Automated Static Analysis - Source Code**

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

*Effectiveness = SOAR Partial*

**Automated Static Analysis**

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

*Effectiveness = SOAR Partial*

**Architecture or Design Review**

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

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

The architecture needs to access and modification attributes for files to only those users who actually require those actions.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-1941** | Executables installed world-writable. *https://www.cve.org/CVERecord?id=CVE-2005-1941* |
| **CVE-2002-1713** | Home directories installed world-readable. *https://www.cve.org/CVERecord?id=CVE-2002-1713* |
| **CVE-2001-1550** | World-writable log files allow information loss; world-readable file has cleartext passwords. *https://www.cve.org/CVERecord?id=CVE-2001-1550* |
| **CVE-2002-1711** | World-readable directory. *https://www.cve.org/CVERecord?id=CVE-2002-1711* |
| **CVE-2002-1844** | Windows product uses insecure permissions when installing on Solaris (genesis: port error). *https://www.cve.org/CVERecord?id=CVE-2002-1844* |
| **CVE-2001-0497** | Insecure permissions for a shared secret key file. Overlaps cryptographic problem. *https://www.cve.org/CVERecord?id=CVE-2001-0497* |
| **CVE-1999-0426** | Default permissions of a device allow IP spoofing. *https://www.cve.org/CVERecord?id=CVE-1999-0426* |

## 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 | 743 | CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO) | 734 | 2368 |
| MemberOf | C | 857 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO) | 844 | 2389 |
| MemberOf | C | 877 | CERT C++ Secure Coding Section 09 - Input Output (FIO) | 868 | 2398 |
| MemberOf | C | 946 | SFP Secondary Cluster: Insecure Resource Permissions | 888 | 2415 |
| MemberOf | C | 1147 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO) | 1133 | 2471 |
| MemberOf | C | 1198 | Privilege Separation and Access Control Issues | 1194 | 2491 |
| MemberOf | V | 1337 | Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses | 1337 | 2610 |

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 1345 | OWASP Top Ten 2021 Category A01:2021 - Broken Access Control | 1344 | 2508 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1376 | ICS Engineering (Construction/Deployment): Security Gaps in Commissioning | 1358 | 2533 |
| MemberOf | V | 1387 | Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses | 1387 | 2618 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |
| MemberOf | V | 1425 | Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses | 1425 | 2621 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Insecure Default Permissions |
| CERT C Secure Coding | FIO06-C | | Create files with appropriate access permissions |
| The CERT Oracle Secure Coding Standard for Java (2011) | FIO01-J | | Create files with appropriate access permission |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.08 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 2.1 |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 1 | Accessing Functionality Not Properly Constrained by ACLs |
| 81 | Web Server Logs Tampering |
| 127 | Directory Indexing |

### References

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

## CWE-277: Insecure Inherited Permissions

**Weakness ID :** 277
**Structure :** Simple
**Abstraction :** Variant

### Description

A product defines a set of insecure permissions that are inherited by objects that are created by the program.

### Relationships

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 | 732 | Incorrect Permission Assignment for Critical Resource | 1559 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 275 | Permission Issues | 2339 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity | Read Application Data<br>Modify Application Data | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-1841** | User's umask is used when creating temp files.<br>*https://www.cve.org/CVERecord?id=CVE-2005-1841* |
| **CVE-2002-1786** | Insecure umask for core dumps [is the umask preserved or assigned?].<br>*https://www.cve.org/CVERecord?id=CVE-2002-1786* |

## 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 | 946 | SFP Secondary Cluster: Insecure Resource Permissions | 888 | 2415 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Insecure inherited permissions |

# CWE-278: Insecure Preserved Inherited Permissions

**Weakness ID :** 278

**Structure :** Simple
**Abstraction :** Variant

## Description

A product inherits a set of insecure permissions for an object, e.g. when copying from an archive file, without user awareness or involvement.

## Relationships

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 | 1559 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 275 | Permission Issues | 2339 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity | Read Application Data<br>Modify Application Data | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-1724** | Does not obey specified permissions when exporting.<br>*https://www.cve.org/CVERecord?id=CVE-2005-1724* |

## 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 | 946 | SFP Secondary Cluster: Insecure Resource Permissions | 888 | 2415 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Insecure preserved inherited permissions |

## CWE-279: Incorrect Execution-Assigned Permissions

**Weakness ID :** 279
**Structure :** Simple
**Abstraction :** Variant

### Description

While it is executing, the product sets the permissions of an object in a way that violates the intended permissions that have been specified by the user.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 732 | Incorrect Permission Assignment for Critical Resource | 1559 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 275 | Permission Issues | 2339 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity | Read Application Data<br>Modify Application Data | |

### Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

### Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2002-0265** | Log files opened read/write. *https://www.cve.org/CVERecord?id=CVE-2002-0265* |
| **CVE-2003-0876** | Log files opened read/write. *https://www.cve.org/CVERecord?id=CVE-2003-0876* |
| **CVE-2002-1694** | Log files opened read/write. *https://www.cve.org/CVERecord?id=CVE-2002-1694* |

### 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 | 743 | CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO) | 734 | 2368 |
| MemberOf | C | 857 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO) | 844 | 2389 |
| MemberOf | C | 877 | CERT C++ Secure Coding Section 09 - Input Output (FIO) | 868 | 2398 |
| MemberOf | C | 946 | SFP Secondary Cluster: Insecure Resource Permissions | 888 | 2415 |
| MemberOf | C | 1147 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO) | 1133 | 2471 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
| --- | --- | --- | --- |
| PLOVER | | | Insecure execution-assigned permissions |
| CERT C Secure Coding | FIO06-C | | Create files with appropriate access permissions |
| The CERT Oracle Secure Coding Standard for Java (2011) | FIO01-J | | Create files with appropriate access permission |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
| --- | --- |
| 81 | Web Server Logs Tampering |

## CWE-280: Improper Handling of Insufficient Permissions or Privileges

**Weakness ID :** 280
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not handle or incorrectly handles when it has insufficient privileges to access resources or functionality as specified by their permissions. This may cause it to follow unexpected code paths that may leave the product in an invalid state.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 755 | Improper Handling of Exceptional Conditions | 1585 |
| PeerOf | C | 636 | Not Failing Securely ('Failing Open') | 1409 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 265 | Privilege Issues | 2338 |
| MemberOf | C | 275 | Permission Issues | 2339 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Other<br>Alter Execution Logic | |

## Potential Mitigations

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

**Phase: Implementation**

Always check to see if you have successfully accessed a resource or system functionality, and use proper error handling if it is unsuccessful. Do this even when you are operating in a highly privileged mode, because errors or environmental conditions might still cause a failure. For example, environments with highly granular permissions/privilege models, such as Windows or Linux capabilities, can cause unexpected failures.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2003-0501** | Special file system allows attackers to prevent ownership/permission change of certain entries by opening the entries before calling a setuid program.<br>*https://www.cve.org/CVERecord?id=CVE-2003-0501* |

| Reference | Description |
| --- | --- |
| **CVE-2004-0148** | FTP server places a user in the root directory when the user's permissions prevent access to the their own home directory. *https://www.cve.org/CVERecord?id=CVE-2004-0148* |

### 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 | 962 | SFP Secondary Cluster: Unchecked Status Condition | 888 | 2421 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Notes

#### Maintenance

CWE-280 and CWE-274 are too similar. It is likely that CWE-274 will be deprecated in the future.

#### Relationship

This can be both primary and resultant. When primary, it can expose a variety of weaknesses because a resource might not have the expected state, and subsequent operations might fail. It is often resultant from Unchecked Error Condition (CWE-391).

#### Theoretical

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

#### Research Gap

This type of issue is under-studied, since researchers often concentrate on whether an object has too many permissions, instead of not enough. These weaknesses are likely to appear in environments with fine-grained models for permissions and privileges, which can include operating systems and other large-scale software packages. However, even highly simplistic permission/privilege models are likely to contain these issues if the developer has not considered the possibility of access failure.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
| --- | --- | --- | --- |
| PLOVER | | | Fails poorly due to insufficient permissions |
| WASC | 17 | | Improper Filesystem Permissions |
| Software Fault Patterns | SFP4 | | Unchecked Status Condition |

## CWE-281: Improper Preservation of Permissions

**Weakness ID :** 281
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not preserve permissions or incorrectly preserves permissions when copying, restoring, or sharing objects, which can cause them to have less restrictive permissions 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 | C | 732 | Incorrect Permission Assignment for Critical Resource | 1559 |

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

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 275 | Permission Issues | 2339 |

## Weakness Ordinalities

**Resultant : This is resultant from errors that prevent the permissions from being preserved.**

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity | Read Application Data<br>Modify Application Data | |

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-2323** | Incorrect ACLs used when restoring backups from directories that use symbolic links.<br>*https://www.cve.org/CVERecord?id=CVE-2002-2323* |
| **CVE-2001-1515** | Automatic modification of permissions inherited from another file system.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1515* |
| **CVE-2005-1920** | Permissions on backup file are created with defaults, possibly less secure than original file.<br>*https://www.cve.org/CVERecord?id=CVE-2005-1920* |
| **CVE-2001-0195** | File is made world-readable when being cloned.<br>*https://www.cve.org/CVERecord?id=CVE-2001-0195* |

## 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 | 946 | SFP Secondary Cluster: Insecure Resource Permissions | 888 | 2415 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Permission preservation failure |

## CWE-282: Improper Ownership Management

**Weakness ID :** 282
**Structure :** Simple
**Abstraction :** Class

### Description

The product assigns the wrong ownership, or does not properly verify the ownership, of an object or 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 | P | 284 | Improper Access Control | 687 |
| ParentOf | B | 283 | Unverified Ownership | 685 |
| ParentOf | B | 708 | Incorrect Ownership Assignment | 1556 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity | |

### Detection Methods

**Automated Static Analysis**

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

*Effectiveness = High*

### Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

## Demonstrative Examples

**Example 1:**

This function is part of a privileged program that takes input from users with potentially lower privileges.

*Example Language: Python* *(Bad)*

```
def killProcess(processID):
  os.kill(processID, signal.SIGKILL)
```

This code does not confirm that the process to be killed is owned by the requesting user, thus allowing an attacker to kill arbitrary processes.

This function remedies the problem by checking the owner of the process before killing it:

*Example Language: Python* *(Good)*

```
def killProcess(processID):
  user = getCurrentUser()
  #Check process owner against requesting user
  if getProcessOwner(processID) == user:
    os.kill(processID, signal.SIGKILL)
    return
  else:
    print("You cannot kill a process you don't own")
    return
```

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-1999-1125** | Program runs setuid root but relies on a configuration file owned by a non-root user.<br>*https://www.cve.org/CVERecord?id=CVE-1999-1125* |

## Affected Resources

- 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 | 944 | SFP Secondary Cluster: Access Management | 888 | 2414 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

**Maintenance**

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Ownership errors |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 17 | Using Malicious Files |
| 35 | Leverage Executable Code in Non-Executable Files |

## CWE-283: Unverified Ownership

**Weakness ID :** 283
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not properly verify that a critical resource is owned by the proper entity.

### Relationships

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 | 282 | Improper Ownership Management | 683 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 840 | Business Logic Errors | 2381 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*An attacker could gain unauthorized access to system resources.* | |

### Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

## Demonstrative Examples

### Example 1:

This function is part of a privileged program that takes input from users with potentially lower privileges.

*Example Language: Python* *(Bad)*

```
def killProcess(processID):
  os.kill(processID, signal.SIGKILL)
```

This code does not confirm that the process to be killed is owned by the requesting user, thus allowing an attacker to kill arbitrary processes.

This function remedies the problem by checking the owner of the process before killing it:

*Example Language: Python* *(Good)*

```
def killProcess(processID):
  user = getCurrentUser()
  #Check process owner against requesting user
  if getProcessOwner(processID) == user:
    os.kill(processID, signal.SIGKILL)
    return
  else:
    print("You cannot kill a process you don't own")
    return
```

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-0178** | Program does not verify the owner of a UNIX socket that is used for sending a password. *https://www.cve.org/CVERecord?id=CVE-2001-0178* |
| **CVE-2004-2012** | Owner of special device not checked, allowing root. *https://www.cve.org/CVERecord?id=CVE-2004-2012* |

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

## Notes

### Relationship

This overlaps insufficient comparison, verification errors, permissions, and privileges.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Unverified Ownership |

## CWE-284: Improper Access Control

**Weakness ID :** 284
**Structure :** Simple
**Abstraction :** Pillar

### Description

The product does not restrict or incorrectly restricts access to a resource from an unauthorized actor.

### Extended Description

Access control involves the use of several protection mechanisms such as:

- Authentication (proving the identity of an actor)
- Authorization (ensuring that a given actor can access a resource), and
- Accountability (tracking of activities that were performed)

When any mechanism is not applied or otherwise fails, attackers can compromise the security of the product by gaining privileges, reading sensitive information, executing commands, evading detection, etc.

There are two distinct behaviors that can introduce access control weaknesses:

- Specification: incorrect privileges, permissions, ownership, etc. are explicitly specified for either the user or the resource (for example, setting a password file to be world-writable, or giving administrator capabilities to a guest user). This action could be performed by the program or the administrator.
- Enforcement: the mechanism contains errors that prevent it from properly enforcing the specified access control requirements (e.g., allowing the user to specify their own privileges, or allowing a syntactically-incorrect ACL to produce insecure settings). This problem occurs within the program itself, in that it does not actually enforce the intended security policy that the administrator specifies.

### Relationships

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 |
|---|---|---|---|---|
| MemberOf | V | 1000 | Research Concepts | 2596 |
| ParentOf | C | 269 | Improper Privilege Management | 653 |
| ParentOf | C | 282 | Improper Ownership Management | 683 |
| ParentOf | C | 285 | Improper Authorization | 691 |
| ParentOf | C | 286 | Incorrect User Management | 698 |
| ParentOf | C | 287 | Improper Authentication | 699 |
| ParentOf | C | 346 | Origin Validation Error | 860 |
| ParentOf | B | 749 | Exposed Dangerous Method or Function | 1572 |
| ParentOf | C | 923 | Improper Restriction of Communication Channel to Intended Endpoints | 1836 |
| ParentOf | B | 1191 | On-Chip Debug and Test Interface With Improper Access Control | 1989 |
| ParentOf | B | 1220 | Insufficient Granularity of Access Control | 2002 |
| ParentOf | B | 1224 | Improper Restriction of Write-Once Bit Fields | 2014 |

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 1231 | Improper Prevention of Lock Bit Modification | 2018 |
| ParentOf | B | 1233 | Security-Sensitive Hardware Controls with Missing Lock Bit Protection | 2023 |
| ParentOf | B | 1242 | Inclusion of Undocumented Features or Chicken Bits | 2044 |
| ParentOf | B | 1252 | CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations | 2068 |
| ParentOf | B | 1257 | Improper Access Control Applied to Mirrored or Aliased Memory Regions | 2079 |
| ParentOf | B | 1259 | Improper Restriction of Security Token Assignment | 2085 |
| ParentOf | B | 1260 | Improper Handling of Overlap Between Protected Memory Ranges | 2087 |
| ParentOf | B | 1262 | Improper Access Control for Register Interface | 2093 |
| ParentOf | C | 1263 | Improper Physical Access Control | 2097 |
| ParentOf | B | 1267 | Policy Uses Obsolete Encoding | 2105 |
| ParentOf | B | 1268 | Policy Privileges are not Assigned Consistently Between Control and Data Agents | 2107 |
| ParentOf | B | 1270 | Generation of Incorrect Security Tokens | 2113 |
| ParentOf | B | 1274 | Improper Access Control for Volatile Memory Containing Boot Code | 2121 |
| ParentOf | B | 1276 | Hardware Child Block Incorrectly Connected to Parent System | 2125 |
| ParentOf | B | 1280 | Access Control Check Implemented After Asset is Accessed | 2134 |
| ParentOf | B | 1283 | Mutable Attestation or Measurement Reporting Data | 2140 |
| ParentOf | B | 1290 | Incorrect Decoding of Security Identifiers | 2155 |
| ParentOf | B | 1292 | Incorrect Conversion of Security Identifiers | 2159 |
| ParentOf | C | 1294 | Insecure Security Identifier Mechanism | 2162 |
| ParentOf | B | 1296 | Incorrect Chaining or Granularity of Debug Components | 2166 |
| ParentOf | B | 1304 | Improperly Preserved Integrity of Hardware Configuration State During a Power Save/Restore Operation | 2188 |
| ParentOf | B | 1311 | Improper Translation of Security Attributes by Fabric Bridge | 2194 |
| ParentOf | B | 1312 | Missing Protection for Mirrored Regions in On-Chip Fabric Firewall | 2196 |
| ParentOf | B | 1313 | Hardware Allows Activation of Test or Debug Logic at Runtime | 2198 |
| ParentOf | B | 1315 | Improper Setting of Bus Controlling Capability in Fabric End-point | 2202 |
| ParentOf | B | 1316 | Fabric-Address Map Allows Programming of Unwarranted Overlaps of Protected and Unprotected Ranges | 2204 |
| ParentOf | B | 1317 | Improper Access Control in Fabric Bridge | 2206 |
| ParentOf | B | 1320 | Improper Protection for Outbound Error Messages and Alert Signals | 2214 |
| ParentOf | B | 1323 | Improper Management of Sensitive Trace Data | 2220 |
| ParentOf | B | 1334 | Unauthorized Error Injection Can Degrade Hardware Redundancy | 2246 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | C | 285 | Improper Authorization | 691 |

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | C | 287 | Improper Authentication | 699 |
| ParentOf | B | 288 | Authentication Bypass Using an Alternate Path or Channel | 707 |
| ParentOf | B | 639 | Authorization Bypass Through User-Controlled Key | 1415 |
| ParentOf | C | 862 | Missing Authorization | 1789 |
| ParentOf | C | 863 | Incorrect Authorization | 1796 |

## Applicable Platforms

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

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

## Alternate Terms

**Authorization** : The terms "access control" and "authorization" are often used interchangeably, although many people have distinct definitions. The CWE usage of "access control" is intended as a general term for the various mechanisms that restrict which users can access which resources, and "authorization" is more narrowly defined. It is unlikely that there will be community consensus on the use of these terms.

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-24985** | A form hosting website only checks the session authentication status for a single form, making it possible to bypass authentication when there are multiple forms *https://www.cve.org/CVERecord?id=CVE-2022-24985* |
| **CVE-2022-29238** | Access-control setting in web-based document collaboration tool is not properly implemented by the code, which prevents listing hidden directories but does not prevent direct requests to files in those directories. *https://www.cve.org/CVERecord?id=CVE-2022-29238* |
| **CVE-2022-23607** | Python-based HTTP library did not scope cookies to a particular domain such that "supercookies" could be sent to any domain on redirect *https://www.cve.org/CVERecord?id=CVE-2022-23607* |
| **CVE-2021-21972** | Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses .. path traversal sequences |

| Reference | Description |
|---|---|
| | (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-21972* |
| **CVE-2021-37415** | IT management product does not perform authentication for some REST API requests, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-37415* |
| **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-2020-10263** | Bluetooth speaker does not require authentication for the debug functionality on the UART port, allowing root shell access *https://www.cve.org/CVERecord?id=CVE-2020-10263* |
| **CVE-2020-13927** | Default setting in workflow management product allows all API requests without authentication, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-13927* |
| **CVE-2010-4624** | Bulletin board applies restrictions on number of images during post creation, but does not enforce this on editing. *https://www.cve.org/CVERecord?id=CVE-2010-4624* |



## Affected Resources

- 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 | 254 | 7PK - Security Features | 700 | 2335 |
| MemberOf | C | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control | 711 | 2356 |
| MemberOf | C | 944 | SFP Secondary Cluster: Access Management | 888 | 2414 |
| MemberOf | C | 1031 | OWASP Top Ten 2017 Category A5 - Broken Access Control | 1026 | 2458 |
| MemberOf | V | 1340 | CISQ Data Protection Measures | 1340 | 2611 |
| MemberOf | C | 1345 | OWASP Top Ten 2021 Category A01:2021 - Broken Access Control | 1344 | 2508 |
| MemberOf | C | 1369 | ICS Supply Chain: IT/OT Convergence/Expansion | 1358 | 2527 |
| MemberOf | C | 1372 | ICS Supply Chain: OT Counterfeit and Malicious Corruption | 1358 | 2530 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

This entry needs more work. Possible sub-categories include: Trusted group includes undesired entities (partially covered by CWE-286) Group can perform undesired actions ACL parse error does not fail closed

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Access Control List (ACL) errors |
| WASC | 2 | | Insufficient Authorization |
| 7 Pernicious Kingdoms | | | Missing Access Control |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 19 | Embedding Scripts within Scripts |
| 441 | Malicious Logic Insertion |
| 478 | Modification of Windows Service Configuration |
| 479 | Malicious Root Certificate |
| 502 | Intent Spoof |
| 503 | WebView Exposure |
| 536 | Data Injected During Configuration |
| 546 | Incomplete Data Deletion in a Multi-Tenant Environment |
| 550 | Install New Service |
| 551 | Modify Existing Service |
| 552 | Install Rootkit |
| 556 | Replace File Extension Handlers |
| 558 | Replace Trusted Executable |
| 562 | Modify Shared File |
| 563 | Add Malicious File to Shared Webroot |
| 564 | Run Software at Logon |
| 578 | Disable Security Software |

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

[REF-1287]MITRE. "Supplemental Details - 2022 CWE Top 25". 2022 June 8. < https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25_supplemental.html#problematicMappingDetails >.2024-11-17.

## CWE-285: Improper Authorization

**Weakness ID :** 285
**Structure :** Simple
**Abstraction :** Class

### Description

The product does not perform or incorrectly performs 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 consistently - or not at all - 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 | P | 284 | Improper Access Control | 687 |
| ParentOf | B | 552 | Files or Directories Accessible to External Parties | 1274 |
| ParentOf | C | 732 | Incorrect Permission Assignment for Critical Resource | 1559 |
| ParentOf | C | 862 | Missing Authorization | 1789 |
| ParentOf | C | 863 | Incorrect Authorization | 1796 |
| ParentOf | V | 926 | Improper Export of Android Application Components | 1843 |
| ParentOf | V | 927 | Use of Implicit Intent for Sensitive Communication | 1846 |
| ParentOf | B | 1230 | Exposure of Sensitive Information Through Metadata | 2017 |
| ParentOf | B | 1256 | Improper Restriction of Software Interfaces to Hardware Features | 2076 |
| ParentOf | B | 1297 | Unprotected Confidential Information on Device is Accessible by OSAT Vendors | 2168 |
| ParentOf | B | 1328 | Security Version Number Mutable to Older Versions | 2229 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |

## Applicable Platforms

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

**Technology** : Web Server *(Prevalence = Often)*

**Technology** : Database Server *(Prevalence = Often)*

## Background Details

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

## Alternate Terms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data | |

| Scope | Impact | Likelihood |
|---|---|---|
| | Read Files or Directories<br><br>*An attacker could read sensitive data, either by reading the data directly from a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.* | |
| Integrity | Modify Application Data<br>Modify Files or Directories<br><br>*An attacker could modify sensitive data, either by writing the data directly to a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.* | |
| Access Control | Gain Privileges or Assume Identity<br><br>*An attacker could gain privileges by modifying or reading critical data directly, or by accessing insufficiently-protected, 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 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) 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 you perform access control checks related to your business logic. These checks may be different than the access control checks that you apply 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.

**Phase: Architecture and Design**

*Strategy = Libraries or Frameworks*

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45].

**Phase: Architecture and Design**

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

**Phase: System Configuration**

**Phase: Installation**

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

## Demonstrative Examples

**Example 1:**

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

*Example Language: PHP* *(Bad)*

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

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

**Example 2:**

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

*Example Language: Perl* *(Bad)*

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

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

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

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-24730** | Go-based continuous deployment product does not check that a user has certain privileges to update or create an app, allowing adversaries to read sensitive repository information *https://www.cve.org/CVERecord?id=CVE-2022-24730* |
| **CVE-2009-3168** | Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords. *https://www.cve.org/CVERecord?id=CVE-2009-3168* |
| **CVE-2009-2960** | 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-2009-3597** | 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-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-2008-5027** | System monitoring software allows users to bypass authorization by creating custom forms. *https://www.cve.org/CVERecord?id=CVE-2008-5027* |
| **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-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-2008-4577** | 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-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-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* |

| Reference | Description |
|---|---|
| **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 | 254 | 7PK - Security Features | 700 | 2335 |
| MemberOf | C | 721 | OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access | 629 | 2355 |
| MemberOf | C | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control | 711 | 2356 |
| MemberOf | C | 753 | 2009 Top 25 - Porous Defenses | 750 | 2374 |
| MemberOf | C | 803 | 2010 Top 25 - Porous Defenses | 800 | 2376 |
| MemberOf | C | 817 | OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access | 809 | 2380 |
| MemberOf | C | 935 | OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control | 928 | 2413 |
| MemberOf | C | 945 | SFP Secondary Cluster: Insecure Resource Access | 888 | 2415 |
| MemberOf | C | 1031 | OWASP Top Ten 2017 Category A5 - Broken Access Control | 1026 | 2458 |
| MemberOf | C | 1345 | OWASP Top Ten 2021 Category A01:2021 - Broken Access Control | 1344 | 2508 |
| MemberOf | C | 1382 | ICS Operations (& Maintenance): Emerging Energy Technologies | 1358 | 2538 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| 7 Pernicious Kingdoms | | | Missing Access Control |
| OWASP Top Ten 2007 | A10 | CWE More Specific | Failure to Restrict URL Access |
| OWASP Top Ten 2004 | A2 | CWE More Specific | Broken Access Control |
| Software Fault Patterns | SFP35 | | Insecure resource access |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 1 | Accessing Functionality Not Properly Constrained by ACLs |
| 5 | Blue Boxing |
| 13 | Subverting Environment Variable Values |
| 17 | Using Malicious Files |
| 39 | Manipulating Opaque Client-based Data Tokens |
| 45 | Buffer Overflow via Symbolic Links |
| 51 | Poison Web Service Registry |
| 59 | Session Credential Falsification through Prediction |
| 60 | Reusing Session IDs (aka Session Replay) |
| 76 | Manipulating Web Input to File System Calls |

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 77 | Manipulating User-Controlled Variables |
| 87 | Forceful Browsing |
| 104 | Cross Zone Scripting |
| 127 | Directory Indexing |
| 402 | Bypassing ATA Password Security |
| 647 | Collect Data from Registries |
| 668 | Key Negotiation of Bluetooth Attack (KNOB) |

### References

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

[REF-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.

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

## CWE-286: Incorrect User Management

**Weakness ID :** 286
**Structure :** Simple
**Abstraction :** Class

### Description

The product does not properly manage a user within its environment.

### Extended Description

Users can be assigned to the wrong group (class) of permissions resulting in unintended access rights to sensitive objects.

### Relationships

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |
| ParentOf | B | 842 | Placement of User into Incorrect Group | 1784 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-36109** | Containerization product does not record a user's supplementary group ID, allowing bypass of group restrictions. *https://www.cve.org/CVERecord?id=CVE-2022-36109* |
| **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* |

### 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 | 944 | SFP Secondary Cluster: Access Management | 888 | 2414 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Notes

**Maintenance**

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

**Maintenance**

This item needs more work. Possible sub-categories include: user in wrong group, and user with insecure profile or "configuration". It also might be better expressed as a category than a weakness.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | User management errors |

## CWE-287: Improper Authentication

**Weakness ID :** 287
**Structure :** Simple
**Abstraction :** Class

## Description

When an actor claims to have a given identity, the product does not prove or insufficiently proves that the claim is correct.

## Relationships

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |
| ParentOf | B | 295 | Improper Certificate Validation | 721 |
| ParentOf | B | 306 | Missing Authentication for Critical Function | 748 |
| ParentOf | B | 645 | Overly Restrictive Account Lockout Mechanism | 1432 |
| ParentOf | C | 1390 | Weak Authentication | 2279 |
| CanFollow | B | 613 | Insufficient Session Expiration | 1380 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 290 | Authentication Bypass by Spoofing | 712 |
| ParentOf | B | 294 | Authentication Bypass by Capture-replay | 719 |
| ParentOf | B | 295 | Improper Certificate Validation | 721 |
| ParentOf | B | 306 | Missing Authentication for Critical Function | 748 |
| ParentOf | B | 307 | Improper Restriction of Excessive Authentication Attempts | 754 |
| ParentOf | B | 521 | Weak Password Requirements | 1231 |
| ParentOf | C | 522 | Insufficiently Protected Credentials | 1234 |
| ParentOf | B | 640 | Weak Password Recovery Mechanism for Forgotten Password | 1418 |
| ParentOf | B | 798 | Use of Hard-coded Credentials | 1699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |

## Applicable Platforms

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

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

## Alternate Terms

**authentification** : An alternate term is "authentification", which appears to be most commonly used by people from non-English-speaking countries.

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

**AuthC** : "AuthC" is used as an abbreviation of "authentication," but it appears to used less frequently than "AuthN."

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Confidentiality<br>Availability<br>Access Control | Read Application Data<br>Gain Privileges or Assume Identity<br>Execute Unauthorized Code or Commands<br>*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

### Automated Static Analysis

Automated static analysis is useful for detecting certain types of authentication. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authentication libraries. Generally, automated static analysis tools have difficulty detecting custom authentication schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an established identity; an automated technique that detects the absence of authentication may report false positives.

*Effectiveness = Limited*

### Manual Static Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Manual static analysis is useful for evaluating the correctness of custom authentication mechanisms.

*Effectiveness = High*

*These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.*

### 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

*Effectiveness = SOAR Partial*

### Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Manual Source Code Review (not inspections)

*Effectiveness = SOAR Partial*

**Automated Static Analysis - Source Code**

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

*Effectiveness = SOAR Partial*

**Automated Static Analysis**

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

*Effectiveness = SOAR Partial*

**Architecture or Design Review**

According to SOAR, the following detection techniques may be useful: Highly cost effective: 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**

*Strategy = Libraries or Frameworks*

Use an authentication framework or library such as the OWASP ESAPI Authentication feature.

## Demonstrative Examples

**Example 1:**

The following code intends to ensure that the user is already logged in. If not, the code performs authentication with the user-provided username and password. If successful, it sets the loggedin and user cookies to "remember" that the user has already logged in. Finally, the code performs administrator tasks if the logged-in user has the "Administrator" username, as recorded in the user cookie.

*Example Language: Perl* *(Bad)*

```
my $q = new CGI;
if ($q->cookie('loggedin') ne "true") {
  if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
    ExitError("Error: you need to log in first");
  }
  else {
    # Set loggedin and user cookies.
    $q->cookie(
      -name => 'loggedin',
      -value => 'true'
      );
    $q->cookie(
      -name => 'user',
      -value => $q->param('username')
      );
  }
}
if ($q->cookie('user') eq "Administrator") {
  DoAdministratorTasks();
}
```

Unfortunately, this code can be bypassed. The attacker can set the cookies independently so that the code does not check the username and password. The attacker could do this with an HTTP request containing headers such as:

*Example Language:* *(Attack)*

```
GET /cgi-bin/vulnerable.cgi HTTP/1.1
Cookie: user=Administrator
Cookie: loggedin=true
[body of request]
```

By setting the loggedin cookie to "true", the attacker bypasses the entire authentication check. By using the "Administrator" value in the user cookie, the attacker also gains privileges to administer the software.

**Example 2:**

In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts [REF-236]. The attacker targeted a member of Twitter's support team and was able to successfully guess the member's password using a brute force attack by guessing a large number of common words. After gaining access as the member of the support staff, the attacker used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts.

**Example 3:**

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 did not use any authentication or used client-side authentication for critical functionality in their OT products.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-35248** | Chat application skips validation when Central Authentication Service (CAS) is enabled, effectively removing the second factor from two-factor authentication *https://www.cve.org/CVERecord?id=CVE-2022-35248* |
| **CVE-2022-36436** | Python-based authentication proxy does not enforce password authentication during the initial handshake, allowing the client to bypass authentication by specifying a 'None' authentication type. *https://www.cve.org/CVERecord?id=CVE-2022-36436* |
| **CVE-2022-30034** | Chain: Web UI for a Python RPC framework does not use regex anchors to validate user login emails (CWE-777), potentially allowing bypass of OAuth (CWE-1390). *https://www.cve.org/CVERecord?id=CVE-2022-30034* |
| **CVE-2022-29951** | TCP-based protocol in Programmable Logic Controller (PLC) has no authentication. *https://www.cve.org/CVERecord?id=CVE-2022-29951* |
| **CVE-2022-29952** | Condition Monitor uses a protocol that does not require authentication. *https://www.cve.org/CVERecord?id=CVE-2022-29952* |
| **CVE-2022-30313** | Safety Instrumented System uses proprietary TCP protocols with no authentication. *https://www.cve.org/CVERecord?id=CVE-2022-30313* |
| **CVE-2022-30317** | Distributed Control System (DCS) uses a protocol that has no authentication. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2022-30317* |
| **CVE-2022-33139** | SCADA system only uses client-side authentication, allowing adversaries to impersonate other users. *https://www.cve.org/CVERecord?id=CVE-2022-33139* |
| **CVE-2021-3116** | Chain: Python-based HTTP Proxy server uses the wrong boolean operators (CWE-480) causing an incorrect comparison (CWE-697) that identifies an authN failure if all three conditions are met instead of only one, allowing bypass of the proxy authentication (CWE-1390) *https://www.cve.org/CVERecord?id=CVE-2021-3116* |
| **CVE-2021-21972** | Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses .. path traversal sequences (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-21972* |
| **CVE-2021-37415** | IT management product does not perform authentication for some REST API requests, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-37415* |
| **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-2020-10263** | Bluetooth speaker does not require authentication for the debug functionality on the UART port, allowing root shell access *https://www.cve.org/CVERecord?id=CVE-2020-10263* |
| **CVE-2020-13927** | Default setting in workflow management product allows all API requests without authentication, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-13927* |
| **CVE-2021-35395** | Stack-based buffer overflows in SFK for wifi chipset used for IoT/embedded devices, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-35395* |
| **CVE-2021-34523** | Mail server does not properly check an access token before executing a Powershell command, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-34523* |
| **CVE-2020-12812** | Chain: user is not prompted for a second authentication factor (CWE-287) when changing the case of their username (CWE-178), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-12812* |
| **CVE-2020-10148** | Authentication bypass by appending specific parameters and values to a URI, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-10148* |
| **CVE-2020-0688** | Mail server does not generate a unique key during installation, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-0688* |
| **CVE-2017-14623** | LDAP Go package allows authentication bypass using an empty password, causing an unauthenticated LDAP bind *https://www.cve.org/CVERecord?id=CVE-2017-14623* |
| **CVE-2009-3421** | login script for guestbook allows bypassing authentication by setting a "login_ok" parameter to 1. *https://www.cve.org/CVERecord?id=CVE-2009-3421* |
| **CVE-2009-2382** | admin script allows authentication bypass by setting a cookie value to "LOGGEDIN". *https://www.cve.org/CVERecord?id=CVE-2009-2382* |
| **CVE-2009-1048** | VOIP product allows authentication bypass using 127.0.0.1 in the Host header. *https://www.cve.org/CVERecord?id=CVE-2009-1048* |

| Reference | Description |
|---|---|
| **CVE-2009-2213** | product uses default "Allow" action, instead of default deny, leading to authentication bypass. *https://www.cve.org/CVERecord?id=CVE-2009-2213* |
| **CVE-2009-2168** | chain: redirect without exit (CWE-698) leads to resultant authentication bypass. *https://www.cve.org/CVERecord?id=CVE-2009-2168* |
| **CVE-2009-3107** | product does not restrict access to a listening port for a critical service, allowing authentication to be bypassed. *https://www.cve.org/CVERecord?id=CVE-2009-3107* |
| **CVE-2009-1596** | product does not properly implement a security-related configuration setting, allowing authentication bypass. *https://www.cve.org/CVERecord?id=CVE-2009-1596* |
| **CVE-2009-2422** | authentication routine returns "nil" instead of "false" in some situations, allowing authentication bypass using an invalid username. *https://www.cve.org/CVERecord?id=CVE-2009-2422* |
| **CVE-2009-3232** | authentication update script does not properly handle when admin does not select any authentication modules, allowing authentication bypass. *https://www.cve.org/CVERecord?id=CVE-2009-3232* |
| **CVE-2009-3231** | use of LDAP authentication with anonymous binds causes empty password to result in successful authentication *https://www.cve.org/CVERecord?id=CVE-2009-3231* |
| **CVE-2005-3435** | product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks. *https://www.cve.org/CVERecord?id=CVE-2005-3435* |
| **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* |

## Functional Areas

- Authentication

## 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 | 635 | Weaknesses Originally Used by NVD from 2008 to 2016 | 635 | 2573 |
| MemberOf | C | 718 | OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management | 629 | 2353 |
| MemberOf | C | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 812 | OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management | 809 | 2378 |
| MemberOf | C | 930 | OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management | 928 | 2410 |
| MemberOf | C | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | C | 1028 | OWASP Top Ten 2017 Category A2 - Broken Authentication | 1026 | 2457 |
| MemberOf | V | 1200 | Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors | 1200 | 2608 |
| MemberOf | V | 1337 | Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses | 1337 | 2610 |

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 1350 | Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses | 1350 | 2615 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1364 | ICS Communications: Zone Boundary Failures | 1358 | 2522 |
| MemberOf | C | 1368 | ICS Dependencies (& Architecture): External Digital Systems | 1358 | 2526 |
| MemberOf | V | 1387 | Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses | 1387 | 2618 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |
| MemberOf | V | 1425 | Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses | 1425 | 2621 |
| MemberOf | V | 1430 | Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses | 1430 | 2622 |

## Notes

### Relationship

This can be resultant from SQL injection vulnerabilities and other issues.

### 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 |
|---|---|---|---|
| PLOVER | | | Authentication Error |
| OWASP Top Ten 2007 | A7 | CWE More Specific | Broken Authentication and Session Management |
| OWASP Top Ten 2004 | A3 | CWE More Specific | Broken Authentication and Session Management |
| WASC | 1 | | Insufficient Authentication |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.1 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.2 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.2 |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 22 | Exploiting Trust in Client |
| 57 | Utilizing REST's Trust in the System Resource to Obtain Sensitive Data |
| 94 | Adversary in the Middle (AiTM) |
| 114 | Authentication Abuse |
| 115 | Authentication Bypass |
| 151 | Identity Spoofing |
| 194 | Fake the Source of Data |
| 593 | Session Hijacking |
| 633 | Token Impersonation |
| 650 | Upload a Web Shell to a Web Server |

### References

[REF-236]Kim Zetter. "Weak Password Brings 'Happiness' to Twitter Hacker". 2009 January 9. < https://www.wired.com/2009/01/professed-twitt/ >.2023-04-07.

[REF-237]OWASP. "Top 10 2007-Broken Authentication and Session Management". 2007. < http://www.owasp.org/index.php/Top_10_2007-A7 >.

[REF-238]OWASP. "Guide to Authentication". < http://www.owasp.org/index.php/Guide_to_Authentication >.

[REF-239]Microsoft. "Authentication". < http://msdn.microsoft.com/en-us/library/aa374735(VS.85).aspx >.

[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-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/ >.

## CWE-288: Authentication Bypass Using an Alternate Path or Channel

**Weakness ID :** 288
**Structure :** Simple
**Abstraction :** Base

### Description

The product requires authentication, but the product has an alternate path or channel that does not require authentication.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 306 | Missing Authentication for Critical Function | 748 |
| ParentOf | B | 425 | Direct Request ('Forced Browsing') | 1032 |
| ParentOf | B | 1299 | Missing Protection Mechanism for Alternate Hardware Interface | 2174 |
| PeerOf | B | 420 | Unprotected Alternate Channel | 1025 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |

## Potential Mitigations

### Phase: Architecture and Design

Funnel all access through a single choke point to simplify how users can access a resource. For every access, perform a check to determine if the user has permissions to access the resource.

## Demonstrative Examples

### Example 1:

Register SECURE_ME is located at address 0xF00. A mirror of this register called COPY_OF_SECURE_ME is at location 0x800F00. The register SECURE_ME is protected from malicious agents and only allows access to select, while COPY_OF_SECURE_ME is not.

Access control is implemented using an allowlist (as indicated by acl_oh_allowlist). The identity of the initiator of the transaction is indicated by the one hot input, incoming_id. This is checked against the acl_oh_allowlist (which contains a list of initiators that are allowed to access the asset).

Though this example is shown in Verilog, it will apply to VHDL as well.

*Example Language: Verilog* *(Informative)*

```
module foo_bar(data_out, data_in, incoming_id, address, clk, rst_n);
output [31:0] data_out;
input [31:0] data_in, incoming_id, address;
input clk, rst_n;
wire write_auth, addr_auth;
reg [31:0] data_out, acl_oh_allowlist, q;
assign write_auth = | (incoming_id & acl_oh_allowlist) ? 1 : 0;
always @*
  acl_oh_allowlist <= 32'h8312;
assign addr_auth = (address == 32'hF00) ? 1: 0;
always @ (posedge clk or negedge rst_n)
  if (!rst_n)
    begin
      q <= 32'h0;
      data_out <= 32'h0;
    end
  else
    begin
      q <= (addr_auth & write_auth) ? data_in: q;
      data_out <= q;
    end
  end
endmodule
```

*Example Language: Verilog* *(Bad)*

```
assign addr_auth = (address == 32'hF00) ? 1: 0;
```

The bugged line of code is repeated in the Bad example above. Weakness arises from the fact that the SECURE_ME register can be modified by writing to the shadow register COPY_OF_SECURE_ME, the address of COPY_OF_SECURE_ME should also be included in the check. That buggy line of code should instead be replaced as shown in the Good Code Snippet below.

*Example Language: Verilog* *(Good)*

```
assign addr_auth = (address == 32'hF00 || address == 32'h800F00) ? 1: 0;
```

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2000-1179** | Router allows remote attackers to read system logs without authentication by directly connecting to the login screen and typing certain control characters. *https://www.cve.org/CVERecord?id=CVE-2000-1179* |
| **CVE-1999-1454** | Attackers with physical access to the machine may bypass the password prompt by pressing the ESC (Escape) key. *https://www.cve.org/CVERecord?id=CVE-1999-1454* |
| **CVE-1999-1077** | OS allows local attackers to bypass the password protection of idled sessions via the programmer's switch or CMD-PWR keyboard sequence, which brings up a debugger that the attacker can use to disable the lock. *https://www.cve.org/CVERecord?id=CVE-1999-1077* |
| **CVE-2003-0304** | Direct request of installation file allows attacker to create administrator accounts. *https://www.cve.org/CVERecord?id=CVE-2003-0304* |
| **CVE-2002-0870** | Attackers may gain additional privileges by directly requesting the web management URL. *https://www.cve.org/CVERecord?id=CVE-2002-0870* |
| **CVE-2002-0066** | Bypass authentication via direct request to named pipe. *https://www.cve.org/CVERecord?id=CVE-2002-0066* |
| **CVE-2003-1035** | User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing. *https://www.cve.org/CVERecord?id=CVE-2003-1035* |

## 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 | 721 | OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access | 629 | 2355 |
| MemberOf | C | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1364 | ICS Communications: Zone Boundary Failures | 1358 | 2522 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Relationship

overlaps Unprotected Alternate Channel

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication Bypass by Alternate Path/Channel |
| OWASP Top Ten 2007 | A10 | CWE More Specific | Failure to Restrict URL Access |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 127 | Directory Indexing |
| 665 | Exploitation of Thunderbolt Protection Flaws |

## CWE-289: Authentication Bypass by Alternate Name

**Weakness ID :** 289
**Structure :** Simple
**Abstraction :** Base

### Description

The product performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor.

### Relationships

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 | 2279 |
| CanFollow | V | 46 | Path Equivalence: 'filename ' (Trailing Space) | 97 |
| CanFollow | V | 52 | Path Equivalence: '/multiple/trailing/slash//' | 104 |
| CanFollow | V | 173 | Improper Handling of Alternate Encoding | 441 |
| CanFollow | B | 178 | Improper Handling of Case Sensitivity | 451 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |

### Potential Mitigations

**Phase: Architecture and Design**

*Strategy = Input Validation*

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

**Phase: Implementation**

*Strategy = Input Validation*

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

**Phase: Implementation**

*Strategy = Input Validation*

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2003-0317** | Protection mechanism that restricts URL access can be bypassed using URL encoding. *https://www.cve.org/CVERecord?id=CVE-2003-0317* |
| **CVE-2004-0847** | Bypass of authentication for files using "\" (backslash) or "%5C" (encoded backslash). *https://www.cve.org/CVERecord?id=CVE-2004-0847* |

## 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 | 2383 |
| MemberOf | C | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1134 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS) | 1133 | 2465 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

**Relationship**

Overlaps equivalent encodings, canonicalization, authorization, multiple trailing slash, trailing space, mixed case, and other equivalence issues.

**Theoretical**

Alternate names are useful in data driven manipulation attacks, not just for authentication.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication bypass by alternate name |
| The CERT Oracle Secure Coding Standard for Java (2011) | IDS01-J | CWE More Specific | Normalize strings before validating them |
| SEI CERT Oracle Coding Standard for Java | IDS01-J | CWE More Specific | Normalize strings before validating them |

## CWE-290: Authentication Bypass by Spoofing

**Weakness ID :** 290
**Structure :** Simple
**Abstraction :** Base

### Description

This attack-focused weakness is caused by incorrectly implemented authentication schemes that are subject to spoofing attacks.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 2279 |
| ParentOf | V | 291 | Reliance on IP Address for Authentication | 715 |
| ParentOf | V | 293 | Using Referer Field for Authentication | 717 |
| ParentOf | V | 350 | Reliance on Reverse DNS Resolution for a Security-Critical Action | 870 |
| PeerOf | C | 602 | Client-Side Enforcement of Server-Side Security | 1359 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 287 | Improper Authentication | 699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>*This weakness can allow an attacker to access resources which are not otherwise accessible without proper authentication.* | |

### Demonstrative Examples

**Example 1:**

The following code authenticates users.

*Example Language: Java* *(Bad)*

```
String sourceIP = request.getRemoteAddr();
if (sourceIP != null && sourceIP.equals(APPROVED_IP)) {
  authenticated = true;
}
```

The authentication mechanism implemented relies on an IP address for source validation. If an attacker is able to spoof the IP, they may be able to bypass the authentication mechanism.

**Example 2:**

Both of these examples check if a request is from a trusted address before responding to the request.

*Example Language: C* *(Bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
    n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
  }
}
```

*Example Language: Java* *(Bad)*

```
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress clientIPAddress = rp.getAddress();
  int port = rp.getPort();
  if (isTrustedAddress(clientIPAddress) & secretKey.equals(in)) {
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
  }
}
```

The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client.

**Example 3:**

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-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-2009-1048** | VOIP product allows authentication bypass using 127.0.0.1 in the Host header. *https://www.cve.org/CVERecord?id=CVE-2009-1048* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 956 | SFP Secondary Cluster: Channel Attack | 888 | 2418 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Relationship

This can be resultant from insufficient verification.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication bypass by spoofing |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 21 | Exploitation of Trusted Identifiers |
| 22 | Exploiting Trust in Client |
| 59 | Session Credential Falsification through Prediction |
| 60 | Reusing Session IDs (aka Session Replay) |
| 94 | Adversary in the Middle (AiTM) |
| 459 | Creating a Rogue Certification Authority Certificate |
| 461 | Web Services API Signature Forgery Leveraging Hash Function Extension Weakness |
| 473 | Signature Spoof |

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 476 | Signature Spoofing by Misrepresentation |
| 667 | Bluetooth Impersonation AttackS (BIAS) |

### References

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

## CWE-291: Reliance on IP Address for Authentication

**Weakness ID :** 291
**Structure :** Simple
**Abstraction :** Variant

### Description

The product uses an IP address for authentication.

### Extended Description

IP addresses can be easily spoofed. Attackers can 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.

### Relationships

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 | 471 | Modification of Assumed-Immutable Data (MAID) | 1129 |
| ChildOf | C | 923 | Improper Restriction of Communication Channel to Intended Endpoints | 1836 |
| ChildOf | B | 290 | Authentication Bypass by Spoofing | 712 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

### Weakness Ordinalities

**Resultant :**

### Applicable Platforms

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

### Likelihood Of Exploit

High

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Non-Repudiation | Hide Activities<br>Gain Privileges or Assume Identity<br><br>*Malicious users can fake authentication information, impersonating any IP address.* | |

## Potential Mitigations

### Phase: Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

## Demonstrative Examples

### Example 1:

Both of these examples check if a request is from a trusted address before responding to the request.

*Example Language: C* *(Bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
    n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
  }
}
```

*Example Language: Java* *(Bad)*

```
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress clientIPAddress = rp.getAddress();
  int port = rp.getPort();
  if (isTrustedAddress(clientIPAddress) & secretKey.equals(in)) {
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
  }
}
```

The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client.

## 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.<br>*https://www.cve.org/CVERecord?id=CVE-2022-30319* |

## MemberOf Relationships

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

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

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Trusting self-reported IP address |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 4 | Using Alternative IP Address Encodings |

## References

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

[REF-1371]"IP address spoofing". 2006 April 7. Wikipedia. < https://en.wikipedia.org/wiki/IP_address_spoofing >.2023-10-21.

# CWE-293: Using Referer Field for Authentication

**Weakness ID :** 293
**Structure :** Simple
**Abstraction :** Variant

## Description

The referer field in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

## Relationships

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 | 290 | Authentication Bypass by Spoofing | 712 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

## Applicable Platforms

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

## Background Details

The referer field in HTML requests can be simply modified by malicious users, rendering it useless as a means of checking the validity of the request in question.

## Alternate Terms

**referrer** : While the proper spelling might be regarded as "referrer," the HTTP RFCs and their implementations use "referer," so this is regarded as the correct spelling.

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*Actions, which may not be authorized otherwise, can be carried out as if they were validated by the server referred to.* | |

## 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

In order to usefully check if a given action is authorized, some means of strong authentication and method protection must be used. Use other means of authorization that cannot be simply spoofed. Possibilities include a username/password or certificate.

## Demonstrative Examples

### Example 1:

The following code samples check a packet's referer in order to decide whether or not an inbound request is from a trusted host.

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

```
String trustedReferer = "http://www.example.com/"
while(true){
  n = read(newsock, buffer, BUFSIZE);
  requestPacket = processPacket(buffer, n);
  if (requestPacket.referer == trustedReferer){
    openNewSecureSession(requestPacket);
  }
}
```

*Example Language: Java* *(Bad)*

```
boolean processConnectionRequest(HttpServletRequest request){
  String referer = request.getHeader("referer")
  String trustedReferer = "http://www.example.com/"
  if(referer.equals(trustedReferer)){
    openPrivilegedConnection(request);
    return true;
  }
  else{
    sendPrivilegeError(request);
    return false;
  }
}
```

These examples check if a request is from a trusted referer before responding to a request, but the code only verifies the referer name as stored in the request packet. An attacker can spoof the referer, thus impersonating a trusted client.

### 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 | 949 | SFP Secondary Cluster: Faulty Endpoint Authentication | 888 | 2416 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Using referrer field for authentication |
| Software Fault Patterns | SFP29 | | Faulty endpoint authentication |

### References

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

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

## CWE-294: Authentication Bypass by Capture-replay

**Weakness ID :** 294
**Structure :** Simple
**Abstraction :** Base

### Description

A capture-replay flaw exists when the design of the product makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes).

### Extended Description

Capture-replay attacks are common and can be difficult to defeat without cryptography. They are a subset of network injection attacks that rely on observing previously-sent valid commands, then changing them slightly if necessary and resending the same commands to the server.

### Relationships

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 | 2279 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | | 287 | Improper Authentication | 699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

## Applicable Platforms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*Messages sent with a capture-relay attack allow access to resources which are not otherwise accessible without proper authentication.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Utilize some sequence or time stamping functionality along with a checksum which takes this into account in order to ensure that messages can be parsed only once.

**Phase: Architecture and Design**

Since any attacker who can listen to traffic can see sequence numbers, it is necessary to sign messages with some kind of cryptography to ensure that sequence numbers are not simply doctored along with content.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-3435** | product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.<br>*https://www.cve.org/CVERecord?id=CVE-2005-3435* |
| **CVE-2007-4961** | Chain: cleartext transmission of the MD5 hash of password (CWE-319) enables attacks against a server that is susceptible to replay (CWE-294).<br>*https://www.cve.org/CVERecord?id=CVE-2007-4961* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 956 | SFP Secondary Cluster: Channel Attack | 888 | 2418 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication bypass by replay |
| CLASP | | | Capture-replay |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 60 | Reusing Session IDs (aka Session Replay) |
| 94 | Adversary in the Middle (AiTM) |
| 102 | Session Sidejacking |
| 509 | Kerberoasting |
| 555 | Remote Services with Stolen Credentials |
| 561 | Windows Admin Shares with Stolen Credentials |
| 644 | Use of Captured Hashes (Pass The Hash) |
| 645 | Use of Captured Tickets (Pass The Ticket) |
| 652 | Use of Known Kerberos Credentials |
| 701 | Browser in the Middle (BiTM) |

### References

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

## CWE-295: Improper Certificate Validation

**Weakness ID :** 295
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not validate, or incorrectly validates, a certificate.

### Extended Description

When a certificate is invalid or malicious, it might allow an attacker to spoof a trusted entity by interfering in the communication path between the host and client. The product might connect to a malicious host while believing it is a trusted host, or the product might be deceived into accepting spoofed data that appears to originate from a trusted host.

### Relationships

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 | 287 | Improper Authentication | 699 |
| ParentOf | B | 296 | Improper Following of a Certificate's Chain of Trust | 726 |
| ParentOf | V | 297 | Improper Validation of Certificate with Host Mismatch | 729 |
| ParentOf | V | 298 | Improper Validation of Certificate Expiration | 733 |
| ParentOf | B | 299 | Improper Check for Certificate Revocation | 734 |
| ParentOf | V | 599 | Missing Validation of OpenSSL Certificate | 1350 |
| PeerOf | B | 322 | Key Exchange without Entity Authentication | 795 |
| PeerOf | B | 322 | Key Exchange without Entity Authentication | 795 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 287 | Improper Authentication | 699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

### Applicable Platforms

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

**Technology** : Mobile *(Prevalence = Undetermined)*

### Background Details

A certificate is a token that associates an identity (principal) to a cryptographic key. Certificates can be used to check if a public key belongs to the assumed owner.

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Authentication | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity | |

### 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*

**Dynamic Analysis with Automated Results Interpretation**

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

*Effectiveness = SOAR Partial*

**Dynamic Analysis with Manual Results Interpretation**

According to SOAR, the following detection techniques may be useful: Highly cost effective: Man-in-the-middle attack tool

*Effectiveness = High*

**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: 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.)

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Implementation**

Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key.

**Phase: Implementation**

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.

## Demonstrative Examples

**Example 1:**

This code checks the certificate of a connected peer.

*Example Language: C* *(Bad)*

```
if ((cert = SSL_get_peer_certificate(ssl)) && host)
  foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
  // certificate looks good, host can be trusted
```

In this case, because the certificate is self-signed, there was no external authority that could prove the identity of the host. The program could be communicating with a different system that is spoofing the host, e.g. by poisoning the DNS cache or using an Adversary-in-the-Middle (AITM) attack to modify the traffic from server to client.

**Example 2:**

The following OpenSSL code obtains a certificate and verifies it.

*Example Language: C* *(Bad)*

```
cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
  // do secret things
}
```

Even though the "verify" step returns X509_V_OK, this step does not include checking the Common Name against the name of the host. That is, there is no guarantee that the certificate is for the desired host. The SSL connection could have been established with a malicious host that provided a valid certificate.

**Example 3:**

The following OpenSSL code ensures that there is a certificate and allows the use of expired certificates.

*Example Language: C* *(Bad)*

```
if (cert = SSL_get_peer(certificate(ssl)) {
  foo=SSL_get_verify_result(ssl);
  if ((X509_V_OK==foo) || (X509_V_ERR_CERT_HAS_EXPIRED==foo))
    //do stuff
```

If the call to SSL_get_verify_result() returns X509_V_ERR_CERT_HAS_EXPIRED, this means that the certificate has expired. As time goes on, there is an increasing chance for attackers to compromise the certificate.

**Example 4:**

The following OpenSSL code ensures that there is a certificate before continuing execution.

*Example Language: C* *(Bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
  // got a certificate, do secret things
```

Because this code does not use SSL_get_verify_results() to check the certificate, it could accept certificates that have been revoked (X509_V_ERR_CERT_REVOKED). The software could be communicating with a malicious host.

**Example 5:**

The following OpenSSL code ensures that the host has a certificate.

*Example Language: C* *(Bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
  // got certificate, host can be trusted
  //foo=SSL_get_verify_result(ssl);
  //if (X509_V_OK==foo) ...
}
```

Note that the code does not call SSL_get_verify_result(ssl), which effectively disables the validation step that checks the certificate.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2019-12496** | A Go framework for robotics, drones, and IoT devices skips verification of root CA certificates by default. *https://www.cve.org/CVERecord?id=CVE-2019-12496* |
| **CVE-2014-1266** | chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing Adversary-in-the-Middle (AITM) attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://www.cve.org/CVERecord?id=CVE-2014-1266* |
| **CVE-2021-22909** | Chain: router's firmware update procedure uses curl with "-k" (insecure) option that disables certificate validation (CWE-295), allowing adversary-in-the-middle (AITM) compromise with a malicious firmware image (CWE-494). *https://www.cve.org/CVERecord?id=CVE-2021-22909* |
| **CVE-2008-4989** | Verification function trusts certificate chains in which the last certificate is self-signed. *https://www.cve.org/CVERecord?id=CVE-2008-4989* |
| **CVE-2012-5821** | Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA) *https://www.cve.org/CVERecord?id=CVE-2012-5821* |
| **CVE-2009-3046** | Web browser does not check if any intermediate certificates are revoked. *https://www.cve.org/CVERecord?id=CVE-2009-3046* |
| **CVE-2011-0199** | Operating system does not check Certificate Revocation List (CRL) in some cases, allowing spoofing using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2011-0199* |

| Reference | Description |
|---|---|
| **CVE-2012-5810** | Mobile banking application does not verify hostname, leading to financial loss. *https://www.cve.org/CVERecord?id=CVE-2012-5810* |
| **CVE-2012-3446** | Cloud-support library written in Python uses incorrect regular expression when matching hostname. *https://www.cve.org/CVERecord?id=CVE-2012-3446* |
| **CVE-2009-2408** | Web browser does not correctly handle '\0' character (NUL) in Common Name, allowing spoofing of https sites. *https://www.cve.org/CVERecord?id=CVE-2009-2408* |
| **CVE-2012-2993** | Smartphone device does not verify hostname, allowing spoofing of mail services. *https://www.cve.org/CVERecord?id=CVE-2012-2993* |
| **CVE-2012-5822** | Application uses third-party library that does not validate hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5822* |
| **CVE-2012-5819** | Cloud storage management application does not validate hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5819* |
| **CVE-2012-5817** | Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5817* |
| **CVE-2010-1378** | chain: incorrect calculation allows attackers to bypass certificate checks. *https://www.cve.org/CVERecord?id=CVE-2010-1378* |
| **CVE-2005-3170** | LDAP client accepts certificates even if they are not from a trusted CA. *https://www.cve.org/CVERecord?id=CVE-2005-3170* |
| **CVE-2009-0265** | chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain. *https://www.cve.org/CVERecord?id=CVE-2009-0265* |
| **CVE-2003-1229** | chain: product checks if client is trusted when it intended to check if the server is trusted, allowing validation of signed code. *https://www.cve.org/CVERecord?id=CVE-2003-1229* |
| **CVE-2002-0862** | Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints. *https://www.cve.org/CVERecord?id=CVE-2002-0862* |
| **CVE-2009-1358** | chain: OS package manager does not check properly check the return value, allowing bypass using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2009-1358* |

## 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 | 731 | OWASP Top Ten 2004 Category A10 - Insecure Configuration Management | 711 | 2360 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | V | 1200 | Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors | 1200 | 2608 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1382 | ICS Operations (& Maintenance): Emerging Energy Technologies | 1358 | 2538 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| OWASP Top Ten 2004 | A10 | CWE More Specific | Insecure Configuration Management |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 459 | Creating a Rogue Certification Authority Certificate |
| 475 | Signature Spoofing by Improper Validation |

### References

[REF-243]Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith and Lars Baumgärtner, Bernd Freisleben. "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security". 2012 October 6. < http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf >.

[REF-244]M. Bishop. "Computer Security: Art and Science". 2003. Addison-Wesley.



## CWE-296: Improper Following of a Certificate's Chain of Trust

**Weakness ID :** 296
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not follow, or incorrectly follows, the chain of trust for a certificate back to a trusted root certificate, resulting in incorrect trust of any resource that is associated with that certificate.

### Extended Description

If a system does not follow the chain of trust of a certificate to a root server, the certificate loses all usefulness as a metric of trust. Essentially, the trust gained from a certificate is derived from a chain of trust -- with a reputable trusted entity at the end of that list. The end user must trust that reputable source, and this reputable source must vouch for the resource in question through the medium of the certificate.

In some cases, this trust traverses several entities who vouch for one another. The entity trusted by the end user is at one end of this trust chain, while the certificate-wielding resource is at the other end of the chain. If the user receives a certificate at the end of one of these trust chains and then proceeds to check only that the first link in the chain, no real trust has been derived, since the entire chain must be traversed back to a trusted source to verify the certificate.

There are several ways in which the chain of trust might be broken, including but not limited to:

- Any certificate in the chain is self-signed, unless it the root.
- Not every intermediate certificate is checked, starting from the original certificate all the way up to the root certificate.
- An intermediate, CA-signed certificate does not have the expected Basic Constraints or other important extensions.
- The root certificate has been compromised or authorized to the wrong party.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 573 | Improper Following of Specification by Caller | 1307 |
| ChildOf | B | 295 | Improper Certificate Validation | 721 |
| PeerOf | V | 370 | Missing Check for Certificate Revocation after Initial Check | 924 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

## Applicable Platforms

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

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Non-Repudiation | Hide Activities<br>*Exploitation of this flaw can lead to the trust of data that may have originated with a spoofed source.* | |
| Integrity<br>Confidentiality<br>Availability<br>Access Control | Gain Privileges or Assume Identity<br>Execute Unauthorized Code or Commands<br>*Data, requests, or actions taken by the attacking entity can be carried out as a spoofed benign entity.* | |

## 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**

Ensure that proper certificate checking is included in the system design.

**Phase: Implementation**

Understand, and properly implement all checks necessary to ensure the integrity of certificate trust integrity.

**Phase: Implementation**

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the full chain of trust.

## Demonstrative Examples

**Example 1:**

This code checks the certificate of a connected peer.

*Example Language: C* *(Bad)*

```
if ((cert = SSL_get_peer_certificate(ssl)) && host)
```

```
    foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
    // certificate looks good, host can be trusted
```

In this case, because the certificate is self-signed, there was no external authority that could prove the identity of the host. The program could be communicating with a different system that is spoofing the host, e.g. by poisoning the DNS cache or using an Adversary-in-the-Middle (AITM) attack to modify the traffic from server to client.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2016-2402** | Server allows bypass of certificate pinning by sending a chain of trust that includes a trusted CA that is not pinned. *https://www.cve.org/CVERecord?id=CVE-2016-2402* |
| **CVE-2008-4989** | Verification function trusts certificate chains in which the last certificate is self-signed. *https://www.cve.org/CVERecord?id=CVE-2008-4989* |
| **CVE-2012-5821** | Chain: Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA). *https://www.cve.org/CVERecord?id=CVE-2012-5821* |
| **CVE-2009-3046** | Web browser does not check if any intermediate certificates are revoked. *https://www.cve.org/CVERecord?id=CVE-2009-3046* |
| **CVE-2009-0265** | chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain. *https://www.cve.org/CVERecord?id=CVE-2009-0265* |
| **CVE-2009-0124** | chain: incorrect check of return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain. *https://www.cve.org/CVERecord?id=CVE-2009-0124* |
| **CVE-2002-0970** | File-transfer software does not validate Basic Constraints of an intermediate CA-signed certificate. *https://www.cve.org/CVERecord?id=CVE-2002-0970* |
| **CVE-2002-0862** | Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints. *https://www.cve.org/CVERecord?id=CVE-2002-0862* |

### 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 948 | SFP Secondary Cluster: Digital Certificate | 888 | 2416 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1382 | ICS Operations (& Maintenance): Emerging Energy Technologies | 1358 | 2538 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure to follow chain of trust in certificate validation |

### References

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

[REF-245]Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh and Vitaly Shmatikov. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software". 2012 October 5. < http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf >.

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

## CWE-297: Improper Validation of Certificate with Host Mismatch

**Weakness ID :** 297
**Structure :** Simple
**Abstraction :** Variant

### Description

The product communicates with a host that provides a certificate, but the product does not properly ensure that the certificate is actually associated with that host.

### Extended Description

Even if a certificate is well-formed, signed, and follows the chain of trust, it may simply be a valid certificate for a different site than the site that the product is interacting with. If the certificate's host-specific data is not properly checked - such as the Common Name (CN) in the Subject or the Subject Alternative Name (SAN) extension of an X.509 certificate - it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data, impersonating a trusted host. In order to ensure data integrity, the certificate must be valid and it must pertain to the site that is being accessed.

Even if the product attempts to check the hostname, it is still possible to incorrectly check the hostname. For example, attackers could create a certificate with a name that begins with a trusted name followed by a NUL byte, which could cause some string-based comparisons to only examine the portion that contains the trusted name.

This weakness can occur even when the product uses Certificate Pinning, if the product does not verify the hostname at the time a certificate is pinned.

### Relationships

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 | 295 | Improper Certificate Validation | 721 |
| ChildOf | C | 923 | Improper Restriction of Communication Channel to Intended Endpoints | 1836 |
| PeerOf | V | 370 | Missing Check for Certificate Revocation after Initial Check | 924 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

## Applicable Platforms

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

**Technology** : Mobile *(Prevalence = Undetermined)*

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*The data read from the system vouched for by the certificate may not be from the expected system.* | |
| Authentication<br>Other | Other<br>*Trust afforded to the system in question - based on the malicious certificate - may allow for spoofing or redirection attacks.* | |

## 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*

### Dynamic Analysis with Manual Results Interpretation

Set up an untrusted endpoint (e.g. a server) with which the product will connect. Create a test certificate that uses an invalid hostname but is signed by a trusted CA and provide this certificate from the untrusted endpoint. If the product performs any operations instead of disconnecting and reporting an error, then this indicates that the hostname is not being checked and the test certificate has been accepted.

### Black Box

When Certificate Pinning is being used in a mobile application, consider using a tool such as Spinner [REF-955]. This methodology might be extensible to other technologies.

## Potential Mitigations

### Phase: Architecture and Design

Fully check the hostname of the certificate and provide the user with adequate information about the nature of the problem and how to proceed.

### Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.

## Demonstrative Examples

### Example 1:

The following OpenSSL code obtains a certificate and verifies it.

*Example Language: C* *(Bad)*

```
cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
  // do secret things
}
```

Even though the "verify" step returns X509_V_OK, this step does not include checking the Common Name against the name of the host. That is, there is no guarantee that the certificate is for the desired host. The SSL connection could have been established with a malicious host that provided a valid certificate.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2012-5810** | Mobile banking application does not verify hostname, leading to financial loss. *https://www.cve.org/CVERecord?id=CVE-2012-5810* |
| **CVE-2012-5811** | Mobile application for printing documents does not verify hostname, allowing attackers to read sensitive documents. *https://www.cve.org/CVERecord?id=CVE-2012-5811* |
| **CVE-2012-5807** | Software for electronic checking does not verify hostname, leading to financial loss. *https://www.cve.org/CVERecord?id=CVE-2012-5807* |
| **CVE-2012-3446** | Cloud-support library written in Python uses incorrect regular expression when matching hostname. *https://www.cve.org/CVERecord?id=CVE-2012-3446* |
| **CVE-2009-2408** | Web browser does not correctly handle '\0' character (NUL) in Common Name, allowing spoofing of https sites. *https://www.cve.org/CVERecord?id=CVE-2009-2408* |
| **CVE-2012-0867** | Database program truncates the Common Name during hostname verification, allowing spoofing. *https://www.cve.org/CVERecord?id=CVE-2012-0867* |
| **CVE-2010-2074** | Incorrect handling of '\0' character (NUL) in hostname verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2010-2074* |
| **CVE-2009-4565** | Mail server's incorrect handling of '\0' character (NUL) in hostname verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2009-4565* |
| **CVE-2009-3767** | LDAP server's incorrect handling of '\0' character (NUL) in hostname verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2009-3767* |
| **CVE-2012-5806** | Payment processing module does not verify hostname when connecting to PayPal using PHP fsockopen function. *https://www.cve.org/CVERecord?id=CVE-2012-5806* |
| **CVE-2012-2993** | Smartphone device does not verify hostname, allowing spoofing of mail services. *https://www.cve.org/CVERecord?id=CVE-2012-2993* |
| **CVE-2012-5804** | E-commerce module does not verify hostname when connecting to payment site. *https://www.cve.org/CVERecord?id=CVE-2012-5804* |
| **CVE-2012-5824** | Chat application does not validate hostname, leading to loss of privacy. *https://www.cve.org/CVERecord?id=CVE-2012-5824* |
| **CVE-2012-5822** | Application uses third-party library that does not validate hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5822* |
| **CVE-2012-5819** | Cloud storage management application does not validate hostname. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2012-5819* |
| **CVE-2012-5817** | Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5817* |
| **CVE-2012-5784** | SOAP platform does not verify the hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5784* |
| **CVE-2012-5782** | PHP library for payments does not verify the hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5782* |
| **CVE-2012-5780** | Merchant SDK for payments does not verify the hostname. *https://www.cve.org/CVERecord?id=CVE-2012-5780* |
| **CVE-2003-0355** | Web browser does not validate Common Name, allowing spoofing of https sites. *https://www.cve.org/CVERecord?id=CVE-2003-0355* |

## 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 | 948 | SFP Secondary Cluster: Digital Certificate | 888 | 2416 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure to validate host-specific certificate data |

## References

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

[REF-245]Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh and Vitaly Shmatikov. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software". 2012 October 5. < http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf >.

[REF-243]Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith and Lars Baumgärtner, Bernd Freisleben. "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security". 2012 October 6. < http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf >.

[REF-249]Kenneth Ballard. "Secure programming with the OpenSSL API, Part 2: Secure handshake". 2005 May 3. < https://developer.ibm.com/tutorials/l-openssl/?mhsrc=ibmsearch_a&mhq=secure%20programming%20with%20the%20openssl%20API >.2023-04-07.

[REF-250]Eric Rescorla. "An Introduction to OpenSSL Programming (Part I)". 2001 October 5. < https://www.linuxjournal.com/article/4822 >.2023-04-07.

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

[REF-955]Chris McMahon Stone, Tom Chothia and Flavio D. Garcia. "Spinner: Semi-Automatic Detection of Pinning without Hostname Verification". < http://www.cs.bham.ac.uk/~garciaf/publications/spinner.pdf >.2018-01-16.

# CWE-298: Improper Validation of Certificate Expiration

**Weakness ID :** 298
**Structure :** Simple
**Abstraction :** Variant

## Description

A certificate expiration is not validated or is incorrectly validated, so trust may be assigned to certificates that have been abandoned due to age.

## Extended Description

When the expiration of a certificate is not taken into account, no trust has necessarily been conveyed through it. Therefore, the validity of the certificate cannot be verified and all benefit of the certificate is lost.

## Relationships

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 | 672 | Operation on a Resource after Expiration or Release | 1488 |
| ChildOf | B | 295 | Improper Certificate Validation | 721 |
| PeerOf | B | 324 | Use of a Key Past its Expiration Date | 799 |
| PeerOf | V | 370 | Missing Check for Certificate Revocation after Initial Check | 924 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

## Applicable Platforms

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

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Other | Other<br>*The data read from the system vouched for by the expired certificate may be flawed due to malicious spoofing.* | |
| Authentication<br>Other | Other<br>*Trust afforded to the system in question - based on the expired certificate - may allow for spoofing attacks.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

**Phase: Implementation**

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the expiration.

## Demonstrative Examples

**Example 1:**

The following OpenSSL code ensures that there is a certificate and allows the use of expired certificates.

*Example Language: C* *(Bad)*

```
if (cert = SSL_get_peer(certificate(ssl)) {
  foo=SSL_get_verify_result(ssl);
  if ((X509_V_OK==foo) || (X509_V_ERR_CERT_HAS_EXPIRED==foo))
    //do stuff
```

If the call to SSL_get_verify_result() returns X509_V_ERR_CERT_HAS_EXPIRED, this means that the certificate has expired. As time goes on, there is an increasing chance for attackers to compromise the certificate.

## 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 948 | SFP Secondary Cluster: Digital Certificate | 888 | 2416 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure to validate certificate expiration |

## References

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

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

# CWE-299: Improper Check for Certificate Revocation

**Weakness ID :** 299
**Structure :** Simple
**Abstraction :** Base

## Description

The product does not check or incorrectly checks the revocation status of a certificate, which may cause it to use a certificate that has been compromised.

## Extended Description

An improper check for certificate revocation is a far more serious flaw than related certificate failures. This is because the use of any revoked certificate is almost certainly malicious. The most common reason for certificate revocation is compromise of the system in question, with the result that no legitimate servers will be using a revoked certificate, unless they are sorely out of sync.

## Relationships

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 | 404 | Improper Resource Shutdown or Release | 987 |
| ChildOf | B | 295 | Improper Certificate Validation | 721 |
| ParentOf | V | 370 | Missing Check for Certificate Revocation after Initial Check | 924 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

## Applicable Platforms

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

## Likelihood Of Exploit

Medium

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*Trust may be assigned to an entity who is not who it claims to be.* | |
| Integrity<br>Other | Other<br>*Data from an untrusted (and possibly malicious) source may be integrated.* | |
| Confidentiality | Read Application Data<br>*Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.* | |

## 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**

Ensure that certificates are checked for revoked status.

**Phase: Implementation**

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the revoked status.

## Demonstrative Examples

**Example 1:**

The following OpenSSL code ensures that there is a certificate before continuing execution.

*Example Language: C* *(Bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
  // got a certificate, do secret things
```

Because this code does not use SSL_get_verify_results() to check the certificate, it could accept certificates that have been revoked (X509_V_ERR_CERT_REVOKED). The product could be communicating with a malicious host.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2011-2014** | LDAP-over-SSL implementation does not check Certificate Revocation List (CRL), allowing spoofing using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2011-2014* |
| **CVE-2011-0199** | Operating system does not check Certificate Revocation List (CRL) in some cases, allowing spoofing using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2011-0199* |
| **CVE-2010-5185** | Antivirus product does not check whether certificates from signed executables have been revoked. *https://www.cve.org/CVERecord?id=CVE-2010-5185* |
| **CVE-2009-3046** | Web browser does not check if any intermediate certificates are revoked. *https://www.cve.org/CVERecord?id=CVE-2009-3046* |
| **CVE-2009-0161** | chain: Ruby module for OCSP misinterprets a response, preventing detection of a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2009-0161* |
| **CVE-2011-2701** | chain: incorrect parsing of replies from OCSP responders allows bypass using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2011-2701* |
| **CVE-2011-0935** | Router can permanently cache certain public keys, which would allow bypass if the certificate is later revoked. *https://www.cve.org/CVERecord?id=CVE-2011-0935* |
| **CVE-2009-1358** | chain: OS package manager does not properly check the return value, allowing bypass using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2009-1358* |
| **CVE-2009-0642** | chain: language interpreter does not properly check the return value from an OSCP function, allowing bypass using a revoked certificate. *https://www.cve.org/CVERecord?id=CVE-2009-0642* |
| **CVE-2008-4679** | chain: web service component does not call the expected method, which prevents a check for revoked certificates. *https://www.cve.org/CVERecord?id=CVE-2008-4679* |
| **CVE-2006-4410** | Certificate revocation list not searched for certain certificates. *https://www.cve.org/CVERecord?id=CVE-2006-4410* |
| **CVE-2006-4409** | Product cannot access certificate revocation list when an HTTP proxy is being used. *https://www.cve.org/CVERecord?id=CVE-2006-4409* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 948 | SFP Secondary Cluster: Digital Certificate | 888 | 2416 |

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

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure to check for certificate revocation |

### References

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

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

## CWE-300: Channel Accessible by Non-Endpoint

**Weakness ID :** 300
**Structure :** Simple
**Abstraction :** Class

### Description

The product does not adequately verify the identity of actors at both ends of a communication channel, or does not adequately ensure the integrity of the channel, in a way that allows the channel to be accessed or influenced by an actor that is not an endpoint.

### Extended Description

In order to establish secure communication between two parties, it is often important to adequately verify the identity of entities at each end of the communication channel. Inadequate or inconsistent verification may result in insufficient or incorrect identification of either communicating entity. This can have negative consequences such as misplaced trust in the entity at the other end of the channel. An attacker can leverage this by interposing between the communicating entities and masquerading as the original entity. In the absence of sufficient verification of identity, such an attacker can eavesdrop and potentially modify the communication between the original entities.

### Relationships

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 | 923 | Improper Restriction of Communication Channel to Intended Endpoints | 1836 |
| PeerOf | C | 602 | Client-Side Enforcement of Server-Side Security | 1359 |
| PeerOf | B | 603 | Use of Client-Side Authentication | 1363 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

### Applicable Platforms

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

### Alternate Terms

**Adversary-in-the-Middle / AITM** :

**Man-in-the-Middle / MITM** :

**Person-in-the-Middle / PITM** :

**Monkey-in-the-Middle** :

**Monster-in-the-Middle** :

**Manipulator-in-the-Middle** :

**On-path attack** :

**Interception attack** :

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Integrity<br>Access Control | Read Application Data<br>Modify Application Data<br>Gain Privileges or Assume Identity<br><br>*An attacker could pose as one of the entities and read or possibly modify the communication.* | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Implementation

Always fully authenticate both ends of any communications channel.

### Phase: Architecture and Design

Adhere to the principle of complete mediation.

### Phase: Implementation

A certificate binds an identity to a cryptographic key to authenticate a communicating party. Often, the certificate takes the encrypted form of the hash of the identity of the subject, the public key, and information such as time of issue or expiration using the issuer's private key. The certificate can be validated by deciphering the certificate with the issuer's public key. See also X.509 certificate signature chains and the PGP certification structure.

## Demonstrative Examples

### Example 1:

In the Java snippet below, data is sent over an unencrypted channel to a remote server.

*Example Language: Java* *(Bad)*

```
Socket sock;
PrintWriter out;
try {
  sock = new Socket(REMOTE_HOST, REMOTE_PORT);
  out = new PrintWriter(echoSocket.getOutputStream(), true);
  // Write data to remote host via socket output stream.
```

```
    ...
}
```

By eavesdropping on the communication channel or posing as the endpoint, an attacker would be able to read all of the transmitted data.

### Observed Examples

| Reference | Description |
|---|---|
| **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* |

### 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 | 859 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) | 844 | 2390 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 956 | SFP Secondary Cluster: Channel Attack | 888 | 2418 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Notes

**Maintenance**

The summary identifies multiple distinct possibilities, suggesting that this is a category that must be broken into more specific weaknesses.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Man-in-the-middle (MITM) |
| WASC | 32 | | Routing Detour |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC06-J | | Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 57 | Utilizing REST's Trust in the System Resource to Obtain Sensitive Data |
| 94 | Adversary in the Middle (AiTM) |
| 466 | Leveraging Active Adversary in the Middle Attacks to Bypass Same Origin Policy |
| 589 | DNS Blocking |
| 590 | IP Address Blocking |
| 612 | WiFi MAC Address Tracking |
| 613 | WiFi SSID Tracking |
| 615 | Evil Twin Wi-Fi Attack |
| 662 | Adversary in the Browser (AiTB) |

### References

[REF-244]M. Bishop. "Computer Security: Art and Science". 2003. Addison-Wesley.

# CWE-301: Reflection Attack in an Authentication Protocol

**Weakness ID :** 301
**Structure :** Simple
**Abstraction :** Base



## Description

Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user.

## Extended Description

A mutual authentication protocol requires each party to respond to a random challenge by the other party by encrypting it with a pre-shared key. Often, however, such protocols employ the same pre-shared key for communication with a number of different entities. A malicious user or an attacker can easily compromise this protocol without possessing the correct key by employing a reflection attack on the protocol.

Reflection attacks capitalize on mutual authentication schemes in order to trick the target into revealing the secret shared between it and another valid user. In a basic mutual-authentication scheme, a secret is known to both the valid user and the server; this allows them to authenticate. In order that they may verify this shared secret without sending it plainly over the wire, they utilize a Diffie-Hellman-style scheme in which they each pick a value, then request the hash of that value as keyed by the shared secret. In a reflection attack, the attacker claims to be a valid user and requests the hash of a random value from the server. When the server returns this value and requests its own value to be hashed, the attacker opens another connection to the server. This time, the hash requested by the attacker is the value which the server requested in the first connection. When the server returns this hashed value, it is used in the first connection, authenticating the attacker successfully as the impersonated valid user.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 2279 |
| PeerOf | C | 327 | Use of a Broken or Risky Cryptographic Algorithm | 806 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

## Applicable Platforms

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

## Likelihood Of Exploit

Medium

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Gain Privileges or Assume Identity<br>*The primary result of reflection attacks is successful authentication with a target machine -- as an impersonated user.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Use different keys for the initiator and responder or of a different type of challenge for the initiator and responder.

**Phase: Architecture and Design**

Let the initiator prove its identity before proceeding.

## Demonstrative Examples

**Example 1:**

The following example demonstrates the weakness.

*Example Language: C* *(Bad)*

```
unsigned char *simple_digest(char *alg,char *buf,unsigned int len, int *olen) {
  const EVP_MD *m;
  EVP_MD_CTX ctx;
  unsigned char *ret;
  OpenSSL_add_all_digests();
  if (!(m = EVP_get_digestbyname(alg))) return NULL;
  if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE))) return NULL;
  EVP_DigestInit(&ctx, m);
  EVP_DigestUpdate(&ctx,buf,len);
  EVP_DigestFinal(&ctx,ret,olen);
  return ret;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
  simple_digest("sha1",password,strlen(password_and_cmd)
  ...
  );
}
```

*Example Language: Java* *(Bad)*

```
String command = new String("some cmd to execute & the password") MessageDigest encer =
MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8"));
byte[] digest = encer.digest();
```

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-3435** | product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.<br>*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 | 718 | OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management | 629 | 2353 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 956 | SFP Secondary Cluster: Channel Attack | 888 | 2418 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Notes

### Maintenance

The term "reflection" is used in multiple ways within CWE and the community, so its usage should be reviewed.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Reflection attack in an auth protocol |
| OWASP Top Ten 2007 | A7 | CWE More Specific | Broken Authentication and Session Management |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 90 | Reflection Attack in Authentication Protocol |

## References

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

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

# CWE-302: Authentication Bypass by Assumed-Immutable Data

**Weakness ID :** 302
**Structure :** Simple
**Abstraction :** Base

## Description

The authentication scheme or implementation uses key data elements that are assumed to be immutable, but can be controlled or modified by the attacker.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 1723 |
| ChildOf | C | 1390 | Weak Authentication | 2279 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Operation**

**Phase: Implementation**

Implement proper protection for immutable data (e.g. environment variable, hidden form fields, etc.)

## Demonstrative Examples

**Example 1:**

In the following example, an "authenticated" cookie is used to determine whether or not a user should be granted access to a system.

*Example Language: Java* *(Bad)*

```
boolean authenticated = new Boolean(getCookieValue("authenticated")).booleanValue();
if (authenticated) {
  ...
}
```

Modifying the value of a cookie on the client-side is trivial, but many developers assume that cookies are essentially immutable.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-0367** | DebPloit *https://www.cve.org/CVERecord?id=CVE-2002-0367* |
| **CVE-2004-0261** | Web auth *https://www.cve.org/CVERecord?id=CVE-2004-0261* |
| **CVE-2002-1730** | Authentication bypass by setting certain cookies to "true". *https://www.cve.org/CVERecord?id=CVE-2002-1730* |
| **CVE-2002-1734** | Authentication bypass by setting certain cookies to "true". *https://www.cve.org/CVERecord?id=CVE-2002-1734* |
| **CVE-2002-2064** | Admin access by setting a cookie. *https://www.cve.org/CVERecord?id=CVE-2002-2064* |
| **CVE-2002-2054** | Gain privileges by setting cookie. *https://www.cve.org/CVERecord?id=CVE-2002-2054* |
| **CVE-2004-1611** | Product trusts authentication information in cookie. *https://www.cve.org/CVERecord?id=CVE-2004-1611* |
| **CVE-2005-1708** | Authentication bypass by setting admin-testing variable to true. *https://www.cve.org/CVERecord?id=CVE-2005-1708* |
| **CVE-2005-1787** | Bypass auth and gain privileges by setting a variable. *https://www.cve.org/CVERecord?id=CVE-2005-1787* |

## 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 859 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) | 844 | 2390 |
| MemberOf | C | 949 | SFP Secondary Cluster: Faulty Endpoint Authentication | 888 | 2416 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication Bypass via Assumed-Immutable Data |
| OWASP Top Ten 2004 | A1 | CWE More Specific | Unvalidated Input |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC02-J | | Do not base security checks on untrusted sources |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 10 | Buffer Overflow via Environment Variables |
| 13 | Subverting Environment Variable Values |
| 21 | Exploitation of Trusted Identifiers |
| 31 | Accessing/Intercepting/Modifying HTTP Cookies |
| 39 | Manipulating Opaque Client-based Data Tokens |
| 45 | Buffer Overflow via Symbolic Links |
| 77 | Manipulating User-Controlled Variables |
| 274 | HTTP Verb Tampering |

## CWE-303: Incorrect Implementation of Authentication Algorithm

**Weakness ID :** 303
**Structure :** Simple
**Abstraction :** Base

### Description

The requirements for the product dictate the use of an established authentication algorithm, but the implementation of the algorithm is incorrect.

### Extended Description

This incorrect implementation may allow authentication to be bypassed.

### Relationships

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 | 2279 |
| ParentOf | B | 304 | Missing Critical Step in Authentication | 745 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2003-0750** | Conditional should have been an 'or' not an 'and'. *https://www.cve.org/CVERecord?id=CVE-2003-0750* |

### 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 | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication Logic Error |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 90 | Reflection Attack in Authentication Protocol |

## CWE-304: Missing Critical Step in Authentication

**Weakness ID :** 304
**Structure :** Simple
**Abstraction :** Base

### Description

The product implements an authentication technique, but it skips a step that weakens the technique.

### Extended Description

Authentication techniques should follow the algorithms that define them exactly, otherwise authentication can be bypassed or more easily subjected to brute force attacks.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 573 | Improper Following of Specification by Caller | 1307 |
| ChildOf | B | 303 | Incorrect Implementation of Authentication Algorithm | 744 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Integrity<br>Confidentiality | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>Read Application Data<br>Execute Unauthorized Code or Commands<br>*This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or allowing attackers to execute arbitrary 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*

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2004-2163** | Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.<br>*https://www.cve.org/CVERecord?id=CVE-2004-2163* |
| **CVE-2005-3327** | Chain: Authentication bypass by skipping the first startup step as required by the protocol.<br>*https://www.cve.org/CVERecord?id=CVE-2005-3327* |

## 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |

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

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Missing Critical Step in Authentication |

## CWE-305: Authentication Bypass by Primary Weakness

**Weakness ID :** 305
**Structure :** Simple
**Abstraction :** Base

### Description

The authentication algorithm is sound, but the implemented mechanism can be bypassed as the result of a separate weakness that is primary to the authentication 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 | | 1390 | Weak Authentication | 2279 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1374** | The provided password is only compared against the first character of the real password.<br>*https://www.cve.org/CVERecord?id=CVE-2002-1374* |
| **CVE-2000-0979** | The password is not properly checked, which allows remote attackers to bypass access controls by sending a 1-byte password that matches the first character of the real password.<br>*https://www.cve.org/CVERecord?id=CVE-2000-0979* |
| **CVE-2001-0088** | Chain: Forum software does not properly initialize an array, which inadvertently sets the password to a single character, allowing remote attackers to easily guess the password and gain administrative privileges. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2001-0088* |

### 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 | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Notes

**Relationship**

Most "authentication bypass" errors are resultant, not primary.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Authentication Bypass by Primary Weakness |

## CWE-306: Missing Authentication for Critical Function

**Weakness ID :** 306
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 287 | Improper Authentication | 699 |
| ParentOf | B | 288 | Authentication Bypass Using an Alternate Path or Channel | 707 |
| ParentOf | B | 322 | Key Exchange without Entity Authentication | 795 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 287 | Improper Authentication | 699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

## Applicable Platforms

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

**Technology** : Cloud Computing *(Prevalence = Undetermined)*

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Other | Gain Privileges or Assume Identity<br>Varies by Context<br><br>*Exposing critical functionality essentially provides an attacker with the privilege level of that functionality. The consequences will depend on the associated functionality, but they can range from reading or modifying sensitive data, accessing administrative or other privileged functionality, or possibly even executing arbitrary code.* | |

## Detection Methods

### Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authentication mechanisms.

### Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authentication. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authentication libraries. Generally, automated static analysis tools have difficulty detecting custom authentication schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an established identity; an automated technique that detects the absence of authentication may report false positives.

*Effectiveness = Limited*

### 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 Cost effective for partial coverage: Attack Modeling

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

Divide the software into anonymous, normal, privileged, and administrative areas. Identify which of these areas require a proven user identity, and use a centralized authentication capability. Identify all potential communication channels, or other means of interaction with the software, to ensure that all channels are appropriately protected, including those channels that are assumed to be accessible only by authorized parties. Developers sometimes perform authentication at the primary channel, but open up a secondary channel that is assumed to be private. For example, a login mechanism may be listening on one network port, but after successful authentication, it may open up a second port where it waits for the connection, but avoids authentication because it assumes that only the authenticated party will connect to the port. In general, if the software or protocol allows a single session or user state to persist across multiple connections or channels, authentication and appropriate credential management need to be used throughout.

**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**

Where possible, avoid implementing custom, "grow-your-own" authentication routines and consider using authentication capabilities as provided by the surrounding framework, operating system, or environment. These capabilities may avoid common weaknesses that are unique to authentication; support automatic auditing and tracking; and make it easier to provide a clear separation between authentication tasks and authorization tasks. In environments such as the World Wide Web, the line between authentication and authorization is sometimes blurred. If custom authentication routines are required instead of those provided by the server, then these routines must be applied to every single page, since these pages could be requested directly.

**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 libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator [REF-45].

**Phase: Implementation**

**Phase: System Configuration**

**Phase: Operation**

When storing data in the cloud (e.g., S3 buckets, Azure blobs, Google Cloud Storage, etc.), use the provider's controls to require strong authentication for users who should be allowed to access the data [REF-1297] [REF-1298] [REF-1302].

## Demonstrative Examples

**Example 1:**

In the following Java example the method createBankAccount is used to create a BankAccount object for a bank management application.

*Example Language: Java* *(Bad)*

```java
public BankAccount createBankAccount(String accountNumber, String accountType,
String accountName, String accountSSN, double balance) {
  BankAccount account = new BankAccount();
  account.setAccountNumber(accountNumber);
  account.setAccountType(accountType);
  account.setAccountOwnerName(accountName);
  account.setAccountOwnerSSN(accountSSN);
  account.setBalance(balance);
  return account;
}
```

However, there is no authentication mechanism to ensure that the user creating this bank account object has the authority to create new bank accounts. Some authentication mechanisms should be used to verify that the user has the authority to create bank account objects.

The following Java code includes a boolean variable and method for authenticating a user. If the user has not been authenticated then the createBankAccount will not create the bank account object.

*Example Language: Java* *(Good)*

```java
private boolean isUserAuthentic = false;
// authenticate user,
// if user is authenticated then set variable to true
// otherwise set variable to false
public boolean authenticateUser(String username, String password) {
  ...
}
public BankAccount createNewBankAccount(String accountNumber, String accountType,
String accountName, String accountSSN, double balance) {
  BankAccount account = null;
  if (isUserAuthentic) {
    account = new BankAccount();
    account.setAccountNumber(accountNumber);
    account.setAccountType(accountType);
    account.setAccountOwnerName(accountName);
    account.setAccountOwnerSSN(accountSSN);
    account.setBalance(balance);
  }
  return account;
}
```

**Example 2:**

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 did not use any authentication for critical functionality in their OT products.

**Example 3:**

In 2021, a web site operated by PeopleGIS stored data of US municipalities in Amazon Web Service (AWS) Simple Storage Service (S3) buckets.

*Example Language: Other* *(Bad)*

A security researcher found 86 S3 buckets that could be accessed without authentication (CWE-306) and stored data unencrypted (CWE-312). These buckets exposed over 1000 GB of data and 1.6 million files including physical addresses, phone numbers, tax documents, pictures of driver's license IDs, etc. [REF-1296] [REF-1295]

While it was not publicly disclosed how the data was protected after discovery, multiple options could have been considered.

*Example Language: Other* *(Good)*

The sensitive information could have been protected by ensuring that the buckets did not have public read access, e.g., by enabling the s3-account-level-public-access-blocks-periodic rule to Block Public Access. In addition, the data could have been encrypted at rest using the appropriate S3 settings, e.g., by enabling server-side encryption using the s3-bucket-server-side-encryption-enabled setting. Other settings are available to further prevent bucket data from being leaked. [REF-1297]

## 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)<br>*https://www.cve.org/CVERecord?id=CVE-2022-31260* |
| **CVE-2022-29951** | TCP-based protocol in Programmable Logic Controller (PLC) has no authentication.<br>*https://www.cve.org/CVERecord?id=CVE-2022-29951* |
| **CVE-2022-29952** | Condition Monitor firmware uses a protocol that does not require authentication.<br>*https://www.cve.org/CVERecord?id=CVE-2022-29952* |
| **CVE-2022-30276** | SCADA-based protocol for bridging WAN and LAN traffic has no authentication.<br>*https://www.cve.org/CVERecord?id=CVE-2022-30276* |
| **CVE-2022-30313** | Safety Instrumented System uses proprietary TCP protocols with no authentication.<br>*https://www.cve.org/CVERecord?id=CVE-2022-30313* |
| **CVE-2022-30317** | Distributed Control System (DCS) uses a protocol that has no authentication.<br>*https://www.cve.org/CVERecord?id=CVE-2022-30317* |
| **CVE-2021-21972** | Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses .. path traversal sequences (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV.<br>*https://www.cve.org/CVERecord?id=CVE-2021-21972* |
| **CVE-2020-10263** | Bluetooth speaker does not require authentication for the debug functionality on the UART port, allowing root shell access<br>*https://www.cve.org/CVERecord?id=CVE-2020-10263* |
| **CVE-2021-23147** | WiFi router does not require authentication for its UART port, allowing adversaries with physical access to execute commands as root<br>*https://www.cve.org/CVERecord?id=CVE-2021-23147* |

| Reference | Description |
|---|---|
| **CVE-2021-37415** | IT management product does not perform authentication for some REST API requests, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-37415* |
| **CVE-2020-13927** | Default setting in workflow management product allows all API requests without authentication, as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2020-13927* |
| **CVE-2002-1810** | MFV. Access TFTP server without authentication and obtain configuration file with sensitive plaintext information. *https://www.cve.org/CVERecord?id=CVE-2002-1810* |
| **CVE-2008-6827** | Agent software running at privileges does not authenticate incoming requests over an unprotected channel, allowing a Shatter" attack. *https://www.cve.org/CVERecord?id=CVE-2008-6827* |
| **CVE-2004-0213** | Product enforces restrictions through a GUI but not through privileged APIs. *https://www.cve.org/CVERecord?id=CVE-2004-0213* |
| **CVE-2020-15483** | monitor device allows access to physical UART debug port without authentication *https://www.cve.org/CVERecord?id=CVE-2020-15483* |
| **CVE-2019-9201** | Programmable Logic Controller (PLC) does not have an authentication feature on its communication protocols. *https://www.cve.org/CVERecord?id=CVE-2019-9201* |

## 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 | 2376 |
| MemberOf | C | 812 | OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management | 809 | 2378 |
| MemberOf | C | 866 | 2011 Top 25 - Porous Defenses | 900 | 2393 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 952 | SFP Secondary Cluster: Missing Authentication | 888 | 2417 |
| MemberOf | V | 1337 | Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses | 1337 | 2610 |
| MemberOf | V | 1350 | Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses | 1350 | 2615 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1364 | ICS Communications: Zone Boundary Failures | 1358 | 2522 |
| MemberOf | C | 1365 | ICS Communications: Unreliability | 1358 | 2523 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1368 | ICS Dependencies (& Architecture): External Digital Systems | 1358 | 2526 |
| MemberOf | V | 1387 | Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses | 1387 | 2618 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |
| MemberOf | V | 1425 | Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses | 1425 | 2621 |
| MemberOf | V | 1430 | Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses | 1430 | 2622 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | No Authentication for Critical Function |
| Software Fault Patterns | SFP31 | | Missing authentication |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.2 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 2.1 |
| ISA/IEC 62443 | Part 4-1 | | Req SR-2 |
| ISA/IEC 62443 | Part 4-1 | | Req SVV-3 |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 12 | Choosing Message Identifier |
| 36 | Using Unpublished Interfaces or Functionality |
| 62 | Cross Site Request Forgery |
| 166 | Force the System to Reset Values |
| 216 | Communication Channel Manipulation |

### References

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

[REF-257]Frank Kim. "Top 25 Series - Rank 19 - Missing Authentication for Critical Function". 2010 February 3. SANS Software Security Institute. < https://www.sans.org/blog/top-25-series-rank-19-missing-authentication-for-critical-function/ >.2023-04-07.

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

[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-1295]WizCase. "Over 80 US Municipalities' Sensitive Information, Including Resident's Personal Data, Left Vulnerable in Massive Data Breach". 2021 July 0. < https://www.wizcase.com/blog/us-municipality-breach-report/ >.

[REF-1296]Jonathan Greig. "1,000 GB of local government data exposed by Massachusetts software company". 2021 July 2. < https://www.zdnet.com/article/1000-gb-of-local-government-data-exposed-by-massachusetts-software-company/ >.

[REF-1297]Amazon. "AWS Foundational Security Best Practices controls". 2022. < https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-controls-reference.html >.2023-04-07.

[REF-1298]Microsoft. "Authentication and authorization in Azure App Service and Azure Functions". 2021 November 3. < https://learn.microsoft.com/en-us/azure/app-service/overview-authentication-authorization >.2022-10-11.

[REF-1302]Google Cloud. "Authentication and authorization use cases". 2022 October 1. < https://cloud.google.com/docs/authentication/use-cases >.2022-10-11.

## CWE-307: Improper Restriction of Excessive Authentication Attempts

**Weakness ID :** 307
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not implement sufficient measures to prevent multiple failed authentication attempts within a short time frame.

## Relationships

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 | 799 | Improper Control of Interaction Frequency | 1708 |
| ChildOf | C | 1390 | Weak Authentication | 2279 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 287 | Improper Authentication | 699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>*An attacker could perform an arbitrary number of authentication attempts using different passwords, and eventually gain access to the targeted account using a brute force attack.* | |

## Detection Methods

### Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

*Effectiveness = High*

### Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer Cost effective for partial coverage: Forced Path Execution

*Effectiveness = High*

### Manual Static Analysis - Source Code

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

*Effectiveness = SOAR Partial*

**Automated Static Analysis**

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

*Effectiveness = SOAR Partial*

**Architecture or Design Review**

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

*Effectiveness = High*

## Potential Mitigations

**Phase: Architecture and Design**

Common protection mechanisms include: Disconnecting the user after a small number of failed attempts Implementing a timeout Locking out a targeted account Requiring a computational task on the user's part.

**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. Consider using libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator. [REF-45]

## Demonstrative Examples

**Example 1:**

In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts [REF-236]. The attacker targeted a member of Twitter's support team and was able to successfully guess the member's password using a brute force attack by guessing a large number of common words. After gaining access as the member of the support staff, the attacker used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts.

**Example 2:**

The following code, extracted from a servlet's doPost() method, performs an authentication lookup every time the servlet is invoked.

*Example Language: Java* *(Bad)*

```
String username = request.getParameter("username");
String password = request.getParameter("password");
int authResult = authenticateUser(username, password);
```

However, the software makes no attempt to restrict excessive authentication attempts.

**Example 3:**

This code attempts to limit the number of login attempts by causing the process to sleep before completing the authentication.

*Example Language: PHP* *(Bad)*

```
$username = $_POST['username'];
$password = $_POST['password'];
sleep(2000);
$isAuthenticated = authenticateUser($username, $password);
```

However, there is no limit on parallel connections, so this does not increase the amount of time an attacker needs to complete an attack.

**Example 4:**

In the following C/C++ example the validateUser method opens a socket connection, reads a username and password from the socket and attempts to authenticate the username and password.

*Example Language: C* *(Bad)*

```
int validateUser(char *host, int port)
{
  int socket = openSocketConnection(host, port);
  if (socket < 0) {
    printf("Unable to open socket connection");
    return(FAIL);
  }
  int isValidUser = 0;
  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);
}
```

The validateUser method will continuously check for a valid username and password without any restriction on the number of authentication attempts made. The method should limit the number of authentication attempts made to prevent brute force attacks as in the following example code.

*Example Language: C* *(Good)*

```
int validateUser(char *host, int port)
{
  ...
  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);
    }
}
```

**Example 5:**

Consider this example from a real-world attack against the iPhone [REF-1218]. An attacker can use brute force methods; each time there is a failed guess, the attacker quickly cuts the power before the failed entry is recorded, effectively bypassing the intended limit on the number of failed authentication attempts. Note that this attack requires removal of the cell phone battery and connecting directly to the phone's power source, and the brute force attack is still time-consuming.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2019-0039** | the REST API for a network OS has a high limit for number of connections, allowing brute force password guessing<br>*https://www.cve.org/CVERecord?id=CVE-2019-0039* |
| **CVE-1999-1152** | Product does not disconnect or timeout after multiple failed logins.<br>*https://www.cve.org/CVERecord?id=CVE-1999-1152* |
| **CVE-2001-1291** | Product does not disconnect or timeout after multiple failed logins.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1291* |
| **CVE-2001-0395** | Product does not disconnect or timeout after multiple failed logins.<br>*https://www.cve.org/CVERecord?id=CVE-2001-0395* |
| **CVE-2001-1339** | Product does not disconnect or timeout after multiple failed logins.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1339* |
| **CVE-2002-0628** | Product does not disconnect or timeout after multiple failed logins.<br>*https://www.cve.org/CVERecord?id=CVE-2002-0628* |
| **CVE-1999-1324** | User accounts not disabled when they exceed a threshold; possibly a resultant problem.<br>*https://www.cve.org/CVERecord?id=CVE-1999-1324* |

### 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 808 | 2010 Top 25 - Weaknesses On the Cusp | 800 | 2376 |
| MemberOf | C | 812 | OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management | 809 | 2378 |
| MemberOf | C | 866 | 2011 Top 25 - Porous Defenses | 900 | 2393 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 955 | SFP Secondary Cluster: Unrestricted Authentication | 888 | 2418 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | AUTHENT.MULTFAIL | | Multiple Failed Authentication Attempts not Prevented |
| Software Fault Patterns | SFP34 | | Unrestricted authentication |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 16 | Dictionary-based Password Attack |
| 49 | Password Brute Forcing |
| 560 | Use of Known Domain Credentials |
| 565 | Password Spraying |
| 600 | Credential Stuffing |
| 652 | Use of Known Kerberos Credentials |
| 653 | Use of Known Operating System Credentials |

### References

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

[REF-236]Kim Zetter. "Weak Password Brings 'Happiness' to Twitter Hacker". 2009 January 9. < https://www.wired.com/2009/01/professed-twitt/ >.2023-04-07.

[REF-1218]Graham Cluley. "This Black Box Can Brute Force Crack iPhone PIN Passcodes". The Mac Security Blog. 2015 March 6. < https://www.intego.com/mac-security-blog/iphone-pin-pass-code/ >.

## CWE-308: Use of Single-factor Authentication

**Weakness ID :** 308
**Structure :** Simple
**Abstraction :** Base

### Description

The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme.

### Extended Description

While the use of multiple authentication schemes is simply piling on more complexity on top of authentication, it is inestimably valuable to have such measures of redundancy. The use of weak, reused, and common passwords is rampant on the internet. Without the added protection of multiple authentication schemes, a single mistake can result in the compromise of an account. For this reason, if multiple schemes are possible and also easy to use, they should be implemented and required.

### Relationships

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 | 654 | Reliance on a Single Factor in a Security Decision | 1448 |
| ChildOf | C | 1390 | Weak Authentication | 2279 |
| PeerOf | B | 309 | Use of Password System for Primary Authentication | 761 |
| PeerOf | B | 309 | Use of Password System for Primary Authentication | 761 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |

## Applicable Platforms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>*If the secret in a single-factor authentication scheme gets compromised, full authentication is possible.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Use multiple independent authentication schemes, which ensures that -- if one of the methods is compromised -- the system itself is still likely safe from compromise.

## Demonstrative Examples

**Example 1:**

In both of these examples, a user is logged in if their given password matches a stored password:

*Example Language: C* *(Bad)*

```
unsigned char *check_passwd(char *plaintext) {
  ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
  //Login if hash matches stored hash
  if (equal(ctext, secret_password())) {
    login_user();
  }
}
```

*Example Language: Java* *(Bad)*

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
  login_user();
}
```

This code relies exclusively on a password mechanism (CWE-309) using only one factor of authentication (CWE-308). If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also uses SHA-1, which is a weak hash (CWE-328). It also does not use a salt (CWE-759).

## Observed Examples

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

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1028 | OWASP Top Ten 2017 Category A2 - Broken Authentication | 1026 | 2457 |
| MemberOf | C | 1368 | ICS Dependencies (& Architecture): External Digital Systems | 1358 | 2526 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Using single-factor authentication |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 16 | Dictionary-based Password Attack |
| 49 | Password Brute Forcing |
| 55 | Rainbow Table Password Cracking |
| 70 | Try Common or Default Usernames and Passwords |
| 509 | Kerberoasting |
| 555 | Remote Services with Stolen Credentials |
| 560 | Use of Known Domain Credentials |
| 561 | Windows Admin Shares with Stolen Credentials |
| 565 | Password Spraying |
| 600 | Credential Stuffing |
| 644 | Use of Captured Hashes (Pass The Hash) |
| 645 | Use of Captured Tickets (Pass The Ticket) |
| 652 | Use of Known Kerberos Credentials |
| 653 | Use of Known Operating System Credentials |

### References

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

## CWE-309: Use of Password System for Primary Authentication

**Weakness ID :** 309
**Structure :** Simple
**Abstraction :** Base

### Description

The use of password systems as the primary means of authentication may be subject to several flaws or shortcomings, each reducing the effectiveness of the mechanism.

### Relationships

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 | 654 | Reliance on a Single Factor in a Security Decision | 1448 |
| ChildOf | C | 1390 | Weak Authentication | 2279 |
| PeerOf | B | 308 | Use of Single-factor Authentication | 759 |
| PeerOf | B | 262 | Not Using Password Aging | 640 |
| PeerOf | B | 308 | Use of Single-factor Authentication | 759 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |



## Applicable Platforms

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

## Background Details

Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks,and such frequent compromise that their use in the most simple implementation is not practical.

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>*A password authentication mechanism error will almost always result in attackers being authorized as valid users.* | |

## Potential Mitigations

**Phase: Architecture and Design**

In order to protect password systems from compromise, the following should be noted: Passwords should be stored safely to prevent insider attack and to ensure that -- if a system is compromised -- the passwords are not retrievable. Due to password reuse, this information may be useful in the compromise of other systems these users work with. In order to protect these passwords, they should be stored encrypted, in a non-reversible state, such that the original text password cannot be extracted from the stored value. Password aging should be strictly enforced to ensure that passwords do not remain unchanged for long periods of time. The longer a password remains in use, the higher the probability that it has been compromised. For this reason, passwords should require refreshing periodically, and users should be informed of the risk of passwords which remain in use for too long. Password strength should be enforced intelligently. Rather than restrict passwords to specific content, or specific length, users should be encouraged to use upper and lower case letters, numbers, and symbols in their passwords. The system should also ensure that no passwords are derived from dictionary words.

**Phase: Architecture and Design**

Use a zero-knowledge password protocol, such as SRP.

**Phase: Architecture and Design**

Ensure that passwords are stored safely and are not reversible.

**Phase: Architecture and Design**

Implement password aging functionality that requires passwords be changed after a certain point.

**Phase: Architecture and Design**

Use a mechanism for determining the strength of a password and notify the user of weak password use.

**Phase: Architecture and Design**

Inform the user of why password protections are in place, how they work to protect data integrity, and why it is important to heed their warnings.

## Demonstrative Examples

**Example 1:**

In both of these examples, a user is logged in if their given password matches a stored password:

*Example Language: C* *(Bad)*

```
unsigned char *check_passwd(char *plaintext) {
  ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
  //Login if hash matches stored hash
  if (equal(ctext, secret_password())) {
    login_user();
  }
}
```

*Example Language: Java* *(Bad)*

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
  login_user();
}
```

This code relies exclusively on a password mechanism (CWE-309) using only one factor of authentication (CWE-308). If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also uses SHA-1, which is a weak hash (CWE-328). It also does not use a salt (CWE-759).

## 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 947 | SFP Secondary Cluster: Authentication Bypass | 888 | 2415 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Using password systems |
| OWASP Top Ten 2004 | A3 | CWE More Specific | Broken Authentication and Session Management |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 16 | Dictionary-based Password Attack |

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 49 | Password Brute Forcing |
| 55 | Rainbow Table Password Cracking |
| 70 | Try Common or Default Usernames and Passwords |
| 509 | Kerberoasting |
| 555 | Remote Services with Stolen Credentials |
| 560 | Use of Known Domain Credentials |
| 561 | Windows Admin Shares with Stolen Credentials |
| 565 | Password Spraying |
| 600 | Credential Stuffing |
| 652 | Use of Known Kerberos Credentials |
| 653 | Use of Known Operating System Credentials |

### References

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

## CWE-311: Missing Encryption of Sensitive Data

**Weakness ID :** 311
**Structure :** Simple
**Abstraction :** Class

### Description

The product does not encrypt sensitive or critical information before storage or transmission.

### Extended Description

The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

### Relationships

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 | 1529 |
| ParentOf | B | 312 | Cleartext Storage of Sensitive Information | 771 |
| ParentOf | B | 319 | Cleartext Transmission of Sensitive Information | 786 |
| PeerOf | C | 327 | Use of a Broken or Risky Cryptographic Algorithm | 806 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 312 | Cleartext Storage of Sensitive Information | 771 |
| ParentOf | B | 319 | Cleartext Transmission of Sensitive Information | 786 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

### Applicable Platforms

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

### Likelihood Of Exploit

High

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data<br><br>*If the application does not use a secure channel, such as SSL, to exchange sensitive information, it is possible for an attacker with access to the network traffic to sniff packets from the connection and uncover the data. This attack is not technically difficult, but does require physical access to some portion of the network over which the sensitive data travels. This access is usually somewhere near where the user is connected to the network (such as a colleague on the company network) but can be anywhere along the path from the user to the end server.* | |
| Confidentiality<br>Integrity | Modify Application Data<br><br>*Omitting the use of encryption in any program which transfers data over a network of any kind should be considered on par with delivering the data sent to each user on the local networks of both the sender and receiver. Worse, this omission allows for the injection of data into a stream of communication between two parties -- with no means for the victims to separate valid data from invalid. In this day of widespread network attacks and password collection sniffers, it is an unnecessary risk to omit encryption from the design of any system which might benefit from it.* | |

### Detection Methods

#### Manual Analysis

The characterizaton of sensitive data often requires domain-specific understanding, so manual methods are useful. However, manual efforts might not achieve desired code coverage within limited time constraints. Black box methods may produce artifacts (e.g. stored data or unencrypted network transfer) that require manual evaluation.

*Effectiveness = High*

#### Automated Analysis

Automated measurement of the entropy of an input/output source may indicate the use or lack of encryption, but human analysis is still required to distinguish intentionally-unencrypted data (e.g. metadata) from sensitive data.

#### 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: Highly cost effective: Network Sniffer Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Man-in-the-middle attack tool

*Effectiveness = High*

**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: 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: 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: Requirements**

Clearly specify which data or resources are valuable enough that they should be protected by encryption. Require that any transmission or storage of this data/resource should use well-vetted encryption algorithms.

**Phase: Architecture and Design**

Ensure that encryption is properly integrated into the system design, including but not necessarily limited to: Encryption that is needed to store or transmit private data of the users of the system Encryption that is needed to protect the system itself from unauthorized disclosure or tampering Identify the separate needs and contexts for encryption: One-way (i.e., only the user or recipient needs to have the key). This can be achieved using public key cryptography, or other techniques in which the encrypting party (i.e., the product) does not need to have access to a private key. Two-way (i.e., the encryption can be automatically performed on behalf of a user, but the key must be available so that the plaintext can be automatically recoverable by that user). This requires storage of the private key in a format that is recoverable only by the user (or perhaps by the operating system) in a way that cannot be recovered by others. Using threat modeling or other techniques, assume that data can be compromised through a separate vulnerability or weakness, and determine where encryption will be most effective. Ensure that data that should be private is not being inadvertently exposed using weaknesses such as insecure permissions (CWE-732). [REF-7]

**Phase: Architecture and Design**

*Strategy = Libraries or Frameworks*

When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis. For example, US government systems require FIPS 140-2 certification. Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak. Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. [REF-267]

**Phase: Architecture and Design**

*Strategy = Separation of Privilege*

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

**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.

**Phase: Implementation**

*Strategy = Attack Surface Reduction*

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible.

*Effectiveness = Defense in Depth*

*This makes it easier to spot places in the code where data is being used that is unencrypted.*

## Demonstrative Examples

**Example 1:**

This code writes a user's login information to a cookie so the user does not have to login again later.

*Example Language: PHP* *(Bad)*

```
function persistLogin($username, $password){
  $data = array("username" => $username, "password"=> $password);
  setcookie ("userdata", $data);
}
```

The code stores the user's username and password in plaintext in a cookie on the user's machine. This exposes the user's login information if their computer is compromised by an attacker. Even if the user's machine is not compromised, this weakness combined with cross-site scripting (CWE-79) could allow an attacker to remotely copy the cookie.

Also note this example code also exhibits Plaintext Storage in a Cookie (CWE-315).

**Example 2:**

The following code attempts to establish a connection, read in a password, then store it to a buffer.

*Example Language: C* *(Bad)*

```
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
```

```
memcpy( (char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
  write(dfd,password_buffer,n);
  ...
```

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

**Example 3:**

The following code attempts to establish a connection to a site to communicate sensitive information.

*Example Language: Java* *(Bad)*

```
try {
  URL u = new URL("http://www.secret.example.org/");
  HttpURLConnection hu = (HttpURLConnection) u.openConnection();
  hu.setRequestMethod("PUT");
  hu.connect();
  OutputStream os = hu.getOutputStream();
  hu.disconnect();
}
catch (IOException e) {
  //...
}
```

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

## 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* |
| **CVE-2009-1466** | password stored in cleartext in a file with insecure permissions *https://www.cve.org/CVERecord?id=CVE-2009-1466* |
| **CVE-2009-0152** | chat program disables SSL in some circumstances even when the user says to use SSL. *https://www.cve.org/CVERecord?id=CVE-2009-0152* |
| **CVE-2009-1603** | Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption *https://www.cve.org/CVERecord?id=CVE-2009-1603* |
| **CVE-2009-0964** | storage of unencrypted passwords in a database *https://www.cve.org/CVERecord?id=CVE-2009-0964* |
| **CVE-2008-6157** | storage of unencrypted passwords in a database *https://www.cve.org/CVERecord?id=CVE-2008-6157* |
| **CVE-2008-6828** | product stores a password in cleartext in memory *https://www.cve.org/CVERecord?id=CVE-2008-6828* |
| **CVE-2008-1567** | storage of a secret key in cleartext in a temporary file *https://www.cve.org/CVERecord?id=CVE-2008-1567* |
| **CVE-2008-0174** | SCADA product uses HTTP Basic Authentication, which is not encrypted *https://www.cve.org/CVERecord?id=CVE-2008-0174* |
| **CVE-2007-5778** | login credentials stored unencrypted in a registry key *https://www.cve.org/CVERecord?id=CVE-2007-5778* |
| **CVE-2002-1949** | Passwords transmitted in cleartext. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2002-1949* |
| **CVE-2008-4122** | Chain: Use of HTTPS cookie without "secure" flag causes it to be transmitted across unencrypted HTTP. *https://www.cve.org/CVERecord?id=CVE-2008-4122* |
| **CVE-2008-3289** | Product sends password hash in cleartext in violation of intended policy. *https://www.cve.org/CVERecord?id=CVE-2008-3289* |
| **CVE-2008-4390** | Remote management feature sends sensitive information including passwords in cleartext. *https://www.cve.org/CVERecord?id=CVE-2008-4390* |
| **CVE-2007-5626** | Backup routine sends password in cleartext in email. *https://www.cve.org/CVERecord?id=CVE-2007-5626* |
| **CVE-2004-1852** | Product transmits Blowfish encryption key in cleartext. *https://www.cve.org/CVERecord?id=CVE-2004-1852* |
| **CVE-2008-0374** | Printer sends configuration information, including administrative password, in cleartext. *https://www.cve.org/CVERecord?id=CVE-2008-0374* |
| **CVE-2007-4961** | Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294). *https://www.cve.org/CVERecord?id=CVE-2007-4961* |
| **CVE-2007-4786** | Product sends passwords in cleartext to a log server. *https://www.cve.org/CVERecord?id=CVE-2007-4786* |
| **CVE-2005-3140** | Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes. *https://www.cve.org/CVERecord?id=CVE-2005-3140* |

## 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 | 719 | OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage | 629 | 2354 |
| MemberOf | C | 720 | OWASP Top Ten 2007 Category A9 - Insecure Communications | 629 | 2354 |
| MemberOf | C | 729 | OWASP Top Ten 2004 Category A8 - Insecure Storage | 711 | 2359 |
| MemberOf | C | 803 | 2010 Top 25 - Porous Defenses | 800 | 2376 |
| MemberOf | C | 816 | OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage | 809 | 2380 |
| MemberOf | C | 818 | OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection | 809 | 2381 |
| MemberOf | C | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 866 | 2011 Top 25 - Porous Defenses | 900 | 2393 |
| MemberOf | C | 930 | OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management | 928 | 2410 |
| MemberOf | C | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | V | 1340 | CISQ Data Protection Measures | 1340 | 2611 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## 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.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure to encrypt data |
| OWASP Top Ten 2007 | A8 | CWE More Specific | Insecure Cryptographic Storage |
| OWASP Top Ten 2007 | A9 | CWE More Specific | Insecure Communications |
| OWASP Top Ten 2004 | A8 | CWE More Specific | Insecure Storage |
| WASC | 4 | | Insufficient Transport Layer Protection |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC00-J | | Use SSLSocket rather than Socket for secure data exchange |
| Software Fault Patterns | SFP23 | | Exposed Data |
| ISA/IEC 62443 | Part 3-3 | | Req SR 4.1 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 4.3 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 4.1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 7.3 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 1.5 |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 31 | Accessing/Intercepting/Modifying HTTP Cookies |
| 37 | Retrieve Embedded Sensitive Data |
| 65 | Sniff Application Code |
| 157 | Sniffing Attacks |
| 158 | Sniffing Network Traffic |
| 204 | Lifting Sensitive Data Embedded in Cache |
| 383 | Harvesting Information via API Event Monitoring |
| 384 | Application API Message Manipulation via Man-in-the-Middle |
| 385 | Transaction or Event Tampering via Application API Manipulation |
| 386 | Application API Navigation Remapping |
| 387 | Navigation Remapping To Propagate Malicious Content |
| 388 | Application API Button Hijacking |
| 477 | Signature Spoofing by Mixing Signed and Unsigned Content |
| 609 | Cellular Traffic Intercept |

## References

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

[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-265]Frank Kim. "Top 25 Series - Rank 10 - Missing Encryption of Sensitive Data". 2010 February 6. SANS Software Security Institute. < https://www.sans.org/blog/top-25-series-rank-10-missing-encryption-of-sensitive-data/ >.2023-04-07.

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

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

## CWE-312: Cleartext Storage of Sensitive Information

**Weakness ID :** 312
**Structure :** Simple
**Abstraction :** Base

### Description

The product stores sensitive information in cleartext within a resource that might be accessible to another control sphere.

### Extended Description

Because the information is stored in cleartext (i.e., unencrypted), attackers could potentially read it. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

When organizations adopt cloud services, it can be easier for attackers to access the data from anywhere on the Internet.

In some systems/environments such as cloud, the use of "double encryption" (at both the software and hardware layer) might be required, and the developer might be solely responsible for both layers, instead of shared responsibility with the administrator of the broader system/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 | 922 | Insecure Storage of Sensitive Information | 1835 |
| ChildOf | C | 311 | Missing Encryption of Sensitive Data | 764 |
| ParentOf | V | 313 | Cleartext Storage in a File or on Disk | 777 |
| ParentOf | V | 314 | Cleartext Storage in the Registry | 779 |
| ParentOf | V | 315 | Cleartext Storage of Sensitive Information in a Cookie | 781 |
| ParentOf | V | 316 | Cleartext Storage of Sensitive Information in Memory | 782 |
| ParentOf | V | 317 | Cleartext Storage of Sensitive Information in GUI | 784 |
| ParentOf | V | 318 | Cleartext Storage of Sensitive Information in Executable | 785 |

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | V | 526 | Cleartext Storage of Sensitive Information in an Environment Variable | 1243 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 311 | Missing Encryption of Sensitive Data | 764 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 199 | Information Management Errors | 2333 |

## Applicable Platforms

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

**Technology** : Cloud Computing *(Prevalence = Undetermined)*

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

**Technology** : Mobile *(Prevalence = Undetermined)*

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data<br>*An attacker with access to the system could read sensitive information stored in cleartext.* | |

## 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

### Phase: System Configuration

### Phase: Operation

When storing data in the cloud (e.g., S3 buckets, Azure blobs, Google Cloud Storage, etc.), use the provider's controls to encrypt the data at rest. [REF-1297] [REF-1299] [REF-1301]

## Demonstrative Examples

### Example 1:

The following code excerpt stores a plaintext user account ID in a browser cookie.

*Example Language: Java* *(Bad)*

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Because the account ID is in plaintext, the user's account information is exposed if their computer is compromised by an attacker.

**Example 2:**

This code writes a user's login information to a cookie so the user does not have to login again later.

*Example Language: PHP* *(Bad)*

```
function persistLogin($username, $password){
  $data = array("username" => $username, "password"=> $password);
  setcookie ("userdata", $data);
}
```

The code stores the user's username and password in plaintext in a cookie on the user's machine. This exposes the user's login information if their computer is compromised by an attacker. Even if the user's machine is not compromised, this weakness combined with cross-site scripting (CWE-79) could allow an attacker to remotely copy the cookie.

Also note this example code also exhibits Plaintext Storage in a Cookie (CWE-315).

**Example 3:**

The following code attempts to establish a connection, read in a password, then store it to a buffer.

*Example Language: C* *(Bad)*

```
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
memcpy( (char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
  write(dfd,password_buffer,n);
  ...
```

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

**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.

At least one OT product stored a password in plaintext.

**Example 6:**

In 2021, a web site operated by PeopleGIS stored data of US municipalities in Amazon Web Service (AWS) Simple Storage Service (S3) buckets.

*Example Language: Other* *(Bad)*

A security researcher found 86 S3 buckets that could be accessed without authentication (CWE-306) and stored data unencrypted (CWE-312). These buckets exposed over 1000 GB of data and 1.6 million files including physical addresses, phone numbers, tax documents, pictures of driver's license IDs, etc. [REF-1296] [REF-1295]

While it was not publicly disclosed how the data was protected after discovery, multiple options could have been considered.

*Example Language: Other* *(Good)*

The sensitive information could have been protected by ensuring that the buckets did not have public read access, e.g., by enabling the s3-account-level-public-access-blocks-periodic rule to Block Public Access. In addition, the data could have been encrypted at rest using the appropriate S3 settings, e.g., by enabling server-side encryption using the s3-bucket-server-side-encryption-enabled setting. Other settings are available to further prevent bucket data from being leaked. [REF-1297]

**Example 7:**

Consider the following PowerShell command examples for encryption scopes of Azure storage objects. In the first example, an encryption scope is set for the storage account.

*Example Language: Shell* *(Bad)*

```
New-AzStorageEncryptionScope -ResourceGroupName "MyResourceGroup" -AccountName "MyStorageAccount" -
EncryptionScopeName testscope -StorageEncryption
```

The result (edited and formatted for readability) might be:

*Example Language: Other* *(Bad)*

```
ResourceGroupName: MyResourceGroup, StorageAccountName: MyStorageAccount
```

However, the empty string under RequireInfrastructureEncryption indicates this service was not enabled at the time of creation, because the -RequireInfrastructureEncryption argument was not specified in the command.

Including the -RequireInfrastructureEncryption argument addresses the issue:

*Example Language: Shell* *(Good)*

```
New-AzStorageEncryptionScope -ResourceGroupName "MyResourceGroup" -AccountName "MyStorageAccount" -
EncryptionScopeName testscope -StorageEncryption -RequireInfrastructureEncryption
```

This produces the report:

*Example Language: Other* *(Result)*

```
ResourceGroupName: MyResourceGroup, StorageAccountName: MyStorageAccount
```

In a scenario where both software and hardware layer encryption is required ("double encryption"), Azure's infrastructure encryption setting can be enabled via the CLI or Portal. An important note is that infrastructure hardware encryption cannot be enabled or disabled after a blob is created. Furthermore, the default value for infrastructure encryption is disabled in blob creations.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-30275** | Remote Terminal Unit (RTU) uses a driver that relies on a password stored in plaintext. *https://www.cve.org/CVERecord?id=CVE-2022-30275* |
| **CVE-2009-2272** | password and username stored in cleartext in a cookie *https://www.cve.org/CVERecord?id=CVE-2009-2272* |
| **CVE-2009-1466** | password stored in cleartext in a file with insecure permissions *https://www.cve.org/CVERecord?id=CVE-2009-1466* |
| **CVE-2009-0152** | chat program disables SSL in some circumstances even when the user says to use SSL. *https://www.cve.org/CVERecord?id=CVE-2009-0152* |
| **CVE-2009-1603** | Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption *https://www.cve.org/CVERecord?id=CVE-2009-1603* |
| **CVE-2009-0964** | storage of unencrypted passwords in a database *https://www.cve.org/CVERecord?id=CVE-2009-0964* |
| **CVE-2008-6157** | storage of unencrypted passwords in a database *https://www.cve.org/CVERecord?id=CVE-2008-6157* |
| **CVE-2008-6828** | product stores a password in cleartext in memory *https://www.cve.org/CVERecord?id=CVE-2008-6828* |
| **CVE-2008-1567** | storage of a secret key in cleartext in a temporary file *https://www.cve.org/CVERecord?id=CVE-2008-1567* |
| **CVE-2008-0174** | SCADA product uses HTTP Basic Authentication, which is not encrypted *https://www.cve.org/CVERecord?id=CVE-2008-0174* |
| **CVE-2007-5778** | login credentials stored unencrypted in a registry key *https://www.cve.org/CVERecord?id=CVE-2007-5778* |
| **CVE-2001-1481** | Plaintext credentials in world-readable file. *https://www.cve.org/CVERecord?id=CVE-2001-1481* |
| **CVE-2005-1828** | Password in cleartext in config file. *https://www.cve.org/CVERecord?id=CVE-2005-1828* |
| **CVE-2005-2209** | Password in cleartext in config file. *https://www.cve.org/CVERecord?id=CVE-2005-2209* |

| Reference | Description |
|---|---|
| **CVE-2002-1696** | Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message. *https://www.cve.org/CVERecord?id=CVE-2002-1696* |
| **CVE-2004-2397** | Plaintext storage of private key and passphrase in log file when user imports the key. *https://www.cve.org/CVERecord?id=CVE-2004-2397* |
| **CVE-2002-1800** | Admin password in plaintext in a cookie. *https://www.cve.org/CVERecord?id=CVE-2002-1800* |
| **CVE-2001-1537** | Default configuration has cleartext usernames/passwords in cookie. *https://www.cve.org/CVERecord?id=CVE-2001-1537* |
| **CVE-2001-1536** | Usernames/passwords in cleartext in cookies. *https://www.cve.org/CVERecord?id=CVE-2001-1536* |
| **CVE-2005-2160** | Authentication information stored in cleartext in a cookie. *https://www.cve.org/CVERecord?id=CVE-2005-2160* |

## 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 | 816 | OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage | 809 | 2380 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1368 | ICS Dependencies (& Architecture): External Digital Systems | 1358 | 2526 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Notes

### Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Plaintext Storage of Sensitive Information |
| Software Fault Patterns | SFP23 | | Exposed Data |
| ISA/IEC 62443 | Part 4-2 | | Req CR 4.1 a) |
| ISA/IEC 62443 | Part 3-3 | | Req SR 4.1 |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 37 | Retrieve Embedded Sensitive Data |

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

[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-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-1295]WizCase. "Over 80 US Municipalities' Sensitive Information, Including Resident's Personal Data, Left Vulnerable in Massive Data Breach". 2021 July 0. < https://www.wizcase.com/blog/us-municipality-breach-report/ >.

[REF-1296]Jonathan Greig. "1,000 GB of local government data exposed by Massachusetts software company". 2021 July 2. < https://www.zdnet.com/article/1000-gb-of-local-government-data-exposed-by-massachusetts-software-company/ >.

[REF-1297]Amazon. "AWS Foundational Security Best Practices controls". 2022. < https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-controls-reference.html >.2023-04-07.

[REF-1299]Microsoft. "Azure encryption overview". 2022 August 8. < https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview >.2022-10-11.

[REF-1301]Google Cloud. "Default encryption at rest". 2022 October 1. < https://cloud.google.com/docs/security/encryption/default-encryption >.2022-10-11.

[REF-1307]Center for Internet Security. "CIS Microsoft Azure Foundations Benchmark version 1.5.0". 2022 August 6. < https://www.cisecurity.org/benchmark/azure >.2023-01-19.

[REF-1310]Microsoft. "Enable infrastructure encryption for double encryption of data". 2022 July 4. < https://learn.microsoft.com/en-us/azure/storage/common/infrastructure-encryption-enable >.2023-01-24.

## CWE-313: Cleartext Storage in a File or on Disk

**Weakness ID :** 313
**Structure :** Simple
**Abstraction :** Variant

### Description

The product stores sensitive information in cleartext in a file, or on disk.

### Extended Description

The sensitive information could be read by attackers with access to the file, or with physical or administrator access to the raw disk. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 312 | Cleartext Storage of Sensitive Information | 771 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data | |

## 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 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.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1481** | Cleartext credentials in world-readable file. *https://www.cve.org/CVERecord?id=CVE-2001-1481* |
| **CVE-2005-1828** | Password in cleartext in config file. *https://www.cve.org/CVERecord?id=CVE-2005-1828* |
| **CVE-2005-2209** | Password in cleartext in config file. *https://www.cve.org/CVERecord?id=CVE-2005-2209* |
| **CVE-2002-1696** | Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message. *https://www.cve.org/CVERecord?id=CVE-2002-1696* |
| **CVE-2004-2397** | Cleartext storage of private key and passphrase in log file when user imports the key. *https://www.cve.org/CVERecord?id=CVE-2004-2397* |

### 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

### Notes

**Terminology**

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Plaintext Storage in File or on Disk |
| Software Fault Patterns | SFP23 | | Exposed Data |

## CWE-314: Cleartext Storage in the Registry

**Weakness ID :** 314
**Structure :** Simple
**Abstraction :** Variant

### Description

The product stores sensitive information in cleartext in the registry.

### Extended Description

Attackers can read the information by accessing the registry key. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 312 | Cleartext Storage of Sensitive Information | 771 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data | |

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-2227** | Cleartext passwords in registry key.<br>*https://www.cve.org/CVERecord?id=CVE-2005-2227* |

## 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Notes

### Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Plaintext Storage in Registry |
| Software Fault Patterns | SFP23 | | Exposed Data |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 37 | Retrieve Embedded Sensitive Data |

# CWE-315: Cleartext Storage of Sensitive Information in a Cookie

**Weakness ID :** 315
**Structure :** Simple
**Abstraction :** Variant

## Description

The product stores sensitive information in cleartext in a cookie.

## Extended Description

Attackers can use widely-available tools to view the cookie and read the sensitive information. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 312 | Cleartext Storage of Sensitive Information | 771 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Demonstrative Examples

### Example 1:

The following code excerpt stores a plaintext user account ID in a browser cookie.

*Example Language: Java* *(Bad)*

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Because the account ID is in plaintext, the user's account information is exposed if their computer is compromised by an attacker.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1800** | Admin password in cleartext in a cookie. *https://www.cve.org/CVERecord?id=CVE-2002-1800* |
| **CVE-2001-1537** | Default configuration has cleartext usernames/passwords in cookie. *https://www.cve.org/CVERecord?id=CVE-2001-1537* |
| **CVE-2001-1536** | Usernames/passwords in cleartext in cookies. *https://www.cve.org/CVERecord?id=CVE-2001-1536* |
| **CVE-2005-2160** | Authentication information stored in cleartext in a cookie. *https://www.cve.org/CVERecord?id=CVE-2005-2160* |

### 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1349 | OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration | 1344 | 2514 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

### Notes

#### Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Plaintext Storage in Cookie |
| Software Fault Patterns | SFP23 | | Exposed Data |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 31 | Accessing/Intercepting/Modifying HTTP Cookies |
| 37 | Retrieve Embedded Sensitive Data |
| 39 | Manipulating Opaque Client-based Data Tokens |
| 74 | Manipulating State |

## CWE-316: Cleartext Storage of Sensitive Information in Memory

**Weakness ID :** 316
**Structure :** Simple
**Abstraction :** Variant

### Description

The product stores sensitive information in cleartext in memory.

### Extended Description

The sensitive memory might be saved to disk, stored in a core dump, or remain uncleared if the product crashes, or if the programmer does not properly clear the memory before freeing it.

It could be argued that such problems are usually only exploitable by those with administrator privileges. However, swapping could cause the memory to be written to disk and leave it accessible to physical attack afterwards. Core dump files might have insecure permissions or be stored in archive files that are accessible to untrusted people. Or, uncleared sensitive memory might be inadvertently exposed to attackers due to another 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 | B | 312 | Cleartext Storage of Sensitive Information | 771 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Memory | |

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1517** | Sensitive authentication information in cleartext in memory.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1517* |
| **CVE-2001-0984** | Password protector leaves passwords in memory when window is minimized, even when "clear password when minimized" is set.<br>*https://www.cve.org/CVERecord?id=CVE-2001-0984* |
| **CVE-2003-0291** | SSH client does not clear credentials from memory.<br>*https://www.cve.org/CVERecord?id=CVE-2003-0291* |

## 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1348 | OWASP Top Ten 2021 Category A04:2021 - Insecure Design | 1344 | 2512 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Notes

### Relationship

This could be a resultant weakness, e.g. if the compiler removes code that was intended to wipe memory.

### Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Plaintext Storage in Memory |
| Software Fault Patterns | SFP23 | | Exposed Data |



# CWE-317: Cleartext Storage of Sensitive Information in GUI

**Weakness ID :** 317
**Structure :** Simple
**Abstraction :** Variant

## Description

The product stores sensitive information in cleartext within the GUI.

## Extended Description

An attacker can often obtain data from a GUI, even if hidden, by using an API to directly access GUI objects such as windows and menus. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 312 | Cleartext Storage of Sensitive Information | 771 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

**Operating_System** : Windows *(Prevalence = Sometimes)*

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Memory<br>Read Application Data | |

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1848** | Unencrypted passwords stored in GUI dialog may allow local users to access the passwords. |

| Reference | Description |
| --- | --- |
| | *https://www.cve.org/CVERecord?id=CVE-2002-1848* |

### 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

### Notes

#### Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
| --- | --- | --- | --- |
| PLOVER | | | Plaintext Storage in GUI |
| Software Fault Patterns | SFP23 | | Exposed Data |

## CWE-318: Cleartext Storage of Sensitive Information in Executable

**Weakness ID :** 318
**Structure :** Simple
**Abstraction :** Variant

### Description

The product stores sensitive information in cleartext in an executable.

### Extended Description

Attackers can reverse engineer binary code to obtain secret data. This is especially easy when the cleartext is plain ASCII. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 312 | Cleartext Storage of Sensitive Information | 771 |

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

| Nature | Type | ID | Name | Page |
| --- | --- | --- | --- | --- |
| MemberOf | C | 1013 | Encrypt Data | 2449 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data | |

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2005-1794** | Product stores RSA private key in a DLL and uses it to sign a certificate, allowing spoofing of servers and Adversary-in-the-Middle (AITM) attacks. *https://www.cve.org/CVERecord?id=CVE-2005-1794* |
| **CVE-2001-1527** | administration passwords in cleartext in executable *https://www.cve.org/CVERecord?id=CVE-2001-1527* |

### 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 | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

### Notes

**Terminology**

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Plaintext Storage in Executable |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 37 | Retrieve Embedded Sensitive Data |
| 65 | Sniff Application Code |

## CWE-319: Cleartext Transmission of Sensitive Information

**Weakness ID :** 319
**Structure :** Simple
**Abstraction :** Base

### Description

The product transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.

### Extended Description

Many communication channels can be "sniffed" (monitored) by adversaries during data transmission. For example, in networking, packets can traverse many intermediary nodes from the source to the destination, whether across the internet, an internal network, the cloud, etc. Some actors might have privileged access to a network interface or any link along the channel, such as a router, but they might not be authorized to collect the underlying data. As a result, network traffic could be sniffed by adversaries, spilling security-critical data.

Applicable communication channels are not limited to software products. Applicable channels include hardware-specific technologies such as internal hardware networks and external debug channels, supporting remote JTAG debugging. When mitigations are not applied to combat adversaries within the product's threat model, this weakness significantly lowers the difficulty of exploitation by such adversaries.

When full communications are recorded or logged, such as with a packet dump, an adversary could attempt to obtain the dump long after the transmission has occurred and try to "sniff" the cleartext from the recorded communications in the dump itself. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 311 | Missing Encryption of Sensitive Data | 764 |
| ParentOf | V | 5 | J2EE Misconfiguration: Data Transmission Without Encryption | 1 |
| ParentOf | V | 614 | Sensitive Cookie in HTTPS Session Without 'Secure' Attribute | 1382 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 311 | Missing Encryption of Sensitive Data | 764 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 199 | Information Management Errors | 2333 |

## Applicable Platforms

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

**Technology** : Cloud Computing *(Prevalence = Undetermined)*

**Technology** : Mobile *(Prevalence = Undetermined)*

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

**Technology** : System on Chip *(Prevalence = Undetermined)*

**Technology** : Test/Debug Hardware *(Prevalence = Often)*

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Confidentiality | Read Application Data<br>Modify Files or Directories<br><br>*Anyone can read the information by gaining access to the channel being used for communication.* | |

## Detection Methods

### Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process, trigger the feature that sends the data, and look for the presence or absence of common cryptographic functions in the call tree. Monitor the network and determine if the data packets contain readable commands. Tools exist for detecting if certain encodings are in use. If the traffic contains high entropy, this might indicate the usage of encryption.

### 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 transmitting, encrypt the data using reliable, confidentiality-protecting cryptographic protocols.

### Phase: Implementation

When using web applications with SSL, use SSL for the entire session from login to logout, not just for the initial login page.

### Phase: Implementation

When designing hardware platforms, ensure that approved encryption algorithms (such as those recommended by NIST) protect paths from security critical data to trusted user applications.

### Phase: Testing

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

### Phase: Operation

Configure servers to use encrypted channels for communication, which may include SSL or other secure protocols.

## Demonstrative Examples

### Example 1:

The following code attempts to establish a connection to a site to communicate sensitive information.

*Example Language: Java* *(Bad)*

```
try {
  URL u = new URL("http://www.secret.example.org/");
  HttpURLConnection hu = (HttpURLConnection) u.openConnection();
  hu.setRequestMethod("PUT");
  hu.connect();
  OutputStream os = hu.getOutputStream();
  hu.disconnect();
}
catch (IOException e) {
  //...
}
```

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

**Example 2:**

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 cleartext transmission of sensitive information in their OT products.

**Example 3:**

A TAP accessible register is read/written by a JTAG based tool, for internal use by authorized users. However, an adversary can connect a probing device and collect the values from the unencrypted channel connecting the JTAG interface to the authorized user, if no additional protections are employed.

**Example 4:**

The following Azure CLI command lists the properties of a particular storage account:

*Example Language: Shell* *(Informative)*

```
az storage account show -g {ResourceGroupName} -n {StorageAccountName}
```

The JSON result might be:

*Example Language: JSON* *(Bad)*

```
{
  "name": "{StorageAccountName}",
  "enableHttpsTrafficOnly": false,
  "type": "Microsoft.Storage/storageAccounts"
}
```

The enableHttpsTrafficOnly value is set to false, because the default setting for Secure transfer is set to Disabled. This allows cloud storage resources to successfully connect and transfer data without the use of encryption (e.g., HTTP, SMB 2.1, SMB 3.0, etc.).

Azure's storage accounts can be configured to only accept requests from secure connections made over HTTPS. The secure transfer setting can be enabled using Azure's Portal (GUI) or programmatically by setting the enableHttpsTrafficOnly property to True on the storage account, such as:

*Example Language: Shell* *(Good)*

```
az storage account update -g {ResourceGroupName} -n {StorageAccountName} --https-only true
```

The change can be confirmed from the result by verifying that the enableHttpsTrafficOnly value is true:

*Example Language: JSON* *(Good)*

```
{
  "name": "{StorageAccountName}",
  "enableHttpsTrafficOnly": true,
  "type": "Microsoft.Storage/storageAccounts"
}
```

Note: to enable secure transfer using Azure's Portal instead of the command line:

1. Open the Create storage account pane in the Azure portal.
2. In the Advanced page, select the Enable secure transfer checkbox.

## Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2022-29519** | Programmable Logic Controller (PLC) sends sensitive information in plaintext, including passwords and session tokens. *https://www.cve.org/CVERecord?id=CVE-2022-29519* |
| **CVE-2022-30312** | Building Controller uses a protocol that transmits authentication credentials in plaintext. *https://www.cve.org/CVERecord?id=CVE-2022-30312* |
| **CVE-2022-31204** | Programmable Logic Controller (PLC) sends password in plaintext. *https://www.cve.org/CVERecord?id=CVE-2022-31204* |
| **CVE-2002-1949** | Passwords transmitted in cleartext. *https://www.cve.org/CVERecord?id=CVE-2002-1949* |
| **CVE-2008-4122** | Chain: Use of HTTPS cookie without "secure" flag causes it to be transmitted across unencrypted HTTP. *https://www.cve.org/CVERecord?id=CVE-2008-4122* |
| **CVE-2008-3289** | Product sends password hash in cleartext in violation of intended policy. *https://www.cve.org/CVERecord?id=CVE-2008-3289* |
| **CVE-2008-4390** | Remote management feature sends sensitive information including passwords in cleartext. *https://www.cve.org/CVERecord?id=CVE-2008-4390* |
| **CVE-2007-5626** | Backup routine sends password in cleartext in email. *https://www.cve.org/CVERecord?id=CVE-2007-5626* |
| **CVE-2004-1852** | Product transmits Blowfish encryption key in cleartext. *https://www.cve.org/CVERecord?id=CVE-2004-1852* |
| **CVE-2008-0374** | Printer sends configuration information, including administrative password, in cleartext. *https://www.cve.org/CVERecord?id=CVE-2008-0374* |
| **CVE-2007-4961** | Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294). *https://www.cve.org/CVERecord?id=CVE-2007-4961* |
| **CVE-2007-4786** | Product sends passwords in cleartext to a log server. *https://www.cve.org/CVERecord?id=CVE-2007-4786* |
| **CVE-2005-3140** | Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes. *https://www.cve.org/CVERecord?id=CVE-2005-3140* |

## 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 | 751 | 2009 Top 25 - Insecure Interaction Between Components | 750 | 2373 |
| MemberOf | C | 818 | OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection | 809 | 2381 |
| MemberOf | C | 858 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER) | 844 | 2390 |
| MemberOf | C | 859 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) | 844 | 2390 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 2421 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | C | 1148 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER) | 1133 | 2472 |
| MemberOf | C | 1207 | Debug and Test Problems | 1194 | 2495 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## 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 |
|---|---|---|---|
| PLOVER | | | Plaintext Transmission of Sensitive Information |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC06-J | | Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar |
| The CERT Oracle Secure Coding Standard for Java (2011) | SER02-J | | Sign then seal sensitive objects before sending them outside a trust boundary |
| Software Fault Patterns | SFP23 | | Exposed Data |
| ISA/IEC 62443 | Part 3-3 | | Req SR 4.1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 4.1B |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 65 | Sniff Application Code |
| 102 | Session Sidejacking |

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 117 | Interception |
| 383 | Harvesting Information via API Event Monitoring |
| 477 | Signature Spoofing by Mixing Signed and Unsigned Content |

### References

[REF-271]OWASP. "Top 10 2007-Insecure Communications". 2007. < http://www.owasp.org/index.php/Top_10_2007-A9 >.

[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-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-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-1307]Center for Internet Security. "CIS Microsoft Azure Foundations Benchmark version 1.5.0". 2022 August 6. < https://www.cisecurity.org/benchmark/azure >.2023-01-19.

[REF-1309]Microsoft. "Require secure transfer to ensure secure connections". 2022 July 4. < https://learn.microsoft.com/en-us/azure/storage/common/storage-require-secure-transfer >.2023-01-24.

## CWE-321: Use of Hard-coded Cryptographic Key

**Weakness ID :** 321
**Structure :** Simple
**Abstraction :** Variant

### Description

The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.

### Relationships

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 | 798 | Use of Hard-coded Credentials | 1699 |
| PeerOf | V | 259 | Use of Hard-coded Password | 630 |
| PeerOf | B | 1291 | Public Key Re-Use for Signing both Debug and Production Code | 2157 |
| CanFollow | C | 656 | Reliance on Security Through Obscurity | 1452 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | B | 798 | Use of Hard-coded Credentials | 1699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | B | 798 | Use of Hard-coded Credentials | 1699 |

## Applicable Platforms

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

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>*If hard-coded cryptographic keys are used, it is almost certain that malicious users will gain access through the account in question.* | |

## 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

Prevention schemes mirror that of hard-coded password storage.

## Demonstrative Examples

### Example 1:

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 2:**

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 keys for critical functionality in their OT products.

## Observed Examples

| Reference | Description |
|---|---|
| **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-2022-30271** | Remote Terminal Unit (RTU) uses a hard-coded SSH private key that is likely to be used by default. *https://www.cve.org/CVERecord?id=CVE-2022-30271* |
| **CVE-2020-10884** | WiFi router service has a hard-coded encryption key, allowing root access *https://www.cve.org/CVERecord?id=CVE-2020-10884* |
| **CVE-2014-2198** | Communications / collaboration product has a hardcoded SSH private key, allowing access to root account *https://www.cve.org/CVERecord?id=CVE-2014-2198* |

## 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 | 719 | OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage | 629 | 2354 |
| MemberOf | C | 720 | OWASP Top Ten 2007 Category A9 - Insecure Communications | 629 | 2354 |
| MemberOf | C | 729 | OWASP Top Ten 2004 Category A8 - Insecure Storage | 711 | 2359 |
| MemberOf | C | 950 | SFP Secondary Cluster: Hardcoded Sensitive Data | 888 | 2417 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |

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

## Notes

### Other

The main difference between the use of hard-coded passwords and the use of hard-coded cryptographic keys is the false sense of security that the former conveys. Many people believe that simply hashing a hard-coded password before storage will protect the information from malicious users. However, many hashes are reversible (or at least vulnerable to brute force attacks) -- and further, many authentication protocols simply request the hash itself, making it no better than a password.

### Maintenance

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

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Use of hard-coded cryptographic key |
| OWASP Top Ten 2007 | A8 | CWE More Specific | Insecure Cryptographic Storage |
| OWASP Top Ten 2007 | A9 | CWE More Specific | Insecure Communications |
| OWASP Top Ten 2004 | A8 | CWE More Specific | Insecure Storage |
| Software Fault Patterns | SFP33 | | Hardcoded sensitive data |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.10 RE(1) |
| ISA/IEC 62443 | Part 2-4 | | Req SP.03.10 RE(3) |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.5 |
| ISA/IEC 62443 | Part 3-3 | | Req SR 4.3 |
| ISA/IEC 62443 | Part 4-1 | | Req SD-1 |
| ISA/IEC 62443 | Part 4-2 | | Req SR 4.3 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 7.3 |

## References

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

[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/ >.

# CWE-322: Key Exchange without Entity Authentication

**Weakness ID :** 322
**Structure :** Simple
**Abstraction :** Base

## Description

The product performs a key exchange with an actor without verifying the identity of that actor.

## Extended Description

Performing a key exchange will preserve the integrity of the information sent between two entities, but this will not guarantee that the entities are who they claim they are. This may enable an attacker to impersonate an actor by modifying traffic between the two entities. Typically, this involves a victim client that contacts a malicious server that is impersonating a trusted server. If the client skips authentication or ignores an authentication failure, the malicious server may request authentication information from the user. The malicious server can then use this authentication information to log in to the trusted server using the victim's credentials, sniff traffic between the victim and trusted server, 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 | B | 306 | Missing Authentication for Critical Function | 748 |
| PeerOf | B | 295 | Improper Certificate Validation | 721 |
| PeerOf | B | 295 | Improper Certificate Validation | 721 |
| CanPrecede | C | 923 | Improper Restriction of Communication Channel to Intended Endpoints | 1836 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1010 | Authenticate Actors | 2445 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1211 | Authentication Errors | 2496 |
| MemberOf | C | 1214 | Data Integrity Issues | 2498 |
| MemberOf | C | 320 | Key Management Errors | 2340 |
| MemberOf | C | 417 | Communication Channel Errors | 2347 |

## Applicable Platforms

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>*No authentication takes place in this process, bypassing an assumed protection of encryption.* | |
| Confidentiality | Read Application Data<br>*The encrypted communication between a user and a trusted host may be subject to sniffing by any actor in the communication path.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Ensure that proper authentication is included in the system design.

**Phase: Implementation**

Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

### Demonstrative Examples

**Example 1:**

Many systems have used Diffie-Hellman key exchange without authenticating the entities exchanging keys, allowing attackers to influence communications by redirecting or interfering with the communication path. Many people using SSL/TLS skip the authentication (often unknowingly).

### MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1396 | Comprehensive Categorization: Access Control | 1400 | 2540 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Key exchange without entity authentication |

### References

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

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

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

## CWE-323: Reusing a Nonce, Key Pair in Encryption

**Weakness ID :** 323
**Structure :** Simple
**Abstraction :** Base

### Description

Nonces should be used for the present occasion and only once.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 856 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 320 | Key Management Errors | 2340 |

## Applicable Platforms

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

## Background Details

Nonces are often bundled with a key in a communication exchange to produce a new session key for each exchange.

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>*Potentially a replay attack, in which an attacker could send the same data twice, could be crafted if nonces are allowed to be reused. This could allow a user to send a message which masquerades as a valid message from a valid user.* | |

## Potential Mitigations

**Phase: Implementation**

Refuse to reuse nonce values.

**Phase: Implementation**

Use techniques such as requiring incrementing, time based and/or challenge response to assure uniqueness of nonces.

## Demonstrative Examples

**Example 1:**

This code takes a password, concatenates it with a nonce, then encrypts it before sending over a network:

*Example Language: C* *(Bad)*

```
void encryptAndSendPassword(char *password){
  char *nonce = "bad";
  ...
  char *data = (unsigned char*)malloc(20);
  int para_size = strlen(nonce) + strlen(password);
  char *paragraph = (char*)malloc(para_size);
  SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);
  sendEncryptedData(data)
}
```

Because the nonce used is always the same, an attacker can impersonate a trusted party by intercepting and resending the encrypted password. This attack avoids the need to learn the unencrypted password.

**Example 2:**

This code sends a command to a remote server, using an encrypted password and nonce to prove the command is from a trusted party:

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

```
String command = new String("some command to execute");
MessageDigest nonce = MessageDigest.getInstance("SHA");
nonce.update(String.valueOf("bad nonce"));
byte[] nonce = nonce.digest();
MessageDigest password = MessageDigest.getInstance("SHA");
password.update(nonce + "secretPassword");
byte[] digest = password.digest();
sendCommand(digest, command)
```

Once again the nonce used is always the same. An attacker may be able to replay previous legitimate commands or execute new arbitrary commands.

### MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Reusing a nonce, key pair in encryption |

### References

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

## CWE-324: Use of a Key Past its Expiration Date

**Weakness ID :** 324
**Structure :** Simple
**Abstraction :** Base

### Description

The product uses a cryptographic key or password past its expiration date, which diminishes its safety significantly by increasing the timing window for cracking attacks against that key.

### Extended Description

While the expiration of keys does not necessarily ensure that they are compromised, it is a significant concern that keys which remain in use for prolonged periods of time have a decreasing probability of integrity. For this reason, it is important to replace keys within a period of time proportional to their strength.

### Relationships

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 | 672 | Operation on a Resource after Expiration or Release | 1488 |
| PeerOf | V | 298 | Improper Validation of Certificate Expiration | 733 |
| PeerOf | B | 262 | Not Using Password Aging | 640 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 255 | Credentials Management Errors | 2336 |
| MemberOf | C | 310 | Cryptographic Issues | 2339 |
| MemberOf | C | 320 | Key Management Errors | 2340 |

## Applicable Platforms

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

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>*The cryptographic key in question may be compromised, providing a malicious user with a method for authenticating as the victim.* | |

## Potential Mitigations

**Phase: Architecture and Design**

Adequate consideration should be put in to the user interface in order to notify users previous to the key's expiration, to explain the importance of new key generation and to walk users through the process as painlessly as possible.

## Demonstrative Examples

**Example 1:**

The following code attempts to verify that a certificate is valid.

*Example Language: C* *(Bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
  foo=SSL_get_verify_result(ssl);
  if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo))
    //do stuff
}
```

The code checks if the certificate is not yet valid, but it fails to check if a certificate is past its expiration date, thus treating expired certificates as valid.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2021-33020** | Picture Archiving and Communication System (PACS) system for hospitals uses a cryptographic key or password past its expiration date *https://www.cve.org/CVERecord?id=CVE-2021-33020* |

### 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 | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Using a key past its expiration date |

### References

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

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

## CWE-325: Missing Cryptographic Step

**Weakness ID :** 325
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not implement a required step in a cryptographic algorithm, resulting in weaker encryption than advertised by the algorithm.

### Relationships

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 | 573 | Improper Following of Specification by Caller | 1307 |
| PeerOf | B | 358 | Improperly Implemented Security Check for Standard | 888 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

## Applicable Platforms

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

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |
| Confidentiality<br>Integrity | Read Application Data<br>Modify Application Data | |
| Accountability,<br>Non-Repudiation | Hide Activities | |

## Demonstrative Examples

**Example 1:**

The example code is taken from the HMAC engine inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1358]. HAMC is a message authentication code (MAC) that uses both a hash and a secret crypto key. The HMAC engine in HACK@DAC SoC uses the SHA-256 module for the calculation of the HMAC for 512 bits messages.

*Example Language: Verilog* *(Bad)*

```
logic [511:0] bigData;
...
hmac hmac(
  .clk_i(clk_i),
  .rst_ni(rst_ni && ~rst_4),
  .init_i(startHash && ~startHash_r),
  .key_i(key),
  .ikey_hash_i(ikey_hash),
  .okey_hash_i(okey_hash),
  .key_hash_bypass_i(key_hash_bypass),
  .message_i(bigData),
  .hash_o(hash),
  .ready_o(ready),
  .hash_valid_o(hashValid)
```

However, this HMAC engine cannot handle messages that are longer than 512 bits. Moreover, a complete HMAC will contain an iterate hash function that breaks up a message into blocks of a fixed size and iterates over them with a compression function (e.g., SHA-256). Therefore, the implementation of the HMAC in OpenPiton SoC is incomplete. Such HMAC engines will not be used in real-world applications as the messages will usually be longer than 512 bits. For instance, OpenTitan offers a comprehensive HMAC implementation that utilizes a FIFO for temporarily storing the truncated message, as detailed in [REF-1359].

To mitigate this, implement the iterative function to break up a message into blocks of a fixed size.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1585** | Missing challenge-response step allows authentication bypass using public key.<br>*https://www.cve.org/CVERecord?id=CVE-2001-1585* |

## Functional Areas

- Cryptography

## 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 | 719 | OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage | 629 | 2354 |
| MemberOf | C | 720 | OWASP Top Ten 2007 Category A9 - Insecure Communications | 629 | 2354 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 958 | SFP Secondary Cluster: Broken Cryptography | 888 | 2419 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | C | 1205 | Security Primitives and Cryptography Issues | 1194 | 2494 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Notes

**Relationship**

Overlaps incomplete/missing security check.

**Relationship**

Can be resultant.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Missing Required Cryptographic Step |
| OWASP Top Ten 2007 | A8 | CWE More Specific | Insecure Cryptographic Storage |
| OWASP Top Ten 2007 | A9 | CWE More Specific | Insecure Communications |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 68 | Subvert Code-signing Facilities |

## References

[REF-1358]"hmac_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/main/piton/design/chip/tile/ariane/src/hmac/hmac_wrapper.sv#L41 >.2023-07-15.

[REF-1359]"HMAC HWIP Technical Specification". 2023. < https://opentitan.org/book/hw/ip/hmac/ >.2023-10-05.

# CWE-326: Inadequate Encryption Strength

**Weakness ID :** 326
**Structure :** Simple
**Abstraction :** Class

## Description

The product stores or transmits sensitive data using an encryption scheme that is theoretically sound, but is not strong enough for the level of protection required.

## Extended Description

A weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of succeeding using current attack methods and resources.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 1529 |
| ParentOf | B | 328 | Use of Weak Hash | 813 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Confidentiality | Bypass Protection Mechanism<br>Read Application Data<br>*An attacker may be able to decrypt the data using brute force attacks.* | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Architecture and Design

Use an encryption scheme that is currently considered to be strong by experts in the field.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1546** | Weak encryption<br>*https://www.cve.org/CVERecord?id=CVE-2001-1546* |
| **CVE-2004-2172** | Weak encryption (chosen plaintext attack)<br>*https://www.cve.org/CVERecord?id=CVE-2004-2172* |
| **CVE-2002-1682** | Weak encryption<br>*https://www.cve.org/CVERecord?id=CVE-2002-1682* |
| **CVE-2002-1697** | Weak encryption produces same ciphertext from the same plaintext blocks.<br>*https://www.cve.org/CVERecord?id=CVE-2002-1697* |
| **CVE-2002-1739** | Weak encryption |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2002-1739* |
| **CVE-2005-2281** | Weak encryption scheme *https://www.cve.org/CVERecord?id=CVE-2005-2281* |
| **CVE-2002-1872** | Weak encryption (XOR) *https://www.cve.org/CVERecord?id=CVE-2002-1872* |
| **CVE-2002-1910** | Weak encryption (reversible algorithm). *https://www.cve.org/CVERecord?id=CVE-2002-1910* |
| **CVE-2002-1946** | Weak encryption (one-to-one mapping). *https://www.cve.org/CVERecord?id=CVE-2002-1946* |
| **CVE-2002-1975** | Encryption error uses fixed salt, simplifying brute force / dictionary attacks (overlaps randomness). *https://www.cve.org/CVERecord?id=CVE-2002-1975* |

## 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 | 719 | OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage | 629 | 2354 |
| MemberOf | C | 720 | OWASP Top Ten 2007 Category A9 - Insecure Communications | 629 | 2354 |
| MemberOf | C | 729 | OWASP Top Ten 2004 Category A8 - Insecure Storage | 711 | 2359 |
| MemberOf | C | 816 | OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage | 809 | 2380 |
| MemberOf | C | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Weak Encryption |
| OWASP Top Ten 2007 | A8 | CWE More Specific | Insecure Cryptographic Storage |
| OWASP Top Ten 2007 | A9 | CWE More Specific | Insecure Communications |
| OWASP Top Ten 2004 | A8 | CWE More Specific | Insecure Storage |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 20 | Encryption Brute Forcing |
| 112 | Brute Force |
| 192 | Protocol Analysis |

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



## CWE-327: Use of a Broken or Risky Cryptographic Algorithm

**Weakness ID :** 327
**Structure :** Simple
**Abstraction :** Class

### Description

The product uses a broken or risky cryptographic algorithm or protocol.

### Extended Description

Cryptographic algorithms are the methods by which data is scrambled to prevent observation or influence by unauthorized actors. Insecure cryptography can be exploited to expose sensitive information, modify data in unexpected ways, spoof identities of other users or devices, or other impacts.

It is very difficult to produce a secure algorithm, and even high-profile algorithms by accomplished cryptographic experts have been broken. Well-known techniques exist to break or weaken various kinds of cryptography. Accordingly, there are a small number of well-understood and heavily studied algorithms that should be used by most products. Using a non-standard or known-insecure algorithm is dangerous because a determined adversary may be able to break the algorithm and compromise whatever data has been protected.

Since the state of cryptography advances so rapidly, it is common for an algorithm to be considered "unsafe" even if it was once thought to be strong. This can happen when new attacks are discovered, or if computing power increases so much that the cryptographic algorithm no longer provides the amount of protection that was originally thought.

For a number of reasons, this weakness is even more challenging to manage with hardware deployment of cryptographic algorithms as opposed to software implementation. First, if a flaw is discovered with hardware-implemented cryptography, the flaw cannot be fixed in most cases without a recall of the product, because hardware is not easily replaceable like software. Second, because the hardware product is expected to work for years, the adversary's computing power will only increase over time.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 1529 |
| ParentOf | B | 328 | Use of Weak Hash | 813 |
| ParentOf | V | 780 | Use of RSA Algorithm without OAEP | 1652 |
| ParentOf | B | 1240 | Use of a Cryptographic Primitive with a Risky Implementation | 2036 |
| PeerOf | C | 311 | Missing Encryption of Sensitive Data | 764 |
| PeerOf | B | 301 | Reflection Attack in an Authentication Protocol | 740 |
| CanFollow | B | 208 | Observable Timing Discrepancy | 537 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 916 | Use of Password Hash With Insufficient Computational Effort | 1822 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

**Language** : Verilog *(Prevalence = Undetermined)*

**Language** : VHDL *(Prevalence = Undetermined)*

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

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

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data<br>*The confidentiality of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.* | |
| Integrity | Modify Application Data<br>*The integrity of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.* | |
| Accountability<br>Non-Repudiation | Hide Activities<br>*If the cryptographic algorithm is used to ensure the identity of the source of the data (such as digital signatures), then a broken algorithm will compromise this scheme and the source of the data cannot be proven.* | |

## Detection Methods

**Automated Analysis**

Automated methods may be useful for recognizing commonly-used libraries or features that have become obsolete.

*Effectiveness = Moderate*

*False negatives may occur if the tool is not aware of the cryptographic libraries in use, or if custom cryptography is being used.*

**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.

**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 Binary / Bytecode simple extractor - strings, ELF readers, etc.

*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: Highly cost effective: Man-in-the-middle attack tool Cost effective for partial coverage: Framework-based Fuzzer Automated Monitored Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

*Effectiveness = High*

**Manual Static Analysis - Source Code**

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

*Effectiveness = High*

**Automated Static Analysis - Source Code**

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

*Effectiveness = High*

**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**

*Strategy = Libraries or Frameworks*

When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis. For example, US government systems require FIPS 140-2 certification [REF-1192]. Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak. Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. [REF-267]

**Phase: Architecture and Design**

Ensure that the design allows one cryptographic algorithm to be replaced with another in the next generation or version. Where possible, use wrappers to make the interfaces uniform. This will make it easier to upgrade to stronger algorithms. With hardware, design the product at the Intellectual Property (IP) level so that one cryptographic algorithm can be replaced with another in the next generation of the hardware product.

*Effectiveness = Defense in Depth*

**Phase: Architecture and Design**

Carefully manage and protect cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography itself is irrelevant.

**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. Industry-standard implementations will save development time and may be more likely to avoid errors that can occur during implementation of cryptographic algorithms. Consider the ESAPI Encryption feature.

**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:**

These code examples use the Data Encryption Standard (DES).

*Example Language: C* *(Bad)*

```
EVP_des_ecb();
```

*Example Language: Java* *(Bad)*

```
Cipher des=Cipher.getInstance("DES...");
des.initEncrypt(key2);
```

*Example Language: PHP* *(Bad)*

```
function encryptPassword($password){
  $iv_size = mcrypt_get_iv_size(MCRYPT_DES, MCRYPT_MODE_ECB);
  $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
  $key = "This is a password encryption key";
  $encryptedPassword = mcrypt_encrypt(MCRYPT_DES, $key, $password, MCRYPT_MODE_ECB, $iv);
  return $encryptedPassword;
}
```

Once considered a strong algorithm, DES now regarded as insufficient for many applications. It has been replaced by Advanced Encryption Standard (AES).

**Example 2:**

Suppose a chip manufacturer decides to implement a hashing scheme for verifying integrity property of certain bitstream, and it chooses to implement a SHA1 hardware accelerator for to implement the scheme.

*Example Language: Other* *(Bad)*

The manufacturer chooses a SHA1 hardware accelerator for to implement the scheme because it already has a working SHA1 Intellectual Property (IP) that the manufacturer had created and used earlier, so this reuse of IP saves design cost.

However, SHA1 was theoretically broken in 2005 and practically broken in 2017 at a cost of $110K. This means an attacker with access to cloud-rented computing power will now be able to provide a malicious bitstream with the same hash value, thereby defeating the purpose for which the hash was used.

This issue could have been avoided with better design.

*Example Language: Other* *(Good)*

The manufacturer could have chosen a cryptographic solution that is recommended by the wide security community (including standard-setting bodies like NIST) and is not expected to be broken (or even better, weakened) within the reasonable life expectancy of the hardware product. In this case, the architects could have used SHA-2 or SHA-3, even if it meant that such choice would cost extra.

**Example 3:**

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 OT products used weak cryptography.

## Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2022-30273** | SCADA-based protocol supports a legacy encryption mode that uses Tiny Encryption Algorithm (TEA) in ECB mode, which leaks patterns in messages and cannot protect integrity<br>*https://www.cve.org/CVERecord?id=CVE-2022-30273* |
| **CVE-2022-30320** | Programmable Logic Controller (PLC) uses a protocol with a cryptographically insecure hashing algorithm for passwords.<br>*https://www.cve.org/CVERecord?id=CVE-2022-30320* |
| **CVE-2008-3775** | Product uses "ROT-25" to obfuscate the password in the registry.<br>*https://www.cve.org/CVERecord?id=CVE-2008-3775* |
| **CVE-2007-4150** | product only uses "XOR" to obfuscate sensitive data<br>*https://www.cve.org/CVERecord?id=CVE-2007-4150* |
| **CVE-2007-5460** | product only uses "XOR" and a fixed key to obfuscate sensitive data<br>*https://www.cve.org/CVERecord?id=CVE-2007-5460* |
| **CVE-2005-4860** | Product substitutes characters with other characters in a fixed way, and also leaves certain input characters unchanged.<br>*https://www.cve.org/CVERecord?id=CVE-2005-4860* |
| **CVE-2002-2058** | Attackers can infer private IP addresses by dividing each octet by the MD5 hash of '20'.<br>*https://www.cve.org/CVERecord?id=CVE-2002-2058* |
| **CVE-2008-3188** | Product uses DES when MD5 has been specified in the configuration, resulting in weaker-than-expected password hashes. |

| Reference | Description |
|---|---|
| | *https://www.cve.org/CVERecord?id=CVE-2008-3188* |
| **CVE-2005-2946** | Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates. *https://www.cve.org/CVERecord?id=CVE-2005-2946* |
| **CVE-2007-6013** | Product uses the hash of a hash for authentication, allowing attackers to gain privileges if they can obtain the original hash. *https://www.cve.org/CVERecord?id=CVE-2007-6013* |

## 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 | 729 | OWASP Top Ten 2004 Category A8 - Insecure Storage | 711 | 2359 |
| MemberOf | C | 753 | 2009 Top 25 - Porous Defenses | 750 | 2374 |
| MemberOf | C | 803 | 2010 Top 25 - Porous Defenses | 800 | 2376 |
| MemberOf | C | 816 | OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage | 809 | 2380 |
| MemberOf | C | 866 | 2011 Top 25 - Porous Defenses | 900 | 2393 |
| MemberOf | C | 883 | CERT C++ Secure Coding Section 49 - Miscellaneous (MSC) | 868 | 2402 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 958 | SFP Secondary Cluster: Broken Cryptography | 888 | 2419 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | C | 1131 | CISQ Quality Measures (2016) - Security | 1128 | 2463 |
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | C | 1170 | SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC) | 1154 | 2484 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Notes

### Maintenance

Since CWE 4.4, various cryptography-related entries, including CWE-327 and CWE-1240, have been slated for extensive research, analysis, and community consultation to define consistent terminology, improve relationships, and reduce overlap or duplication. As of CWE 4.6, this work is still ongoing.

### Maintenance

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

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Using a broken or risky cryptographic algorithm |
| OWASP Top Ten 2004 | A8 | CWE More Specific | Insecure Storage |
| CERT C Secure Coding | MSC30-C | CWE More Abstract | Do not use the rand() function for generating pseudorandom numbers |
| CERT C Secure Coding | MSC32-C | CWE More Abstract | Properly seed pseudorandom number generators |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC02-J | | Generate strong random numbers |
| OMG ASCSM | ASCSM-CWE-327 | | |
| ISA/IEC 62443 | Part 3-3 | | Req SR 4.3 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 4.3 |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 20 | Encryption Brute Forcing |
| 97 | Cryptanalysis |
| 459 | Creating a Rogue Certification Authority Certificate |
| 473 | Signature Spoof |
| 475 | Signature Spoofing by Improper Validation |
| 608 | Cryptanalysis of Cellular Encryption |
| 614 | Rooting SIM Cards |

### References

[REF-280]Bruce Schneier. "Applied Cryptography". 1996. John Wiley & Sons. < https://www.schneier.com/books/applied-cryptography >.2023-04-07.

[REF-281]Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. "Handbook of Applied Cryptography". 1996 October. < https://cacr.uwaterloo.ca/hac/ >.2023-04-07.

[REF-282]C Matthew Curtin. "Avoiding bogus encryption products: Snake Oil FAQ". 1998 April 0. < http://www.faqs.org/faqs/cryptography-faq/snake-oil/ >.

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-284]Paul F. Roberts. "Microsoft Scraps Old Encryption in New Code". 2005 September 5. < https://www.eweek.com/security/microsoft-scraps-old-encryption-in-new-code/ >.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-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-287]Johannes Ullrich. "Top 25 Series - Rank 24 - Use of a Broken or Risky Cryptographic Algorithm". 2010 March 5. SANS Software Security Institute. < https://www.sans.org/blog/top-25-series-use-of-a-broken-or-risky-cryptographic-algorithm/ >.2023-04-07.

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

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

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

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

[REF-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/ >.

## CWE-328: Use of Weak Hash

**Weakness ID :** 328
**Structure :** Simple
**Abstraction :** Base

### Description

The product uses an algorithm that produces a digest (output value) that does not meet security expectations for a hash function that allows an adversary to reasonably determine the original input (preimage attack), find another input that can produce the same hash (2nd preimage attack), or find multiple inputs that evaluate to the same hash (birthday attack).

### Extended Description

A hash function is defined as an algorithm that maps arbitrarily sized data into a fixed-sized digest (output) such that the following properties hold:

1. The algorithm is not invertible (also called "one-way" or "not reversible")
2. The algorithm is deterministic; the same input produces the same digest every time

Building on this definition, a cryptographic hash function must also ensure that a malicious actor cannot leverage the hash function to have a reasonable chance of success at determining any of the following:

1. the original input (preimage attack), given only the digest
2. another input that can produce the same digest (2nd preimage attack), given the original input
3. a set of two or more inputs that evaluate to the same digest (birthday attack), given the actor can arbitrarily choose the inputs to be hashed and can do so a reasonable amount of times

What is regarded as "reasonable" varies by context and threat model, but in general, "reasonable" could cover any attack that is more efficient than brute force (i.e., on average, attempting half of all possible combinations). Note that some attacks might be more efficient than brute force but are still not regarded as achievable in the real world.

Any algorithm that does not meet the above conditions will generally be considered weak for general use in hashing.

In addition to algorithmic weaknesses, a hash function can be made weak by using the hash in a security context that breaks its security guarantees. For example, using a hash function without a salt for storing passwords (that are sufficiently short) could enable an adversary to create a "rainbow table" [REF-637] to recover the password under certain conditions; this attack works against such hash functions as MD5, SHA-1, and SHA-2.

## Relationships

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 | 326 | Inadequate Encryption Strength | 803 |
| ChildOf | C | 327 | Use of a Broken or Risky Cryptographic Algorithm | 806 |
| ParentOf | B | 916 | Use of Password Hash With Insufficient Computational Effort | 1822 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

## Applicable Platforms

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

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: 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*

## Demonstrative Examples

**Example 1:**

In both of these examples, a user is logged in if their given password matches a stored password:

*Example Language: C* *(Bad)*

```
unsigned char *check_passwd(char *plaintext) {
  ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
  //Login if hash matches stored hash
  if (equal(ctext, secret_password())) {
    login_user();
  }
}
```

*Example Language: Java* *(Bad)*

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
  login_user();
}
```

This code relies exclusively on a password mechanism (CWE-309) using only one factor of authentication (CWE-308). If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also uses SHA-1, which is a weak hash (CWE-328). It also does not use a salt (CWE-759).

**Example 2:**

In 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.

At least one OT product used weak hashes.

**Example 3:**

The example code below is taken from the JTAG access control mechanism of the Hack@DAC'21 buggy OpenPiton SoC [REF-1360]. Access to JTAG allows users to access sensitive information in the system. Hence, access to JTAG is controlled using cryptographic authentication of the users. In this example (see the vulnerable code source), the password checker uses HMAC-SHA256 for authentication. It takes a 512-bit secret message from the user, hashes it using HMAC, and compares its output with the expected output to determine the authenticity of the user.

*Example Language: Verilog* *(Bad)*

```
...
logic [31:0] data_d, data_q
logic [512-1:0] pass_data;
```

```
...
   Write: begin
      ...
         if (pass_mode) begin
            pass_data = { {60{8'h00}}, data_d};
            state_d = PassChk;
            pass_mode = 1'b0;
         ...
   end
...
```

The vulnerable code shows an incorrect implementation of the HMAC authentication where it only uses the least significant 32 bits of the secret message for the authentication (the remaining 480 bits are hard coded as zeros). As a result, the system is susceptible to brute-force attacks where the attacker only needs to determine 32 bits of the secret message instead of 512 bits, weakening the cryptographic protocol.

To mitigate, remove the zero padding and use all 512 bits of the secret message for HMAC authentication [REF-1361].

*Example Language: Verilog* *(Good)*

```
...
logic [512-1:0] data_d, data_q
logic [512-1:0] pass_data;
...
   Write: begin
      ...
         if (pass_mode) begin
            pass_data = data_d;
            state_d = PassChk;
            pass_mode = 1'b0;
         ...
   end
...
```

## Observed Examples

| Reference | Description |
| --- | --- |
| **CVE-2022-30320** | Programmable Logic Controller (PLC) uses a protocol with a cryptographically insecure hashing algorithm for passwords. *https://www.cve.org/CVERecord?id=CVE-2022-30320* |
| **CVE-2005-4900** | SHA-1 algorithm is not collision-resistant. *https://www.cve.org/CVERecord?id=CVE-2005-4900* |
| **CVE-2020-25685** | DNS product uses a weak hash (CRC32 or SHA-1) of the query name, allowing attacker to forge responses by computing domain names with the same hash. *https://www.cve.org/CVERecord?id=CVE-2020-25685* |
| **CVE-2012-6707** | blogging product uses MD5-based algorithm for passwords. *https://www.cve.org/CVERecord?id=CVE-2012-6707* |
| **CVE-2019-14855** | forging of certificate signatures using SHA-1 collisions. *https://www.cve.org/CVERecord?id=CVE-2019-14855* |
| **CVE-2017-15999** | mobile app for backup sends SHA-1 hash of password in cleartext. *https://www.cve.org/CVERecord?id=CVE-2017-15999* |
| **CVE-2006-4068** | Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks. *https://www.cve.org/CVERecord?id=CVE-2006-4068* |

## MemberOf Relationships

This MemberOf relationships table shows additional CWE 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 | 934 | OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure | 928 | 2412 |
| MemberOf | C | 958 | SFP Secondary Cluster: Broken Cryptography | 888 | 2419 |
| MemberOf | C | 1029 | OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure | 1026 | 2457 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

## Notes

### Maintenance

Since CWE 4.4, various cryptography-related entries including CWE-328 have been slated for extensive research, analysis, and community consultation to define consistent terminology, improve relationships, and reduce overlap or duplication. As of CWE 4.6, this work is still ongoing.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Reversible One-Way Hash |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 68 | Subvert Code-signing Facilities |
| 461 | Web Services API Signature Forgery Leveraging Hash Function Extension Weakness |

## References

[REF-289]Alexander Sotirov et al.. "MD5 considered harmful today". < http://www.phreedom.org/research/rogue-ca/ >.2023-04-07.

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

[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-637]"Rainbow table". 2009 March 3. Wikipedia. < https://en.wikipedia.org/wiki/Rainbow_table >.2023-04-07.

[REF-1243]Bruce Schneier. "Cryptanalysis of SHA-1". 2005 February 8. < https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html >.2021-10-25.

[REF-1244]Dan Goodin. "At death's door for years, widely used SHA1 function is now dead". 2017 February 3. Ars Technica. < https://arstechnica.com/information-technology/2017/02/at-deaths-door-for-years-widely-used-sha1-function-is-now-dead/ >.2021-10-25.

[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-1360]"dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L82 >.2023-07-15.

[REF-1361]"fix cwe_1205 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/c4f4b832218b50c406dbf9f425d3b654117c1355/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L82 >.2023-07-22.

# CWE-329: Generation of Predictable IV with CBC Mode

**Weakness ID :** 329
**Structure :** Simple
**Abstraction :** Variant

## Description

The product generates and uses a predictable initialization Vector (IV) with Cipher Block Chaining (CBC) Mode, which causes algorithms to be susceptible to dictionary attacks when they are encrypted under the same key.

## Extended Description

CBC mode eliminates a weakness of Electronic Code Book (ECB) mode by allowing identical plaintext blocks to be encrypted to different ciphertext blocks. This is possible by the XOR-ing of an IV with the initial plaintext block so that every plaintext block in the chain is XOR'd with a different value before encryption. If IVs are reused, then identical plaintexts would be encrypted to identical ciphertexts. However, even if IVs are not identical but are predictable, then they still break the security of CBC mode against Chosen Plaintext Attacks (CPA).

## Relationships

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 | 573 | Improper Following of Specification by Caller | 1307 |
| ChildOf | B | 1204 | Generation of Weak Initialization Vector (IV) | 1996 |

## Applicable Platforms

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

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

## Background Details

CBC mode is a commonly used mode of operation for a block cipher. It works by XOR-ing an IV with the initial block of a plaintext prior to encryption and then XOR-ing each successive block of plaintext with the previous block of ciphertext before encryption.

C_0 = IV
C_i = E_k{M_i XOR C_{i-1}}

When used properly, CBC mode provides security against chosen plaintext attacks. Having an unpredictable IV is a crucial underpinning of this. See [REF-1171].

## Likelihood Of Exploit

Medium

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality | Read Application Data<br>*If the IV is not properly initialized, data that is encrypted can be compromised and leak information.* | |

## 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

NIST recommends two methods of generating unpredictable IVs for CBC mode [REF-1172]. The first is to generate the IV randomly. The second method is to encrypt a nonce with the same key and cipher to be used to encrypt the plaintext. In this case the nonce must be unique but can be predictable, since the block cipher will act as a pseudo random permutation.

## Demonstrative Examples

### Example 1:

In the following examples, CBC mode is used when encrypting data:

*Example Language: C* *(Bad)*

```
EVP_CIPHER_CTX ctx;
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv,0,EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx,EVP_bf_cbc(), key,iv);
```

*Example Language: Java* *(Bad)*

```
public class SymmetricCipherTest {
  public static void main() {
    byte[] text ="Secret".getBytes();
    byte[] iv ={
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
    };
    KeyGenerator kg = KeyGenerator.getInstance("DES");
    kg.init(56);
    SecretKey key = kg.generateKey();
    Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
    IvParameterSpec ips = new IvParameterSpec(iv);
    cipher.init(Cipher.ENCRYPT_MODE, key, ips);
    return cipher.doFinal(inpBytes);
  }
}
```

In both of these examples, the initialization vector (IV) is always a block of zeros. This makes the resulting cipher text much more predictable and susceptible to a dictionary attack.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2020-5408** | encryption functionality in an authentication framework uses a fixed null IV with CBC mode, allowing attackers to decrypt traffic in applications that use this functionality<br>*https://www.cve.org/CVERecord?id=CVE-2020-5408* |
| **CVE-2017-17704** | messages for a door-unlocking product use a fixed IV in CBC mode, which is the same after each restart<br>*https://www.cve.org/CVERecord?id=CVE-2017-17704* |
| **CVE-2017-11133** | application uses AES in CBC mode, but the pseudo-random secret and IV are generated using math.random, which is not cryptographically strong.<br>*https://www.cve.org/CVERecord?id=CVE-2017-11133* |
| **CVE-2007-3528** | Blowfish-CBC implementation constructs an IV where each byte is calculated modulo 8 instead of modulo 256, resulting in less than 12 bits for the effective IV length, and less than 4096 possible IV values.<br>*https://www.cve.org/CVERecord?id=CVE-2007-3528* |
| **CVE-2011-3389** | BEAST attack in SSL 3.0 / TLS 1.0. In CBC mode, chained initialization vectors are non-random, allowing decryption of HTTPS traffic using a chosen plaintext attack.<br>*https://www.cve.org/CVERecord?id=CVE-2011-3389* |

## Functional Areas

- Cryptography

## 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 | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1370 | ICS Supply Chain: Common Mode Frailties | 1358 | 2528 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Not using a random IV with CBC mode |

### References

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

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

[REF-1171]Matthew Green. "Why IND-CPA implies randomized encryption". 2018 August 4. < https://blog.cryptographyengineering.com/why-ind-cpa-implies-randomized-encryption/ >.

[REF-1172]NIST. "Recommendation for Block Cipher Modes of Operation". 2001 December. < https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf >.2023-04-07.

## CWE-330: Use of Insufficiently Random Values

**Weakness ID :** 330
**Structure :** Simple
**Abstraction :** Class

### Description

The product uses insufficiently random numbers or values in a security context that depends on unpredictable numbers.

### Extended Description

When product generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 1529 |
| ParentOf | B | 331 | Insufficient Entropy | 828 |
| ParentOf | B | 334 | Small Space of Random Values | 834 |
| ParentOf | B | 335 | Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) | 836 |
| ParentOf | B | 338 | Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) | 844 |
| ParentOf | C | 340 | Generation of Predictable Numbers or Identifiers | 849 |

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 344 | Use of Invariant Value in Dynamically Changing Context | 856 |
| ParentOf | B | 1204 | Generation of Weak Initialization Vector (IV) | 1996 |
| ParentOf | B | 1241 | Use of Predictable Algorithm in Random Number Generator | 2042 |
| CanPrecede | B | 804 | Guessable CAPTCHA | 1710 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 331 | Insufficient Entropy | 828 |
| ParentOf | B | 335 | Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) | 836 |
| ParentOf | B | 338 | Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) | 844 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Weakness Ordinalities

**Primary :**

## Applicable Platforms

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

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

## Background Details

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated. There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value.

## Likelihood Of Exploit

High

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Confidentiality<br>Other | Other<br><br>*When a protection mechanism relies on random values to restrict access to a sensitive resource, such as a session ID or a seed for generating a cryptographic key, then the resource being protected could be accessed by guessing the ID or key.* | |
| Access Control<br>Other | Bypass Protection Mechanism<br>Other<br><br>*If product relies on unique, unguessable IDs to identify a resource, an attacker might be able to guess an ID for a resource that is owned by another user. The attacker could then read the resource, or pre-create a resource with the* | |

| Scope | Impact | Likelihood |
|---|---|---|
| | *same ID to prevent the legitimate program from properly sending the resource to the intended user. For example, a product might maintain session information in a file whose name is based on a username. An attacker could pre-create this file for a victim user, then set the permissions so that the application cannot generate the session for the victim, preventing the victim from using the application.* | |
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br><br>*When an authorization or authentication mechanism relies on random values to restrict access to restricted functionality, such as a session ID or a seed for generating a cryptographic key, then an attacker may access the restricted functionality by guessing the ID or key.* | |

## Detection Methods

**Black Box**

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and look for library functions that indicate when randomness is being used. Run the process multiple times to see if the seed changes. Look for accesses of devices or equivalent resources that are commonly used for strong (or weak) randomness, such as /dev/urandom on Linux. Look for library or system calls that access predictable information such as process IDs and system time.

**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 Manual Results Interpretation**

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Man-in-the-middle attack tool

*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: 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.)

*Effectiveness = High*

### Potential Mitigations

#### Phase: Architecture and Design

Use a well-vetted algorithm that is currently considered to be strong by experts in the field, and select well-tested implementations with adequate length seeds. In general, if a pseudo-random number generator is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

#### Phase: Implementation

Consider a PRNG that re-seeds itself as needed from high quality pseudo-random output sources, such as hardware devices.

#### Phase: Testing

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

#### Phase: Architecture and Design

#### Phase: Requirements

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

#### Phase: Testing

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

### Demonstrative Examples

#### Example 1:

This code attempts to generate a unique random identifier for a user's session.

*Example Language: PHP* *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

**Example 2:**

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

*Example Language: Java* *(Bad)*

```
String GenerateReceiptURL(String baseUrl) {
  Random ranGen = new Random();
  ranGen.setSeed((new Date()).getTime());
  return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2021-3692** | PHP framework uses mt_rand() function (Marsenne Twister) when generating tokens<br>*https://www.cve.org/CVERecord?id=CVE-2021-3692* |
| **CVE-2020-7010** | Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time.<br>*https://www.cve.org/CVERecord?id=CVE-2020-7010* |
| **CVE-2009-3278** | Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks.<br>*https://www.cve.org/CVERecord?id=CVE-2009-3278* |
| **CVE-2009-3238** | Random number generator can repeatedly generate the same value.<br>*https://www.cve.org/CVERecord?id=CVE-2009-3238* |
| **CVE-2009-2367** | Web application generates predictable session IDs, allowing session hijacking.<br>*https://www.cve.org/CVERecord?id=CVE-2009-2367* |
| **CVE-2009-2158** | Password recovery utility generates a relatively small number of random passwords, simplifying brute force attacks.<br>*https://www.cve.org/CVERecord?id=CVE-2009-2158* |
| **CVE-2009-0255** | Cryptographic key created with a seed based on the system time.<br>*https://www.cve.org/CVERecord?id=CVE-2009-0255* |
| **CVE-2008-5162** | Kernel function does not have a good entropy source just after boot.<br>*https://www.cve.org/CVERecord?id=CVE-2008-5162* |
| **CVE-2008-4905** | Blogging software uses a hard-coded salt when calculating a password hash.<br>*https://www.cve.org/CVERecord?id=CVE-2008-4905* |
| **CVE-2008-4929** | Bulletin board application uses insufficiently random names for uploaded files, allowing other users to access private files.<br>*https://www.cve.org/CVERecord?id=CVE-2008-4929* |
| **CVE-2008-3612** | Handheld device uses predictable TCP sequence numbers, allowing spoofing or hijacking of TCP connections.<br>*https://www.cve.org/CVERecord?id=CVE-2008-3612* |
| **CVE-2008-2433** | Web management console generates session IDs based on the login time, making it easier to conduct session hijacking.<br>*https://www.cve.org/CVERecord?id=CVE-2008-2433* |
| **CVE-2008-0166** | SSL library uses a weak random number generator that only generates 65,536 unique keys.<br>*https://www.cve.org/CVERecord?id=CVE-2008-0166* |

| Reference | Description |
|---|---|
| **CVE-2008-2108** | Chain: insufficient precision causes extra zero bits to be assigned, reducing entropy for an API function that generates random numbers. *https://www.cve.org/CVERecord?id=CVE-2008-2108* |
| **CVE-2008-2108** | Chain: insufficient precision (CWE-1339) in random-number generator causes some zero bits to be reliably generated, reducing the amount of entropy (CWE-331) *https://www.cve.org/CVERecord?id=CVE-2008-2108* |
| **CVE-2008-2020** | CAPTCHA implementation does not produce enough different images, allowing bypass using a database of all possible checksums. *https://www.cve.org/CVERecord?id=CVE-2008-2020* |
| **CVE-2008-0087** | DNS client uses predictable DNS transaction IDs, allowing DNS spoofing. *https://www.cve.org/CVERecord?id=CVE-2008-0087* |
| **CVE-2008-0141** | Application generates passwords that are based on the time of day. *https://www.cve.org/CVERecord?id=CVE-2008-0141* |

## Functional Areas

- Cryptography
- Authentication
- Session Management

## 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 | 2335 |
| MemberOf | C | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control | 711 | 2356 |
| MemberOf | C | 747 | CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC) | 734 | 2371 |
| MemberOf | C | 753 | 2009 Top 25 - Porous Defenses | 750 | 2374 |
| MemberOf | C | 808 | 2010 Top 25 - Weaknesses On the Cusp | 800 | 2376 |
| MemberOf | C | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 867 | 2011 Top 25 - Weaknesses On the Cusp | 900 | 2393 |
| MemberOf | C | 883 | CERT C++ Secure Coding Section 49 - Miscellaneous (MSC) | 868 | 2402 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | C | 1169 | SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON) | 1154 | 2483 |
| MemberOf | C | 1170 | SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC) | 1154 | 2484 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Relationship

This can be primary to many other weaknesses such as cryptographic errors, authentication errors, symlink following, information leaks, and others.

### Maintenance

As of CWE 4.3, CWE-330 and its descendants are being investigated by the CWE crypto team to identify gaps related to randomness and unpredictability, as well as the relationships between randomness and cryptographic primitives. This "subtree analysis" might result in the addition or deprecation of existing entries; the reorganization of relationships in some views, e.g. the research view (CWE-1000); more consistent use of terminology; and/or significant modifications to related entries.

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Randomness and Predictability |
| 7 Pernicious Kingdoms | | | Insecure Randomness |
| OWASP Top Ten 2004 | A2 | CWE More Specific | Broken Access Control |
| CERT C Secure Coding | CON33-C | Imprecise | Avoid race conditions when using library functions |
| CERT C Secure Coding | MSC30-C | CWE More Abstract | Do not use the rand() function for generating pseudorandom numbers |
| CERT C Secure Coding | MSC32-C | CWE More Abstract | Properly seed pseudorandom number generators |
| WASC | 11 | | Brute Force |
| WASC | 18 | | Credential/Session Prediction |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC02-J | | Generate strong random numbers |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 59 | Session Credential Falsification through Prediction |
| 112 | Brute Force |
| 485 | Signature Spoofing by Key Recreation |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

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

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

## CWE-331: Insufficient Entropy

**Weakness ID :** 331
**Structure :** Simple
**Abstraction :** Base

### Description

The product uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than 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 | C | 330 | Use of Insufficiently Random Values | 821 |
| ParentOf | V | 332 | Insufficient Entropy in PRNG | 830 |
| ParentOf | V | 333 | Improper Handling of Insufficient Entropy in TRNG | 832 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 330 | Use of Insufficiently Random Values | 821 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Other | Bypass Protection Mechanism<br>Other<br>*An attacker could guess the random numbers generated and could gain unauthorized access to a system if the random numbers are used for authentication and authorization.* | |

### Potential Mitigations

**Phase: Implementation**

Determine the necessary entropy to adequately provide for randomness and predictability. This can be achieved by increasing the number of bits of objects such as keys and seeds.

### Demonstrative Examples

**Example 1:**

This code generates a unique random identifier for a user's session.

*Example Language: PHP* *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

**Example 2:**

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

*Example Language: Java* *(Bad)*

```
String GenerateReceiptURL(String baseUrl) {
  Random ranGen = new Random();
  ranGen.setSeed((new Date()).getTime());
  return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-0950** | Insufficiently random data used to generate session tokens using C rand(). Also, for certificate/key generation, uses a source that does not block when entropy is low. *https://www.cve.org/CVERecord?id=CVE-2001-0950* |
| **CVE-2008-2108** | Chain: insufficient precision (CWE-1339) in random-number generator causes some zero bits to be reliably generated, reducing the amount of entropy (CWE-331) *https://www.cve.org/CVERecord?id=CVE-2008-2108* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1170 | SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC) | 1154 | 2484 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

**Maintenance**

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Insufficient Entropy |
| WASC | 11 | | Brute Force |
| CERT C Secure Coding | MSC32-C | Exact | Properly seed pseudorandom number generators |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 59 | Session Credential Falsification through Prediction |

### References

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

## CWE-332: Insufficient Entropy in PRNG

**Weakness ID :** 332
**Structure :** Simple
**Abstraction :** Variant

### Description

The lack of entropy available for, or used by, a Pseudo-Random Number Generator (PRNG) can be a stability and security threat.

### Relationships

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 | 331 | Insufficient Entropy | 828 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

### Applicable Platforms

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

### Likelihood Of Exploit

Medium

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Availability | DoS: Crash, Exit, or Restart<br>*If a pseudo-random number generator is using a limited entropy source which runs out (if the generator fails closed), the program may pause or crash.* | |
| Access Control<br>Other | Bypass Protection Mechanism<br>Other<br>*If a PRNG is using a limited entropy source which runs out, and the generator fails open, the generator could produce predictable random numbers. Potentially a weak source of random numbers could weaken the encryption method used for authentication of users.* | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Phase: Implementation**

Consider a PRNG that re-seeds itself as needed from high-quality pseudo-random output, such as hardware devices.

**Phase: Architecture and Design**

When deciding which PRNG to use, look at its sources of entropy. Depending on what your security needs are, you may need to use a random number generator that always uses strong random data -- i.e., a random number generator that attempts to be strong but will fail in a weak way or will always provide some middle ground of protection through techniques like re-seeding. Generally, something that always provides a predictable amount of strength is preferable.

## Observed Examples

| Reference | Description |
|---|---|
| **[REF-1374]** | Chain: JavaScript-based cryptocurrency library can fall back to the insecure Math.random() function instead of reporting a failure (CWE-392), thus reducing the entropy (CWE-332) and leading to generation of non-unique cryptographic keys for Bitcoin wallets (CWE-1391)<br>*https://www.unciphered.com/blog/randstorm-you-cant-patch-a-house-of-cards* |
| **CVE-2019-1715** | security product has insufficient entropy in the DRBG, allowing collisions and private key discovery<br>*https://www.cve.org/CVERecord?id=CVE-2019-1715* |

## 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 | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Insufficient entropy in PRNG |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC02-J | | Generate strong random numbers |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

[REF-1374]Unciphered. "Randstorm: You Can't Patch a House of Cards". 2023 November 4. < https://www.unciphered.com/blog/randstorm-you-cant-patch-a-house-of-cards >.2023-11-15.

# CWE-333: Improper Handling of Insufficient Entropy in TRNG

**Weakness ID :** 333
**Structure :** Simple
**Abstraction :** Variant

## Description

True random number generators (TRNG) generally have a limited source of entropy and therefore can fail or block.

## Extended Description

The rate at which true random numbers can be generated is limited. It is important that one uses them only when they are needed for security.

## Relationships

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 | 755 | Improper Handling of Exceptional Conditions | 1585 |
| ChildOf | B | 331 | Insufficient Entropy | 828 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Likelihood Of Exploit

Low

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Availability | DoS: Crash, Exit, or Restart<br>*A program may crash or block if it runs out of random numbers.* | |

## Potential Mitigations

**Phase: Implementation**

Rather than failing on a lack of random numbers, it is often preferable to wait for more numbers to be created.

## Demonstrative Examples

**Example 1:**

This code uses a TRNG to generate a unique session id for new connections to a server:

*Example Language: C* *(Bad)*

```
while (1){
  if (haveNewConnection()){
    if (hwRandom()){
      int sessionID = hwRandom();
      createNewConnection(sessionID);
    } } }
```

This code does not attempt to limit the number of new connections or make sure the TRNG can successfully generate a new random number. An attacker may be able to create many new connections and exhaust the entropy of the TRNG. The TRNG may then block and cause the program to crash or hang.

## 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 | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

**Maintenance**

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Failure of TRNG |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC02-J | | Generate strong random numbers |

### References

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

## CWE-334: Small Space of Random Values

**Weakness ID :** 334
**Structure :** Simple
**Abstraction :** Base

### Description

The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 330 | Use of Insufficiently Random Values | 821 |
| ParentOf | V | 6 | J2EE Misconfiguration: Insufficient Session-ID Length | 2 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Other | Bypass Protection Mechanism<br>Other<br>*An attacker could easily guess the values used. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.* | |

## Potential Mitigations

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

## Demonstrative Examples

**Example 1:**

The following XML example code is a deployment descriptor for a Java web application deployed on a Sun Java Application Server. This deployment descriptor includes a session configuration property for configuring the session ID length.

*Example Language: XML* *(Bad)*

```
<sun-web-app>
  ...
  <session-config>
    <session-properties>
      <property name="idLengthBytes" value="8">
        <description>The number of bytes in this web module's session ID.</description>
      </property>
    </session-properties>
  </session-config>
  ...
</sun-web-app>
```

This deployment descriptor has set the session ID length for this Java web application to 8 bytes (or 64 bits). The session ID length for Java web applications should be set to 16 bytes (128 bits) to prevent attackers from guessing and/or stealing a session ID and taking over a user's session.

Note for most application servers including the Sun Java Application Server the session ID length is by default set to 128 bits and should not be changed. And for many application servers the session ID length cannot be changed from this default setting. Check your application server documentation for the session ID length default setting and configuration options to ensure that the session ID length is set to 128 bits.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-0583** | Product uses 5 alphanumeric characters for filenames of expense claim reports, stored under web root. *https://www.cve.org/CVERecord?id=CVE-2002-0583* |
| **CVE-2002-0903** | Product uses small number of random numbers for a code to approve an action, and also uses predictable new user IDs, allowing attackers to hijack new accounts. *https://www.cve.org/CVERecord?id=CVE-2002-0903* |
| **CVE-2003-1230** | SYN cookies implementation only uses 32-bit keys, making it easier to brute force ISN. *https://www.cve.org/CVERecord?id=CVE-2003-1230* |
| **CVE-2004-0230** | Complex predictability / randomness (reduced space). *https://www.cve.org/CVERecord?id=CVE-2004-0230* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Small Space of Random Values |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

# CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)

**Weakness ID :** 335
**Structure :** Simple
**Abstraction :** Base

## Description

The product uses a Pseudo-Random Number Generator (PRNG) but does not correctly manage seeds.

## Extended Description

PRNGs are deterministic and, while their output appears random, they cannot actually create entropy. They rely on cryptographically secure and unique seeds for entropy so proper seeding is critical to the secure operation of the PRNG.

Management of seeds could be broken down into two main areas:

- (1) protecting seeds as cryptographic material (such as a cryptographic key);
- (2) whenever possible, using a uniquely generated seed from a cryptographically secure source

PRNGs require a seed as input to generate a stream of numbers that are functionally indistinguishable from random numbers. While the output is, in many cases, sufficient for cryptographic uses, the output of any PRNG is directly determined by the seed provided as input. If the seed can be ascertained by a third party, the entire output of the PRNG can be made known to them. As such, the seed should be kept secret and should ideally not be able to be guessed.

For example, the current time may be a poor seed. Knowing the approximate time the PRNG was seeded greatly reduces the possible key space.

Seeds do not necessarily need to be unique, but reusing seeds may open up attacks if the seed is discovered.

## Relationships

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 | 330 | Use of Insufficiently Random Values | 821 |
| ParentOf | V | 336 | Same Seed in Pseudo-Random Number Generator (PRNG) | 839 |
| ParentOf | V | 337 | Predictable Seed in Pseudo-Random Number Generator (PRNG) | 841 |
| ParentOf | V | 339 | Small Seed Space in PRNG | 847 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | C | 330 | Use of Insufficiently Random Values | 821 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Other | Bypass Protection Mechanism<br>Other<br>*If a PRNG is used incorrectly, such as using the same seed for each initialization or using a predictable seed, then an attacker may be able to easily guess the seed and thus the random numbers. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.* | |

## Demonstrative Examples

**Example 1:**

The following code uses a statistical PRNG to generate account IDs.

*Example Language: Java* *(Bad)*

```
private static final long SEED = 1234567890;
```

```
public int generateAccountID() {
  Random random = new Random(SEED);
  return random.nextInt();
}
```

Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

**Example 2:**

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

*Example Language: Java* *(Bad)*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

*Example Language: C* *(Bad)*

```
srand(time());
int randNum = rand();
```

An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

**Example 3:**

This code grabs some random bytes and uses them for a seed in a PRNG, in order to generate a new cryptographic key.

*Example Language: Python* *(Bad)*

```
# getting 2 bytes of randomness for the seeding the PRNG
seed = os.urandom(2)
random.seed(a=seed)
key = random.getrandbits(128)
```

Since only 2 bytes are used as a seed, an attacker will only need to guess 2^16 (65,536) values before being able to replicate the state of the PRNG.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2020-7010** | Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time. *https://www.cve.org/CVERecord?id=CVE-2020-7010* |
| **CVE-2019-11495** | server uses erlang:now() to seed the PRNG, which results in a small search space for potential random seeds *https://www.cve.org/CVERecord?id=CVE-2019-11495* |
| **CVE-2018-12520** | Product's PRNG is not seeded for the generation of session IDs *https://www.cve.org/CVERecord?id=CVE-2018-12520* |
| **CVE-2016-10180** | Router's PIN generation is based on rand(time(0)) seeding. *https://www.cve.org/CVERecord?id=CVE-2016-10180* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | PRNG Seed Error |

## References

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

# CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG)

**Weakness ID :** 336
**Structure :** Simple
**Abstraction :** Variant

## Description

A Pseudo-Random Number Generator (PRNG) uses the same seed each time the product is initialized.

## Extended Description

Given the deterministic nature of PRNGs, using the same seed for each initialization will lead to the same output in the same order. If an attacker can guess (or knows) the seed, then the attacker may be able to determine the random numbers that will be produced from the PRNG.

## Relationships

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 | 335 | Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) | 836 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other<br>Access Control | Other<br>Bypass Protection Mechanism | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Architecture and Design

Do not reuse PRNG seeds. Consider a PRNG that periodically re-seeds itself as needed from a high quality pseudo-random output, such as hardware devices.

### Phase: Architecture and Design

### Phase: Requirements

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems, or use the more recent FIPS 140-3 [REF-1192] if possible.

## Demonstrative Examples

### Example 1:

The following code uses a statistical PRNG to generate account IDs.

*Example Language: Java* *(Bad)*

```
private static final long SEED = 1234567890;
public int generateAccountID() {
  Random random = new Random(SEED);
  return random.nextInt();
}
```

Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

### Example 2:

This code attempts to generate a unique random identifier for a user's session.

*Example Language: PHP* *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

If the user IDs are generated sequentially, or otherwise restricted to a narrow range of values, then this example also exhibits a Small Seed Space (CWE-339).

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-39218** | SDK for JavaScript app builder for serverless code uses the same fixed seed for a PRNG, allowing cryptography bypass *https://www.cve.org/CVERecord?id=CVE-2022-39218* |

## 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 | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Same Seed in PRNG |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC02-J | | Generate strong random numbers |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

## CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)

**Weakness ID :** 337
**Structure :** Simple
**Abstraction :** Variant

## Description

A Pseudo-Random Number Generator (PRNG) is initialized from a predictable seed, such as the process ID or system time.

## Extended Description

The use of predictable seeds significantly reduces the number of possible seeds that an attacker would need to test in order to predict which random numbers will be generated by the PRNG.

## Relationships

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 | 335 | Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) | 836 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Potential Mitigations

Use non-predictable inputs for seed generation.

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems, or use the more recent FIPS 140-3 [REF-1192] if possible.

**Phase: Implementation**

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

## Demonstrative Examples

**Example 1:**

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

*Example Language: Java* *(Bad)*

```
Random random = new Random(System.currentTimeMillis());
```

```
int accountID = random.nextInt();
```

*Example Language: C* *(Bad)*

```
srand(time());
int randNum = rand();
```

An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2020-7010** | Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time. *https://www.cve.org/CVERecord?id=CVE-2020-7010* |
| **CVE-2019-11495** | server uses erlang:now() to seed the PRNG, which results in a small search space for potential random seeds *https://www.cve.org/CVERecord?id=CVE-2019-11495* |
| **CVE-2008-0166** | The removal of a couple lines of code caused Debian's OpenSSL Package to only use the current process ID for seeding a PRNG *https://www.cve.org/CVERecord?id=CVE-2008-0166* |
| **CVE-2016-10180** | Router's PIN generation is based on rand(time(0)) seeding. *https://www.cve.org/CVERecord?id=CVE-2016-10180* |
| **CVE-2018-9057** | cloud provider product uses a non-cryptographically secure PRNG and seeds it with the current time *https://www.cve.org/CVERecord?id=CVE-2018-9057* |

## 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 | 861 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) | 844 | 2391 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1152 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) | 1133 | 2474 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Predictable Seed in PRNG |
| The CERT Oracle Secure Coding Standard for Java (2011) | MSC02-J | | Generate strong random numbers |

### References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

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



## CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)

**Weakness ID :** 338
**Structure :** Simple
**Abstraction :** Base

### Description

The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG's algorithm is not cryptographically strong.

### Extended Description

When a non-cryptographic PRNG is used in a cryptographic context, it can expose the cryptography to certain types of attacks.

Often a pseudo-random number generator (PRNG) is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms that use random numbers. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system. While such PRNGs might have very useful features, these same features could be used to break the cryptography.

### Relationships

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 | | 330 | Use of Insufficiently Random Values | 821 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | | 330 | Use of Insufficiently Random Values | 821 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

## Applicable Platforms

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

## Likelihood Of Exploit

Medium

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>*If a PRNG is used for authentication and authorization, such as a session ID or a seed for generating a cryptographic key, then an attacker may be able to easily guess the ID or cryptographic key and gain access to restricted functionality.* | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Implementation

Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CyptGenRandom on Windows, or hw_rand() on Linux.

## Demonstrative Examples

### Example 1:

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

*Example Language: Java* *(Bad)*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

*Example Language: C* *(Bad)*

```
srand(time());
int randNum = rand();
```

The random number functions used in these examples, rand() and Random.nextInt(), are not considered cryptographically strong. An attacker may be able to predict the random numbers generated by these functions. Note that these example also exhibit CWE-337 (Predictable Seed in PRNG).

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2021-3692** | PHP framework uses mt_rand() function (Marsenne Twister) when generating tokens<br>*https://www.cve.org/CVERecord?id=CVE-2021-3692* |
| **CVE-2009-3278** | Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks.<br>*https://www.cve.org/CVERecord?id=CVE-2009-3278* |
| **CVE-2009-3238** | Random number generator can repeatedly generate the same value.<br>*https://www.cve.org/CVERecord?id=CVE-2009-3238* |
| **CVE-2009-2367** | Web application generates predictable session IDs, allowing session hijacking.<br>*https://www.cve.org/CVERecord?id=CVE-2009-2367* |
| **CVE-2008-0166** | SSL library uses a weak random number generator that only generates 65,536 unique keys.<br>*https://www.cve.org/CVERecord?id=CVE-2008-0166* |

## MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1170 | SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC) | 1154 | 2484 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| CLASP | | | Non-cryptographic PRNG |
| CERT C Secure Coding | MSC30-C | CWE More Abstract | Do not use the rand() function for generating pseudorandom numbers |

## References

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

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

## CWE-339: Small Seed Space in PRNG

**Weakness ID :** 339
**Structure :** Simple
**Abstraction :** Variant

### Description

A Pseudo-Random Number Generator (PRNG) uses a relatively small seed space, which makes it more susceptible to brute force attacks.

### Extended Description

PRNGs are entirely deterministic once seeded, so it should be extremely difficult to guess the seed. If an attacker can collect the outputs of a PRNG and then brute force the seed by trying every possibility to see which seed matches the observed output, then the attacker will know the output of any subsequent calls to the PRNG. A small seed space implies that the attacker will have far fewer possible values to try to exhaust all possibilities.

### Relationships

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 | 335 | Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) | 836 |
| PeerOf | B | 341 | Predictable from Observable State | 850 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

### Potential Mitigations

**Phase: Architecture and Design**

Use well vetted pseudo-random number generating algorithms with adequate length seeds. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems, or use the more recent FIPS 140-3 [REF-1192] if possible.

## Demonstrative Examples

### Example 1:

This code grabs some random bytes and uses them for a seed in a PRNG, in order to generate a new cryptographic key.

*Example Language: Python* *(Bad)*

```
# getting 2 bytes of randomness for the seeding the PRNG
seed = os.urandom(2)
random.seed(a=seed)
key = random.getrandbits(128)
```

Since only 2 bytes are used as a seed, an attacker will only need to guess 2^16 (65,536) values before being able to replicate the state of the PRNG.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2019-10908** | product generates passwords via org.apache.commons.lang.RandomStringUtils, which uses java.util.Random internally. This PRNG has only a 48-bit seed. *https://www.cve.org/CVERecord?id=CVE-2019-10908* |

## 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 | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

This entry may have a chaining relationship with predictable from observable state (CWE-341).

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Small Seed Space in PRNG |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < https://csrc.nist.gov/publications/detail/fips/140/3/final >.

# CWE-340: Generation of Predictable Numbers or Identifiers

**Weakness ID :** 340
**Structure :** Simple
**Abstraction :** Class

## Description

The product uses a scheme that generates numbers or identifiers that are more predictable than required.

## Relationships

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 | 330 | Use of Insufficiently Random Values | 821 |
| ParentOf | B | 341 | Predictable from Observable State | 850 |
| ParentOf | B | 342 | Predictable Exact Value from Previous Values | 852 |
| ParentOf | B | 343 | Predictable Value Range from Previous Values | 854 |
| CanPrecede | | 384 | Session Fixation | 943 |

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Demonstrative Examples

**Example 1:**

This code generates a unique random identifier for a user's session.

*Example Language: PHP* *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-29330** | Product for administering PBX systems uses predictable identifiers and timestamps for filenames (CWE-340) which allows attackers to access files via direct request (CWE-425). *https://www.cve.org/CVERecord?id=CVE-2022-29330* |

| Reference | Description |
|---|---|
| CVE-2001-1141 | PRNG allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers. *https://www.cve.org/CVERecord?id=CVE-2001-1141* |
| CVE-1999-0074 | Listening TCP ports are sequentially allocated, allowing spoofing attacks. *https://www.cve.org/CVERecord?id=CVE-1999-0074* |

### 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 | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

### Notes

**Maintenance**

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Predictability problems |
| WASC | 11 | | Brute Force |

### References

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



## CWE-341: Predictable from Observable State

**Weakness ID :** 341
**Structure :** Simple
**Abstraction :** Base

### Description

A number or object is predictable based on observations that the attacker can make about the state of the system or network, such as time, process ID, 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 | 340 | Generation of Predictable Numbers or Identifiers | 849 |
| PeerOf | V | 339 | Small Seed Space in PRNG | 847 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1011 | Authorize Actors | 2446 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context<br>*This weakness could be exploited by an attacker in a number ways depending on the context. If a predictable number is used to generate IDs or keys that are used within protection mechanisms, then an attacker could gain unauthorized access to the system. If predictable filenames are used for storing sensitive information, then an attacker might gain access to the system and may be able to gain access to the information in the file.* | |

## Potential Mitigations

**Phase: Implementation**

Increase the entropy used to seed a PRNG.

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Phase: Implementation**

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

## Demonstrative Examples

**Example 1:**

This code generates a unique random identifier for a user's session.

*Example Language: PHP* *(Bad)*

```
function generateSessionID($userID){
  srand($userID);
  return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-0389** | Mail server stores private mail messages with predictable filenames in a world-executable directory, which allows local users to read private mailing list archives. *https://www.cve.org/CVERecord?id=CVE-2002-0389* |
| **CVE-2001-1141** | PRNG allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers. *https://www.cve.org/CVERecord?id=CVE-2001-1141* |
| **CVE-2000-0335** | DNS resolver library uses predictable IDs, which allows a local attacker to spoof DNS query results. *https://www.cve.org/CVERecord?id=CVE-2000-0335* |
| **CVE-2005-1636** | MFV. predictable filename and insecure permissions allows file modification to execute SQL queries. *https://www.cve.org/CVERecord?id=CVE-2005-1636* |

### MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

### Notes

**Maintenance**

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Predictable from Observable State |

### References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

## CWE-342: Predictable Exact Value from Previous Values

**Weakness ID :** 342

**Structure :** Simple
**Abstraction :** Base

## Description

An exact value or random number can be precisely predicted by observing previous 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 | | 340 | Generation of Predictable Numbers or Identifiers | 849 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Potential Mitigations

Increase the entropy used to seed a PRNG.

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Phase: Implementation**

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1463** | Firewall generates easily predictable initial sequence numbers (ISN), which allows remote attackers to spoof connections. *https://www.cve.org/CVERecord?id=CVE-2002-1463* |
| **CVE-1999-0074** | Listening TCP ports are sequentially allocated, allowing spoofing attacks. *https://www.cve.org/CVERecord?id=CVE-1999-0074* |
| **CVE-1999-0077** | Predictable TCP sequence numbers allow spoofing. *https://www.cve.org/CVERecord?id=CVE-1999-0077* |
| **CVE-2000-0335** | DNS resolver uses predictable IDs, allowing a local user to spoof DNS query results. *https://www.cve.org/CVERecord?id=CVE-2000-0335* |

## 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 | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

### Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Predictable Exact Value from Previous Values |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

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

# CWE-343: Predictable Value Range from Previous Values

**Weakness ID :** 343
**Structure :** Simple
**Abstraction :** Base

## Description

The product's random number generator produces a series of values which, when observed, can be used to infer a relatively small range of possibilities for the next value that could be generated.

## Extended Description

The output of a random number generator should not be predictable based on observations of previous values. In some cases, an attacker cannot predict the exact value that will be produced next, but can narrow down the possibilities significantly. This reduces the amount of effort to perform a brute force attack. For example, suppose the product generates random numbers between 1 and 100, but it always produces a larger value until it reaches 100. If the generator produces an 80, then the attacker knows that the next value will be somewhere between 81 and 100. Instead of 100 possibilities, the attacker only needs to consider 20.

## Relationships

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 | | 340 | Generation of Predictable Numbers or Identifiers | 849 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Potential Mitigations

Increase the entropy used to seed a PRNG.

**Phase: Architecture and Design**

**Phase: Requirements**

*Strategy = Libraries or Frameworks*

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Phase: Implementation**

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

## 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 | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

**Maintenance**

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Predictable Value Range from Previous Values |

## References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

[REF-320]Michal Zalewski. "Strange Attractors and TCP/IP Sequence Number Analysis". 2001. < https://lcamtuf.coredump.cx/oldtcp/tcpseq.html >.2023-04-07.

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



# CWE-344: Use of Invariant Value in Dynamically Changing Context

**Weakness ID :** 344
**Structure :** Simple
**Abstraction :** Base

## Description

The product uses a constant value, name, or reference, but this value can (or should) vary across different environments.

## Relationships

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 | 330 | Use of Insufficiently Random Values | 821 |
| ParentOf | B | 323 | Reusing a Nonce, Key Pair in Encryption | 797 |
| ParentOf | V | 587 | Assignment of a Fixed Address to a Pointer | 1330 |
| ParentOf | B | 798 | Use of Hard-coded Credentials | 1699 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1213 | Random Number Issues | 2498 |

## Weakness Ordinalities

**Primary :**

**Resultant :**

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Other | Varies by Context | |

## Demonstrative Examples

**Example 1:**

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

*Example Language: C* *(Bad)*

```
int VerifyAdmin(char *password) {
  if (strcmp(password, "Mew!")) {
```

```
    printf("Incorrect Password!\n");
    return(0)
  }
  printf("Entering Diagnostic Mode...\n");
  return(1);
}
```

*Example Language: Java* *(Bad)*

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

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

**Example 2:**

This code assumes a particular function will always be found at a particular address. It assigns a pointer to that address and calls the function.

*Example Language: C* *(Bad)*

```
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

The same function may not always be found at the same memory address. This could lead to a crash, or an attacker may alter the memory at the expected address, leading to arbitrary code execution.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-0980** | Component for web browser writes an error message to a known location, which can then be referenced by attackers to process HTML/script in a less restrictive context *https://www.cve.org/CVERecord?id=CVE-2002-0980* |

## 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 | 905 | SFP Primary Cluster: Predictability | 888 | 2409 |
| MemberOf | C | 1414 | Comprehensive Categorization: Randomness | 1400 | 2564 |

## Notes

**Relationship**

overlaps default configuration.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Static Value in Unpredictable Context |

### References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf >.2023-04-07.

## CWE-345: Insufficient Verification of Data Authenticity

**Weakness ID :** 345
**Structure :** Simple
**Abstraction :** Class

### Description

The product does not sufficiently verify the origin or authenticity of data, in a way that causes it to accept invalid data.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 1529 |
| ParentOf | C | 346 | Origin Validation Error | 860 |
| ParentOf | B | 347 | Improper Verification of Cryptographic Signature | 864 |
| ParentOf | B | 348 | Use of Less Trusted Source | 866 |
| ParentOf | B | 349 | Acceptance of Extraneous Untrusted Data With Trusted Data | 868 |
| ParentOf | B | 351 | Insufficient Type Distinction | 873 |
| ParentOf | | 352 | Cross-Site Request Forgery (CSRF) | 875 |
| ParentOf | B | 353 | Missing Support for Integrity Check | 881 |
| ParentOf | B | 354 | Improper Validation of Integrity Check Value | 883 |
| ParentOf | B | 360 | Trust of System Event Data | 894 |
| ParentOf | B | 494 | Download of Code Without Integrity Check | 1192 |
| ParentOf | V | 616 | Incomplete Identification of Uploaded File Variables (PHP) | 1385 |
| ParentOf | V | 646 | Reliance on File Name or Extension of Externally-Supplied File | 1434 |
| ParentOf | B | 649 | Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking | 1439 |
| ParentOf | B | 924 | Improper Enforcement of Message Integrity During Transmission in a Communication Channel | 1839 |
| ParentOf | B | 1293 | Missing Source Correlation of Multiple Independent Data | 2161 |
| PeerOf | C | 20 | Improper Input Validation | 20 |
| PeerOf | B | 1304 | Improperly Preserved Integrity of Hardware Configuration State During a Power Save/Restore Operation | 2188 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | C | 346 | Origin Validation Error | 860 |
| ParentOf | B | 347 | Improper Verification of Cryptographic Signature | 864 |
| ParentOf | | 352 | Cross-Site Request Forgery (CSRF) | 875 |

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ParentOf | B | 354 | Improper Validation of Integrity Check Value | 883 |
| ParentOf | B | 924 | Improper Enforcement of Message Integrity During Transmission in a Communication Channel | 1839 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

## Applicable Platforms

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

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Integrity<br>Other | Varies by Context<br>Unexpected State | |

## 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:**

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 did not sign firmware images.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2022-30260** | Distributed Control System (DCS) does not sign firmware images and only relies on insecure checksums for integrity checks<br>*https://www.cve.org/CVERecord?id=CVE-2022-30260* |
| **CVE-2022-30267** | Distributed Control System (DCS) does not sign firmware images and only relies on insecure checksums for integrity checks<br>*https://www.cve.org/CVERecord?id=CVE-2022-30267* |
| **CVE-2022-30272** | Remote Terminal Unit (RTU) does not use signatures for firmware images and relies on insecure checksums<br>*https://www.cve.org/CVERecord?id=CVE-2022-30272* |

## 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 | 724 | OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management | 711 | 2356 |
| MemberOf | C | 949 | SFP Secondary Cluster: Faulty Endpoint Authentication | 888 | 2416 |
| MemberOf | V | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 2597 |
| MemberOf | C | 1354 | OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures | 1344 | 2516 |
| MemberOf | C | 1411 | Comprehensive Categorization: Insufficient Verification of Data Authenticity | 1400 | 2559 |

## Notes

### Relationship

"origin validation" could fall under this.

### Maintenance

The specific ways in which the origin is not properly identified should be laid out as separate weaknesses. In some sense, this is more like a category.

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Insufficient Verification of Data |
| OWASP Top Ten 2004 | A3 | CWE More Specific | Broken Authentication and Session Management |
| WASC | 12 | | Content Spoofing |

## Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 111 | JSON Hijacking (aka JavaScript Hijacking) |
| 141 | Cache Poisoning |
| 142 | DNS Cache Poisoning |
| 148 | Content Spoofing |
| 218 | Spoofing of UDDI/ebXML Messages |
| 384 | Application API Message Manipulation via Man-in-the-Middle |
| 385 | Transaction or Event Tampering via Application API Manipulation |
| 386 | Application API Navigation Remapping |
| 387 | Navigation Remapping To Propagate Malicious Content |
| 388 | Application API Button Hijacking |
| 665 | Exploitation of Thunderbolt Protection Flaws |
| 701 | Browser in the Middle (BiTM) |

## References

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

[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/ >.

# CWE-346: Origin Validation Error

**Weakness ID :** 346
**Structure :** Simple
**Abstraction :** Class

## Description

The product does not properly verify that the source of data or communication is valid.

## Relationships

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

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control | 687 |
| ChildOf | C | 345 | Insufficient Verification of Data Authenticity | 858 |
| ParentOf | B | 940 | Improper Verification of Source of a Communication Channel | 1852 |
| ParentOf | V | 1385 | Missing Origin Validation in WebSockets | 2271 |
| PeerOf | C | 451 | User Interface (UI) Misrepresentation of Critical Information | 1087 |

*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 | 858 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1014 | Identify Actors | 2450 |

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

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

## Weakness Ordinalities

**Primary :**

**Resultant :**

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Other | Gain Privileges or Assume Identity<br>Varies by Context<br>*An attacker can access any functionality that is inadvertently accessible to the source.* | |

## 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<br>*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<br>*https://www.cve.org/CVERecord?id=CVE-2005-0877* |
| **CVE-2001-1452** | DNS server caches glue records received from non-delegated name servers<br>*https://www.cve.org/CVERecord?id=CVE-2001-1452* |
| **CVE-2005-2188** | user ID obtained from untrusted source (URL)<br>*https://www.cve.org/CVERecord?id=CVE-2005-2188* |
| **CVE-2003-0174** | LDAP service does not verify if a particular attribute was set by the LDAP server<br>*https://www.cve.org/CVERecord?id=CVE-2003-0174* |
| **CVE-1999-1549** | product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements.<br>*https://www.cve.org/CVERecord?id=CVE-1999-1549* |
| **CVE-2003-0981** | product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.<br>*https://www.cve.org/CVERecord?id=CVE-2003-0981* |

## 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 | 949 | SFP Secondary Cluster: Faulty Endpoint Authentication | 888 | 2416 |
| MemberOf | C | 1353 | OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures | 1344 | 2515 |
| MemberOf | C | 1382 | ICS Operations (& Maintenance): Emerging Energy Technologies | 1358 | 2538 |
| MemberOf | C | 1411 | Comprehensive Categorization: Insufficient Verification of Data Authenticity | 1400 | 2559 |

## Notes

### Maintenance

This entry has some significant overlap with other CWE entries and may need some clarification. See terminology notes.

### Terminology

The "Origin Validation Error" term was originally used in a 1995 thesis [REF-324]. Although not formally defined, an issue is considered to be an origin validation error if either (1) "an object [accepts] input from an unauthorized subject," or (2) "the system [fails] to properly or completely authenticate a subject." A later section says that an origin validation error can occur when the system (1) "does not properly authenticate a user or process" or (2) "does not properly authenticate the shared data or libraries." The only example provided in the thesis (covered by OSVDB:57615) involves a setuid program running command-line arguments without dropping privileges. So, this definition (and its examples in the thesis) effectively cover other weaknesses such as CWE-287 (Improper Authentication), CWE-285 (Improper Authorization), and CWE-250

(Execution with Unnecessary Privileges). There appears to be little usage of this term today, except in the SecurityFocus vulnerability database, where the term is used for a variety of issues, including web-browser problems that allow violation of the Same Origin Policy and improper validation of the source of an incoming message.

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Origin Validation Error |
| ISA/IEC 62443 | Part 3-3 | | Req SR 2.12 RE(1) |
| ISA/IEC 62443 | Part 4-1 | | Req SD-1 |
| ISA/IEC 62443 | Part 4-1 | | Req SR-2 |
| ISA/IEC 62443 | Part 4-1 | | Req SVV-1 |
| ISA/IEC 62443 | Part 4-2 | | Req CR 2.12 RE(1) |
| ISA/IEC 62443 | Part 4-2 | | Req CR 3.1 RE(1) |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 21 | Exploitation of Trusted Identifiers |
| 59 | Session Credential Falsification through Prediction |
| 60 | Reusing Session IDs (aka Session Replay) |
| 75 | Manipulating Writeable Configuration Files |
| 76 | Manipulating Web Input to File System Calls |
| 89 | Pharming |
| 111 | JSON Hijacking (aka JavaScript Hijacking) |
| 141 | Cache Poisoning |
| 142 | DNS Cache Poisoning |
| 160 | Exploit Script-Based APIs |
| 384 | Application API Message Manipulation via Man-in-the-Middle |
| 385 | Transaction or Event Tampering via Application API Manipulation |
| 386 | Application API Navigation Remapping |
| 387 | Navigation Remapping To Propagate Malicious Content |
| 388 | Application API Button Hijacking |
| 510 | SaaS User Request Forgery |

### References

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

## CWE-347: Improper Verification of Cryptographic Signature

**Weakness ID :** 347
**Structure :** Simple
**Abstraction :** Base

### Description

The product does not verify, or incorrectly verifies, the cryptographic signature for data.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | 858 |

*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 | 858 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1013 | Encrypt Data | 2449 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1214 | Data Integrity Issues | 2498 |
| MemberOf | C | 310 | Cryptographic Issues | 2339 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Integrity<br>Confidentiality | Gain Privileges or Assume Identity<br>Modify Application Data<br>Execute Unauthorized Code or Commands<br><br>*An attacker could gain access to sensitive data and possibly execute unauthorized 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*

## Demonstrative Examples

### Example 1:

In the following code, a JarFile object is created from a downloaded file.

*Example Language: Java* *(Bad)*

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

The JAR file that was potentially downloaded from an untrusted source is created without verifying the signature (if present). An alternate constructor that accepts a boolean verify parameter should be used instead.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-1796** | Does not properly verify signatures for "trusted" entities.<br>*https://www.cve.org/CVERecord?id=CVE-2002-1796* |

| Reference | Description |
|---|---|
| **CVE-2005-2181** | Insufficient verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2005-2181* |
| **CVE-2005-2182** | Insufficient verification allows spoofing. *https://www.cve.org/CVERecord?id=CVE-2005-2182* |
| **CVE-2002-1706** | Accepts a configuration file without a Message Integrity Check (MIC) signature. *https://www.cve.org/CVERecord?id=CVE-2002-1706* |

### 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 | 859 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) | 844 | 2390 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 959 | SFP Secondary Cluster: Weak Cryptography | 888 | 2419 |
| MemberOf | C | 1346 | OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures | 1344 | 2509 |
| MemberOf | C | 1402 | Comprehensive Categorization: Encryption | 1400 | 2548 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Improperly Verified Signature |
| The CERT Oracle Secure Coding Standard for Java (2011) | SEC06-J | | Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar |
| ISA/IEC 62443 | Part 3-3 | | Req SR 1.9 |
| ISA/IEC 62443 | Part 4-1 | | Req SM-6 |
| ISA/IEC 62443 | Part 4-2 | | Req EDR 3.12 |
| ISA/IEC 62443 | Part 4-2 | | Req NDR 3.12 |
| ISA/IEC 62443 | Part 4-2 | | Req HDR 3.12 |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 463 | Padding Oracle Crypto Attack |
| 475 | Signature Spoofing by Improper Validation |

## CWE-348: Use of Less Trusted Source

**Weakness ID :** 348
**Structure :** Simple
**Abstraction :** Base

### Description

The product has two different sources of the same data or information, but it uses the source that has less support for verification, is less trusted, or is less resistant to attack.

### Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 | | 345 | Insufficient Verification of Data Authenticity | 858 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1214 | Data Integrity Issues | 2498 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control | Bypass Protection Mechanism<br>Gain Privileges or Assume Identity<br>*An attacker could utilize the untrusted data source to bypass protection mechanisms and gain access to sensitive data.* | |

### Demonstrative Examples

**Example 1:**

This code attempts to limit the access of a page to certain IP Addresses. It checks the 'HTTP_X_FORWARDED_FOR' header in case an authorized user is sending the request through a proxy.

*Example Language: PHP* *(Bad)*

```
$requestingIP = '0.0.0.0';
if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)) {
  $requestingIP = $_SERVER['HTTP_X_FORWARDED_FOR'];
else{
  $requestingIP = $_SERVER['REMOTE_ADDR'];
}
if(in_array($requestingIP,$ipAllowlist)){
  generatePage();
  return;
}
else{
  echo "You are not authorized to view this page";
  return;
}
```

The 'HTTP_X_FORWARDED_FOR' header can be user controlled and so should never be trusted. An attacker can falsify the header to gain access to the page.

This fixed code only trusts the 'REMOTE_ADDR' header and so avoids the issue:

*Example Language: PHP* *(Good)*

```
$requestingIP = '0.0.0.0';
if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)) {
  echo "This application cannot be accessed through a proxy.";
  return;
else{
  $requestingIP = $_SERVER['REMOTE_ADDR'];
}
...
```

Be aware that 'REMOTE_ADDR' can still be spoofed. This may seem useless because the server will send the response to the fake address and not the attacker, but this may still be enough to conduct an attack. For example, if the generatePage() function in this code is resource intensive, an attacker could flood the server with fake requests using an authorized IP and consume significant resources. This could be a serious DoS attack even though the attacker would never see the page's sensitive content.

### Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-0860** | Product uses IP address provided by a client, instead of obtaining it from the packet headers, allowing easier spoofing. *https://www.cve.org/CVERecord?id=CVE-2001-0860* |
| **CVE-2004-1950** | Web product uses the IP address in the X-Forwarded-For HTTP header instead of a server variable that uses the connecting IP address, allowing filter bypass. *https://www.cve.org/CVERecord?id=CVE-2004-1950* |
| **CVE-2001-0908** | Product logs IP address specified by the client instead of obtaining it from the packet headers, allowing information hiding. *https://www.cve.org/CVERecord?id=CVE-2001-0908* |
| **CVE-2006-1126** | PHP application uses IP address from X-Forwarded-For HTTP header, instead of REMOTE_ADDR. *https://www.cve.org/CVERecord?id=CVE-2006-1126* |

### MemberOf Relationships

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

| Nature | Type | ID | Name | V | Page |
|---|---|---|---|---|---|
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 975 | SFP Secondary Cluster: Architecture | 888 | 2427 |
| MemberOf | C | 1411 | Comprehensive Categorization: Insufficient Verification of Data Authenticity | 1400 | 2559 |

### Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Use of Less Trusted Source |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 73 | User-Controlled Filename |
| 76 | Manipulating Web Input to File System Calls |
| 85 | AJAX Footprinting |
| 141 | Cache Poisoning |
| 142 | DNS Cache Poisoning |

## CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data

**Weakness ID :** 349
**Structure :** Simple
**Abstraction :** Base

### Description

The product, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were 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 | C | 345 | Insufficient Verification of Data Authenticity | 858 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1019 | Validate Inputs | 2454 |

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

| Nature | Type | ID | Name | Page |
|---|---|---|---|---|
| MemberOf | C | 1214 | Data Integrity Issues | 2498 |

## Applicable Platforms

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

## Common Consequences

| Scope | Impact | Likelihood |
|---|---|---|
| Access Control<br>Integrity | Bypass Protection Mechanism<br>Modify Application Data<br>*An attacker could package untrusted data with trusted data to bypass protection mechanisms to gain access to and possibly modify sensitive data.* | |

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2002-0018** | Does not verify that trusted entity is authoritative for all entities in its response.<br>*https://www.cve.org/CVERecord?id=CVE-2002-0018* |
| **CVE-2006-5462** | use of extra data in a signature allows certificate signature forging<br>*https://www.cve.org/CVERecord?id=CVE-2006-5462* |

## 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 | 860 | The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV) | 844 | 2391 |
| MemberOf | V | 884 | CWE Cross-section | 884 | 2588 |
| MemberOf | C | 977 | SFP Secondary Cluster: Design | 888 | 2428 |
| MemberOf | C | 1150 | SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV) | 1133 | 2473 |
| MemberOf | C | 1365 | ICS Communications: Unreliability | 1358 | 2523 |
| MemberOf | C | 1366 | ICS Communications: Frail Security in Protocols | 1358 | 2524 |
| MemberOf | C | 1373 | ICS Engineering (Construction/Deployment): Trust Model Problems | 1358 | 2531 |
| MemberOf | C | 1411 | Comprehensive Categorization: Insufficient Verification of Data Authenticity | 1400 | 2559 |

## Taxonomy Mappings

| Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
|---|---|---|---|
| PLOVER | | | Untrusted Data Appended with Trusted Data |
| The CERT Oracle Secure Coding Standard for Java (2011) | ENV01-J | | Place all security-sensitive code in a single JAR and sign and seal it |

### Related Attack Patterns

| CAPEC-ID | Attack Pattern Name |
|---|---|
| 75 | Manipulating Writeable Configuration Files |
| 141 | Cache Poisoning |
| 142 | DNS Cache Poisoning |



## CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action

**Weakness ID :** 350
**Structure :** Simple
**Abstraction :** Variant

### Description

The product performs reverse DNS resolution on an IP address to obtain the hostname and make a security decision, but it does not properly ensure that the IP address is truly associated with the hostname.

### Extended Description

Since DNS names can be easily spoofed or misreported, and it may be difficult for the product to detect if a trusted DNS server has been compromised, DNS names do not constitute a valid authentication mechanism.

When the product performs a reverse DNS resolution for an IP address, if an attacker controls the DNS server for that IP address, then the attacker can cause the server to return an arbitrary hostname. As a result, the attacker may be able to bypass authentication, cause the wrong hostname to be recorded in log files to hide activities, or perform other attacks.

Attackers can spoof DNS names by either (1) compromising a DNS server and modifying its records (sometimes called DNS cache poisoning), or (2) having legitimate control over a DNS server associated with their IP address.

### Relationships

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 | 1723 |
| ChildOf | B | 290 | Authentication Bypass by Spoofing | 712 |
| CanPrecede | C | 923 | Improper Restriction of Communication Channel to Intended Endpoints | 1836 |

### Applicable Platforms

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

### Common Consequences

| Scope | Impact | Likelihood |
| --- | --- | --- |
| Access Control | Gain Privileges or Assume Identity<br>Bypass Protection Mechanism<br>*Malicious users can fake authentication information by providing false DNS information.* | |

## Detection Methods

### Automated Static Analysis

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

*Effectiveness = High*

## Potential Mitigations

### Phase: Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

### Phase: Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

## Demonstrative Examples

### Example 1:

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.

**Example 2:**

In these examples, a connection is established if a request is made by a trusted host.

*Example Language: C* *(Bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  h=gethostbyname(inet_ntoa(cliAddr.sin_addr));
  if (h->h_name==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

*Example Language: Java* *(Bad)*

```
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress IPAddress = rp.getAddress();
  int port = rp.getPort();
  if ((rp.getHostName()==...) & (in==...)) {
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port);
    outSock.send(sp);
  }
}
```

These examples check if a request is from a trusted host before responding to a request, but the code only verifies the hostname as stored in the request packet. An attacker can spoof the hostname, thus impersonating a trusted client.

## Observed Examples

| Reference | Description |
|---|---|
| **CVE-2001-1488** | Does not do double-reverse lookup to prevent DNS spoofing. *https://www.cve.org/CVERecord?id=CVE-2001-1488* |
| **CVE-2001-1500** | Does not verify reverse-resolved hostnames in DNS. *https://www.cve.org/CVERecord?id=CVE-2001-1500* |
| **CVE-2000-1221** | Authentication bypass using spoofed reverse-resolved DNS hostnames. *https://www.cve.org/CVERecord?id=CVE-2000-1221* |
| **CVE-2002-0804** | Authentication bypass using spoofed reverse-resolved DNS hostnames. *https://www.cve.org/CVERecord?id=CVE-2002-0804* |
| **CVE-2001-1155** | Filter does not properly check the result of a reverse DNS lookup, which could allow remote attackers to bypass intended access restrictions via DNS spoofing. *https://www.cve.org/CVERecord?id=CVE-2001-1155* |
| **CVE-2004-0892** | Reverse DNS lookup used to spoof trusted content in intermediary. |