

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-10		

References

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

CWE-1093: Excessively Complex Data Representation

Weakness ID : 1093

Structure : Simple

Abstraction : Class

Description

The product uses an unnecessarily complex internal representation for its data structures or interrelationships between those structures.

Extended Description

This issue makes it more difficult to understand or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1549
ParentOf	B	1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements	1877
ParentOf	B	1055	Multiple Inheritance from Concrete Classes	1890
ParentOf	B	1074	Class with Excessively Deep Inheritance	1914
ParentOf	B	1086	Class with Excessive Number of Child Classes	1926

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-1094: Excessive Index Range Scan for a Data Resource

Weakness ID : 1094

Structure : Simple

Abstraction : Base

Description

The product contains an index range scan for a large data table, but the scan can cover a large number of rows.

Extended Description

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "excessive index range" may vary for each product or developer, CISQ recommends a threshold of 1000000 table rows and a threshold of 10 for the index range.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	405	Asymmetric Resource Consumption (Amplification)	986

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

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

Weakness Ordinalities




Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1132	CISQ Quality Measures (2016) - Performance Efficiency	1128	2443
MemberOf		1309	CISQ Quality Measures - Efficiency	1305	2486
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-7		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < <https://www.omg.org/spec/ASCPEM/> >.2023-04-07.

CWE-1095: Loop Condition Value Update within the Loop

Weakness ID : 1095

Structure : Simple

Abstraction : Base

Description

The product uses a loop with a control flow condition based on a value that is updated within the body of the loop.

Extended Description

This issue makes it more difficult to understand and/or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	View	Page
MemberOf		1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf		1307	CISQ Quality Measures - Maintainability	1305	2484
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-5		

References

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

CWE-1096: Singleton Class Instance Creation without Proper Locking or Synchronization

Weakness ID : 1096

Structure : Simple

Abstraction : Variant

Description

The product implements a Singleton design pattern but does not use appropriate locking or other synchronization mechanism to ensure that the singleton class is only instantiated once.

Extended Description

This issue can prevent the product from running reliably, e.g. by making the instantiation process non-thread-safe and introducing deadlock (CWE-833) or livelock conditions. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		820	Missing Synchronization	1720

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1448

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

Nature	Type	ID	Name	Page
ChildOf		662	Improper Synchronization	1448

Weakness Ordinalities



Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-12		

References

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

CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element

Weakness ID : 1097

Structure : Simple

Abstraction : Base

Description

The product uses a storable data element that does not have all of the associated functions or methods that are necessary to support comparison.

Extended Description


For example, with Java, a class that is made persistent requires both hashCode() and equals() methods to be defined.

This issue can prevent the product from running reliably, due to incorrect or unexpected comparison results. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1916

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

Nature	Type	ID	Name	Page
ChildOf		595	Comparison of Object References Instead of Object Contents	1334

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-4		

References

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

CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element

Weakness ID : 1098

Structure : Simple

Abstraction : Base

Description

The code contains a data element with a pointer that does not have an associated copy or constructor method.


Extended Description

This issue can prevent the product from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1916

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities





Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1129	CISQ Quality Measures (2016) - Reliability	1128	2440
MemberOf		1306	CISQ Quality Measures - Reliability	1305	2483
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-6		

References

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

CWE-1099: Inconsistent Naming Conventions for Identifiers

Weakness ID : 1099

Structure : Simple

Abstraction : Base

Description

The product's code, documentation, or other artifacts do not consistently use the same naming conventions for variables, callables, groups of related callables, I/O capabilities, data types, file names, or similar types of elements.

Extended Description

This issue makes it more difficult to understand and/or maintain the product due to inconsistencies, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1100: Insufficient Isolation of System-Dependent Functions

Weakness ID : 1100

Structure : Simple

Abstraction : Base

Description

The product or code does not isolate system-dependent functionality into separate standalone modules.


Extended Description

This issue makes it more difficult to maintain and/or port the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1061	Insufficient Encapsulation	1898

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1101: Reliance on Runtime Component in Generated Code

Weakness ID : 1101

Structure : Simple

Abstraction : Base

Description

The product uses automatically-generated code that cannot be executed without a specific runtime support component.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1549

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

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Scope	Impact	Likelihood
-------	--------	------------

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1102: Reliance on Machine-Dependent Data Representation

Weakness ID : 1102

Structure : Simple

Abstraction : Base

Description

The code uses a data representation that relies on low-level data representation or constructs that may vary across different processors, physical machines, OSes, or other physical components.



Extended Description

This issue makes it more difficult to maintain and/or port the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1582
PeerOf		1105	Insufficient Encapsulation of Machine-Dependent Functionality	1945

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities


Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1103: Use of Platform-Dependent Third Party Components

Weakness ID : 1103

Structure : Simple

Abstraction : Base

Description

The product relies on third-party components that do not provide equivalent functionality across all desirable platforms.


Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1582

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1104: Use of Unmaintained Third Party Components

Weakness ID : 1104

Structure : Simple

Abstraction : Base

Description

The product relies on third-party components that are not actively supported or maintained by the original developer or a trusted proxy for the original developer.

Extended Description

Reliance on components that are no longer maintained can make it difficult or impossible to fix significant bugs, vulnerabilities, or quality issues. In effect, unmaintained code can become obsolete.

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1357	Reliance on Insufficiently Trustworthy Component	2254

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

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

Weakness Ordinalities

Indirect :

Applicable Platforms

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

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

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1352	OWASP Top Ten 2021 Category A06:2021 - Vulnerable and Outdated Components	1344	2494
MemberOf		1368	ICS Dependencies (& Architecture): External Digital Systems	1358	2505
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2544

References

[REF-1212]"A06:2021 - Vulnerable and Outdated Components". 2021 September 4. OWASP. <
https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ >.

CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality

Weakness ID : 1105

Structure : Simple

Abstraction : Base

Description

The product or code uses machine-dependent functionality, but it does not sufficiently encapsulate or isolate this functionality from the rest of the code.





Extended Description

This issue makes it more difficult to port or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1061	Insufficient Encapsulation	1898
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1582
ParentOf		188	Reliance on Data/Memory Layout	470
PeerOf		1102	Reliance on Machine-Dependent Data Representation	1942

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

Nature	Type	ID	Name	Page
MemberOf		1227	Encapsulation Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Demonstrative Examples

Example 1:

In this example function, the memory address of variable b is derived by adding 1 to the address of variable a. This derived address is then used to assign the value 0 to b.

Example Language: C

(Bad)

```
void example() {  
    char a;  
    char b;  
    *(&a + 1) = 0;  
}
```

Here, b may not be one byte past a. It may be one byte in front of a. Or, they may have three bytes between them because they are aligned on 32-bit boundaries.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1106: Insufficient Use of Symbolic Constants

Weakness ID : 1106

Structure : Simple

Abstraction : Base

Description

The source code uses literal constants that may need to change or evolve over time, instead of using symbolic constants.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1107: Insufficient Isolation of Symbolic Constant Definitions

Weakness ID : 1107

Structure : Simple

Abstraction : Base

Description

The source code uses symbolic constants, but it does not sufficiently place the definitions of these constants into a more centralized or isolated location.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1108: Excessive Reliance on Global Variables

Weakness ID : 1108

Structure : Simple

Abstraction : Base

Description

The code is structured in a way that relies too much on using or setting global variables throughout various points in the code, instead of preserving the associated information in a narrower, more local context.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1076	Insufficient Adherence to Expected Conventions	1916

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Detection Methods

Automated Static Analysis

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

Effectiveness = High

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1109: Use of Same Variable for Multiple Purposes

Weakness ID : 1109

Structure : Simple

Abstraction : Base

Description

The code contains a callable, block, or other code element in which the same variable is used to control more than one unique task or store more than one instance of data.

Extended Description

Use of the same variable for multiple purposes can make it more difficult for a person to read or understand the code, potentially hiding other quality issues.

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1110: Incomplete Design Documentation

Weakness ID : 1110
Structure : Simple
Abstraction : Base

Description

The product's design documentation does not adequately describe control flow, data flow, system initialization, relationships between tasks, components, rationales, or other important aspects of the design.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1894

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

Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2480

Weakness Ordinalities

Indirect :

Applicable Platforms

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

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

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2511
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1111: Incomplete I/O Documentation

Weakness ID : 1111

Structure : Simple

Abstraction : Base

Description

The product's documentation does not adequately define inputs, outputs, or system/software interfaces.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1894

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



Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2480

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1375	ICS Engineering (Construction/Deployment): Gaps in Details/Data	1358	2511
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1112: Incomplete Documentation of Program Execution

Weakness ID : 1112

Structure : Simple

Abstraction : Base

Description

The document does not fully define all mechanisms that are used to control or influence how product-specific programs are executed.

Extended Description

This includes environmental variables, configuration files, registry keys, command-line switches or options, or system settings.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1894

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

Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2480

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1113: Inappropriate Comment Style

Weakness ID : 1113

Structure : Simple

Abstraction : Base

Description

The source code uses comment styles or formats that are inconsistent or do not follow expected standards for the product.

Extended Description

This issue makes it more difficult to maintain the product due to insufficient legibility, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1114: Inappropriate Whitespace Style

Weakness ID : 1114

Structure : Simple
Abstraction : Base

Description

The source code contains whitespace that is inconsistent across the code or does not follow expected standards for the product.

Extended Description

This issue makes it more difficult to understand and maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1115: Source Code Element without Standard Prologue

Weakness ID : 1115
Structure : Simple
Abstraction : Base

Description

The source code contains elements such as source files that do not consistently provide a prologue or header that has been standardized for the project.

Extended Description

The lack of a prologue can make it more difficult to accurately and quickly understand the associated code. Standard prologues or headers may contain information such as module name, version number, author, date, purpose, function, assumptions, limitations, accuracy considerations, etc.

This issue makes it more difficult to maintain the product due to insufficient analyzability, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1116: Inaccurate Comments

Weakness ID : 1116

Structure : Simple

Abstraction : Base

Description

The source code contains comments that do not accurately describe or explain aspects of the portion of the code with which the comment is associated.

Extended Description

When a comment does not accurately reflect the associated code elements, this can introduce confusion to a reviewer (due to inconsistencies) or make it more difficult and less efficient to validate that the code is implementing the intended behavior correctly.

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Potential Mitigations

Phase: Implementation

Verify that each comment accurately reflects what is intended to happen during execution of the code.

Demonstrative Examples

Example 1:

In the following Java example the code performs a calculation to determine how much medicine to administer. A comment is provided to give insight into what the calculation should be doing. Unfortunately the comment does not match the actual code and thus leaves the reader to wonder which is correct.

Example Language: Java

(Bad)

```
public class Main {
    public static void main(String[] args) {
        int pt_weight = 83;
        int mg_per_kg = 3;
        int daily_dose = 0;
        // Add the patient weight and Mg/Kg to calculate the correct daily dose
        daily_dose = pt_weight * mg_per_kg;
        return dosage;
    }
}
```

In the correction below, the code functionality has been verified, and the comment has been corrected to reflect the proper calculation.

Example Language: Java

(Good)

```

public class Main {
    public static void main(String[] args) {
        int pt_weight = 83;
        int mg_per_kg = 3;
        int daily_dose = 0;
        // Multiply the patient weight and Mg/Kg to calculate the correct daily dose
        daily_dose = pt_weight * mg_per_kg;
        return dosage;
    }
}

```

Note that in real-world code, these values should be validated to disallow negative numbers, prevent integer overflow, etc.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENTS >.2023-04-07.

CWE-1117: Callable with Insufficient Behavioral Summary

Weakness ID : 1117

Structure : Simple

Abstraction : Base

Description

The code contains a function or method whose signature and/or associated inline documentation does not sufficiently describe the callable's inputs, outputs, side effects, assumptions, or return codes.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	1918

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.2023-04-07.

CWE-1118: Insufficient Documentation of Error Handling Techniques

Weakness ID : 1118

Structure : Simple

Abstraction : Base

Description

The documentation does not sufficiently describe the techniques that are used for error handling, exception processing, or similar mechanisms.

Extended Description

Documentation may need to cover error handling techniques at multiple layers, such as module, executable, compilable code unit, or callable.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1059	Insufficient Technical Documentation	1894

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

Nature	Type	ID	Name	Page
MemberOf		1225	Documentation Issues	2480

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1119: Excessive Use of Unconditional Branching

Weakness ID : 1119

Structure : Simple

Abstraction : Base

Description

The code uses too many unconditional branches (such as "goto").

Extended Description

This issue makes it more difficult to understand and/or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_>.2023-04-07.

CWE-1120: Excessive Code Complexity

Weakness ID : 1120

Structure : Simple

Abstraction : Class

Description

The code is too complex, as calculated using a well-defined, quantitative measure.

Extended Description

This issue makes it more difficult to understand and/or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

This issue can make the product perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1549
ParentOf	B	1047	Modules with Circular Dependencies	1882
ParentOf	B	1056	Invokable Control Element with Variadic Parameters	1891
ParentOf	B	1060	Excessive Number of Inefficient Server-Side Data Accesses	1897
ParentOf	B	1064	Invokable Control Element with Signature Containing an Excessive Number of Parameters	1902
ParentOf	B	1075	Unconditional Control Flow Transfer outside of Switch Block	1915
ParentOf	B	1080	Source Code File with Excessive Number of Lines of Code	1920
ParentOf	B	1095	Loop Condition Value Update within the Loop	1935
ParentOf	B	1119	Excessive Use of Unconditional Branching	1959
ParentOf	B	1121	Excessive McCabe Cyclomatic Complexity	1961
ParentOf	B	1122	Excessive Halstead Complexity	1962
ParentOf	B	1123	Excessive Use of Self-Modifying Code	1963
ParentOf	B	1124	Excessively Deep Nesting	1964
ParentOf	B	1125	Excessive Attack Surface	1965

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	
Other	Reduce Performance	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-1121: Excessive McCabe Cyclomatic Complexity

Weakness ID : 1121

Structure : Simple

Abstraction : Base

Description

The code contains McCabe cyclomatic complexity that exceeds a desirable maximum.

Extended Description

This issue makes it more difficult to understand and/or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures (2016) - Maintainability	1128	2441
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-11		

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

[REF-964]Wikipedia. "Cyclomatic Complexity". 2018 April 3. < https://en.wikipedia.org/wiki/Cyclomatic_complexity >.

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

CWE-1122: Excessive Halstead Complexity

Weakness ID : 1122

Structure : Simple

Abstraction : Base

Description

The code is structured in a way that a Halstead complexity measure exceeds a desirable maximum.

Extended Description

A variety of Halstead complexity measures exist, such as program vocabulary size or volume.

This issue makes it more difficult to understand and/or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

[REF-965]Wikipedia. "Halstead complexity measures". 2017 November 2. < https://en.wikipedia.org/wiki/Halstead_complexity_measures >.

CWE-1123: Excessive Use of Self-Modifying Code

Weakness ID : 1123

Structure : Simple

Abstraction : Base

Description

The product uses too much self-modifying code.

Extended Description

This issue makes it more difficult to understand or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1124: Excessively Deep Nesting

Weakness ID : 1124

Structure : Simple

Abstraction : Base

Description

The code contains a callable or other code grouping in which the nesting / branching is too deep.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	G	1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT >.2023-04-07.

CWE-1125: Excessive Attack Surface

Weakness ID : 1125

Structure : Simple

Abstraction : Base

Description

The product has an attack surface whose quantitative measurement exceeds a desirable maximum.

Extended Description

Originating from software security, an "attack surface" measure typically reflects the number of input points and output points that can be utilized by an untrusted party, i.e. a potential attacker. A larger attack surface provides more places to attack, and more opportunities for developers to introduce weaknesses. In some cases, this measure may reflect other aspects of quality besides security; e.g., a product with many inputs and outputs may require a large number of tests in order to improve code coverage.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1960

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

Nature	Type	ID	Name	Page
MemberOf		1226	Complexity Issues	2481

Weakness Ordinalities

Indirect :

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-966]Pratyusa Manadhata. "An Attack Surface Metric". 2008 November. < <http://reports-archive.adm.cs.cmu.edu/anon/2008/CMU-CS-08-152.pdf> >.

[REF-967]Pratyusa Manadhata and Jeannette M. Wing. "Measuring a System's Attack Surface". 2004. < <http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/ManadhataWing04.pdf> >.

CWE-1126: Declaration of Variable with Unnecessarily Wide Scope

Weakness ID : 1126

Structure : Simple

Abstraction : Base

Description

The source code declares a variable in one scope, but the variable is only used within a narrower scope.

Extended Description

This issue makes it more difficult to understand and/or maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		710	Improper Adherence to Coding Standards	1549

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-1127: Compilation with Insufficient Warnings or Errors

Weakness ID : 1127

Structure : Simple

Abstraction : Base

Description

The code is compiled without sufficient warnings enabled, which may prevent the detection of subtle bugs or quality issues.

Extended Description

This issue makes it more difficult to maintain the product, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	I P	710	Improper Adherence to Coding Standards	1549

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

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

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-1164: Irrelevant Code

Weakness ID : 1164

Structure : Simple

Abstraction : Class

Description

The product contains code that is not essential for execution, i.e. makes no state changes and has no side effects that alter data or control flow, such that removal of the code would have no impact to functionality or correctness.

Extended Description

Irrelevant code could include dead code, initialization that is not used, empty blocks, code that could be entirely removed due to optimization, etc.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1549
ParentOf	Y	107	Struts: Unused Validation Form	259
ParentOf	Y	110	Struts: Validator Without Form Field	264
ParentOf	B	561	Dead Code	1275
ParentOf	B	563	Assignment to Variable without Use	1280
ParentOf	B	1071	Empty Code Block	1910

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	
Other	Reduce Performance	

Demonstrative Examples

Example 1:

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

Example Language: C++

(Bad)

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

Example 2:

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

Example Language: C

(Bad)

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

Observed Examples

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

CWE-1173: Improper Use of Validation Framework

Weakness ID : 1173

Structure : Simple

Abstraction : Base

Description

The product does not use, or incorrectly uses, an input validation framework that is provided by the source language or an independent library.









Extended Description

Many modern coding languages provide developers with input validation frameworks to make the task of input validation easier and less error-prone. These frameworks will automatically check all input against specified criteria and direct execution to error handlers when invalid input is received. The improper use (i.e., an incorrect implementation or missing altogether) of these frameworks is not directly exploitable, but can lead to an exploitable condition if proper input validation is not performed later in the product. Not using provided input validation frameworks can also hurt the maintainability of code as future developers may not recognize the downstream input validation being used in the place of the validation framework.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
ParentOf		102	Struts: Duplicate Validation Forms	246
ParentOf		105	Struts: Form Field Without Validator	253
ParentOf		106	Struts: Plug-in Framework not in Use	256
ParentOf		108	Struts: Unvalidated Action Form	261
ParentOf		109	Struts: Validator Turned Off	263
ParentOf		554	ASP.NET Misconfiguration: Not Using Input Validation Framework	1269
ParentOf		1174	ASP.NET Misconfiguration: Improper Model Validation	1970

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478

Weakness Ordinalities

Indirect :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

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

Detection Methods

Automated Static Analysis

Some instances of improper input validation can be detected using automated static analysis. A static analysis tool might allow the user to specify which application-specific methods or functions perform input validation; the tool might also have built-in knowledge of validation frameworks such as Struts. The tool may then suppress or de-prioritize any associated warnings. This allows the analyst to focus on areas of the software in which input validation does not appear to be present. Except in the cases described in the previous paragraph, automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Potential Mitigations

Phase: Implementation

Properly use provided input validation frameworks.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design	1344	2491
MemberOf	C	1406	Comprehensive Categorization: Improper Input Validation	1400	2531

CWE-1174: ASP.NET Misconfiguration: Improper Model Validation

Weakness ID : 1174

Structure : Simple

Abstraction : Variant

Description

The ASP.NET application does not use, or incorrectly uses, the model validation framework.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	E	1173	Improper Use of Validation Framework	1969

Weakness Ordinalities

Indirect :

Applicable Platforms




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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State <i>Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.</i>	

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1349	OWASP Top Ten 2021 Category A05:2021 - Security Misconfiguration	1344	2493
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

CWE-1176: Inefficient CPU Computation

Weakness ID : 1176

Structure : Simple

Abstraction : Class

Description

The product performs CPU computations using algorithms that are not as efficient as they could be for the needs of the developer, i.e., the computations can be optimized further.







Extended Description

This issue can make the product perform more slowly, possibly in ways that are noticeable to the users. If an attacker can influence the amount of computation that must be performed, e.g. by triggering worst-case complexity, then this performance problem might introduce a vulnerability.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	986
ParentOf		1042	Static Member Data Element outside of a Singleton Class Element	1876
ParentOf		1046	Creation of Immutable Text Using String Concatenation	1881
ParentOf		1049	Excessive Data Query Operations in a Large Data Table	1884
ParentOf		1063	Creation of Class Instance within a Static Code Block	1901
ParentOf		1067	Excessive Execution of Sequential Searches of Data Resource	1905

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	
Other	Reduce Performance	

Observed Examples

Reference	Description
CVE-2022-37734	Chain: lexer in Java-based GraphQL server does not enforce maximum of tokens early enough (CWE-696), allowing excessive CPU consumption (CWE-1176) https://www.cve.org/CVERecord?id=CVE-2022-37734

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

References

[REF-1008]Wikipedia. "Computational complexity theory". < https://en.wikipedia.org/wiki/Computational_complexity_theory >.

CWE-1177: Use of Prohibited Code

Weakness ID : 1177

Structure : Simple

Abstraction : Class

Description

The product uses a function, library, or third party component that has been explicitly prohibited, whether by the developer or the customer.

Extended Description

The developer - or customers - may wish to restrict or eliminate use of a function, library, or third party component for any number of reasons, including real or suspected vulnerabilities; difficulty to use securely; export controls or license requirements; obsolete or poorly-maintained code; internal code being scheduled for deprecation; etc.

To reduce risk of vulnerabilities, the developer might maintain a list of "banned" functions that programmers must avoid using because the functions are difficult or impossible to use securely. This issue can also make the product more costly and difficult to maintain.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		710	Improper Adherence to Coding Standards	1549

Nature	Type	ID	Name	Page
ParentOf		242	Use of Inherently Dangerous Function	586
ParentOf		676	Use of Potentially Dangerous Function	1489

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Demonstrative Examples

Example 1:

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

Example Language: C

(Bad)

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

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

Example 2:

The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

Example Language: C

(Bad)

```
void manipulate_string(char * string){
    char buf[24];
    strcpy(buf, string);
    ...
}
```

However, the programmer does not ensure that the size of the data pointed to by string will fit in the local buffer and copies the data with the potentially dangerous strcpy() function. This may result in a buffer overflow condition if an attacker can influence the contents of the string parameter.

Observed Examples

Reference	Description
CVE-2007-1470	Library has multiple buffer overflows using sprintf() and strcpy() https://www.cve.org/CVERecord?id=CVE-2007-1470
CVE-2007-4004	FTP client uses inherently insecure gets() function and is setuid root on some systems, allowing buffer overflow https://www.cve.org/CVERecord?id=CVE-2007-4004

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

References

[REF-1009]Tim Rains. "Microsoft's Free Security Tools - banned.h". 2012 August 0. < <https://www.microsoft.com/en-us/security/blog/2012/08/30/microsofts-free-security-tools-banned-h/> >.2023-04-07.

[REF-1010]Michael Howard. "Microsoft's Free Security Tools - banned.h". 2011 June. < <https://www.microsoft.com/en-us/security/blog/2012/08/30/microsofts-free-security-tools-banned-h/> >.2023-04-07.

CWE-1188: Initialization of a Resource with an Insecure Default

Weakness ID : 1188

Structure : Simple

Abstraction : Base

Description

The product initializes or sets a resource with a default that is intended to be changed by the administrator, but the default is not secure.



Extended Description

Developers often choose default values that leave the product as open and easy to use as possible out-of-the-box, under the assumption that the administrator can (or should) change the default value. However, this ease-of-use comes at a cost when the default is insecure and the administrator does not change it.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1419	Incorrect Initialization of Resource	2280
ParentOf		453	Insecure Default Variable Initialization	1083

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1456

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

Nature	Type	ID	Name	Page
MemberOf		399	Resource Management Errors	2324
MemberOf		452	Initialization and Cleanup Errors	2327

Weakness Ordinalities

Primary :

Demonstrative Examples

Example 1:

This code attempts to login a user using credentials from a POST request:

Example Language: PHP

(Bad)

```
// $user and $pass automatically set from POST request
if (login_user($user,$pass)) {
    $authorized = true;
}
...
if ($authorized) {
    generatePage();
}
```

Because the `$authorized` variable is never initialized, PHP will automatically set `$authorized` to any value included in the POST request if `register_globals` is enabled. An attacker can send a POST request with an unexpected third value 'authorized' set to 'true' and gain authorized status without supplying valid credentials.

Here is a fixed version:

Example Language: PHP

(Good)

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$authorized = false;
if (login_user($user,$pass)) {
    $authorized = true;
}
...
```


This code avoids the issue by initializing the `$authorized` variable to false and explicitly retrieving the login credentials from the `$_POST` variable. Regardless, `register_globals` should never be enabled and is disabled by default in current versions of PHP.

Observed Examples

Reference	Description
CVE-2022-36349	insecure default variable initialization in BIOS firmware for a hardware board allows DoS https://www.cve.org/CVERecord?id=CVE-2022-36349
CVE-2022-42467	A generic database browser interface has a default mode that exposes a web server to the network, allowing queries to the database. https://www.cve.org/CVERecord?id=CVE-2022-42467

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

This entry improves organization of concepts under initialization. The typical CWE model is to cover "Missing" and "Incorrect" behaviors. Arguably, this entry could be named as "Incorrect" instead of "Insecure." This might be changed in the near future.

Related Attack Patterns

CAPEC-ID Attack Pattern Name

665 Exploitation of Thunderbolt Protection Flaws

CWE-1189: Improper Isolation of Shared Resources on System-on-a-Chip (SoC)**Weakness ID :** 1189**Structure :** Simple**Abstraction :** Base**Description**

The System-On-a-Chip (SoC) does not properly isolate shared resources between trusted and untrusted agents.





Extended Description

A System-On-a-Chip (SoC) has a lot of functionality, but it may have a limited number of pins or pads. A pin can only perform one function at a time. However, it can be configured to perform multiple different functions. This technique is called pin multiplexing. Similarly, several resources on the chip may be shared to multiplex and support different features or functions. When such resources are shared between trusted and untrusted agents, untrusted agents may be able to access the assets intended to be accessed only by the trusted agents.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1469
ChildOf		653	Improper Isolation or Compartmentalization	1437
ParentOf		1303	Non-Transparent Sharing of Microarchitectural Resources	2174
PeerOf		1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2225

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1331	Improper Isolation of Shared Resources in Network On Chip (NoC)	2225

Weakness Ordinalities

Primary :

Applicable Platforms

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	
	<i>If resources being used by a trusted user are shared with an untrusted user, the untrusted user may be able</i>	

Scope	Impact	Likelihood
	<i>to modify the functionality of the shared resource of the trusted user.</i>	
Integrity	Quality Degradation <i>The functionality of the shared resource may be intentionally degraded.</i>	

Detection Methods

Automated Dynamic Analysis

Pre-silicon / post-silicon: Test access to shared systems resources (memory ranges, control registers, etc.) from untrusted software to verify that the assets are not incorrectly exposed to untrusted agents. Note that access to shared resources can be dynamically allowed or revoked based on system flows. Security testing should cover such dynamic shared resource allocation and access control modification flows.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

When sharing resources, avoid mixing agents of varying trust levels. Untrusted agents should not share resources with trusted agents.

Demonstrative Examples

Example 1:

Consider the following SoC design. The Hardware Root of Trust (HROt) local SRAM is memory mapped in the core{0-N} address space. The HROt allows or disallows access to private memory ranges, thus allowing the sram to function as a mailbox for communication between untrusted and trusted HROt partitions.






We assume that the threat is from malicious software in the untrusted domain. We assume this software has access to the core{0-N} memory map and can be running at any privilege level on the untrusted cores. The capability of this threat in this example is communication to and from the mailbox region of SRAM modulated by the hrot_iface. To address this threat, information must not enter or exit the shared region of SRAM through hrot_iface when in secure or privileged mode.

Observed Examples

Reference	Description
CVE-2020-8698	Processor has improper isolation of shared resources allowing for information disclosure. https://www.cve.org/CVERecord?id=CVE-2020-8698
CVE-2019-6260	Baseboard Management Controller (BMC) device implements Advanced High-performance Bus (AHB) bridges that do not require authentication for arbitrary read and write access to the BMC's physical address space from the host, and possibly the network [REF-1138]. https://www.cve.org/CVERecord?id=CVE-2019-6260

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf		1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf		1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
124	Shared Resource Manipulation

References

[REF-1036]Ali Abbasi and Majid Hashemi. "Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack". 2016. < <https://www.blackhat.com/docs/eu-16/materials/eu-16-Abbasi-Ghost-In-The-PLC-Designing-An-Undetectable-Programmable-Logic-Controller-Rootkit-wp.pdf> >.

[REF-1138]Stewart Smith. "CVE-2019-6260: Gaining control of BMC from the host processor". 2019. < <https://www.flamingspork.com/blog/2019/01/23/cve-2019-6260:-gaining-control-of-bmc-from-the-host-processor/> >.

CWE-1190: DMA Device Enabled Too Early in Boot Phase

Weakness ID : 1190

Structure : Simple

Abstraction : Base

Description

The product enables a Direct Memory Access (DMA) capable device before the security configuration settings are established, which allows an attacker to extract data from or gain privileges on the product.

Extended Description

DMA is included in a number of devices because it allows data transfer between the computer and the connected device, using direct hardware access to read or write directly to main memory without any OS interaction. An attacker could exploit this to access secrets. Several virtualization-based mitigations have been introduced to thwart DMA attacks. These are usually configured/setup during boot time. However, certain IPs that are powered up before boot is complete (known as early boot IPs) may be DMA capable. Such IPs, if not trusted, could launch DMA attacks and gain access to assets that should otherwise be protected.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		696	Incorrect Behavior Order	1527

Applicable Platforms

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Modify Memory <i>DMA devices have direct write access to main memory and due to time of attack will be able to bypass OS or Bootloader access control.</i>	High

Potential Mitigations

Phase: Architecture and Design

Utilize an IOMMU to orchestrate IO access from the start of the boot process.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2469
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1038]"DMA attack". 2019 October 9. < https://en.wikipedia.org/wiki/DMA_attack >.

[REF-1039]A. Theodore Marketos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore and Robert N. M. Watson. "Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals". 2019 February 5. < https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_05A-1_Marketos_paper.pdf >.

[REF-1040]Maximillian Dornseif, Michael Becher and Christian N. Klein. "FireWire all your memory are belong to us". 2005. < <http://www.orkspace.net/secdocs/Conferences/CanSecWest/2005/0wn3d%20by%20an%20iPod%20-%20Firewire1394%20Issues.pdf> >.2023-04-07.

[REF-1041]Rory Breuk, Albert Spruyt and Adam Boileau. "Integrating DMA attacks in exploitation frameworks". 2012 February 0. < https://www.os3.nl/_media/2011-2012/courses/rp1/p14_report.pdf >.

[REF-1042]Maximillian Dornseif. "Owned by an iPod". 2004. < <https://web.archive.org/web/20060505224959/https://pacsec.jp/psj04/psj04-dornseif-e.ppt> >.2023-04-07.

[REF-1044]Dmytro Oleksiuk. "My aimful life". 2015 September 2. < <http://blog.cr4.sh/2015/09/breaking-uefi-security-with-software.html> >.

[REF-1046]A. Theodore Marketos and Adam Boileau. "Hit by a Bus:Physical Access Attacks with Firewire". 2006. < https://security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf >.

CWE-1191: On-Chip Debug and Test Interface With Improper Access Control

Weakness ID : 1191

Structure : Simple

Abstraction : Base

Description

The chip does not implement or does not correctly perform access control to check whether users are authorized to access internal registers and test modes through the physical debug/test interface.

Extended Description

A device's internal information may be accessed through a scan chain of interconnected internal registers, usually through a JTAG interface. The JTAG interface provides access to these registers in a serial fashion in the form of a scan chain for the purposes of debugging programs running on a device. Since almost all information contained within a device may be accessed over this interface, device manufacturers typically insert some form of authentication and authorization to prevent unintended use of this sensitive information. This mechanism is implemented in addition to on-chip protections that are already present.



If authorization, authentication, or some other form of access control is not implemented or not implemented correctly, a user may be able to bypass on-chip protection mechanisms through the debug interface.

Sometimes, designers choose not to expose the debug pins on the motherboard. Instead, they choose to hide these pins in the intermediate layers of the board. This is primarily done to work around the lack of debug authorization inside the chip. In such a scenario (without debug authorization), when the debug interface is exposed, chip internals are accessible to an attacker.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680
PeerOf		1263	Improper Physical Access Control	2085

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1299	Missing Protection Mechanism for Alternate Hardware Interface	2162

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	High
Confidentiality	Read Memory	High
Authorization	Execute Unauthorized Code or Commands	High
Integrity	Modify Memory	High
Integrity	Modify Application Data	High
Access Control	Bypass Protection Mechanism	High

Detection Methods

Dynamic Analysis with Manual Results Interpretation

Authentication and authorization of debug and test interfaces should be part of the architecture and design review process. Withholding of private register documentation from the debug and test interface public specification ("Security by obscurity") should not be considered as sufficient security.

Dynamic Analysis with Manual Results Interpretation

Dynamic tests should be done in the pre-silicon and post-silicon stages to verify that the debug and test interfaces are not open by default.

Fuzzing

Tests that fuzz Debug and Test Interfaces should ensure that no access without appropriate authentication and authorization is possible.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

If feasible, the manufacturer should disable the JTAG interface or implement authentication and authorization for the JTAG interface. If authentication logic is added, it should be resistant to timing attacks. Security-sensitive data stored in registers, such as keys, etc. should be cleared when entering debug mode.

Effectiveness = High

Demonstrative Examples

Example 1:

A home, WiFi-router device implements a login prompt which prevents an unauthorized user from issuing any commands on the device until appropriate credentials are provided. The credentials are protected on the device and are checked for strength against attack.

Example Language: Other

(Bad)

If the JTAG interface on this device is not hidden by the manufacturer, the interface may be identified using tools such as JTAGulator. If it is hidden but not disabled, it can be exposed by physically wiring to the board.

By issuing a "halt" command before the OS starts, the unauthorized user pauses the watchdog timer and prevents the router from restarting (once the watchdog timer would have expired). Having paused the router, an unauthorized user is able to execute code and inspect and modify data in the device, even extracting all of the router's firmware. This allows the user to examine the router and potentially exploit it.

JTAG is useful to chip and device manufacturers during design, testing, and production and is included in nearly every product. Without proper authentication and authorization, the interface may allow tampering with a product.

Example Language: Other

(Good)

In order to prevent exposing the debugging interface, manufacturers might try to obfuscate the JTAG interface or blow device internal fuses to disable the JTAG interface. Adding authentication and authorization to this interface makes use by unauthorized individuals much more difficult.

Example 2:

The following example code is a snippet from the JTAG wrapper module in the RISC-V debug module of the HACK@DAC'21 Openpiton SoC [REF-1355]. To make sure that the JTAG is accessed securely, the developers have included a primary authentication mechanism based on a password.

The developers employed a Finite State Machine (FSM) to implement this authentication. When a user intends to read from or write to the JTAG module, they must input a password.

In the subsequent state of the FSM module, the entered password undergoes Hash-based Message Authentication Code (HMAC) calculation using an internal HMAC submodule. Once the HMAC for the entered password is computed by the HMAC submodule, the FSM transitions to the next state, where it compares the computed HMAC with the expected HMAC for the password.

If the computed HMAC matches the expected HMAC, the FSM grants the user permission to perform read or write operations on the JTAG module. [REF-1352]

Example Language: Verilog

(Bad)

```
...
    PassChkValid: begin
        if(hashValid) begin
            if(exp_hash == pass_hash) begin
                pass_check = 1'b1;
            end else begin
                pass_check = 1'b0;
            end
            state_d = Idle;
        end else begin
            state_d = PassChkValid;
        end
    end
end
...
```

However, in the given vulnerable part of the code, the JTAG module has not defined a limitation for several continuous wrong password attempts. This omission poses a significant security risk, allowing attackers to carry out brute-force attacks without restrictions.

Without a limitation on wrong password attempts, an attacker can repeatedly guess different passwords until they gain unauthorized access to the JTAG module. This leads to various malicious activities, such as unauthorized read from or write to debug module interface.

To mitigate the mentioned vulnerability, developers need to implement a restriction on the number of consecutive incorrect password attempts allowed by the JTAG module, which can achieve by incorporating a mechanism that temporarily locks the module after a certain number of failed attempts.[REF-1353][REF-1354]

Example Language: Verilog

(Good)

```
...
case (state_q)
    Idle: begin
    ...
```



```

else if ( (dm::dtm_op_e'(dmi.op) == dm::DTM_PASS) && (miss_pass_check_cnt_q != 2'b11) )
begin
    state_d = Write;
    pass_mode = 1'b1;
end
...
end
...
PassChkValid: begin
    if(hashValid) begin
        if(exp_hash == pass_hash) begin
            pass_check = 1'b1;
        end else begin
            pass_check = 1'b0;
            miss_pass_check_cnt_d = miss_pass_check_cnt_q + 1
        end
        state_d = Idle;
    end else begin
        state_d = PassChkValid;
    end
end
...

```

Example 3:

The example code below is taken from the JTAG access control mechanism of the HACK@DAC'21 buggy OpenPiton SoC [REF-1364]. Access to JTAG allows users to access sensitive information in the system. Hence, access to JTAG is controlled using cryptographic authentication of the users. In this example (see the vulnerable code source), the password checker uses HMAC-SHA256 for authentication. It takes a 512-bit secret message from the user, hashes it using HMAC, and compares its output with the expected output to determine the authenticity of the user.

Example Language: Verilog

(Bad)

```

...
logic [31-1:0] data_d, data_q;
...
logic [512-1:0] pass_data;
...
Write: begin
    ...
    if (pass_mode) begin
        pass_data = { {60{8'h00}}, data_d};
        state_d = PassChk;
        pass_mode = 1'b0;
    end
    ...
end
...

```

The vulnerable code shows an incorrect implementation of the HMAC authentication where it only uses the least significant 32 bits of the secret message for the authentication (the remaining 480 bits are hard coded as zeros). As a result, the system is susceptible to brute-force attacks on the access control mechanism of JTAG, where the attacker only needs to determine 32 bits of the secret message instead of 512 bits.

To mitigate this issue, remove the zero padding and use all 512 bits of the secret message for HMAC authentication [REF-1365].

Example Language: Verilog

(Good)

```

...
logic [512-1:0] data_d, data_q;
...
logic [512-1:0] pass_data;
...

```

```

Write: begin
...
    if (pass_mode) begin
        pass_data = data_d;
        state_d = PassChk;
        pass_mode = 1'b0;
    ...
end
...




```

Observed Examples

Reference	Description
CVE-2019-18827	chain: JTAG interface is not disabled (CWE-1191) during ROM code execution, introducing a race condition (CWE-362) to extract encryption keys https://www.cve.org/CVERecord?id=CVE-2019-18827

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Relationship

CWE-1191 and CWE-1244 both involve physical debug access, but the weaknesses are different. CWE-1191 is effectively about missing authorization for a debug interface, i.e. JTAG. CWE-1244 is about providing internal assets with the wrong debug access level, exposing the asset to untrusted debug agents.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1037]Kurt Rosenfeld and Ramesh Karri. "Attacks and Defenses for JTAG". 2010 February. < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5406671> >.

[REF-1043]Gopal Vishwakarma and Wonjun Lee. "Exploiting JTAG and Its Mitigation in IOT: A Survey". 2018 December 3. < <https://www.mdpi.com/1999-5903/10/12/121/pdf> >.2023-04-07.

[REF-1084]Gopal Vishwakarma and Wonjun Lee. "JTAG Explained (finally!): Why "IoT", Software Security Engineers, and Manufacturers Should Care". < <https://www.mdpi.com/1999-5903/10/12/121/pdf> >.2023-04-07.

[REF-1085]Bob Molyneaux, Mark McDermott and Anil Sabbavarapu. "Design for Testability & Design for Debug". < http://users.ece.utexas.edu/~mcdermot/vlsi-2/Lecture_17.pdf >.

[REF-1355]Florian Zaruba. "dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L192:L204 >.2023-09-18.

[REF-1354]Florian Zaruba. "Fix CWE-1191 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/58f984d492fdb0369c82ef10fcbbaa4b9850f9fb/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L200 >.2023-09-18.

[REF-1353]Florian Zaruba. "Fix CWE-1191 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/58f984d492fdb0369c82ef10fcbbaa4b9850f9fb/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L131 >.2023-09-18.

[REF-1352]Florian Zaruba. "dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L118:L204 >.2023-09-18.

[REF-1364]"dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L82 >.2023-07-15.

[REF-1365]"fix cwe_1205 in dmi_jtag.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/c4f4b832218b50c406dbf9f425d3b654117c1355/piton/design/chip/tile/ariane/src/riscv-dbg/src/dmi_jtag.sv#L158 >.2023-07-22.

CWE-1192: Improper Identifier for IP Block used in System-On-Chip (SOC)

Weakness ID : 1192

Structure : Simple

Abstraction : Base

Description

The System-on-Chip (SoC) does not have unique, immutable identifiers for each of its components.

Extended Description

A System-on-Chip (SoC) comprises several components (IP) with varied trust requirements. It is required that each IP is identified uniquely and should distinguish itself from other entities in the SoC without any ambiguity. The unique secured identity is required for various purposes. Most of the time the identity is used to route a transaction or perform certain actions, including resetting, retrieving a sensitive information, and acting upon or on behalf of something else.

There are several variants of this weakness:

- A "missing" identifier is when the SoC does not define any mechanism to uniquely identify the IP.
- An "insufficient" identifier might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended.
- A "misconfigured" mechanism occurs when a mechanism is available but not implemented correctly.
- An "ignored" identifier occurs when the SoC/IP has not applied any policies or does not act upon the identifier securely.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		657	Violation of Secure Design Principles	1446

Applicable Platforms**Language** : Not Language-Specific (*Prevalence = Undetermined*)**Technology** : System on Chip (*Prevalence = Undetermined*)**Common Consequences**



Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	High

Potential Mitigations**Phase: Architecture and Design***Strategy = Separation of Privilege*

Every identity generated in the SoC should be unique and immutable in hardware. The actions that an IP is trusted or not trusted should be clearly defined, implemented, configured, and tested. If the definition is implemented via a policy, then the policy should be immutable or protected with clear authentication and authorization.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf		1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
113	Interface Manipulation

CWE-1193: Power-On of Untrusted Execution Core Before Enabling Fabric Access Control**Weakness ID** : 1193**Structure** : Simple**Abstraction** : Base**Description**

The product enables components that contain untrusted firmware before memory and fabric access controls have been enabled.

Extended Description

After initial reset, System-on-Chip (SoC) fabric access controls and other security features need to be programmed by trusted firmware as part of the boot sequence. If untrusted IPs or peripheral microcontrollers are enabled first, then the untrusted component can master transactions on the hardware bus and target memory or other assets to compromise the SoC boot firmware.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		696	Incorrect Behavior Order	1527

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	High
	<i>An untrusted component can master transactions on the HW bus and target memory or other assets to compromise the SoC boot firmware.</i>	




Potential Mitigations

Phase: Architecture and Design

The boot sequence should enable fabric access controls and memory protections before enabling third-party hardware IPs and peripheral microcontrollers that use untrusted firmware.

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1196	Security Flow Issues	1194	2469
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1130]Mark Ermolov, Positive Technologies. "Intel x86 Root of Trust: loss of trust". 2020 March 5. < <https://blog.ptsecurity.com/2020/03/intelx86-root-of-trust-loss-of-trust.html> >.

[REF-1042]Maximillian Dornseif. "Owned by an iPod". 2004. < <https://web.archive.org/web/20060505224959/https://pacsec.jp/psj04/psj04-dornseif-e.ppt> >. 2023-04-07.

CWE-1204: Generation of Weak Initialization Vector (IV)

Weakness ID : 1204

Structure : Simple

Abstraction : Base

Description

The product uses a cryptographic primitive that uses an Initialization Vector (IV), but the product does not generate IVs that are sufficiently unpredictable or unique according to the expected cryptographic requirements for that primitive.



Extended Description

By design, some cryptographic primitives (such as block ciphers) require that IVs must have certain properties for the uniqueness and/or unpredictability of an IV. Primitives may vary in how important these properties are. If these properties are not maintained, e.g. by a bug in the code, then the cryptography may be weakened or broken by attacking the IVs themselves.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	814
ParentOf		329	Generation of Predictable IV with CBC Mode	811

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

Nature	Type	ID	Name	Page
MemberOf		310	Cryptographic Issues	2318

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>If the IV is not properly initialized, data that is encrypted can be compromised and information about the data can be leaked. See [REF-1179].</i>	

Potential Mitigations

Phase: Implementation

Different cipher modes have different requirements for their IVs. When choosing and implementing a mode, it is important to understand those requirements in order to keep security guarantees intact. Generally, it is safest to generate a random IV, since it will be both unpredictable and have a very low chance of being non-unique. IVs do not have to be kept secret, so if generating duplicate IVs is a concern, a list of already-used IVs can be kept and checked against. NIST offers recommendations on generation of IVs for modes of which they have approved. These include options for when random IVs are not practical. For CBC, CFB, and OFB, see [REF-1175]; for GCM, see [REF-1178].

Demonstrative Examples

Example 1:

In the following examples, CBC mode is used when encrypting data:

Example Language: C

(Bad)

```
EVP_CIPHER_CTX ctx;
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv,0,EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx,EVP_bf_cbc(), key,iv);
```

Example Language: Java

(Bad)

```
public class SymmetricCipherTest {
    public static void main() {
        byte[] text ="Secret".getBytes();
        byte[] iv ={
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
        };
```

```

KeyGenerator kg = KeyGenerator.getInstance("DES");
kg.init(56);
SecretKey key = kg.generateKey();
Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
IvParameterSpec ips = new IvParameterSpec(iv);
cipher.init(Cipher.ENCRYPT_MODE, key, ips);
return cipher.doFinal(inpBytes);
}
}

```

In both of these examples, the initialization vector (IV) is always a block of zeros. This makes the resulting cipher text much more predictable and susceptible to a dictionary attack.

Example 2:

The Wired Equivalent Privacy (WEP) protocol used in the 802.11 wireless standard only supported 40-bit keys, and the IVs were only 24 bits, increasing the chances that the same IV would be reused for multiple messages. The IV was included in plaintext as part of the packet, making it directly observable to attackers. Only 5000 messages are needed before a collision occurs due to the "birthday paradox" [REF-1176]. Some implementations would reuse the same IV for each packet. This IV reuse made it much easier for attackers to recover plaintext from two packets with the same IV, using well-understood attacks, especially if the plaintext was known for one of the packets [REF-1175].

Observed Examples

Reference	Description
CVE-2020-1472	ZeroLogon vulnerability - use of a static IV of all zeroes in AES-CFB8 mode https://www.cve.org/CVERecord?id=CVE-2020-1472
CVE-2011-3389	BEAST attack in SSL 3.0 / TLS 1.0. In CBC mode, chained initialization vectors are non-random, allowing decryption of HTTPS traffic using a chosen plaintext attack. https://www.cve.org/CVERecord?id=CVE-2011-3389
CVE-2001-0161	wireless router does not use 6 of the 24 bits for WEP encryption, making it easier for attackers to decrypt traffic https://www.cve.org/CVERecord?id=CVE-2001-0161
CVE-2001-0160	WEP card generates predictable IV values, making it easier for attackers to decrypt traffic https://www.cve.org/CVERecord?id=CVE-2001-0160
CVE-2017-3225	device bootloader uses a zero initialization vector during AES-CBC https://www.cve.org/CVERecord?id=CVE-2017-3225
CVE-2016-6485	crypto framework uses PHP rand function - which is not cryptographically secure - for an initialization vector https://www.cve.org/CVERecord?id=CVE-2016-6485
CVE-2014-5386	encryption routine does not seed the random number generator, causing the same initialization vector to be generated repeatedly https://www.cve.org/CVERecord?id=CVE-2014-5386
CVE-2020-5408	encryption functionality in an authentication framework uses a fixed null IV with CBC mode, allowing attackers to decrypt traffic in applications that use this functionality https://www.cve.org/CVERecord?id=CVE-2020-5408
CVE-2017-17704	messages for a door-unlocking product use a fixed IV in CBC mode, which is the same after each restart https://www.cve.org/CVERecord?id=CVE-2017-17704
CVE-2017-11133	application uses AES in CBC mode, but the pseudo-random secret and IV are generated using math.random, which is not cryptographically strong. https://www.cve.org/CVERecord?id=CVE-2017-11133

Reference	Description
CVE-2007-3528	Blowfish-CBC implementation constructs an IV where each byte is calculated modulo 8 instead of modulo 256, resulting in less than 12 bits for the effective IV length, and less than 4096 possible IV values. https://www.cve.org/CVERecord?id=CVE-2007-3528

Functional Areas

- Cryptography

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
20	Encryption Brute Forcing
97	Cryptanalysis

References

[REF-1175]Nikita Borisov, Ian Goldberg and David Wagner. "Intercepting Mobile Communications: The Insecurity of 802.11". Proceedings of the Seventh Annual International Conference on Mobile Computing And Networking. 2001 July. ACM. < <http://www.isaac.cs.berkeley.edu/isaac/mobicom.pdf> >.

[REF-1175]Nikita Borisov, Ian Goldberg and David Wagner. "Intercepting Mobile Communications: The Insecurity of 802.11". Proceedings of the Seventh Annual International Conference on Mobile Computing And Networking. 2001 July. ACM. < <http://www.isaac.cs.berkeley.edu/isaac/mobicom.pdf> >.

[REF-1176]Wikipedia. "Birthday problem". 2021 March 6. < https://en.wikipedia.org/wiki/Birthday_problem >.

[REF-1177]Wikipedia. "Initialization Vector". 2021 March 8. < https://en.wikipedia.org/wiki/Initialization_vector >.

[REF-1178]NIST. "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC". 2007 November. < <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf> >.2023-04-07.

[REF-1179]Arxum Path Security. "CBC Mode is Malleable. Don't trust it for Authentication". 2019 October 6. < <https://arxumpathsecurity.com/blog/2019/10/16/cbc-mode-is-malleable-dont-trust-it-for-authentication> >.2023-04-07.

CWE-1209: Failure to Disable Reserved Bits

Weakness ID : 1209**Structure** : Simple**Abstraction** : Base

Description

The reserved bits in a hardware design are not disabled prior to production. Typically, reserved bits are used for future capabilities and should not support any functional logic in the design. However, designers might covertly use these bits to debug or further develop new capabilities in production hardware. Adversaries with access to these bits will write to them in hopes of compromising hardware state.

Extended Description

Reserved bits are labeled as such so they can be allocated for a later purpose. They are not to do anything in the current design. However, designers might want to use these bits to debug or control/configure a future capability to help minimize time to market (TTM). If the logic being controlled by these bits is still enabled in production, an adversary could use the logic to induce unwanted/unsupported behavior in the hardware.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	710	Improper Adherence to Coding Standards	1549

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context <i>This type of weakness all depends on the capabilities of the logic being controlled or configured by the reserved bits</i>	
Integrity		
Availability		
Access Control		
Accountability		
Authentication		
Authorization		
Non-Repudiation		

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Include a feature to disable reserved bits.

Phase: Integration

Any writes to these reserve bits are blocked (e.g., ignored, access-protected, etc.), or an exception can be asserted.

Demonstrative Examples

Example 1:

Assume a hardware Intellectual Property (IP) has address space 0x0-0x0F for its configuration registers, with the last one labeled reserved (i.e. 0x0F). Therefore inside the Finite State Machine (FSM), the code is as follows:

Example Language: Verilog (Bad)

```
reg gpio_out = 0; //gpio should remain low for normal operation
case (register_address)
  4'b1111 : //0x0F
    begin
      gpio_out = 1;
    end
```

An adversary may perform writes to reserved address space in hopes of changing the behavior of the hardware. In the code above, the GPIO pin should remain low for normal operation. However, it can be asserted by accessing the reserved address space (0x0F). This may be a concern if the GPIO state is being used as an indicator of health (e.g. if asserted the hardware may respond by shutting down or resetting the system, which may not be the correct action the system should perform).

In the code below, the condition "register_address = 0X0F" is commented out, and a default is provided that will catch any values of register_address not explicitly accounted for and take no action with regards to gpio_out. This means that an attacker who is able to write 0X0F to register_address will not enable any undocumented "features" in the process.

Example Language: Verilog (Good)

```
reg gpio_out = 0; //gpio should remain low for normal operation
case (register_address)
  //4'b1111 : //0x0F
  default: gpio_out = gpio_out;
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces

CWE-1220: Insufficient Granularity of Access Control

Weakness ID : 1220
Structure : Simple
Abstraction : Base

Description

The product implements access controls via a policy or other feature with the intention to disable or restrict accesses (reads and/or writes) to assets in a system from untrusted agents. However, implemented access controls lack required granularity, which renders the control policy too broad because it allows accesses from unauthorized agents to the security-sensitive assets.

Extended Description


Integrated circuits and hardware engines can expose accesses to assets (device configuration, keys, etc.) to trusted firmware or a software module (commonly set by BIOS/bootloader). This access is typically access-controlled. Upon a power reset, the hardware or system usually starts with default values in registers, and the trusted firmware (Boot firmware) configures the necessary access-control protection.

A common weakness that can exist in such protection schemes is that access controls or policies are not granular enough. This condition allows agents beyond trusted agents to access assets and could lead to a loss of functionality or the ability to set up the device securely. This further results in security risks from leaked, sensitive, key material to modification of device configuration.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680
ParentOf		1222	Insufficient Granularity of Address Regions Protected by Register Locks	1999

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2476

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	Execute Unauthorized Code or Commands	
Access Control	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	
	Other	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Access-control-policy protections must be reviewed for design inconsistency and common weaknesses. Access-control-policy definition and programming flow must be tested in pre-silicon, post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider a system with a register for storing AES key for encryption or decryption. The key is 128 bits, implemented as a set of four 32-bit registers. The key registers are assets and registers, AES_KEY_READ_POLICY and AES_KEY_WRITE_POLICY, and are defined to provide necessary access controls.

The read-policy register defines which agents can read the AES-key registers, and write-policy register defines which agents can program or write to those registers. Each register is a 32-bit register, and it can support access control for a maximum of 32 agents. The number of the bit when set (i.e., "1") allows respective action from an agent whose identity matches the number of the bit and, if "0" (i.e., Clear), disallows the respective action to that corresponding agent.

Example Language: Other

(Bad)

In the above example, there is only one policy register that controls access to both read and write accesses to the AES-key registers, and thus the design is not granular enough to separate read and writes access for different agents. Here, agent with identities "1" and "2" can both read and write.

A good design should be granular enough to provide separate access controls to separate actions. Access control for reads should be separate from writes. Below is an example of such implementation where two policy registers are defined for each of these actions. The policy is defined such that: the AES-key registers can only be read or used by a crypto agent with identity "1" when bit #1 is set. The AES-key registers can only be programmed by a trusted firmware with identity "2" when bit #2 is set.

Example Language:

(Good)

Example 2:

Within the AXI node interface wrapper module in the RISC-V AXI module of the HACK@DAC'19 CVA6 SoC [REF-1346], an access control mechanism is employed to regulate the access of different privileged users to peripherals.

The AXI ensures that only users with appropriate privileges can access specific peripherals. For instance, a ROM module is accessible exclusively with Machine privilege, and AXI enforces that users attempting to read data from the ROM must possess machine privilege; otherwise, access to the ROM is denied. The access control information and configurations are stored in a ROM.

Example Language: Verilog

(Bad)

```
...
for (i=0; i<NB_SUBORDINATE; i++)
begin
    for (j=0; j<NB_MANAGER; j++)
    begin
        assign connectivity_map_o[i][j] = access_ctrl_i[i][j][priv_lvl_i] || ((j==6) && access_ctrl_i[i][7][priv_lvl_i]);
    end
end
...
```

However, in the example code above, while assigning distinct privileges to AXI manager and subordinates, both the Platform-Level Interrupt Controller Specification (PLIC) and the Core-local Interrupt Controller (CLINT) (which are peripheral numbers 6 and 7 respectively) utilize the same access control configuration. This common configuration diminishes the granularity of the AXI access control mechanism.

In certain situations, it might be necessary to grant higher privileges for accessing the PLIC than those required for accessing the CLINT. Unfortunately, this differentiation is overlooked, allowing an attacker to access the PLIC with lower privileges than intended.

As a consequence, unprivileged code can read and write to the PLIC even when it was not intended to do so. In the worst-case scenario, the attacker could manipulate interrupt priorities, potentially modifying the system's behavior or availability.

To address the aforementioned vulnerability, developers must enhance the AXI access control granularity by implementing distinct access control entries for the Platform-Level Interrupt Controller (PLIC) and the Core-local Interrupt Controller (CLINT). By doing so, different privilege levels can be defined for accessing PLIC and CLINT, effectively thwarting the potential attacks previously highlighted. This approach ensures a more robust and secure system, safeguarding against unauthorized access and manipulation of interrupt priorities. [REF-1347]

Example Language: Verilog

(Good)

```
...
  for (i=0; i<NB_SUBORDINATE; i++)
    begin
      for (j=0; j<NB_MANAGER; j++)
        begin
          assign connectivity_map_o[i][j] = access_ctrl_i[i][j][priv_lvl_i];
        end
      end
    end
  end
...
```

Example 3:

Consider the following SoC design. The sram in HRoT has an address range that is readable and writable by unprivileged software and it has an area that is only readable by unprivileged software. The tbus interconnect enforces access control for subordinates on the bus but uses only one bit to control both read and write access. Address 0xA0000000 - 0xA000FFFF is readable and writable by the untrusted cores core{0-N} and address 0xA0010000 - 0xA001FFFF is only readable by the untrusted cores core{0-N}.

The security policy access control is not granular enough, as it uses one bit to enable both read and write access. This gives write access to an area that should only be readable by unprivileged agents.

Access control logic should differentiate between read and write access and to have sufficient address granularity.

Observed Examples

Reference	Description
CVE-2022-24985	A form hosting website only checks the session authentication status for a single form, making it possible to bypass authentication when there are multiple forms https://www.cve.org/CVERecord?id=CVE-2022-24985
CVE-2021-36934	An operating system has an overly permission Access Control List on some system files, including those related to user passwords https://www.cve.org/CVERecord?id=CVE-2021-36934

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1346]"axi_node_intf_wrap.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/619e9fb0ef32ee1e01ad76b8732a156572c65700/src/axi_node/src/axi_node_intf_wrap.sv#L430 >.2023-09-18.

[REF-1347]"axi_node_intf_wrap.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/2078f2552194eda37ba87e54cbfef10f1aa41fa5/src/axi_node/src/axi_node_intf_wrap.sv#L430 >.2023-09-18.

CWE-1221: Incorrect Register Defaults or Module Parameters

Weakness ID : 1221

Structure : Simple

Abstraction : Base

Description

Hardware description language code incorrectly defines register defaults or hardware Intellectual Property (IP) parameters to insecure values.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to defined default values that are hard coded in the hardware description language (HDL) code of the hardware unit. Hardware descriptive languages also support definition of parameter variables, which can be defined in code during instantiation of the hardware IP module. Such parameters are generally used to configure a specific instance of a hardware IP in the design.

The system security settings of a hardware design can be affected by incorrectly defined default values or IP parameters. The hardware IP would be in an insecure state at power reset, and this can be exposed or exploited by untrusted software running on the system. Both register defaults and parameters are hardcoded values, which cannot be changed using software or firmware patches but must be changed in hardware silicon. Thus, such security issues are considerably more difficult to address later in the lifecycle. Hardware designs can have a large number of such parameters and register defaults settings, and it is important to have design tool support to check these settings in an automated way and be able to identify which settings are security sensitive.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1419	Incorrect Initialization of Resource	2280

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context	
Integrity	<i>Degradation of system functionality, or loss of access</i>	
Availability	<i>control enforcement can occur.</i>	
Access Control		

Potential Mitigations

Phase: Architecture and Design

During hardware design, all the system parameters and register defaults must be reviewed to identify security sensitive settings.

Phase: Implementation

The default values of these security sensitive settings need to be defined as part of the design review phase.

Phase: Testing

Testing phase should use automated tools to test that values are configured per design specifications.

Demonstrative Examples

Example 1:

Consider example design module system verilog code shown below. The register_example module is an example parameterized module that defines two parameters, REGISTER_WIDTH and REGISTER_DEFAULT. Register_example module defines a Secure_mode setting, which when set makes the register content read-only and not modifiable by software writes. register_top module instantiates two registers, Insecure_Device_ID_1 and Insecure_Device_ID_2. Generally, registers containing device identifier values are required to be read only to prevent any possibility of software modifying these values.

Example Language: Verilog

(Bad)

```
// Parameterized Register module example
// Secure_mode : REGISTER_DEFAULT[0] : When set to 1 register is read only and not writable//
module register_example
#(
    parameter REGISTER_WIDTH = 8, // Parameter defines width of register, default 8 bits
    parameter [REGISTER_WIDTH-1:0] REGISTER_DEFAULT = 2**REGISTER_WIDTH -2 // Default value of register
    computed from Width. Sets all bits to 1s except bit 0 (Secure _mode)
)
(
    input [REGISTER_WIDTH-1:0] Data_in,
    input Clk,
    input resetn,
    input write,
    output reg [REGISTER_WIDTH-1:0] Data_out
);
    reg Secure_mode;
    always @(posedge Clk or negedge resetn)
```

```

if (~resetn)
begin
    Data_out <= REGISTER_DEFAULT; // Register content set to Default at reset
    Secure_mode <= REGISTER_DEFAULT[0]; // Register Secure_mode set at reset
end
else if (write & ~Secure_mode)
begin
    Data_out <= Data_in;
end
endmodule
module register_top
(
input Clk,
input resetn,
input write,
input [31:0] Data_in,
output reg [31:0] Secure_reg,
output reg [31:0] Insecure_reg
);
register_example #(
    .REGISTER_WIDTH (32),
    .REGISTER_DEFAULT (1224) // Incorrect Default value used bit 0 is 0.
) Insecure_Device_ID_1 (
    .Data_in (Data_in),
    .Data_out (Secure_reg),
    .Clk (Clk),
    .resetn (resetn),
    .write (write)
);
register_example #(
    .REGISTER_WIDTH (32) // Default not defined 2^32-2 value will be used as default.
) Insecure_Device_ID_2 (
    .Data_in (Data_in),
    .Data_out (Insecure_reg),
    .Clk (Clk),
    .resetn (resetn),
    .write (write)
);
endmodule

```

These example instantiations show how, in a hardware design, it would be possible to instantiate the register module with insecure defaults and parameters.

In the example design, both registers will be software writable since Secure_mode is defined as zero.

Example Language: Verilog

(Good)

```

register_example #(
    .REGISTER_WIDTH (32),
    .REGISTER_DEFAULT (1225) // Correct default value set, to enable Secure_mode
) Secure_Device_ID_example (
    .Data_in (Data_in),
    .Data_out (Secure_reg),
    .Clk (Clk),
    .resetn (resetn),
    .write (write)
);

```

Example 2:

The example code is taken from the fuse memory inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1356]. Fuse memory can be used to store key hashes, password hashes, and configuration information. For example, the password hashes of JTAG and HMAC are stored in the fuse memory in the OpenPiton design.

During the firmware setup phase, data in the Fuse memory are transferred into the registers of the corresponding SoC peripherals for initialization. However, if the offset to access the password hash is set incorrectly, programs cannot access the correct password hash from the fuse memory, breaking the functionalities of the peripherals and even exposing sensitive information through other peripherals.

Example Language: Verilog

(Bad)

```
parameter MEM_SIZE = 100;
localparam JTAG_OFFSET = 81;
const logic [MEM_SIZE-1:0][31:0] mem = {
    // JTAG expected hamc hash
    32'h49ac13af, 32'h1276f1b8, 32'h6703193a, 32'h65eb531b,
    32'h3025ccca, 32'h3e8861f4, 32'h329edfe5, 32'h98f763b4,
    ...
    assign jtag_hash_o = {mem[JTAG_OFFSET-1],mem[JTAG_OFFSET-2],mem[JTAG_OFFSET-3],
    mem[JTAG_OFFSET-4],mem[JTAG_OFFSET-5],mem[JTAG_OFFSET-6],mem[JTAG_OFFSET-7],mem[JTAG_OFFSET-8]};
    ...
}
```

The following vulnerable code accesses the JTAG password hash from the fuse memory. However, the JTAG_OFFSET is incorrect, and the fuse memory outputs the wrong values to jtag_hash_o. Moreover, setting incorrect offset gives the ability to attackers to access JTAG by knowing other low-privileged peripherals' passwords.

To mitigate this, change JTAG_OFFSET to the correct address of the JTAG key [REF-1357].



Example Language: Verilog

(Good)

```
parameter MEM_SIZE = 100;
localparam JTAG_OFFSET = 100;
```

MemberOf Relationships

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

Nature	Type	ID	Name	<input checked="" type="checkbox"/>	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
166	Force the System to Reset Values

References

[REF-1356]"fuse_mem.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/main/piton/design/chip/tile/ariane/src/fuse_mem/fuse_mem.sv#L14-L15 >.2023-07-15.

[REF-1357]"fix CWE 1221 in fuse_mem.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/compare/main...cwe_1221_in_fuse_mem#diff-d7275edeac22f76691a31c83f005d0177359ad710ad6549ece3d069ed043ef21 >.2023-07-24.

CWE-1222: Insufficient Granularity of Address Regions Protected by Register Locks

Weakness ID : 1222

Structure : Simple

Abstraction : Variant

Description

The product defines a large address region protected from modification by the same register lock control bit. This results in a conflict between the functional requirement that some addresses need to be writable by software during operation and the security requirement that the system configuration lock bit must be set during the boot process.

Extended Description

Integrated circuits and hardware IPs can expose the device configuration controls that need to be programmed after device power reset by a trusted firmware or software module (commonly set by BIOS/bootloader) and then locked from any further modification. In hardware design, this is commonly implemented using a programmable lock bit which enables/disables writing to a protected set of registers or address regions. When the programmable lock bit is set, the relevant address region can be implemented as a hardcoded value in hardware logic that cannot be changed later.

A problem can arise wherein the protected region definition is not granular enough. After the programmable lock bit has been set, then this new functionality cannot be implemented without change to the hardware design.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1220	Insufficient Granularity of Access Control	1992

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Other	
	System security configuration cannot be defined in a way that does not conflict with functional requirements of device.	

Potential Mitigations

Phase: Architecture and Design

The defining of protected locked registers should be reviewed or tested early in the design phase with software teams to ensure software flows are not blocked by the security locks. As an alternative to using register lock control bits and fixed access control regions, the hardware design could use programmable security access control configuration so that device trusted firmware can configure and change the protected regions based on software usage and security models.

Demonstrative Examples

2000

Example 1:

For example, consider a hardware unit with a 32 kilobyte configuration address space where the first 8 kilobyte address contains security sensitive controls that must only be writable by device bootloader. One way to protect the security configuration could be to define a 32 bit system configuration locking register (SYS_LOCK) where each bit lock locks the corresponding 1 kilobyte region.

Example Language: Other

(Bad)

If a register exists within the first kilobyte address range (e.g. SW_MODE, address 0x310) and needs to be software writable at runtime, then this register cannot be written in a securely configured system since SYS_LOCK register lock bit 0 must be set to protect other security settings (e.g. SECURITY_FEATURE_ENABLE, address 0x0004). The only fix would be to change the hardware logic or not set the security lock bit.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
679	Exploitation of Improperly Configured or Implemented Memory Protections

CWE-1223: Race Condition for Write-Once Attributes

Weakness ID : 1223

Structure : Simple

Abstraction : Base

Description

A write-once register in hardware design is programmable by an untrusted software component earlier than the trusted software component, resulting in a race condition issue.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to defined default values that are hard coded in the hardware description language (HDL) code of the hardware unit. A common security protection method used to protect register settings from modification by software is to make them write-once. This means the hardware implementation only allows writing to such registers once, and they become read-only after having been written once by software. This is useful to allow initial boot software to configure systems settings to secure values while blocking runtime software from modifying such hardware settings.

Implementation issues in hardware design of such controls can expose such registers to a race condition security flaw. For example, consider a hardware design that has two different software/firmware modules executing in parallel. One module is trusted (module A) and another is untrusted (module B). In this design it could be possible for Module B to send write cycles to the write-once register before Module A. Since the field is write-once the programmed value from Module A will be ignored and the pre-empted value programmed by Module B will be used by hardware.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>System configuration cannot be programmed in a secure way.</i>	

Potential Mitigations

Phase: Architecture and Design

During hardware design all register write-once or sticky fields must be evaluated for proper configuration.

Phase: Testing

The testing phase should use automated tools to test that values are not reprogrammable and that write-once fields lock on writing zeros.

Demonstrative Examples

Example 1:

consider the example design module system verilog code shown below.

register_write_once_example module is an example of register that has a write-once field defined. Bit 0 field captures the write_once_status value.

Example Language: Verilog

(Bad)

```
module register_write_once_example
(
    input [15:0] Data_in,
    input Clk,
    input ip_resetrn,
    input global_resetrn,
    input write,
    output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetrn)
if (~ip_resetrn)
begin
    Data_out <= 16'h0000;
    Write_once_status <= 1'b0;
end
else if (write & ~Write_once_status)
begin
    Data_out <= Data_in & 16'hFFFE; // Input data written to register after masking bit 0
```

```

    Write_once_status <= 1'b1; // Write once status set after first write.
  end
  else if (~write)
  begin
    Data_out[15:1] <= Data_out[15:1];
    Data_out[0] <= Write_once_status;
  end
endmodule

```

The first system component that sends a write cycle to this register can program the value. This could result in a race condition security issue in the SoC design, if an untrusted agent is running in the system in parallel with the trusted component that is expected to program the register.

Example Language:

(Good)

Trusted firmware or software trying to set the write-once field:

- Must confirm the Write_once_status (bit 0) value is zero, before programming register. If another agent has programmed the register before, then Write_once_status value will be one.
- After writing to the register, the trusted software can issue a read to confirm that the valid setting has been programmed.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions

CWE-1224: Improper Restriction of Write-Once Bit Fields

Weakness ID : 1224

Structure : Simple

Abstraction : Base

Description

The hardware design control register "sticky bits" or write-once bit fields are improperly implemented, such that they can be reprogrammed by software.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to define default values that are hard coded in the hardware description language (HDL) code of the hardware unit. A common security protection method used to protect register settings from modification by software is to make the settings write-once or "sticky." This allows writing to such registers only once, whereupon they become read-only. This is useful to allow initial boot software to configure systems settings to secure values while blocking runtime software from modifying such hardware settings.

Failure to implement write-once restrictions in hardware design can expose such registers to being re-programmed by software and written multiple times. For example, write-once fields could be implemented to only be write-protected if they have been set to value "1", wherein they would work as "write-1-once" and not "write-once".

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	System configuration cannot be programmed in a secure way.	
Integrity		
Availability		
Access Control		

Potential Mitigations

Phase: Architecture and Design

During hardware design all register write-once or sticky fields must be evaluated for proper configuration.

Phase: Testing

The testing phase should use automated tools to test that values are not reprogrammable and that write-once fields lock on writing zeros.

Demonstrative Examples

Example 1:

Consider the example design module system verilog code shown below. register_write_once_example module is an example of register that has a write-once field defined. Bit 0 field captures the write_once_status value. This implementation can be for a register that is defined by specification to be a write-once register, since the write_once_status field gets written by input data bit 0 on first write.

Example Language: Verilog

(Bad)

```
module register_write_once_example
(
  input [15:0] Data_in,
  input Clk,
  input ip_resetrn,
  input global_resetrn,
  input write,
  output reg [15:0] Data_out
);
  reg Write_once_status;
  always @(posedge Clk or negedge ip_resetrn)
```

```
if (~ip_resetrn)
begin
    Data_out <= 16'h0000;
    Write_once_status <= 1'b0;
end
else if (write & ~Write_once_status)
begin
    Data_out <= Data_in & 16'hFFFE;
    Write_once_status <= Data_in[0]; // Input bit 0 sets Write_once_status
end
else if (~write)
begin
    Data_out[15:1] <= Data_out[15:1];
    Data_out[0] <= Write_once_status;
end
endmodule
```

The above example only locks further writes if write_once_status bit is written to one. So it acts as write_1-Once instead of the write-once attribute.

Example Language: Verilog

(Good)

```
module register_write_once_example
(
    input [15:0] Data_in,
    input Clk,
    input ip_resetrn,
    input global_resetrn,
    input write,
    output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetrn)
    if (~ip_resetrn)
    begin
        Data_out <= 16'h0000;
        Write_once_status <= 1'b0;
    end
    else if (write & ~Write_once_status)
    begin
        Data_out <= Data_in & 16'hFFFE;
        Write_once_status <= 1'b1; // Write once status set on first write, independent of input
    end
    else if (~write)
    begin
        Data_out[15:1] <= Data_out[15:1];
        Data_out[0] <= Write_once_status;
    end
end
endmodule
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
680	Exploitation of Improperly Controlled Registers

CWE-1229: Creation of Emergent Resource

Weakness ID : 1229

Structure : Simple

Abstraction : Class

Description

The product manages resources or behaves in a way that indirectly creates a new, distinct resource that can be used by attackers in violation of the intended policy.

Extended Description

A product is only expected to behave in a way that was specifically intended by the developer. Resource allocation and management is expected to be performed explicitly by the associated code. However, in systems with complex behavior, the product might indirectly produce new kinds of resources that were never intended in the original design. For example, a covert channel is a resource that was never explicitly intended by the developer, but it is useful to attackers. "Parasitic computing," while not necessarily malicious in nature, effectively tricks a product into performing unintended computations on behalf of another party.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	⊕	514	Covert Channel	1218

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	⊕	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

References

[REF-1049]Wikipedia. "Parasitic computing". < https://en.wikipedia.org/wiki/Parasitic_computing >.

CWE-1230: Exposure of Sensitive Information Through Metadata

Weakness ID : 1230

Structure : Simple

Abstraction : Base

Description

The product prevents direct access to a resource containing sensitive information, but it does not sufficiently limit access to metadata that is derived from the original, sensitive information.

Extended Description

Developers might correctly prevent unauthorized access to a database or other resource containing sensitive information, but they might not consider that portions of the original information might also be recorded in metadata, search indices, statistical reports, or other resources. If these resources are not also restricted, then attackers might be able to extract some or all of the original information, or otherwise infer some details. For example, an attacker could specify search terms that are known to be unique to a particular person, or view metadata such as activity or creation dates in order to identify usage patterns.



Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		285	Improper Authorization	684
ParentOf		202	Exposure of Sensitive Information Through Data Queries	516
ParentOf		612	Improper Authorization of Index Containing Sensitive Information	1370

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

Nature	Type	ID	Name	Page
MemberOf		1212	Authorization Errors	2476
MemberOf		199	Information Management Errors	2312

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

CWE-1231: Improper Prevention of Lock Bit Modification

Weakness ID : 1231

Structure : Simple

Abstraction : Base

Description

The product uses a trusted lock bit for restricting access to registers, address regions, or other resources, but the product does not prevent the value of the lock bit from being modified after it has been set.

Extended Description

In integrated circuits and hardware intellectual property (IP) cores, device configuration controls are commonly programmed after a device power reset by a trusted firmware or software module (e.g., BIOS/bootloader) and then locked from any further modification.

This behavior is commonly implemented using a trusted lock bit. When set, the lock bit disables writes to a protected set of registers or address regions. Design or coding errors in the implementation of the lock bit protection feature may allow the lock bit to be modified or cleared by software after it has been set. Attackers might be able to unlock the system and features that the bit is intended to protect.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	IP	284	Improper Access Control	680

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory	High
	<i>Registers protected by lock bit can be modified even when lock is set.</i>	

Detection Methods

Manual Analysis

Set the lock bit. Power cycle the device. Attempt to clear the lock bit. If the information is changed, implement a design fix. Retest. Also, attempt to indirectly clear the lock bit or bypass it.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security lock bit protections must be reviewed for design inconsistency and common weaknesses. Security lock programming flow and lock properties must be tested in pre-silicon and post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider the example design below for a digital thermal sensor that detects overheating of the silicon and triggers system shutdown. The system critical temperature limit (CRITICAL_TEMP_LIMIT) and thermal sensor calibration (TEMP_SENSOR_CALIB) data have to be programmed by firmware, and then the register needs to be locked (TEMP_SENSOR_LOCK).

Example Language: Other

(Bad)

In this example, note that if the system heats to critical temperature, the response of the system is controlled by the TEMP_HW_SHUTDOWN bit [1], which is not lockable. Thus, the intended security property of the critical temperature sensor cannot be fully protected, since software can misconfigure the TEMP_HW_SHUTDOWN register even after the lock bit is set to disable the shutdown response.

Example Language:

(Good)

To fix this weakness, one could change the TEMP_HW_SHUTDOWN field to be locked by TEMP_SENSOR_LOCK.

Example 2:

The following example code is a snippet from the register locks inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1350]. Register locks help prevent SoC peripherals' registers from malicious use of resources. The registers that can potentially leak secret data are locked by register locks.

In the vulnerable code, the reglk_mem is used for locking information. If one of its bits toggle to 1, the corresponding peripheral's registers will be locked. In the context of the HACK@DAC System-on-Chip (SoC), it is pertinent to note the existence of two distinct categories of reset signals.

First, there is a global reset signal denoted as "rst_ni," which possesses the capability to simultaneously reset all peripherals to their respective initial states.

Second, we have peripheral-specific reset signals, such as "rst_9," which exclusively reset individual peripherals back to their initial states. The administration of these reset signals is the responsibility of the reset controller module.

Example Language: Verilog

(Bad)

```
always @(posedge clk_i)
begin
    if(~(rst_ni && ~jtag_unlock && ~rst_9))
    begin
        for (j=0; j < 6; j=j+1) begin
            reglk_mem[j] <= 'h0;
        end
    end
end
...
```

In the buggy SoC architecture during HACK@DAC'21, a critical issue arises within the reset controller module. Specifically, the reset controller can inadvertently transmit a peripheral reset signal to the register lock within the user privilege domain.

This unintentional action can result in the reset of the register locks, potentially exposing private data from all other peripherals, rendering them accessible and readable.

To mitigate the issue, remove the extra reset signal rst_9 from the register lock if condition. [REF-1351]

Example Language: Verilog

(Good)

```
always @(posedge clk_i)
begin
    if(~(rst_ni && ~jtag_unlock))
    begin
        for (j=0; j < 6; j=j+1) begin
            reglk_mem[j] <= 'h0;
        end
    end
end
...
```

Observed Examples

Reference	Description
CVE-2017-6283	chip reset clears critical read/write lock permissions for RSA function https://www.cve.org/CVERecord?id=CVE-2017-6283

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf	C	1372	ICS Supply Chain: OT Counterfeit and Malicious Corruption	1358	2509
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
680	Exploitation of Improperly Controlled Registers

References

[REF-1350]"reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L80C1-L80C48 >.2023-09-18.

[REF-1351]"fix cwe 1199 in reglk". 2023. < <https://github.com/HACK-EVENT/hackatdac21/commit/5928add42895b57341ae8fc1f9b8351c35aed865#diff-1c2b09dd092a56e5fb2be431a3849e72ff489d2ae4f4a6bb5> >.2023-09-18.

CWE-1232: Improper Lock Behavior After Power State Transition

Weakness ID : 1232**Structure** : Simple**Abstraction** : Base

Description

Register lock bit protection disables changes to system configuration once the bit is set. Some of the protected registers or lock bits become programmable after power state transitions (e.g., Entry and wake from low power sleep modes) causing the system configuration to be changeable.

Extended Description

Devices may allow device configuration controls which need to be programmed after device power reset via a trusted firmware or software module (commonly set by BIOS/bootloader) and then

locked from any further modification. This action is commonly implemented using a programmable lock bit, which, when set, disables writes to a protected set of registers or address regions.

After a power state transition, the lock bit is set to unlocked. Some common weaknesses that can exist in such a protection scheme are that the lock gets cleared, the values of the protected registers get reset, or the lock become programmable.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1464

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security Lock bit protections should be reviewed for behavior across supported power state transitions. Security lock programming flow and lock properties should be tested in pre-silicon and post-silicon testing including testing across power transitions.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider the memory configuration settings of a system that uses DDR3 DRAM memory. Protecting the DRAM memory configuration from modification by software is required to ensure that system memory access control protections cannot be bypassed. This can be done by using lock bit protection that locks all of the memory configuration registers. The memory configuration lock can be set by the BIOS during the boot process.

If such a system also supports a rapid power on mode like hibernate, the DRAM data must be saved to a disk before power is removed and restored back to the DRAM once the system powers back up and before the OS resumes operation after returning from hibernate.

To support the hibernate transition back to the operating state, the DRAM memory configuration must be reprogrammed even though it was locked previously. As the hibernate resume does a partial reboot, the memory configuration could be altered before the memory lock is set. Functionally the hibernate resume flow requires a bypass of the lock-based protection. The

memory configuration must be securely stored and restored by trusted system firmware. Lock settings and system configuration must be restored to the same state it was in before the device entered into the hibernate mode.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
166	Force the System to Reset Values

CWE-1233: Security-Sensitive Hardware Controls with Missing Lock Bit Protection

Weakness ID : 1233

Structure : Simple

Abstraction : Base

Description

The product uses a register lock bit protection mechanism, but it does not ensure that the lock bit prevents modification of system registers or controls that perform changes to important hardware system configuration.

Extended Description

Integrated circuits and hardware intellectual properties (IPs) might provide device configuration controls that need to be programmed after device power reset by a trusted firmware or software module, commonly set by BIOS/bootloader. After reset, there can be an expectation that the controls cannot be used to perform any further modification. This behavior is commonly implemented using a trusted lock bit, which can be set to disable writes to a protected set of registers or address regions. The lock protection is intended to prevent modification of certain system configuration (e.g., memory/memory protection unit configuration).

However, if the lock bit does not effectively write-protect all system registers or controls that could modify the protected system configuration, then an adversary may be able to use software to access the registers/controls and modify the protected hardware configuration.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1464
ChildOf	P	284	Improper Access Control	680

Weakness Ordinalities

Primary :**Applicable Platforms****Language** : Not Language-Specific (*Prevalence = Undetermined*)**Operating_System** : Not OS-Specific (*Prevalence = Undetermined*)**Architecture** : Not Architecture-Specific (*Prevalence = Undetermined*)**Technology** : Not Technology-Specific (*Prevalence = Undetermined*)**Common Consequences**

Scope	Impact	Likelihood
Access Control	Modify Memory	
	<i>System Configuration protected by the lock bit can be modified even when the lock is set.</i>	

Detection Methods**Manual Analysis**

Set the lock bit. Attempt to modify the information protected by the lock bit. If the information is changed, implement a design fix. Retest. Also, attempt to indirectly clear the lock bit or bypass it.

Effectiveness = High

Potential Mitigations**Phase: Architecture and Design****Phase: Implementation****Phase: Testing**

Security lock bit protections must be reviewed for design inconsistency and common weaknesses. Security lock programming flow and lock properties must be tested in pre-silicon and post-silicon testing.

Demonstrative Examples**Example 1:**

Consider the example design below for a digital thermal sensor that detects overheating of the silicon and triggers system shutdown. The system critical temperature limit (CRITICAL_TEMP_LIMIT) and thermal sensor calibration (TEMP_SENSOR_CALIB) data have to be programmed by the firmware.

Example Language: Other

(Bad)

In this example note that only the CRITICAL_TEMP_LIMIT register is protected by the TEMP_SENSOR_LOCK bit, while the security design intent is to protect any modification of the critical temperature detection and response.

The response of the system, if the system heats to a critical temperature, is controlled by TEMP_HW_SHUTDOWN bit [1], which is not lockable. Also, the TEMP_SENSOR_CALIB register is not protected by the lock bit.

By modifying the temperature sensor calibration, the conversion of the sensor data to a degree centigrade can be changed, such that the current temperature will never be detected to exceed critical temperature value programmed by the protected lock.

Similarly, by modifying the TEMP_HW_SHUTDOWN.Enable bit, the system response detection of the current temperature exceeding critical temperature can be disabled.

Example Language:

(Good)

Change TEMP_HW_SHUTDOWN and TEMP_SENSOR_CALIB controls to be locked by TEMP_SENSOR_LOCK.

Observed Examples

Reference	Description
CVE-2018-9085	Certain servers leave a write protection lock bit unset after boot, potentially allowing modification of parts of flash memory. https://www.cve.org/CVERecord?id=CVE-2018-9085
CVE-2014-8273	Chain: chipset has a race condition (CWE-362) between when an interrupt handler detects an attempt to write-enable the BIOS (in violation of the lock bit), and when the handler resets the write-enable bit back to 0, allowing attackers to issue BIOS writes during the timing window [REF-1237]. https://www.cve.org/CVERecord?id=CVE-2014-8273

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf	C	1372	ICS Supply Chain: OT Counterfeit and Malicious Corruption	1358	2509
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
176	Configuration/Environment Manipulation
680	Exploitation of Improperly Controlled Registers

References

[REF-1237]CERT Coordination Center. "Intel BIOS locking mechanism contains race condition that enables write protection bypass". 2015 January 5. < <https://www.kb.cert.org/vuls/id/766164/> >.

CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks

Weakness ID : 1234

Structure : Simple

Abstraction : Base

Description

System configuration protection may be bypassed during debug mode.

Extended Description


Device configuration controls are commonly programmed after a device power reset by a trusted firmware or software module (e.g., BIOS/bootloader) and then locked from any further modification. This is commonly implemented using a trusted lock bit, which when set, disables writes to a protected set of registers or address regions. The lock protection is intended to prevent modification of certain system configuration (e.g., memory/memory protection unit configuration). If debug features supported by hardware or internal modes/system states are supported in

the hardware design, modification of the lock protection may be allowed allowing access and modification of configuration information.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1464

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism <i>Bypass of lock bit allows access and modification of system configuration even when the lock bit is set.</i>	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security Lock bit protections should be reviewed for any bypass/override modes supported. Any supported override modes either should be removed or protected using authenticated debug modes. Security lock programming flow and lock properties should be tested in pre-silicon and post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider the example Locked_override_register example. This register module supports a lock mode that blocks any writes after lock is set to 1. However, it also allows override of the lock protection when scan_mode or debug_unlocked modes are active.

Example Language: Verilog

(Bad)

```
module Locked_register_example
(
  input [15:0] Data_in,
  input Clk,
  input resetn,
  input write,
  input Lock,
  input scan_mode,
  input debug_unlocked,
  output reg [15:0] Data_out
)
```

```

);
reg lock_status;
always @(posedge Clk or negedge resetn)
  if (~resetn) // Register is reset resetn
    begin
      lock_status <= 1'b0;
    end
  else if (Lock)
    begin
      lock_status <= 1'b1;
    end
  else if (~Lock)
    begin
      lock_status <= lock_status
    end
always @(posedge Clk or negedge resetn)
  if (~resetn) // Register is reset resetn
    begin
      Data_out <= 16'h0000;
    end
  else if (write & (~lock_status | scan_mode | debug_unlocked) ) // Register protected by Lock bit input, overrides supported
    for scan_mode & debug_unlocked
    begin
      Data_out <= Data_in;
    end
  else if (~write)
    begin
      Data_out <= Data_out;
    end
endmodule

```

If either the scan_mode or the debug_unlocked modes can be triggered by software, then the lock protection may be bypassed.

Example Language:

(Good)

Either remove the debug and scan mode overrides or protect enabling of these modes so that only trusted and authorized users may enable these modes.

Example 2:

The following example code [REF-1375] is taken from the register lock security peripheral of the HACK@DAC'21 buggy OpenPiton SoC. It demonstrates how to lock read or write access to security-critical hardware registers (e.g., crypto keys, system integrity code, etc.). The configuration to lock all the sensitive registers in the SoC is managed through the reglk_mem registers. These reglk_mem registers are reset when the hardware powers up and configured during boot up. Malicious users, even with kernel-level software privilege, do not get access to the sensitive contents that are locked down. Hence, the security of the entire system can potentially be compromised if the register lock configurations are corrupted or if the register locks are disabled.

Example Language: Verilog

(Bad)

```

...
always @(posedge clk_i)
  begin
    if (~(rst_ni && ~jtag_unlock && ~rst_9))
      begin
        for (j=0; j < 6; j=j+1) begin
          reglk_mem[j] <= 'h0;
        end
      end
    end
  ...

```

The example code [REF-1375] illustrates an instance of a vulnerable implementation of register locks in the SoC. In this flawed implementation [REF-1375], the reglk_mem registers are also being reset when the system enters debug mode (indicated by the jtag_unlock signal). Consequently, users can simply put the processor in debug mode to access sensitive contents that are supposed to be protected by the register lock feature.

This can be mitigated by excluding debug mode signals from the reset logic of security-critical register locks as demonstrated in the following code snippet [REF-1376].




Example Language: Verilog

(Good)

```
...
always @(posedge clk_i)
begin
    if(~(rst_ni && ~rst_9))
    begin
        for (j=0; j < 6; j=j+1) begin
            reglk_mem[j] <= 'h0;
        end
    end
end
...
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
176	Configuration/Environment Manipulation

References

[REF-1375]"reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/cde1d9d6888bffb21d4b405cce61b19c58dd3c/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L80C1-L80C48 >.2023-12-13.

[REF-1376]"Fix for reglk_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/20238068b385d7ab704cabfb95ff95dd6e56e1c2/piton/design/chip/tile/ariane/src/reglk/reglk_wrapper.sv#L80 >.2023-12-13.

CWE-1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations

Weakness ID : 1235

Structure : Simple

Abstraction : Base

Description

The code uses boxed primitives, which may introduce inefficiencies into performance-critical operations.

Extended Description


Languages such as Java and C# support automatic conversion through their respective compilers from primitive types into objects of the corresponding wrapper classes, and vice versa. For example, a compiler might convert an int to Integer (called autoboxing) or an Integer to int (called unboxing). This eliminates forcing the programmer to perform these conversions manually, which makes the code cleaner.

However, this feature comes at a cost of performance and can lead to resource exhaustion and impact availability when used with generic collections. Therefore, they should not be used for scientific computing or other performance critical operations. They are only suited to support "impedance mismatch" between reference types and primitives.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	964

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

Nature	Type	ID	Name	Page
MemberOf		1006	Bad Coding Practices	2422

Applicable Platforms

Language : Java (*Prevalence = Undetermined*)

Language : C# (*Prevalence = Undetermined*)

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) Reduce Performance <i>Incorrect autoboxing/unboxing would result in reduced performance, which sometimes can lead to resource consumption issues.</i>	Low

Potential Mitigations

Phase: Implementation

Use of boxed primitives should be limited to certain situations such as when calling methods with typed parameters. Examine the use of boxed primitives prior to use. Use SparseArrays or ArrayMap instead of HashMap to avoid performance overhead.

Demonstrative Examples

Example 1:

Java has a boxed primitive for each primitive type. A long can be represented with the boxed primitive Long. Issues arise where boxed primitives are used when not strictly necessary.

*Example Language: Java**(Bad)*

```

Long count = 0L;
for (long i = 0; i < Integer.MAX_VALUE; i++) {
    count += i;
}

```

In the above loop, we see that the count variable is declared as a boxed primitive. This causes autoboxing on the line that increments. This causes execution to be magnitudes less performant (time and possibly space) than if the "long" primitive was used to declare the count variable, which can impact availability of a resource.

Example 2:

This code uses primitive long which fixes the issue.

*Example Language: Java**(Good)*

```

long count = 0L;
for (long i = 0; i < Integer.MAX_VALUE; i++) {
    count += i;
}

```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Oracle Coding Standard for Java	EXP04-J		Do not pass arguments to certain Java Collections Framework methods that are a different type than the collection parameter type
ISA/IEC 62443	Part 4-1		Req SI-2

References

[REF-1051]"Oracle Java Documentation". < <https://docs.oracle.com/javase/1.5.0/docs/guide/language/autoboxing.html> >.

[REF-1052]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 02. Expressions (EXP)". < <https://wiki.sei.cmu.edu/confluence/display/java/EXP04-J.+Do+not+pass+arguments+to+certain+Java+Collections+Framework+methods+that+are+a+different+type+than+the+collection+parameter+type> >.

CWE-1236: Improper Neutralization of Formula Elements in a CSV File**Weakness ID :** 1236**Structure :** Simple**Abstraction :** Base**Description**

The product saves user-provided information into a Comma-Separated Value (CSV) file, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as a command when the file is opened by a spreadsheet product.

Extended Description

User-provided data is often saved to traditional databases. This data can be exported to a CSV file, which allows users to read the data using spreadsheet software such as Excel, Numbers, or Calc. This software interprets entries beginning with '=' as formulas, which are then executed by the spreadsheet software. The software's formula language often allows methods to access hyperlinks or the local command line, and frequently allows enough characters to invoke an entire script. Attackers can populate data fields which, when saved to a CSV file, may attempt information exfiltration or other malicious activity when automatically executed by the spreadsheet software.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137

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

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	137

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

Nature	Type	ID	Name	Page
MemberOf		137	Data Neutralization Issues	2311

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Other (*Prevalence = Undetermined*)

Alternate Terms

CSV Injection :

Formula Injection :

Excel Macro Injection :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Execute Unauthorized Code or Commands <i>Current versions of Excel warn users of untrusted content.</i>	Low

Potential Mitigations

Phase: Implementation

When generating CSV output, ensure that formula-sensitive metacharacters are effectively escaped or removed from all data before storage in the resultant CSV. Risky characters include '=' (equal), '+' (plus), '-' (minus), and '@' (at).

Effectiveness = Moderate

Unfortunately, there is no perfect solution, since different spreadsheet products act differently.

Phase: Implementation

If a field starts with a formula character, prepend it with a ' (single apostrophe), which prevents Excel from executing the formula.

Effectiveness = Moderate

It is not clear how effective this mitigation is with other spreadsheet software.

Phase: Architecture and Design

Certain implementations of spreadsheet software might disallow formulas from executing if the file is untrusted, or if the file is not authored by the current user.

Effectiveness = Limited

This mitigation has limited effectiveness because it often depends on end users opening spreadsheet software safely.

Demonstrative Examples

Example 1:

Hyperlinks or other commands can be executed when a cell begins with the formula identifier, '='

Example Language: Other

(Attack)

```
=HYPERLINK(link_location, [friendly_name])
```

Stripping the leading equals sign, or simply not executing formulas from untrusted sources, impedes malicious activity.

Example Language:

(Good)

```
HYPERLINK(link_location, [friendly_name])
```

Observed Examples

Reference	Description
CVE-2019-12134	Low privileged user can trigger CSV injection through a contact form field value https://www.cve.org/CVERecord?id=CVE-2019-12134
CVE-2019-4521	Cloud management product allows arbitrary command execution via CSV injection https://www.cve.org/CVERecord?id=CVE-2019-4521
CVE-2019-17661	CSV injection in content management system via formula code in a first or last name https://www.cve.org/CVERecord?id=CVE-2019-17661

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1409	Comprehensive Categorization: Injection	1400	2535

References

[REF-21]OWASP. "CSV Injection". 2020 February 2. < https://owasp.org/www-community/attacks/CSV_Injection >.

[REF-22]Jamie Rougvie. "Data Extraction to Command Execution CSV Injection". 2019 September 6. < <https://www.veracode.com/blog/secure-development/data-extraction-command-execution-csv-injection> >.

[REF-23]George Mauer. "The Absurdly Underestimated Dangers of CSV Injection". 2017 October 7. < <http://georgemauer.net/2017/10/07/csv-injection.html> >.

[REF-24]James Kettle. "Comma Separated Vulnerabilities". 2014 August 9. < <https://rstforums.com/forum/topic/82690-comma-separated-vulnerabilities/> >.2023-04-07.

CWE-1239: Improper Zeroization of Hardware Register

Weakness ID : 1239

Structure : Simple

Abstraction : Variant

Description

The hardware product does not properly clear sensitive information from built-in registers when the user of the hardware block changes.

Extended Description

Hardware logic operates on data stored in registers local to the hardware block. Most hardware IPs, including cryptographic accelerators, rely on registers to buffer I/O, store intermediate values, and interface with software. The result of this is that sensitive information, such as passwords or encryption keys, can exist in locations not transparent to the user of the hardware logic. When a different entity obtains access to the IP due to a change in operating mode or conditions, the new entity can extract information belonging to the previous user if no mechanisms are in place to clear register contents. It is important to clear information stored in the hardware if a physical attack on the product is detected, or if the user of the hardware block changes. The process of clearing register contents in a hardware IP is referred to as zeroization in standards for cryptographic hardware modules such as FIPS-140-2 [REF-267].

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	226	Sensitive Information in Resource Not Removed Before Reuse	562

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf	B	226	Sensitive Information in Resource Not Removed Before Reuse	562

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context <i>The consequences will depend on the information disclosed due to the vulnerability.</i>	

Potential Mitigations

Phase: Architecture and Design

Every register potentially containing sensitive information must have a policy specifying how and when information is cleared, in addition to clarifying if it is the responsibility of the hardware logic or IP user to initiate the zeroization procedure at the appropriate time.

Demonstrative Examples

Example 1:

Suppose a hardware IP for implementing an encryption routine works as expected, but it leaves the intermediate results in some registers that can be accessed. Exactly why this access happens is immaterial - it might be unintentional or intentional, where the designer wanted a "quick fix" for something.

Example 2:

The example code below [REF-1379] is taken from the SHA256 Interface/wrapper controller module of the HACK@DAC'21 buggy OpenPiton SoC. Within the wrapper module there are a set of 16 memory-mapped registers referenced data[0] to data[15]. These registers are 32 bits in size and are used to store the data received on the AXI Lite interface for hashing. Once both the message to be hashed and a request to start the hash computation are received, the values of these registers will be forwarded to the underlying SHA256 module for processing. Once forwarded, the values in these registers no longer need to be retained. In fact, if not cleared or overwritten, these sensitive values can be read over the AXI Lite interface, potentially compromising any previously confidential data stored therein.

Example Language: Verilog

(Bad)

```
...
// Implement SHA256 I/O memory map interface
// Write side
always @(posedge clk_i)
begin
    if(~(rst_ni && ~rst_3))
    begin
        startHash <= 0;
        newMessage <= 0;
        data[0] <= 0;
        data[1] <= 0;
        data[2] <= 0;
        ...
        data[14] <= 0;
        data[15] <= 0;
    end
end
...
```

In the previous code snippet [REF-1379] there is the lack of a data clearance mechanism for the memory-mapped I/O registers after their utilization. These registers get cleared only when a reset condition is met. This condition is met when either the global negative-edge reset input signal (rst_ni) or the dedicated reset input signal for SHA256 peripheral (rst_3) is active. In other

words, if either of these reset signals is true, the registers will be cleared. However, in cases where there is not a reset condition these registers retain their values until the next hash operation. It is during the time between an old hash operation and a new hash operation that that data is open to unauthorized disclosure.

To correct the issue of data persisting between hash operations, the memory mapped I/O registers need to be cleared once the values written in these registers are propagated to the SHA256 module. This could be done for example by adding a new condition to zeroize the memory mapped I/O registers once the hash value is computed, i.e., hashValid signal asserted, as shown in the good code example below [REF-1380]. This fix will clear the memory-mapped I/O registers after the data has been provided as input to the SHA engine.

Example Language: Verilog (Good)

```
...
// Implement SHA256 I/O memory map interface
// Write side
always @(posedge clk_i)
begin
    if(~(rst_ni && ~rst_3))
    begin
        startHash <= 0;
        newMessage <= 0;
        data[0] <= 0;
        data[1] <= 0;
        data[2] <= 0;
        ...
        data[14] <= 0;
        data[15] <= 0;
    end
    else if(hashValid && ~hashValid_r)
    begin
        data[0] <= 0;
        data[1] <= 0;
        data[2] <= 0;
        ...
        data[14] <= 0;
        data[15] <= 0;
    end
end
...
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data
150	Collect Data from Common Resource Locations
204	Lifting Sensitive Data Embedded in Cache
545	Pull Data from System Resources

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf> >.2023-04-07.

[REF-1055]Peter Gutmann. "Data Remanence in Semiconductor Devices". 10th USENIX Security Symposium. 2001 August. < https://www.usenix.org/legacy/events/sec01/full_papers/gutmann/gutmann.pdf >.

[REF-1379]"sha256_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/sha256/sha256_wrapper.sv#L94-L116 >.2023-12-13.

[REF-1380]"Fix for sha256_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/e8ba396b5c7cec9031e0e0e18ac547f32cd0ed50/piton/design/chip/tile/ariane/src/sha256/sha256_wrapper.sv#L98C1-L139C18 >.2023-12-13.

CWE-1240: Use of a Cryptographic Primitive with a Risky Implementation

Weakness ID : 1240

Structure : Simple

Abstraction : Base

Description

To fulfill the need for a cryptographic primitive, the product implements a cryptographic algorithm using a non-standard, unproven, or disallowed/non-compliant cryptographic implementation.

Extended Description

Cryptographic protocols and systems depend on cryptographic primitives (and associated algorithms) as their basic building blocks. Some common examples of primitives are digital signatures, one-way hash functions, ciphers, and public key cryptography; however, the notion of "primitive" can vary depending on point of view. See "Terminology Notes" for further explanation of some concepts.

Cryptographic primitives are defined to accomplish one very specific task in a precisely defined and mathematically reliable fashion. For example, suppose that for a specific cryptographic primitive (such as an encryption routine), the consensus is that the primitive can only be broken after trying out N different inputs (where the larger the value of N, the stronger the cryptography). For an encryption scheme like AES-256, one would expect N to be so large as to be infeasible to execute in a reasonable amount of time.

If a vulnerability is ever found that shows that one can break a cryptographic primitive in significantly less than the expected number of attempts, then that primitive is considered weakened (or sometimes in extreme cases, colloquially it is "broken"). As a result, anything using this cryptographic primitive would now be considered insecure or risky. Thus, even breaking or weakening a seemingly small cryptographic primitive has the potential to render the whole system vulnerable, due to its reliance on the primitive. A historical example can be found in TLS when using DES. One would colloquially call DES the cryptographic primitive for transport encryption in this version of TLS. In the past, DES was considered strong, because no weaknesses were found in it; importantly, DES has a key length of 56 bits. Trying $N=2^{56}$ keys was considered impractical for most actors. Unfortunately, attacking a system with 56-bit keys is now practical via brute force, which makes defeating DES encryption practical. It is now practical for an adversary to read any information sent under this version of TLS and use this information to attack the system. As a result, it can be claimed that this use of TLS is weak, and that any system depending on TLS with DES could potentially render the entire system vulnerable to attack.

Cryptographic primitives and associated algorithms are only considered safe after extensive research and review from experienced cryptographers from academia, industry, and government entities looking for any possible flaws. Furthermore, cryptographic primitives and associated algorithms are frequently reevaluated for safety when new mathematical and attack techniques are discovered. As a result and over time, even well-known cryptographic primitives can lose their

compliance status with the discovery of novel attacks that might either defeat the algorithm or reduce its robustness significantly.

If ad-hoc cryptographic primitives are implemented, it is almost certain that the implementation will be vulnerable to attacks that are well understood by cryptographers, resulting in the exposure of sensitive information and other consequences.

This weakness is even more difficult to manage for hardware-implemented deployment of cryptographic algorithms. First, because hardware is not patchable as easily as software, any flaw discovered after release and production typically cannot be fixed without a recall of the product. Secondly, the hardware product is often expected to work for years, during which time computation power available to the attacker only increases. Therefore, for hardware implementations of cryptographic primitives, it is absolutely essential that only strong, proven cryptographic primitives are used.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		327	Use of a Broken or Risky Cryptographic Algorithm	799

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

Nature	Type	ID	Name	Page
MemberOf		310	Cryptographic Issues	2318

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data <i>Incorrect usage of crypto primitives could render the supposedly encrypted data as unencrypted plaintext in the worst case.</i>	High

Detection Methods

Architecture or Design Review

Review requirements, documentation, and product design to ensure that primitives are consistent with the strongest-available recommendations from trusted parties. If the product appears to be using custom or proprietary implementations that have not had sufficient public review and approval, then this is a significant concern.

Effectiveness = High

Manual Analysis

Analyze the product to ensure that implementations for each primitive do not contain any known vulnerabilities and are not using any known-weak algorithms, including MD4, MD5, SHA1, DES, etc.

Effectiveness = Moderate

Dynamic Analysis with Manual Results Interpretation

For hardware, during the implementation (pre-Silicon / post-Silicon) phase, dynamic tests should be done to ensure that outputs from cryptographic routines are indeed working properly, such as test vectors provided by NIST [REF-1236].

Effectiveness = Moderate

Dynamic Analysis with Manual Results Interpretation

It needs to be determined if the output of a cryptographic primitive is lacking entropy, which is one clear sign that something went wrong with the crypto implementation. There exist many methods of measuring the entropy of a bytestream, from sophisticated ones (like calculating Shannon's entropy of a sequence of characters) to crude ones (by compressing it and comparing the size of the original bytestream vs. the compressed - a truly random byte stream should not be compressible and hence the uncompressed and compressed bytestreams should be nearly identical in size).

Effectiveness = Moderate

Potential Mitigations

Phase: Requirements

Require compliance with the strongest-available recommendations from trusted parties, and require that compliance must be kept up-to-date, since recommendations evolve over time. For example, US government systems require FIPS 140-3 certification, which supersedes FIPS 140-2 [REF-1192] [REF-1226].

Effectiveness = High

Phase: Architecture and Design

Ensure that the architecture/design uses the strongest-available primitives and algorithms from trusted parties. For example, US government systems require FIPS 140-3 certification, which supersedes FIPS 140-2 [REF-1192] [REF-1226].

Effectiveness = High

Phase: Architecture and Design

Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. As with all cryptographic mechanisms, the source code should be available for analysis. If the algorithm may be compromised when attackers find out how it works, then it is especially weak.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Try not to use cryptographic algorithms in novel ways or with new modes of operation even when you "know" it is secure. For example, using SHA-2 chaining to create a 1-time pad for encryption might sound like a good idea, but one should not do this.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Ensure that the design can replace one cryptographic primitive or algorithm with another in the next generation ("cryptographic agility"). Where possible, use wrappers to make the interfaces uniform. This will make it easier to upgrade to stronger algorithms. This is especially important for

hardware, which can be more difficult to upgrade quickly than software; design the hardware at a replaceable block level.

Effectiveness = Defense in Depth

Phase: Architecture and Design

Do not use outdated or non-compliant cryptography algorithms. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong [REF-267].

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Phase: Implementation

Do not use a linear-feedback shift register (LFSR) or other legacy methods as a substitute for an accepted and standard Random Number Generator.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Phase: Implementation

Do not use a checksum as a substitute for a cryptographically generated hash.

Effectiveness = Discouraged Common Practice

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted cryptographic library or framework. Industry-standard implementations will save development time and are more likely to avoid errors that can occur during implementation of cryptographic algorithms. However, the library/framework could be used incorrectly during implementation.

Effectiveness = High

Phase: Architecture and Design

Phase: Implementation

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for the prevention of common attacks.

Effectiveness = Moderate

Phase: Architecture and Design

Phase: Implementation

Do not store keys in areas accessible to untrusted agents. Carefully manage and protect the cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography algorithm is irrelevant.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

Re-using random values may compromise security.

Example Language:

(Bad)

Suppose an Encryption algorithm needs a random value for a key. Instead of using a DRNG (Deterministic Random Number Generator), the designer uses a linear-feedback shift register (LFSR) to generate the value.

While an LFSR may provide pseudo-random number generation service, the entropy (measure of randomness) of the resulting output may be less than that of an accepted DRNG (like that used in dev/urandom). Thus, using an LFSR weakens the strength of the cryptographic system, because it may be possible for an attacker to guess the LFSR output and subsequently the encryption key.

Example Language:

(Good)

If a cryptographic algorithm expects a random number as its input, provide one. Do not provide a pseudo-random value.

Observed Examples

Reference	Description
CVE-2020-4778	software uses MD5, which is less safe than the default SHA-256 used by related products https://www.cve.org/CVERecord?id=CVE-2020-4778
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates. https://www.cve.org/CVERecord?id=CVE-2005-2946
CVE-2019-3907	identity card uses MD5 hash of a salt and password https://www.cve.org/CVERecord?id=CVE-2019-3907
CVE-2021-34687	personal key is transmitted over the network using a substitution cipher https://www.cve.org/CVERecord?id=CVE-2021-34687
CVE-2020-14254	product does not disable TLS-RSA cipher suites, allowing decryption of traffic if TLS 2.0 and secure ciphers are not enabled. https://www.cve.org/CVERecord?id=CVE-2020-14254
CVE-2019-1543	SSL/TLS library generates 16-byte nonces but reduces them to 12 byte nonces for the ChaCha20-Poly1305 cipher, converting them in a way that violates the cipher's requirements for unique nonces. https://www.cve.org/CVERecord?id=CVE-2019-1543
CVE-2017-9267	LDAP interface allows use of weak ciphers https://www.cve.org/CVERecord?id=CVE-2017-9267
CVE-2017-7971	SCADA product allows "use of outdated cipher suites" https://www.cve.org/CVERecord?id=CVE-2017-7971
CVE-2020-6616	Chip implementing Bluetooth uses a low-entropy PRNG instead of a hardware RNG, allowing spoofing. https://www.cve.org/CVERecord?id=CVE-2020-6616
CVE-2019-1715	security product has insufficient entropy in the DRBG, allowing collisions and private key discovery https://www.cve.org/CVERecord?id=CVE-2019-1715
CVE-2014-4192	Dual_EC_DRBG implementation in RSA toolkit does not correctly handle certain byte requests, simplifying plaintext recovery https://www.cve.org/CVERecord?id=CVE-2014-4192
CVE-2007-6755	Recommendation for Dual_EC_DRBG algorithm contains point Q constants that could simplify decryption https://www.cve.org/CVERecord?id=CVE-2007-6755

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2473
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf	C	1402	Comprehensive Categorization: Encryption	1400	2527

Notes

Terminology

Terminology for cryptography varies widely, from informal and colloquial to mathematically-defined, with different precision and formalism depending on whether the stakeholder is a developer, cryptologist, etc. Yet there is a need for CWE to be self-consistent while remaining understandable and acceptable to multiple audiences. As of CWE 4.6, CWE terminology around "primitives" and "algorithms" is emerging as shown by the following example, subject to future consultation and agreement within the CWE and cryptography communities. Suppose one wishes to send encrypted data using a CLI tool such as OpenSSL. One might choose to use AES with a 256-bit key and require tamper protection (GCM mode, for instance). For compatibility's sake, one might also choose the ciphertext to be formatted to the PKCS#5 standard. In this case, the "cryptographic system" would be AES-256-GCM with PKCS#5 formatting. The "cryptographic function" would be AES-256 in the GCM mode of operation, and the "algorithm" would be AES. Colloquially, one would say that AES (and sometimes AES-256) is the "cryptographic primitive," because it is the algorithm that realizes the concept of symmetric encryption (without modes of operation or other protocol related modifications). In practice, developers and architects typically refer to base cryptographic algorithms (AES, SHA, etc.) as cryptographic primitives.

Maintenance

Since CWE 4.4, various cryptography-related entries, including CWE-327 and CWE-1240, have been slated for extensive research, analysis, and community consultation to define consistent terminology, improve relationships, and reduce overlap or duplication. As of CWE 4.6, this work is still ongoing.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
97	Cryptanalysis

References

- [REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf> >.2023-04-07.
- [REF-1227]Wikipedia. "Cryptographic primitive". < https://en.wikipedia.org/wiki/Cryptographic_primitive >.
- [REF-1226]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-2: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < <https://csrc.nist.gov/publications/detail/fips/140/2/final> >.
- [REF-1192]Information Technology Laboratory, National Institute of Standards and Technology. "FIPS PUB 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2019 March 2. < <https://csrc.nist.gov/publications/detail/fips/140/3/final> >.
- [REF-1236]NIST. "CAVP Testing: Individual Component Testing". < <https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/component-testing> >.

CWE-1241: Use of Predictable Algorithm in Random Number Generator

Weakness ID : 1241**Structure :** Simple**Abstraction :** Base

Description

The device uses an algorithm that is predictable and generates a pseudo-random number.

Extended Description

Pseudo-random number generator algorithms are predictable because their registers have a finite number of possible states, which eventually lead to repeating patterns. As a result, pseudo-random number generators (PRNGs) can compromise their randomness or expose their internal state to various attacks, such as reverse engineering or tampering. It is highly recommended to use hardware-based true random number generators (TRNGs) to ensure the security of encryption schemes. TRNGs generate unpredictable, unbiased, and independent random numbers because they employ physical phenomena, e.g., electrical noise, as sources to generate random numbers.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	814

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

Nature	Type	ID	Name	Page
MemberOf		1213	Random Number Issues	2477

Applicable Platforms

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	High

Potential Mitigations

Phase: Architecture and Design

A true random number generator should be specified for cryptographic algorithms.

Phase: Implementation

A true random number generator should be implemented for cryptographic algorithms.

Demonstrative Examples

Example 1:

Suppose a cryptographic function expects random value to be supplied for the crypto algorithm.

During the implementation phase, due to space constraint, a cryptographically secure random-number-generator could not be used, and instead of using a TRNG (True Random Number Generator), a LFSR (Linear Feedback Shift Register) is used to generate a random value. While an LFSR will provide a pseudo-random number, its entropy (measure of randomness) is insufficient for a cryptographic algorithm.

Example 2:

The example code is taken from the PRNG inside the buggy OpenPiton SoC of HACK@DAC'21 [REF-1370]. The SoC implements a pseudo-random number generator using a Linear Feedback Shift Register (LFSR).

An example of LFSR with the polynomial function $P(x) = x^6 + x^4 + x^3 + 1$ is shown in the figure.

Example Language: Verilog

(Bad)

```
reg in_sr, entropy16_valid;
reg [15:0] entropy16;
assign entropy16_o = entropy16;
assign entropy16_valid_o = entropy16_valid;
always @ (*)
begin
    in_sr = ^ (poly_i [15:0] & entropy16 [15:0]);
end
```

A LFSR's input bit is determined by the output of a linear function of two or more of its previous states. Therefore, given a long cycle, a LFSR-based PRNG will enter a repeating cycle, which is predictable.

Observed Examples

Reference	Description
CVE-2021-3692	PHP framework uses mt_rand() function (Mersenne Twister) when generating tokens https://www.cve.org/CVERecord?id=CVE-2021-3692

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2473
MemberOf	C	1414	Comprehensive Categorization: Randomness	1400	2543

Notes

Maintenance

As of CWE 4.5, terminology related to randomness, entropy, and predictability can vary widely. Within the developer and other communities, "randomness" is used heavily. However, within cryptography, "entropy" is distinct, typically implied as a measurement. There are no commonly-used definitions, even within standards documents and cryptography papers. Future versions of CWE will attempt to define these terms and, if necessary, distinguish between them in ways that are appropriate for different communities but do not reduce the usability of CWE for mapping, understanding, or other scenarios.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
97	Cryptanalysis

References

[REF-1370]"rng_16.v". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/main/piton/design/chip/tile/ariane/src/rand_num/rng_16.v#L12-L22 >.2023-07-15.

CWE-1242: Inclusion of Undocumented Features or Chicken Bits

Weakness ID : 1242

Structure : Simple

Abstraction : Base

Description

The device includes chicken bits or undocumented features that can create entry points for unauthorized actors.

Extended Description

A common design practice is to use undocumented bits on a device that can be used to disable certain functional security features. These bits are commonly referred to as "chicken bits". They can facilitate quick identification and isolation of faulty components, features that negatively affect performance, or features that do not provide the required controllability for debug and test. Another way to achieve this is through implementation of undocumented features. An attacker might exploit these interfaces for unauthorized access.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Integrity	Read Memory	
Availability	Execute Unauthorized Code or Commands	
Access Control	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

The implementation of chicken bits in a released product is highly discouraged. If implemented at all, ensure that they are disabled in production devices. All interfaces to a device should be documented.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider a device that comes with various security measures, such as secure boot. The secure-boot process performs firmware-integrity verification at boot time, and this code is stored in a separate SPI-flash device. However, this code contains undocumented "special access features" intended to be used only for performing failure analysis and intended to only be unlocked by the device designer.

Example Language: Other

(Bad)

Attackers dump the code from the device and then perform reverse engineering to analyze the code. The undocumented, special-access features are identified, and attackers can activate them by sending specific commands via UART before secure-boot phase completes. Using these hidden features, attackers can perform reads and writes to memory via the UART interface. At runtime, the attackers can also execute arbitrary code and dump the entire memory contents.

Remove all chicken bits and hidden features that are exposed to attackers. Add authorization schemes that rely on cryptographic primitives to access any features that the manufacturer does not want to expose. Clearly document all interfaces.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf		1371	ICS Supply Chain: Poorly Documented or Undocumented Features	1358	2508
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-1		Req SVV-3
ISA/IEC 62443	Part 4-2		Req CR 2.12

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
36	Using Unpublished Interfaces or Functionality
212	Functionality Misuse

References

[REF-1071]Ali Abbasi, Tobias Scharnowski and Thorsten Holz. "Doors of Durin: The Veiled Gate to Siemens S7 Silicon". < <https://i.blackhat.com/eu-19/Wednesday/eu-19-Abbasi-Doors-Of-Durin-The-Veiled-Gate-To-Siemens-S7-Silicon.pdf> >.

[REF-1072]Sergei Skorobogatov and Christopher Woods. "Breakthrough Silicon Scanning Discovers Backdoor in Military Chip". < https://www.cl.cam.ac.uk/~sps32/Silicon_scan_draft.pdf >.

[REF-1073]Chris Domas. "God Mode Unlocked: Hardware Backdoors in x86 CPUs". < <https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPU.pdf> >.

[REF-1074]Jonathan Brossard. "Hardware Backdooring is Practical". < https://media.blackhat.com/bh-us-12/Briefings/Brossard/BH_US_12_Brossard_Backdoor_Hacking_Slides.pdf >.

[REF-1075]Sergei Skorobogatov. "Security, Reliability, and Backdoors". < https://www.cl.cam.ac.uk/~sps32/SG_talk_SRB.pdf >.

CWE-1243: Sensitive Non-Volatile Information Not Protected During Debug

Weakness ID : 1243

Structure : Simple

Abstraction : Base

Description

Access to security-sensitive information stored in fuses is not limited during debug.


Extended Description

Several security-sensitive values are programmed into fuses to be used during early-boot flows or later at runtime. Examples of these security-sensitive values include root keys, encryption keys, manufacturing-specific information, chip-manufacturer-specific information, and original-equipment-manufacturer (OEM) data. After the chip is powered on, these values are sensed from fuses and stored in temporary locations such as registers and local memories. These locations are typically access-control protected from untrusted agents capable of accessing them. Even to trusted agents, only read-access is provided. However, these locations are not blocked during debug operations, allowing a user to access this sensitive information.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1263	Improper Physical Access Control	2085

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Disable access to security-sensitive information stored in fuses directly and also reflected from temporary storage locations when in debug mode.

Demonstrative Examples

Example 1:

Sensitive manufacturing data (such as die information) are stored in fuses. When the chip powers on, these values are read from the fuses and stored in microarchitectural registers. These registers are only given read access to trusted software running on the core. Untrusted software running on the core is not allowed to access these registers.

*Example Language: Other**(Bad)*

All microarchitectural registers in this chip can be accessed through the debug interface. As a result, even an untrusted debugger can access this data and retrieve sensitive manufacturing data.

*Example Language:**(Good)*

Registers used to store sensitive values read from fuses should be blocked during debug. These registers should be disconnected from the debug interface.

Example 2:

The example code below is taken from one of the AES cryptographic accelerators of the HACK@DAC'21 buggy OpenPiton SoC [REF-1366]. The operating system (OS) uses three AES keys to encrypt and decrypt sensitive data using this accelerator. These keys are sensitive data stored in fuses. The security of the OS will be compromised if any of these AES keys are leaked. During system bootup, these AES keys are sensed from fuses and stored in temporary hardware registers of the AES peripheral. Access to these temporary registers is disconnected during the debug state to prevent them from leaking through debug access. In this example (see the vulnerable code source), the registers key0, key1, and key2 are used to store the three AES keys (which are accessed through key_big0, key_big1, and key_big2 signals). The OS selects one of these three keys through the key_big signal, which is used by the AES engine.

*Example Language: Verilog**(Bad)*

```
...
assign key_big0 = debug_mode_i ? 192'b0 : {key0[0],
key0[1], key0[2], key0[3], key0[4], key0[5]};
assign key_big1 = debug_mode_i ? 192'b0 : {key1[0],
key1[1], key1[2], key1[3], key1[4], key1[5]};
assign key_big2 = {key2[0], key2[1], key2[2],
key2[3], key2[4], key2[5]};
...
assign key_big = key_sel[1] ? key_big2 : ( key_sel[0] ?
key_big1 : key_big0 );
...
```

The above code illustrates an instance of a vulnerable implementation for blocking AES key mechanism when the system is in debug mode (i.e., when debug_mode_i is asserted). During debug mode, key accesses through key_big0 and key_big1 are effectively disconnected, as their values are set to zero. However, the key accessed via the key_big2 signal remains accessible, creating a potential pathway for sensitive fuse data leakage, specifically AES key2, during debug mode. Furthermore, even though it is not strictly necessary to disconnect the key_big signal when entering debug mode (since disconnecting key_big0, key_big1, and key_big2 will inherently disconnect key_big), it is advisable, in line with the defense-in-depth strategy, to also sever the connection to key_big. This additional security measure adds an extra layer of protection and safeguards the AES keys against potential future modifications to the key_big logic.



To mitigate this, disconnect access through key_big2 and key_big during debug mode [REF-1367].

*Example Language: Verilog**(Good)*

```
...
assign key_big0 = debug_mode_i ? 192'b0 : {key0[0],
key0[1], key0[2], key0[3], key0[4], key0[5]};
assign key_big1 = debug_mode_i ? 192'b0 : {key1[0],
key1[1], key1[2], key1[3], key1[4], key1[5]};
assign key_big2 = debug_mode_i ? 192'b0 : {key2[0],
key2[1], key2[2], key2[3], key2[4], key2[5]};
...
assign key_big = debug_mode_i ? 192'b0 : ( key_sel[1] ?
key_big2 : ( key_sel[0] ? key_big1 : key_big0 ) );
...
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
116	Excavation
545	Pull Data from System Resources

References

[REF-1366]"aes0_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/71103971e8204de6a61afc17d3653292517d32bf/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L56C1-L57C1 >.2023-07-15.

[REF-1367]"fix cwe_1243 in aes0_wrapper.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/cde1d9d6888bffab21d4b405cce6f1b19c58dd3c/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L56 >.2023-09-28.

CWE-1244: Internal Asset Exposed to Unsafe Debug Access Level or State

Weakness ID : 1244

Structure : Simple

Abstraction : Base

Description

The product uses physical debug or test interfaces with support for multiple access levels, but it assigns the wrong debug access level to an internal asset, providing unintended access to the asset from untrusted debug agents.

Extended Description

Debug authorization can have multiple levels of access, defined such that different system internal assets are accessible based on the current authorized debug level. Other than debugger authentication (e.g., using passwords or challenges), the authorization can also be based on the system state or boot stage. For example, full system debug access might only be allowed early in boot after a system reset to ensure that previous session data is not accessible to the authenticated debugger.

If this protection mechanism does not ensure that internal assets have the correct debug access level during each boot stage or change in system state, an attacker could obtain sensitive information from the internal asset using a debugger.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		863	Incorrect Authorization	1787

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Integrity	Modify Memory	
Authorization Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Detection Methods

Manual Analysis

Check 2 devices for their passcode to authenticate access to JTAG/debugging ports. If the passcodes are missing or the same, update the design to fix and retest. Check communications over JTAG/debugging ports for encryption. If the communications are not encrypted, fix the design and retest.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

For security-sensitive assets accessible over debug/test interfaces, only allow trusted agents.

Effectiveness = High

Phase: Architecture and Design

Apply blinding [REF-1219] or masking techniques in strategic areas.

Effectiveness = Limited

Phase: Implementation

Add shielding or tamper-resistant protections to the device, which increases the difficulty and cost for accessing debug/test interfaces.

Effectiveness = Limited

Demonstrative Examples

Example 1:

The JTAG interface is used to perform debugging and provide CPU core access for developers. JTAG-access protection is implemented as part of the JTAG_SHIELD bit in the hw_digctl_ctrl register. This register has no default value at power up and is set only after the system boots from ROM and control is transferred to the user software.

*Example Language: Other**(Bad)*

This means that since the end user has access to JTAG at system reset and during ROM code execution before control is transferred to user software, a JTAG user can modify the boot flow and subsequently disclose all CPU information, including data-encryption keys.

*Example Language:**(Informative)*

The default value of this register bit should be set to 1 to prevent the JTAG from being enabled at system reset.

Example 2:

The example code below is taken from the CVA6 processor core of the HACK@DAC'21 buggy OpenPiton SoC. Debug access allows users to access internal hardware registers that are otherwise not exposed for user access or restricted access through access control protocols. Hence, requests to enter debug mode are checked and authorized only if the processor has sufficient privileges. In addition, debug accesses are also locked behind password checkers. Thus, the processor enters debug mode only when the privilege level requirement is met, and the correct debug password is provided.

The following code [REF-1377] illustrates an instance of a vulnerable implementation of debug mode. The core correctly checks if the debug requests have sufficient privileges and enables the debug_mode_d and debug_mode_q signals. It also correctly checks for debug password and enables umode_i signal.

*Example Language: Verilog**(Bad)*

```
module csr_regfile #(
...
    // check that we actually want to enter debug depending on the privilege level we are currently in
    unique case (priv_lvl_o)
        riscv::PRIV_LVL_M: begin
            debug_mode_d = dcsr_q.ebreakm;
        ...
        riscv::PRIV_LVL_U: begin
            debug_mode_d = dcsr_q.ebreaku;
        ...
    assign priv_lvl_o = (debug_mode_q || umode_i) ? riscv::PRIV_LVL_M : priv_lvl_q;
    ...
    debug_mode_q <= debug_mode_d;
    ...
endmodule
```

However, it grants debug access and changes the privilege level, priv_lvl_o, even when one of the two checks is satisfied and the other is not. Because of this, debug access can be granted by simply requesting with sufficient privileges (i.e., debug_mode_q is enabled) and failing the password check (i.e., umode_i is disabled). This allows an attacker to bypass the debug password checking and gain debug access to the core, compromising the security of the processor.

A fix to this issue is to only change the privilege level of the processor when both checks are satisfied, i.e., the request has enough privileges (i.e., debug_mode_q is enabled) and the password checking is successful (i.e., umode_i is enabled) [REF-1378].

*Example Language: Verilog**(Good)*

```
module csr_regfile #(
...
    // check that we actually want to enter debug depending on the privilege level we are currently in
    unique case (priv_lvl_o)
        riscv::PRIV_LVL_M: begin
            debug_mode_d = dcsr_q.ebreakm;
        ...
        riscv::PRIV_LVL_U: begin
            debug_mode_d = dcsr_q.ebreaku;
        ...
    endmodule
```

```

debug_mode_d = dcsr_q.ebreaku;
...
assign priv_lvl_o = (debug_mode_q && umode_i) ? riscv::PRIV_LVL_M : priv_lvl_q;
...
debug_mode_q <= debug_mode_d;
...




```

Observed Examples

Reference	Description
CVE-2019-18827	After ROM code execution, JTAG access is disabled. But before the ROM code is executed, JTAG access is possible, allowing a user full system access. This allows a user to modify the boot flow and successfully bypass the secure-boot process. https://www.cve.org/CVERecord?id=CVE-2019-18827

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Relationship

CWE-1191 and CWE-1244 both involve physical debug access, but the weaknesses are different. CWE-1191 is effectively about missing authorization for a debug interface, i.e. JTAG. CWE-1244 is about providing internal assets with the wrong debug access level, exposing the asset to untrusted debug agents.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
114	Authentication Abuse

References

[REF-1056]F-Secure Labs. "Multiple Vulnerabilities in Barco Clickshare: JTAG access is not permanently disabled". < <https://labs.f-secure.com/advisories/multiple-vulnerabilities-in-barco-clickshare/> >.

[REF-1057]Kurt Rosenfeld and Ramesh Karri. "Attacks and Defenses for JTAG". < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5406671> >.

[REF-1219]Monodeep Kar, Arvind Singh, Santosh Ghosh, Sanu Mathew, Anand Rajan, Vivek De, Raheem Beyah and Saibal Mukhopadhyay. "Blindsight: Blinding EM Side-Channel Leakage using Built-In Fully Integrated Inductive Voltage Regulator". 2018 February. < <https://arxiv.org/pdf/1802.09096.pdf> >.2023-04-07.

[REF-1377]"csr_regfile.sv line 938". 2021. < https://github.com/HACK-EVENT/hackatdac19/blob/57e7b2109c1ea2451914878df2e6ca740c2dcf34/src/csr_regfile.sv#L938 >.2023-12-13.

[REF-1378]"Fix for csr_regfile.sv line 938". 2021. < https://github.com/HACK-EVENT/hackatdac19/blob/a7b61209e56c48eec585eeedea8413997ec71e4a/src/csr_regfile.sv#L938C31-L938C56 >.2023-12-13.

CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic

Weakness ID : 1245
Structure : Simple
Abstraction : Base

Description

Faulty finite state machines (FSMs) in the hardware logic allow an attacker to put the system in an undefined state, to cause a denial of service (DoS) or gain privileges on the victim's system.

Extended Description

The functionality and security of the system heavily depend on the implementation of FSMs. FSMs can be used to indicate the current security state of the system. Lots of secure data operations and data transfers rely on the state reported by the FSM. Faulty FSM designs that do not account for all states, either through undefined states (left as don't cares) or through incorrect implementation, might lead an attacker to drive the system into an unstable state from which the system cannot recover without a reset, thus causing a DoS. Depending on what the FSM is used for, an attacker might also gain additional privileges to launch further attacks and compromise the security guarantees.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		684	Incorrect Provision of Specified Functionality	1505

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Availability	Unexpected State	
Access Control	DoS: Crash, Exit, or Restart DoS: Instability Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Define all possible states and handle all unused states through default statements. Ensure that system defaults to a secure state.

Effectiveness = High

Demonstrative Examples

Example 1:

The Finite State Machine (FSM) shown in the "bad" code snippet below assigns the output ("out") based on the value of state, which is determined based on the user provided input ("user_input").

Example Language: Verilog (Bad)

```
module fsm_1(out, user_input, clk, rst_n);
input [2:0] user_input;
input clk, rst_n;
output reg [2:0] out;
reg [1:0] state;
always @ (posedge clk or negedge rst_n )
begin
    if (!rst_n)
        state = 3'h0;
    else
        case (user_input)
            3'h0:
            3'h1:
            3'h2:
            3'h3: state = 2'h3;
            3'h4: state = 2'h2;
            3'h5: state = 2'h1;
        endcase
    end
    out <= {1'h1, state};
endmodule
```

The case statement does not include a default to handle the scenario when the user provides inputs of 3'h6 and 3'h7. Those inputs push the system to an undefined state and might cause a crash (denial of service) or any other unanticipated outcome.

Adding a default statement to handle undefined inputs mitigates this issue. This is shown in the "Good" code snippet below. The default statement is in bold.

Example Language: Verilog (Good)

```
case (user_input)
    3'h0:
    3'h1:
    3'h2:
    3'h3: state = 2'h3;
    3'h4: state = 2'h2;
    3'h5: state = 2'h1;
    default: state = 2'h0;
endcase
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices	1400	2538

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State

References

[REF-1060]Farimah Farahmandi and Prabhat Mishra. "FSM Anomaly Detection using Formal Analysis". < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8119228&tag=1> >.

CWE-1246: Improper Write Handling in Limited-write Non-Volatile Memories

Weakness ID : 1246

Structure : Simple

Abstraction : Base

Description

The product does not implement or incorrectly implements wear leveling operations in limited-write non-volatile memories.


Extended Description

Non-volatile memories such as NAND Flash, EEPROM, etc. have individually erasable segments, each of which can be put through a limited number of program/erase or write cycles. For example, the device can only endure a limited number of writes, after which the device becomes unreliable. In order to wear out the cells in a uniform manner, non-volatile memory and storage products based on the above-mentioned technologies implement a technique called wear leveling. Once a set threshold is reached, wear leveling maps writes of a logical block to a different physical block. This prevents a single physical block from prematurely failing due to a high concentration of writes. If wear leveling is improperly implemented, attackers may be able to programmatically cause the storage to become unreliable within a much shorter time than would normally be expected.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	964

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Storage Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Instability	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Include secure wear leveling algorithms and ensure they may not be bypassed.

Effectiveness = High

Demonstrative Examples

Example 1:

An attacker can render a memory line unusable by repeatedly causing a write to the memory line.

Below is example code from [REF-1058] that the user can execute repeatedly to cause line failure. W is the maximum associativity of any cache in the system; S is the size of the largest cache in the system.

Example Language: C++

(Attack)

```
// Do aligned alloc of (W+1) arrays each of size S
while(1) {
    for (ii = 0; ii < W + 1; ii++)
        array[ii].element[0]++;
}
```

Without wear leveling, the above attack will be successful. Simple randomization of blocks will not suffice as instead of the original physical block, the randomized physical block will be worn out.

Example Language:

(Good)

Wear leveling must be used to even out writes to the device.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1202	Memory and Storage Issues	1194	2472
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-1		Req SI-1
ISA/IEC 62443	Part 4-1		Req SVV-3

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
212	Functionality Misuse

References

[REF-1058]Moinuddin Qureshi, Michele Franchescini, Vijayalakshmi Srinivasan, Luis Lastras, Bulent Abali and John Karidis. "Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling". < <https://researcher.watson.ibm.com/researcher/files/us-moinqureshi/papers-sgap.pdf> >.2023-04-07.

[REF-1059]Micron. "Bad Block Management in NAND Flash Memory". < https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2959_bbm_in_nand_flash.pdf >.

CWE-1247: Improper Protection Against Voltage and Clock Glitches

Weakness ID : 1247**Structure :** Simple**Abstraction :** Base

Description

The device does not contain or contains incorrectly implemented circuitry or sensors to detect and mitigate voltage and clock glitches and protect sensitive information or software contained on the device.


Extended Description

A device might support features such as secure boot which are supplemented with hardware and firmware support. This involves establishing a chain of trust, starting with an immutable root of trust by checking the signature of the next stage (culminating with the OS and runtime software) against a golden value before transferring control. The intermediate stages typically set up the system in a secure state by configuring several access control settings. Similarly, security logic for exercising a debug or testing interface may be implemented in hardware, firmware, or both. A device needs to guard against fault attacks such as voltage glitches and clock glitches that an attacker may employ in an attempt to compromise the system.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1384	Improper Handling of Physical or Environmental Conditions	2257

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1332	Improper Handling of Faults that Lead to Instruction Skips	2227

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Technology : System on Chip (*Prevalence = Undetermined*)

Technology : Power Management Hardware (*Prevalence = Undetermined*)

Technology : Clock/Counter Hardware (*Prevalence = Undetermined*)

Technology : Sensor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Gain Privileges or Assume Identity	
Integrity	Bypass Protection Mechanism	
Availability	Read Memory	
Access Control	Modify Memory	

Scope	Impact	Likelihood
	Execute Unauthorized Code or Commands	

Detection Methods

Manual Analysis

Put the processor in an infinite loop, which is then followed by instructions that should not ever be executed, since the loop is not expected to exit. After the loop, toggle an I/O bit (for oscilloscope monitoring purposes), print a console message, and reenter the loop. Note that to ensure that the loop exit is actually captured, many NOP instructions should be coded after the loop branch instruction and before the I/O bit toggle and the print statement. Margining the clock consists of varying the clock frequency until an anomaly occurs. This could be a continuous frequency change or it could be a single cycle. The single cycle method is described here. For every 1000th clock pulse, the clock cycle is shortened by 10 percent. If no effect is observed, the width is shortened by 20%. This process is continued in 10% increments up to and including 50%. Note that the cycle time may be increased as well, down to seconds per cycle. Separately, the voltage is margined. Note that the voltage could be increased or decreased. Increasing the voltage has limits, as the circuitry may not be able to withstand a drastically increased voltage. This process starts with a 5% reduction of the DC supply to the CPU chip for 5 millisecond repeated at 1KHz. If this has no effect, the process is repeated, but a 10% reduction is used. This process is repeated at 10% increments down to a 50% reduction. If no effects are observed at 5 millisecond, the whole process is repeated using a 10 millisecond pulse. If no effects are observed, the process is repeated in 10 millisecond increments out to 100 millisecond pulses. While these are suggested starting points for testing circuitry for weaknesses, the limits may need to be pushed further at the risk of device damage. See [REF-1217] for descriptions of Smart Card attacks against a clock (section 14.6.2) and using a voltage glitch (section 15.5.3).

Effectiveness = Moderate

Dynamic Analysis with Manual Results Interpretation

During the implementation phase where actual hardware is available, specialized hardware tools and apparatus such as ChipWhisperer may be used to check if the platform is indeed susceptible to voltage and clock glitching attacks.

Architecture or Design Review

Review if the protections against glitching merely transfer the attack target. For example, suppose a critical authentication routine that an attacker would want to bypass is given the protection of modifying certain artifacts from within that specific routine (so that if the routine is bypassed, one can examine the artifacts and figure out that an attack must have happened). However, if the attacker has the ability to bypass the critical authentication routine, they might also have the ability to bypass the other protection routine that checks the artifacts. Basically, depending on these kind of protections is akin to resorting to "Security by Obscurity".

Architecture or Design Review

Many SoCs come equipped with a built-in Dynamic Voltage and Frequency Scaling (DVFS) that can control the voltage and clocks via software alone. However, there have been demonstrated attacks (like Plundervolt and CLKSCREW) that target this DVFS [REF-1081] [REF-1082]. During the design and implementation phases, one needs to check if the interface to this power management feature is available from unprivileged SW (CWE-1256), which would make the attack very easy.

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

At the circuit-level, using Tunable Replica Circuits (TRCs) or special flip-flops such as Razor flip-flops helps mitigate glitch attacks. Working at the SoC or platform base, level sensors may be

implemented to detect glitches. Implementing redundancy in security-sensitive code (e.g., where checks are performed) also can help with mitigation of glitch attacks.

Demonstrative Examples

Example 1:

Below is a representative snippet of C code that is part of the secure-boot flow. A signature of the runtime-firmware image is calculated and compared against a golden value. If the signatures match, the bootloader loads runtime firmware. If there is no match, an error halt occurs. If the underlying hardware executing this code does not contain any circuitry or sensors to detect voltage or clock glitches, an attacker might launch a fault-injection attack right when the signature check is happening (at the location marked with the comment), causing a bypass of the signature-checking process.

Example Language: C

(Bad)

```
...
if (signature_matches) // <-Glitch Here
{
    load_runtime_firmware();
}
else
{
    do_not_load_runtime_firmware();
}
...
```

After bypassing secure boot, an attacker can gain access to system assets to which the attacker should not have access.

Example Language:

(Good)

If the underlying hardware detects a voltage or clock glitch, the information can be used to prevent the glitch from being successful.

Observed Examples





Reference	Description
CVE-2019-17391	Lack of anti-glitch protections allows an attacker to launch a physical attack to bypass the secure boot and read protected eFuses. https://www.cve.org/CVERecord?id=CVE-2019-17391
CVE-2021-33478	IP communication firmware allows access to a boot shell via certain impulses https://www.cve.org/CVERecord?id=CVE-2021-33478

Functional Areas

- Power
- Clock

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf		1365	ICS Communications: Unreliability	1358	2502
MemberOf		1367	ICS Dependencies (& Architecture): External Physical Systems	1358	2504
MemberOf		1388	Physical Access Issues and Concerns	1194	2518

Nature	Type	ID	Name	V	Page
MemberOf	C	1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
624	Hardware Fault Injection
625	Mobile Device Fault Injection

References

[REF-1061]Keith Bowman, James Tschanz, Chris Wilkerson, Shih-Lien Lu, Tanay Karnik, Vivek De and Shekhar Borkar. "Circuit Techniques for Dynamic Variation Tolerance". < <https://dl.acm.org/doi/10.1145/1629911.1629915> >.2023-04-07.

[REF-1062]Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner and Trevor Mudge. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation". < <https://web.eecs.umich.edu/~taustin/papers/MICRO36-Razor.pdf> >.

[REF-1063]James Tschanz, Keith Bowman, Steve Walstra, Marty Agostinelli, Tanay Karnik and Vivek De. "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance". < <https://ieeexplore.ieee.org/document/5205410> >.

[REF-1064]Bilgiday Yuce, Nahid F. Ghalaty, Chinmay Deshpande, Conor Patrick, Leyla Nazhandali and Patrick Schaumont. "FAME: Fault-attack Aware Microprocessor Extensions for Hardware Fault Detection and Software Fault Response". < <https://dl.acm.org/doi/10.1145/2948618.2948626> >.2023-04-07.

[REF-1065]Keith A. Bowman, James W. Tschanz, Shih-Lien L. Lu, Paolo A. Aseron, Muhammad M. Khellah, Arijit Raychowdhury, Bibiche M. Geuskens, Carlos Tokunaga, Chris B. Wilkerson, Tanay Karnik and Vivek De. "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance". < <https://ieeexplore.ieee.org/document/5654663> >.

[REF-1066]Niek Timmers and Albert Spruyt. "Bypassing Secure Boot Using Fault Injection". < <https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf> >.

[REF-1217]Ross Anderson. "Security Engineering". 2001. < <https://www.cl.cam.ac.uk/~rja14/musicfiles/manuscripts/SEv1.pdf> >.

[REF-1217]Ross Anderson. "Security Engineering". 2001. < <https://www.cl.cam.ac.uk/~rja14/musicfiles/manuscripts/SEv1.pdf> >.

[REF-1081]Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Frank Piessens and Daniel Gruss. "Plundervolt". < <https://plundervolt.com/> >.

[REF-1082]Adrian Tang, Simha Sethumadhavan and Salvatore Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". < <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-tang.pdf> >.

[REF-1285]Texas Instruments. "Physical Security Attacks Against Silicon Devices". 2022 January 1. < <https://www.ti.com/lit/an/swra739/swra739.pdf?ts=1644234570420> >.

[REF-1286]Lennert Wouters, Benedikt Gierlichs and Bart Preneel. "On The Susceptibility of Texas Instruments SimpleLink Platform Microcontrollers to Non-Invasive Physical Attacks". 2022 March 4. < <https://eprint.iacr.org/2022/328.pdf> >.

CWE-1248: Semiconductor Defects in Hardware Logic with Security-Sensitive Implications

Weakness ID : 1248

Structure : Simple

Abstraction : Base

Description

The security-sensitive hardware module contains semiconductor defects.

Extended Description

A semiconductor device can fail for various reasons. While some are manufacturing and packaging defects, the rest are due to prolonged use or usage under extreme conditions. Some mechanisms that lead to semiconductor defects include encapsulation failure, die-attach failure, wire-bond failure, bulk-silicon defects, oxide-layer faults, aluminum-metal faults (including electromigration, corrosion of aluminum, etc.), and thermal/electrical stress. These defects manifest as faults on chip-internal signals or registers, have the effect of inputs, outputs, or intermediate signals being always 0 or always 1, and do not switch as expected. If such faults occur in security-sensitive hardware modules, the security objectives of the hardware module may be compromised.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	693	Protection Mechanism Failure	1520

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Instability	
Access Control		

Potential Mitigations

Phase: Testing

While semiconductor-manufacturing companies implement several mechanisms to continuously improve the semiconductor manufacturing process to ensure reduction of defects, some defects can only be fixed after manufacturing. Post-manufacturing testing of silicon die is critical. Fault models such as stuck-at-0 or stuck-at-1 must be used to develop post-manufacturing test cases and achieve good coverage. Once the silicon packaging is done, extensive post-silicon testing must be performed to ensure that hardware logic implementing security functionalities is defect-free.

Phase: Operation

Operating the hardware outside device specification, such as at extremely high temperatures, voltage, etc., accelerates semiconductor degradation and results in defects. When these defects

manifest as faults in security-critical, hardware modules, it results in compromise of security guarantees. Thus, operating the device within the specification is important.

Demonstrative Examples

Example 1:

The network-on-chip implements a firewall for access control to peripherals from all IP cores capable of mastering transactions.

Example Language: Other

(Bad)

A manufacturing defect in this logic manifests itself as a logical fault, which always sets the output of the filter to "allow" access.

Post-manufacture testing must be performed to ensure that hardware logic implementing security functionalities is defect-free.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2469
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf	C	1388	Physical Access Issues and Concerns	1194	2518
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
624	Hardware Fault Injection
625	Mobile Device Fault Injection

References

[REF-1067]Brian Bailey. "Why Chips Die". < <https://semiengineering.com/why-chips-die/> >.

[REF-1068]V. Lakshminarayan. "What causes semiconductor devices to fail". < Original >.2023-04-07.

CWE-1249: Application-Level Admin Tool with Inconsistent View of Underlying Operating System

Weakness ID : 1249

Structure : Simple

Abstraction : Base

Description

The product provides an application for administrators to manage parts of the underlying operating system, but the application does not accurately identify all of the relevant entities or resources that exist in the OS; that is, the application's model of the OS's state is inconsistent with the OS's actual state.

Extended Description

Many products provide web-based applications or other interfaces for managing the underlying operating system. This is common with cloud, network access devices, home networking, and other

systems. When the management tool does not accurately represent what is in the OS - such as user accounts - then the administrator might not see suspicious activities that would be noticed otherwise.

For example, numerous systems utilize a web front-end for administrative control. They also offer the ability to add, alter, and drop users with various privileges as it relates to the functionality of the system. A potential architectural weakness may exist where the user information reflected in the web interface does not mirror the users in the underlying operating system. Many web UI or REST APIs use the underlying operating system for authentication; the system's logic may also track an additional set of user capabilities within configuration files and datasets for authorization capabilities. When there is a discrepancy between the user information in the UI or REST API's interface system and the underlying operating system's user listing, this may introduce a weakness into the system. For example, if an attacker compromises the OS and adds a new user account - a "ghost" account - then the attacker could escape detection if the management tool does not list the newly-added account.

This discrepancy could be exploited in several ways:

- A rogue admin could insert a new account into a system that will persist if they are terminated or wish to take action on a system that cannot be directly associated with them.
- An attacker can leverage a separate command injection attack available through the web interface to insert a ghost account with shell privileges such as ssh.
- An attacker can leverage existing web interface APIs, manipulated in such a way that a new user is inserted into the operating system, and the user web account is either partially created or not at all.
- An attacker could create an admin account which is viewable by an administrator, use this account to create the ghost account, delete logs and delete the first created admin account.

Many of these attacker scenarios can be realized by leveraging separate vulnerabilities related to XSS, command injection, authentication bypass, or logic flaws on the various systems.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1250	Improper Preservation of Consistency Between Independent Representations of Shared State	2052

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

Technology : Web Based (*Prevalence = Undetermined*)

Alternate Terms

Ghost in the Shell :

Common Consequences

Scope	Impact	Likelihood
Access Control	Varies by Context	
Accountability	Hide Activities	

Scope	Impact	Likelihood
Other	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Ensure that the admin tool refreshes its model of the underlying OS on a regular basis, and note any inconsistencies with configuration files or other data sources that are expected to have the same data.

Demonstrative Examples

Example 1:

Suppose that an attacker successfully gains root privileges on a Linux system and adds a new 'user2' account:

Example Language: Other

(Attack)

```
echo "user2:x:0:0::/root:/" >> /etc/passwd;  
echo "user2:$6$ldvyrM6VJnG8Su5U$1gmW3Nm.IO4vxTQDQ1C8urm72JcAdOHZQwqiH/  
nRtL8dPY80xS4Ovsv5bPCMWnXKKWwmsocSWXupUf17LB3oS.:17256:0:99999:7:::" >> /etc/shadow;
```

This new user2 account would not be noticed on the web interface, if the interface does not refresh its data of available users.

It could be argued that for this specific example, an attacker with root privileges would be likely to compromise the admin tool or otherwise feed it with false data. However, this example shows how the discrepancy in critical data can help attackers to escape detection.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2544

References

[REF-1070]Tony Martin. "Ghost in the Shell Weakness". 2020 February 3. < <https://friendsglobal.com/ghost-in-the-shell/ghost-in-the-shell-weakness/> >.2023-04-07.

CWE-1250: Improper Preservation of Consistency Between Independent Representations of Shared State

Weakness ID : 1250

Structure : Simple

Abstraction : Base

Description

The product has or supports multiple distributed components or sub-systems that are each required to keep their own local copy of shared data - such as state or cache - but the product does not ensure that all local copies remain consistent with each other.

Extended Description

In highly distributed environments, or on systems with distinct physical components that operate independently, there is often a need for each component to store and update its own local copy of key data such as state or cache, so that all components have the same "view" of the overall system

and operate in a coordinated fashion. For example, users of a social media service or a massively multiplayer online game might be using their own personal computers while also interacting with different physical hosts in a globally distributed service, but all participants must be able to have the same "view" of the world. Alternately, a processor's Memory Management Unit (MMU) might have "shadow" MMUs to distribute its workload, and all shadow MMUs are expected to have the same accessible ranges of memory.

In such environments, it becomes critical for the product to ensure that this "shared state" is consistently modified across all distributed systems. If state is not consistently maintained across all systems, then critical transactions might take place out of order, or some users might not get the same data as other users. When this inconsistency affects correctness of operations, it can introduce vulnerabilities in mechanisms that depend on consistent state.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	664	Improper Control of a Resource Through its Lifetime	1454
ParentOf	[B]	1249	Application-Level Admin Tool with Inconsistent View of Underlying Operating System	2050
ParentOf	[B]	1251	Mirrored Regions with Different Values	2054

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Cloud Computing (*Prevalence = Undetermined*)

Technology : Security Hardware (*Prevalence = Undetermined*)

Demonstrative Examples

Example 1:

Suppose a processor's Memory Management Unit (MMU) has 5 other shadow MMUs to distribute its workload for its various cores. Each MMU has the start address and end address of "accessible" memory. Any time this accessible range changes (as per the processor's boot status), the main MMU sends an update message to all the shadow MMUs.

Suppose the interconnect fabric does not prioritize such "update" packets over other general traffic packets. This introduces a race condition. If an attacker can flood the target with enough messages so that some of those attack packets reach the target before the new access ranges gets updated, then the attacker can leverage this scenario.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	[C]	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	[C]	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Research Gap

Issues related to state and cache - creation, preservation, and update - are a significant gap in CWE that is expected to be addressed in future versions. It likely has relationships to concurrency and synchronization, incorrect behavior order, and other areas that already have some coverage in CWE, although the focus has typically been on independent processes on the same operating system - not on independent systems that are all a part of a larger system-of-systems.

References

[REF-1069]Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu and Haryadi S. Gunawi. "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems". 2016. < <https://ucare.cs.uchicago.edu/pdf/asplos16-TaxDC.pdf> >.

CWE-1251: Mirrored Regions with Different Values

Weakness ID : 1251**Structure** : Simple**Abstraction** : Base

Description

The product's architecture mirrors regions without ensuring that their contents always stay in sync.

Extended Description

Having mirrored regions with different values might result in the exposure of sensitive information or possibly system compromise.

In the interest of increased performance, one might need to duplicate a resource. A cache memory is a common example of this concept, which keeps a "local" copy of a data element in the high speed cache memory. Unfortunately, this speed improvement comes with a downside, since the product needs to ensure that the local copy always mirrors the original copy truthfully. If they get out of sync, the computational result is no longer true.

During hardware design, memory is not the only item which gets mirrored. There are many other entities that get mirrored, as well: registers, memory regions, and, in some cases, even whole computational units. For example, within a multi-core processor, if all memory accesses for each and every core goes through a single Memory-Management Unit (MMU) then the MMU will become a performance bottleneck. In such cases, duplicating local MMUs that will serve only a subset of the cores rather than all of them may resolve the performance issue. These local copies are also called "shadow copies" or "mirrored copies."

If the original resource never changed, local duplicate copies getting out of sync would never be an issue. However, the values of the original copy will sometimes change. When the original copy changes, the mirrored copies must also change, and change fast.

This situation of shadow-copy-possibly-out-of-sync-with-original-copy might occur as a result of multiple scenarios, including the following:


- After the values in the original copy change, due to some reason the original copy does not send the "update" request to its shadow copies.
- After the values in the original copy change, the original copy dutifully sends the "update" request to its shadow copies, but due to some reason the shadow copy does not "execute" this update request.

- After the values in the original copy change, the original copy sends the "update" request to its shadow copies, and the shadow copy executes this update request faithfully. However, during the small time period when the original copy has "new" values and the shadow copy is still holding the "old" values, an attacker can exploit the old values. Then it becomes a race condition between the attacker and the update process of who can reach the target, shadow copy first, and, if the attacker reaches first, the attacker wins.
- The attacker might send a "spoofed" update request to the target shadow copy, pretending that this update request is coming from the original copy. This spoofed request might cause the targeted shadow copy to update its values to some attacker-friendly values, while the original copies remain unchanged by the attacker.
- Suppose a situation where the original copy has a system of reverting back to its original value if it does not hear back from all the shadow copies that such copies have successfully completed the update request. In such a case, an attack might occur as follows: (1) the original copy might send an update request; (2) the shadow copy updates it; (3) the shadow copy sends back the successful completion message; (4) through a separate issue, the attacker is able to intercept the shadow copy's completion message. In this case, the original copy thinks that the update did not succeed, hence it reverts to its original value. Now there is a situation where the original copy has the "old" value, and the shadow copy has the "new" value.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1250	Improper Preservation of Consistency Between Independent Representations of Shared State	2052

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1312	Missing Protection for Mirrored Regions in On-Chip Fabric Firewall	2184

Applicable Platforms

Language : VHDL (*Prevalence = Undetermined*)

Language : Verilog (*Prevalence = Undetermined*)

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Whenever there are multiple, physically different copies of the same value that might change and the process to update them is not instantaneous and atomic, it is impossible to assert that the original and shadow copies will always be in sync - there will always be a time period when they are out of sync. To mitigate the consequential risk, the recommendations essentially are: Make this out-of-sync time period as small as possible, and Make the update process as robust as possible.

Effectiveness = Moderate

Demonstrative Examples



Example 1:

Suppose a processor's Memory Management Unit (MMU) has 5 other shadow MMUs to distribute its workload for its various cores. Each MMU has the start address and end address of "accessible" memory. Any time this accessible range changes (as per the processor's boot status), the main MMU sends an update message to all the shadow MMUs.

Suppose the interconnect fabric does not prioritize such "update" packets over other general traffic packets. This introduces a race condition. If an attacker can flood the target with enough messages so that some of those attack packets reach the target before the new access ranges gets updated, then the attacker can leverage this scenario.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1202	Memory and Storage Issues	1194	2472
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2544

Notes

Research Gap

Issues related to state and cache - creation, preservation, and update - are a significant gap in CWE that is expected to be addressed in future versions. It has relationships to concurrency and synchronization, incorrect behavior order, and other areas that already have some coverage in CWE, although the focus has typically been on independent processes on the same operating system - not on independent systems that are all a part of a larger system-of-systems.

CWE-1252: CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations

Weakness ID : 1252

Structure : Simple

Abstraction : Base

Description

The CPU is not configured to provide hardware support for exclusivity of write and execute operations on memory. This allows an attacker to execute data from all of memory.

Extended Description

CPUs provide a special bit that supports exclusivity of write and execute operations. This bit is used to segregate areas of memory to either mark them as code (instructions, which can be

executed) or data (which should not be executed). In this way, if a user can write to a region of memory, the user cannot execute from that region and vice versa. This exclusivity provided by special hardware bit is leveraged by the operating system to protect executable space. While this bit is available in most modern processors by default, in some CPUs the exclusivity is implemented via a memory-protection unit (MPU) and memory-management unit (MMU) in which memory regions can be carved out with exact read, write, and execute permissions. However, if the CPU does not have an MMU/MPU, then there is no write exclusivity. Without configuring exclusivity of operations via segregated areas of memory, an attacker may be able to inject malicious code onto memory and later execute it.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Microcontroller Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Implement a dedicated bit that can be leveraged by the Operating System to mark data areas as non-executable. If such a bit is not available in the CPU, implement MMU/MPU (memory management unit / memory protection unit).

Phase: Integration

If MMU/MPU are not available, then the firewalls need to be implemented in the SoC interconnect to mimic the write-exclusivity operation.

Demonstrative Examples

Example 1:

MCS51 Microcontroller (based on 8051) does not have a special bit to support write exclusivity. It also does not have an MMU/MPU support. The Cortex-M CPU has an optional MPU that supports up to 8 regions.

Example Language: Other

(Bad)

The optional MPU is not configured.

If the MPU is not configured, then an attacker will be able to inject malicious data into memory and execute it.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1201	Core and Compute Issues	1194	2471
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
679	Exploitation of Improperly Configured or Implemented Memory Protections

References

[REF-1076]ARM. "Cortex-R4 Manual". < <https://developer.arm.com/Processors/Cortex-M4> >.2023-04-07.

[REF-1077]Intel. "MCS 51 Microcontroller Family User's Manual". < <http://web.mit.edu/6.115/www/document/8051.pdf> >.

[REF-1078]ARM. "Memory Protection Unit (MPU)". < https://web.archive.org/web/20200630034848/https://static.docs.arm.com/100699/0100/armv8m_architecture_memory_protection_unit_100699_0100_00_en.pdf >.2023-04-07.

CWE-1253: Incorrect Selection of Fuse Values

Weakness ID : 1253

Structure : Simple

Abstraction : Base

Description

The logic level used to set a system to a secure state relies on a fuse being unblown. An attacker can set the system to an insecure state merely by blowing the fuse.

Extended Description

Fuses are often used to store secret data, including security configuration data. When not blown, a fuse is considered to store a logic 0, and, when blown, it indicates a logic 1. Fuses are generally considered to be one-directional, i.e., once blown to logic 1, it cannot be reset to logic 0. However, if the logic used to determine system-security state (by leveraging the values sensed from the fuses) uses negative logic, an attacker might blow the fuse and drive the system to an insecure state.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	
Authorization	Gain Privileges or Assume Identity	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Read Memory	
Integrity	Modify Memory Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Logic should be designed in a way that blown fuses do not put the product into an insecure state that can be leveraged by an attacker.

Demonstrative Examples

Example 1:

A chip implements a secure boot and uses the sensed value of a fuse "do_secure_boot" to determine whether to perform a secure boot or not. If this fuse value is "0", the system performs secure boot. Otherwise, it does not perform secure boot.




An attacker blows the "do_secure_boot" fuse to "1". After reset, the attacker loads a custom bootloader, and, since the fuse value is now "1", the system does not perform secure boot, and the attacker can execute their custom firmware image.

Since by default, a fuse-configuration value is a "0", an attacker can blow it to a "1" with inexpensive hardware.

If the logic is reversed, an attacker cannot easily reset the fuse. Note that, with specialized and expensive equipment, an attacker with full physical access might be able to "unblow" the fuse value to a "0".

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State

References

[REF-1080]Christopher Tarnovsky. "Security Failures in Secure Devices". < <https://www.blackhat.com/presentations/bh-europe-08/Tarnovsky/Presentation/bh-eu-08-tarnovsky.pdf> >.

CWE-1254: Incorrect Comparison Logic Granularity

Weakness ID : 1254

Structure : Simple

Abstraction : Base

Description

The product's comparison logic is performed over a series of steps rather than across the entire string in one operation. If there is a comparison logic failure on one of these steps, the operation may be vulnerable to a timing attack that can result in the interception of the process for nefarious purposes.

Extended Description

Comparison logic is used to compare a variety of objects including passwords, Message Authentication Codes (MACs), and responses to verification challenges. When comparison logic is implemented at a finer granularity (e.g., byte-by-byte comparison) and breaks in the case of a comparison failure, an attacker can exploit this implementation to identify when exactly the failure occurred. With multiple attempts, the attacker may be able to guess the correct password/response to challenge and elevate their privileges.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1530
ChildOf	B	208	Observable Timing Discrepancy	529
PeerOf	B	1261	Improper Handling of Single Event Upsets	2079

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Authorization	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

The hardware designer should ensure that comparison logic is implemented so as to compare in one operation instead in smaller chunks.

Demonstrative Examples

Example 1:

2060

Consider an example hardware module that checks a user-provided password to grant access to a user. The user-provided password is compared against a golden value in a byte-by-byte manner.

Example Language: Verilog (Bad)

```
always_comb @ (posedge clk)
begin
  assign check_pass[3:0] = 4'b0;
  for (i = 0; i < 4; i++) begin
    if (entered_pass[(i*8 - 1) : i] eq golden_pass[(i*8 - 1) : i])
      assign check_pass[i] = 1;
      continue;
    else
      assign check_pass[i] = 0;
      break;
    end
  assign grant_access = (check_pass == 4'b1111) ? 1'b1: 1'b0;
end
```

Since the code breaks on an incorrect entry of password, an attacker can guess the correct password for that byte-check iteration with few repeat attempts.

To fix this weakness, either the comparison of the entire string should be done all at once, or the attacker is not given an indication whether pass or fail happened by allowing the comparison to run through all bits before the grant_access signal is set.

Example Language: (Good)

```
always_comb @ (posedge clk)
begin
  assign check_pass[3:0] = 4'b0;
  for (i = 0; i < 4; i++) begin
    if (entered_pass[(i*8 - 1) : i] eq golden_pass[(i*8 - 1) : i])
      assign check_pass[i] = 1;
      continue;
    else
      assign check_pass[i] = 0;
      continue;
    end
  assign grant_access = (check_pass == 4'b1111) ? 1'b1: 1'b0;
end
```

Observed Examples

Reference	Description
CVE-2019-10482	Smartphone OS uses comparison functions that are not in constant time, allowing side channels https://www.cve.org/CVERecord?id=CVE-2019-10482
CVE-2019-10071	Java-oriented framework compares HMAC signatures using String.equals() instead of a constant-time algorithm, causing timing discrepancies https://www.cve.org/CVERecord?id=CVE-2019-10071
CVE-2014-0984	Password-checking function in router terminates validation of a password entry when it encounters the first incorrect character, which allows remote attackers to obtain passwords via a brute-force attack that relies on timing differences in responses to incorrect password guesses, aka a timing side-channel attack. https://www.cve.org/CVERecord?id=CVE-2014-0984

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions

References

[REF-1079]Joe Fitzpatrick. "SCA4n00bz - Timing-based Sidechannel Attacks for Hardware N00bz workshop". < <https://github.com/securelyfitz/SCA4n00bz> >.

CWE-1255: Comparison Logic is Vulnerable to Power Side-Channel Attacks

Weakness ID : 1255

Structure : Simple

Abstraction : Variant

Description

A device's real time power consumption may be monitored during security token evaluation and the information gleaned may be used to determine the value of the reference token.

Extended Description

The power consumed by a device may be instrumented and monitored in real time. If the algorithm for evaluating security tokens is not sufficiently robust, the power consumption may vary by token entry comparison against the reference value. Further, if retries are unlimited, the power difference between a "good" entry and a "bad" entry may be observed and used to determine whether each entry itself is correct thereby allowing unauthorized parties to calculate the reference 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	E	1300	Improper Protection of Physical Side Channels	2165

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf	E	1259	Improper Restriction of Security Token Assignment	2073

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	

Scope	Impact	Likelihood
Integrity	Read Memory	
Availability	Read Files or Directories	
Access Control	Modify Files or Directories	
Accountability	Execute Unauthorized Code or Commands	
Authentication	Gain Privileges or Assume Identity	
Authorization	Bypass Protection Mechanism	
Non-Repudiation	Read Application Data	
	Modify Application Data	
	Hide Activities	
	<i>As compromising a security token may result in complete system control, the impacts are relatively universal</i>	

Potential Mitigations

Phase: Architecture and Design

The design phase must consider each check of a security token against a standard and the amount of power consumed during the check of a good token versus a bad token. The alternative is an all at once check where a retry counter is incremented PRIOR to the check.

Phase: Architecture and Design

Another potential mitigation is to parallelize shifting of secret data (see example 2 below). Note that the wider the bus the more effective the result.

Phase: Architecture and Design

An additional potential mitigation is to add random data to each crypto operation then subtract it out afterwards. This is highly effective but costly in performance, area, and power consumption. It also requires a random number generator.

Phase: Implementation

If the architecture is unable to prevent the attack, using filtering components may reduce the ability to implement an attack, however, consideration must be given to the physical removal of the filter elements.

Phase: Integration

During integration, avoid use of a single secret for an extended period (e.g. frequent key updates). This limits the amount of data compromised but at the cost of complexity of use.

Demonstrative Examples

Example 1:

Consider an example hardware module that checks a user-provided password (or PIN) to grant access to a user. The user-provided password is compared against a stored value byte-by-byte.

Example Language: C

(Bad)

```
static nonvolatile password_tries = NUM_RETRIES;
do
    while (password_tries == 0) ; // Hang here if no more password tries
    password_ok = 0;
    for (i = 0; i < NUM_PW_DIGITS; i++)
        if (GetPasswordByte() == stored_password[i])
            password_ok |= 1; // Power consumption is different here
        else
            password_ok |= 0; // than from here
    end
    if (password_ok > 0)
        password_tries = NUM_RETRIES;
        break_to_Ok_to_proceed
    password_tries--;
```

```
while (true)
// Password OK
```

Since the algorithm uses a different number of 1's and 0's for password validation, a different amount of power is consumed for the good byte versus the bad byte comparison. Using this information, an attacker may be able to guess the correct password for that byte-by-byte iteration with several repeated attempts by stopping the password evaluation before it completes.

Among various options for mitigating the string comparison is obscuring the power consumption by having opposing bit flips during bit operations. Note that in this example, the initial change of the bit values could still provide power indication depending upon the hardware itself. This possibility needs to be measured for verification.

Example Language: C

(Good)

```
static nonvolatile password_tries = NUM_RETRIES;
do
    while (password_tries == 0) ; // Hang here if no more password tries
    password_tries--; // Put retry code here to catch partial retries
    password_ok = 0;
    for (i = 0; i < NUM_PW_DIGITS; i++)
        if (GetPasswordByte() == stored_password[i])
            password_ok |= 0x10; // Power consumption here
        else
            password_ok |= 0x01; // is now the same here
    end
    if ((password_ok & 1) == 0)
        password_tries = NUM_RETRIES;
        break_to_Ok_to_proceed
while (true)
// Password OK
```

Example 2:

This code demonstrates the transfer of a secret key using Serial-In/Serial-Out shift. It's easy to extract the secret using simple power analysis as each shift gives data on a single bit of the key.

Example Language: Verilog

(Bad)

```
module siso(clk,rst,a,q);
    input a;
    input clk,rst;
    output q;
    reg q;
    always@(posedge clk,posedge rst)
    begin
        if(rst==1'b1)
            q<1'b0;
        else
            q<a;
        end
    endmodule
```

This code demonstrates the transfer of a secret key using a Parallel-In/Parallel-Out shift. In a parallel shift, data confounded by multiple bits of the key, not just one.

Example Language: Verilog

(Good)

```
module pipo(clk,rst,a,q);
    input clk,rst;
    input[3:0]a;
    output[3:0]q;
    reg[3:0]q;
    always@(posedge clk,posedge rst)
    begin
```

```

if (rst==1'b1)
  q<4'b0000;
else
  q<a;
end
endmodule

```

Observed Examples

Reference	Description
CVE-2020-12788	CMAC verification vulnerable to timing and power attacks. https://www.cve.org/CVERecord?id=CVE-2020-12788

Functional Areas

- Power

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf	C	1388	Physical Access Issues and Concerns	1194	2518
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
189	Black Box Reverse Engineering

References

[REF-1184]Wikipedia. "Power Analysis". < https://en.wikipedia.org/wiki/Power_analysis >.

CWE-1256: Improper Restriction of Software Interfaces to Hardware Features

Weakness ID : 1256

Structure : Simple

Abstraction : Base

Description

The product provides software-controllable device functionality for capabilities such as power and clock management, but it does not properly limit functionality that can lead to modification of hardware memory or register bits, or the ability to observe physical side channels.

Extended Description

It is frequently assumed that physical attacks such as fault injection and side-channel analysis require an attacker to have physical access to the target device. This assumption may be false if the device has improperly secured power management features, or similar features. For mobile devices, minimizing power consumption is critical, but these devices run a wide variety of applications with different performance requirements. Software-controllable mechanisms to dynamically scale device voltage and frequency and monitor power consumption are common features in today's chipsets, but they also enable attackers to mount fault injection and side-channel attacks without having physical access to the device.

Fault injection attacks involve strategic manipulation of bits in a device to achieve a desired effect such as skipping an authentication step, elevating privileges, or altering the output of a cryptographic operation. Manipulation of the device clock and voltage supply is a well-known technique to inject faults and is cheap to implement with physical device access. Poorly protected power management features allow these attacks to be performed from software. Other features, such as the ability to write repeatedly to DRAM at a rapid rate from unprivileged software, can result in bit flips in other memory locations (Rowhammer, [REF-1083]).

Side channel analysis requires gathering measurement traces of physical quantities such as power consumption. Modern processors often include power metering capabilities in the hardware itself (e.g., Intel RAPL) which if not adequately protected enable attackers to gather measurements necessary for performing side-channel attacks from software.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		285	Improper Authorization	684

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Power Management Hardware (*Prevalence = Undetermined*)

Technology : Clock/Counter Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory Modify Application Data Bypass Protection Mechanism	

Detection Methods

Manual Analysis

Perform a security evaluation of system-level architecture and design with software-aided physical attacks in scope.

Automated Dynamic Analysis

Use custom software to change registers that control clock settings or power settings to try to bypass security locks, or repeatedly write DRAM to try to change adjacent locations. This can be effective in extracting or changing data. The drawback is that it cannot be run before manufacturing, and it may require specialized software.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Ensure proper access control mechanisms protect software-controllable features altering physical operating conditions such as clock frequency and voltage.

Demonstrative Examples

Example 1:

This example considers the Rowhammer problem [REF-1083]. The Rowhammer issue was caused by a program in a tight loop writing repeatedly to a location to which the program was allowed to write but causing an adjacent memory location value to change.

Example Language: Other

(Bad)

Continuously writing the same value to the same address causes the value of an adjacent location to change value.

Preventing the loop required to defeat the Rowhammer exploit is not always possible:

Example Language: Other

(Good)

Redesign the RAM devices to reduce inter capacitive coupling making the Rowhammer exploit impossible.

While the redesign may be possible for new devices, a redesign is not possible in existing devices. There is also the possibility that reducing capacitance with a relayout would impact the density of the device resulting in a less capable, more costly device.

Example 2:

Suppose a hardware design implements a set of software-accessible registers for scaling clock frequency and voltage but does not control access to these registers. Attackers may cause register and memory changes and race conditions by changing the clock or voltage of the device under their control.

Example 3:

Consider the following SoC design. Security-critical settings for scaling clock frequency and voltage are available in a range of registers bounded by [PRIV_END_ADDR : PRIV_START_ADDR] in the tmcu.csr module in the HW Root of Trust. These values are writable based on the lock_bit register in the same module. The lock_bit is only writable by privileged software running on the tmcu.

We assume that untrusted software running on any of the Core{0-N} processors has access to the input and output ports of the hrot_iface. If untrusted software can clear the lock_bit or write the clock frequency and voltage registers due to inadequate protection, a fault injection attack could be performed.

Observed Examples

Reference	Description
CVE-2019-11157	Plundervolt: Improper conditions check in voltage settings for some Intel(R) Processors may allow a privileged user to potentially enable escalation of privilege and/or information disclosure via local access [REF-1081]. https://www.cve.org/CVERecord?id=CVE-2019-11157
CVE-2020-8694	PLATYPUS Attack: Insufficient access control in the Linux kernel driver for some Intel processors allows information disclosure. https://www.cve.org/CVERecord?id=CVE-2020-8694




Reference	Description
CVE-2020-8695	Observable discrepancy in the RAPL interface for some Intel processors allows information disclosure. https://www.cve.org/CVERecord?id=CVE-2020-8695
CVE-2020-12912	AMD extension to a Linux service does not require privileged access to the RAPL interface, allowing side-channel attacks. https://www.cve.org/CVERecord?id=CVE-2020-12912
CVE-2015-0565	NaCl in 2015 allowed the CLFLUSH instruction, making Rowhammer attacks possible. https://www.cve.org/CVERecord?id=CVE-2015-0565

Functional Areas

- Power
- Clock

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
624	Hardware Fault Injection
625	Mobile Device Fault Injection

References

[REF-1081]Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Frank Piessens and Daniel Gruss. "Plundervolt". < <https://plundervolt.com/> >.

[REF-1082]Adrian Tang, Simha Sethumadhavan and Salvatore Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". < <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-tang.pdf> >.

[REF-1083]Yoongu Kim, Ross Daly, Jeremie Kim, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai and Onur Mutlu. "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors". < <https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf> >.

[REF-1225]Project Zero. "Exploiting the DRAM rowhammer bug to gain kernel privileges". 2015 March 9. < <https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html> >.

[REF-1217]Ross Anderson. "Security Engineering". 2001. < <https://www.cl.cam.ac.uk/~rja14/musicfiles/manuscripts/SEv1.pdf> >.

CWE-1257: Improper Access Control Applied to Mirrored or Aliased Memory Regions

Weakness ID : 1257

Structure : Simple

Abstraction : Base

Description

Aliased or mirrored memory regions in hardware designs may have inconsistent read/write permissions enforced by the hardware. A possible result is that an untrusted agent is blocked from accessing a memory region but is not blocked from accessing the corresponding aliased memory region.

Extended Description

Hardware product designs often need to implement memory protection features that enable privileged software to define isolated memory regions and access control (read/write) policies. Isolated memory regions can be defined on different memory spaces in a design (e.g. system physical address, virtual address, memory mapped IO).

Each memory cell should be mapped and assigned a system address that the core software can use to read/write to that memory. It is possible to map the same memory cell to multiple system addresses such that read/write to any of the aliased system addresses would be decoded to the same memory cell.

This is commonly done in hardware designs for redundancy and simplifying address decoding logic. If one of the memory regions is corrupted or faulty, then that hardware can switch to using the data in the mirrored memory region. Memory aliases can also be created in the system address map if the address decoder unit ignores higher order address bits when mapping a smaller address region into the full system address.

A common security weakness that can exist in such memory mapping is that aliased memory regions could have different read/write access protections enforced by the hardware such that an untrusted agent is blocked from accessing a memory address but is not blocked from accessing the corresponding aliased memory address. Such inconsistency can then be used to bypass the access protection of the primary memory block and read or modify the protected memory.

An untrusted agent could also possibly create memory aliases in the system address map for malicious purposes if it is able to change the mapping of an address region or modify memory region sizes.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680
CanPrecede		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Technology : Microcontroller Hardware (*Prevalence = Undetermined*)

Technology : Network on Chip Hardware (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	High
Integrity	Modify Memory	High
Availability	DoS: Instability	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

The checks should be applied for consistency access rights between primary memory regions and any mirrored or aliased memory regions. If different memory protection units (MPU) are protecting the aliased regions, their protected range definitions and policies should be synchronized.

Phase: Architecture and Design

Phase: Implementation

The controls that allow enabling memory aliases or changing the size of mapped memory regions should only be programmable by trusted software components.

Demonstrative Examples

Example 1:

In a System-on-a-Chip (SoC) design the system fabric uses 16 bit addresses. An IP unit (Unit_A) has 4 kilobyte of internal memory which is mapped into a 16 kilobyte address range in the system fabric address map.

To protect the register controls in Unit_A unprivileged software is blocked from accessing addresses between 0x0000 - 0x0FFF.

The address decoder of Unit_A masks off the higher order address bits and decodes only the lower 12 bits for computing the offset into the 4 kilobyte internal memory space.

Example Language: Other

(Bad)

In this design the aliased memory address ranges are these:

0x0000 - 0x0FFF

0x1000 - 0x1FFF

0x2000 - 0x2FFF

0x3000 - 0x3FFF

The same register can be accessed using four different addresses: 0x0000, 0x1000, 0x2000, 0x3000.

The system address filter only blocks access to range 0x0000 - 0x0FFF and does not block access to the aliased addresses in 0x1000 - 0x3FFF range. Thus, untrusted software can leverage the aliased memory addresses to bypass the memory protection.

Example Language: Other



(Good)

In this design the aliased memory addresses (0x1000 - 0x3FFF) could be blocked from all system software access since they are not used by software.

Alternately, the MPU logic can be changed to apply the memory protection policies to the full address range mapped to Unit_A (0x0000 - 0x3FFF).

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1202	Memory and Storage Issues	1194	2472
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
456	Infected Memory
679	Exploitation of Improperly Configured or Implemented Memory Protections

CWE-1258: Exposure of Sensitive System Information Due to Uncleared Debug Information

Weakness ID : 1258

Structure : Simple

Abstraction : Base

Description

The hardware does not fully clear security-sensitive values, such as keys and intermediate values in cryptographic operations, when debug mode is entered.



Extended Description

Security sensitive values, keys, intermediate steps of cryptographic operations, etc. are stored in temporary registers in the hardware. If these values are not cleared when debug mode is entered they may be accessed by a debugger allowing sensitive information to be accessible by untrusted parties.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		200	Exposure of Sensitive Information to an Unauthorized Actor	504
ChildOf		212	Improper Removal of Sensitive Information Before Storage or Transfer	544

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Whenever debug mode is enabled, all registers containing sensitive assets must be cleared.

Demonstrative Examples

Example 1:

A cryptographic core in a System-On-a-Chip (SoC) is used for cryptographic acceleration and implements several cryptographic operations (e.g., computation of AES encryption and decryption, SHA-256, HMAC, etc.). The keys for these operations or the intermediate values are stored in registers internal to the cryptographic core. These internal registers are in the Memory Mapped Input Output (MMIO) space and are blocked from access by software and other untrusted agents on the SoC. These registers are accessible through the debug and test interface.

Example Language: Other

(Bad)

In the above scenario, registers that store keys and intermediate values of cryptographic operations are not cleared when system enters debug mode. An untrusted actor running a debugger may read the contents of these registers and gain access to secret keys and other sensitive cryptographic information.

Example Language: Other

(Good)



Whenever the chip enters debug mode, all registers containing security-sensitive data are be cleared rendering them unreadable.

Observed Examples

Reference	Description
CVE-2021-33080	Uncleared debug information in memory accelerator for SSD product exposes sensitive system information https://www.cve.org/CVERecord?id=CVE-2021-33080
CVE-2022-31162	Rust library leaks Oauth client details in application debug logs https://www.cve.org/CVERecord?id=CVE-2022-31162

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data

CAPEC-ID Attack Pattern Name

150	Collect Data from Common Resource Locations
204	Lifting Sensitive Data Embedded in Cache
545	Pull Data from System Resources

CWE-1259: Improper Restriction of Security Token Assignment**Weakness ID :** 1259**Structure :** Simple**Abstraction :** Base**Description**

The System-On-A-Chip (SoC) implements a Security Token mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. However, the Security Tokens are improperly protected.

Extended Description

Systems-On-A-Chip (Integrated circuits and hardware engines) implement Security Tokens to differentiate and identify which actions originated from which agent. These actions may be one of the directives: 'read', 'write', 'program', 'reset', 'fetch', 'compute', etc. Security Tokens are assigned to every agent in the System that is capable of generating an action or receiving an action from another agent. Multiple Security Tokens may be assigned to an agent and may be unique based on the agent's trust level or allowed privileges. Since the Security Tokens are integral for the maintenance of security in an SoC, they need to be protected properly. A common weakness afflicting Security Tokens is improperly restricting the assignment to trusted components. Consequently, an improperly protected Security Token may be able to be programmed by a malicious agent (i.e., the Security Token is mutable) to spoof the action as if it originated from a trusted agent.



Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf		1294	Insecure Security Identifier Mechanism	2150
PeerOf		1255	Comparison Logic is Vulnerable to Power Side-Channel Attacks	2062

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Processor HardwareNot Technology-Specific (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Files or Directories	High
Integrity	Execute Unauthorized Code or Commands	
Availability	Bypass Protection Mechanism	
Access Control	Gain Privileges or Assume Identity	
	Modify Memory	
	Modify Memory	
	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Security Token assignment review checks for design inconsistency and common weaknesses. Security-Token definition and programming flow is tested in both pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

For example, consider a system with a register for storing an AES key for encryption and decryption. The key is of 128 bits implemented as a set of four 32-bit registers. The key register assets have an associated control register, AES_KEY_ACCESS_POLICY, which provides the necessary access controls. This access-policy register defines which agents may engage in a transaction, and the type of transaction, with the AES-key registers. Each bit in this 32-bit register defines a security Token. There could be a maximum of 32 security Tokens that are allowed access to the AES-key registers. The number of the bit when set (i.e., "1") allows respective action from an agent whose identity matches the number of the bit and, if "0" (i.e., Clear), disallows the respective action to that corresponding agent.

Let's assume the system has two agents: a Main-controller and an Aux-controller. The respective Security Tokens are "1" and "2".

An agent with Security Token "1" has access to AES_ENC_DEC_KEY_0 through AES_ENC_DEC_KEY_3 registers. As per the above access policy, the AES-Key-access policy allows access to the AES-key registers if the security Token is "1".

Example Language: Other

(Bad)

The Aux-controller could program its Security Token to "1" from "2".

The SoC does not properly protect the Security Token of the agents, and, hence, the Aux-controller in the above example can spoof the transaction (i.e., send the transaction as if it is coming from the Main-controller to access the AES-Key registers)

Example Language: Other

(Good)

The SoC needs to protect the Security Tokens. None of the agents in the SoC should have the ability to change the Security Token.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes

2074

Maintenance

This entry is still under development and will continue to see updates and content improvements. Currently it is expressed as a general absence of a protection mechanism as opposed to a specific mistake, and the entry's name and description could be interpreted as applying to software.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces
681	Exploitation of Improperly Controlled Hardware Security Identifiers

CWE-1260: Improper Handling of Overlap Between Protected Memory Ranges

Weakness ID : 1260

Structure : Simple

Abstraction : Base

Description

The product allows address regions to overlap, which can result in the bypassing of intended memory protection.

Extended Description

Isolated memory regions and access control (read/write) policies are used by hardware to protect privileged software. Software components are often allowed to change or remap memory region definitions in order to enable flexible and dynamically changeable memory management by system software.

If a software component running at lower privilege can program a memory address region to overlap with other memory regions used by software running at higher privilege, privilege escalation may be available to attackers. The memory protection unit (MPU) logic can incorrectly handle such an address overlap and allow the lower-privilege software to read or write into the protected memory region, resulting in privilege escalation attack. An address overlap weakness can also be used to launch a denial of service attack on the higher-privilege software memory regions.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680
CanPrecede	🟢	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	293

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Instability	

Detection Methods

Manual Analysis

Create a high privilege memory block of any arbitrary size. Attempt to create a lower privilege memory block with an overlap of the high privilege memory block. If the creation attempt works, fix the hardware. Repeat the test.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Ensure that memory regions are isolated as intended and that access control (read/write) policies are used by hardware to protect privileged software.

Phase: Implementation

For all of the programmable memory protection regions, the memory protection unit (MPU) design can define a priority scheme. For example: if three memory regions can be programmed (Region_0, Region_1, and Region_2), the design can enforce a priority scheme, such that, if a system address is within multiple regions, then the region with the lowest ID takes priority and the access-control policy of that region will be applied. In some MPU designs, the priority scheme can also be programmed by trusted software. Hardware logic or trusted firmware can also check for region definitions and block programming of memory regions with overlapping addresses. The memory-access-control-check filter can also be designed to apply a policy filter to all of the overlapping ranges, i.e., if an address is within Region_0 and Region_1, then access to this address is only granted if both Region_0 and Region_1 policies allow the access.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider a design with a 16-bit address that has two software privilege levels: Privileged_SW and Non_privileged_SW. To isolate the system memory regions accessible by these two privilege levels, the design supports three memory regions: Region_0, Region_1, and Region_2.

Each region is defined by two 32 bit registers: its range and its access policy.

- Address_range[15:0]: specifies the Base address of the region
- Address_range[31:16]: specifies the size of the region
- Access_policy[31:0]: specifies what types of software can access a region and which actions are allowed

Certain bits of the access policy are defined symbolically as follows:

- Access_policy.read_np: if set to one, allows reads from Non_privileged_SW
- Access_policy.write_np: if set to one, allows writes from Non_privileged_SW
- Access_policy.execute_np: if set to one, allows code execution by Non_privileged_SW
- Access_policy.read_p: if set to one, allows reads from Privileged_SW
- Access_policy.write_p: if set to one, allows writes from Privileged_SW
- Access_policy.execute_p: if set to one, allows code execution by Privileged_SW

For any requests from software, an address-protection filter checks the address range and access policies for each of the three regions, and only allows software access if all three filters allow access.

Consider the following goals for access control as intended by the designer:

- Region_0 & Region_1: registers are programmable by Privileged_SW
- Region_2: registers are programmable by Non_privileged_SW

The intention is that Non_privileged_SW cannot modify memory region and policies defined by Privileged_SW in Region_0 and Region_1. Thus, it cannot read or write the memory regions that Privileged_SW is using.

Example Language:

(Bad)

Non_privileged_SW can program the Address_range register for Region_2 so that its address overlaps with the ranges defined by Region_0 or Region_1. Using this capability, it is possible for Non_privileged_SW to block any memory region from being accessed by Privileged_SW, i.e., Region_0 and Region_1.

This design could be improved in several ways.

Example Language:

(Good)

Ensure that software accesses to memory regions are only permitted if all three filters permit access. Additionally, the scheme could define a memory region priority to ensure that Region_2 (the memory region defined by Non_privileged_SW) cannot overlap Region_0 or Region_1 (which are used by Privileged_SW).

Example 2:

The example code below is taken from the IOMMU controller module of the HACK@DAC'19 buggy CVA6 SoC [REF-1338]. The static memory map is composed of a set of Memory-Mapped Input/Output (MMIO) regions covering different IP agents within the SoC. Each region is defined by two 64-bit variables representing the base address and size of the memory region (XXXBase and XXXLength).

In this example, we have 12 IP agents, and only 4 of them are called out for illustration purposes in the code snippets. Access to the AES IP MMIO region is considered privileged as it provides access to AES secret key, internal states, or decrypted data.

Example Language: Verilog

(Bad)

```
...
localparam logic[63:0] PLICLength = 64'h03FF_FFFF;
localparam logic[63:0] UARTLength = 64'h0011_1000;
localparam logic[63:0] AESLength = 64'h0000_1000;
localparam logic[63:0] SPILength = 64'h0080_0000;
...
typedef enum logic [63:0] {
    ...
    PLICBase = 64'h0C00_0000,
    UARTBase = 64'h1000_0000,
    AESBase = 64'h1010_0000,
    SPIBase = 64'h2000_0000,
    ...
}
```

The vulnerable code allows the overlap between the protected MMIO region of the AES peripheral and the unprotected UART MMIO region. As a result, unprivileged users can access the protected region of the AES IP. In the given vulnerable example UART MMIO region starts at address 64'h1000_0000 and ends at address 64'h1011_1000 (UARTBase is 64'h1000_0000, and the size of the region is provided by the UARTLength of 64'h0011_1000).

On the other hand, the AES MMIO region starts at address 64'h1010_0000 and ends at address 64'h1010_1000, which implies an overlap between the two peripherals' memory regions. Thus, any user with access to the UART can read or write the AES MMIO region, e.g., the AES secret key.

To mitigate this issue, remove the overlapping address regions by decreasing the size of the UART memory region or adjusting memory bases for all the remaining peripherals. [REF-1339]

Example Language: Verilog

(Good)







```
...
localparam logic[63:0] PLICLength = 64'h03FF_FFFF;
localparam logic[63:0] UARTLength = 64'h0000_1000;
localparam logic[63:0] AESLength = 64'h0000_1000;
localparam logic[63:0] SPILength = 64'h0080_0000;
...
typedef enum logic [63:0] {
    ...
    PLICBase = 64'h0C00_0000,
    UARTBase = 64'h1000_0000,
    AESBase = 64'h1010_0000,
    SPIBase = 64'h2000_0000,
    ...
}
```

Observed Examples

Reference	Description
CVE-2008-7096	virtualization product allows compromise of hardware product by accessing certain remapping registers. https://www.cve.org/CVERecord?id=CVE-2008-7096
[REF-1100]	processor design flaw allows ring 0 code to access more privileged rings by causing a register window to overlap a range of protected system RAM [REF-1100] https://github.com/xoreaxeaxeax/sinkhole/blob/master/us-15-Domas-TheMemorySinkhole-wp.pdf

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues		2470
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List		2592
MemberOf		1396	Comprehensive Categorization: Access Control		2519

Notes

Maintenance

As of CWE 4.6, CWE-1260 and CWE-1316 are siblings under view 1000, but CWE-1260 might be a parent of CWE-1316. More analysis is warranted.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
456	Infected Memory
679	Exploitation of Improperly Configured or Implemented Memory Protections

References

[REF-1100]Christopher Domas. "The Memory Sinkhole". 2015 July 0. < <https://github.com/xoreaxeaxeax/sinkhole/blob/master/us-15-Domas-TheMemorySinkhole-wp.pdf> >.

[REF-1338]"Hackatdac19 ariane_soc_pkg.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/619e9fb0ef32ee1e01ad76b8732a156572c65700/tb/ariane_soc_pkg.sv#L44:L62 >.2023-06-21.

[REF-1339]Florian Zaruba, Michael Schaffner and Andreas Traber. "csr_regfile.sv". 2019. < https://github.com/openhwgroup/cva6/blob/7951802a0147aedb21e8f2f6dc1e1e9c4ee857a2/src/csr_regfile.sv#L45 >.2023-06-21.

CWE-1261: Improper Handling of Single Event Upsets

Weakness ID : 1261

Structure : Simple

Abstraction : Base

Description

The hardware logic does not effectively handle when single-event upsets (SEUs) occur.



Extended Description

Technology trends such as CMOS-transistor down-sizing, use of new materials, and system-on-chip architectures continue to increase the sensitivity of systems to soft errors. These errors are random, and their causes might be internal (e.g., interconnect coupling) or external (e.g., cosmic radiation). These soft errors are not permanent in nature and cause temporary bit flips known as single-event upsets (SEUs). SEUs are induced errors in circuits caused when charged particles lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs that cause temporary failures. If these failures occur in security-sensitive modules in a chip, it might compromise the security guarantees of the chip. For instance, these temporary failures could be bit flips that change the privilege of a regular user to root.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		1384	Improper Handling of Physical or Environmental Conditions	2257
PeerOf		1254	Incorrect Comparison Logic Granularity	2060

Applicable Platforms

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

Operating System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Access Control	DoS: Instability Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Implement triple-modular redundancy around security-sensitive modules.

Phase: Architecture and Design

SEUs mostly affect SRAMs. For SRAMs storing security-critical data, implement Error-Correcting-Codes (ECC) and Address Interleaving.

Demonstrative Examples

Example 1:

This is an example from [REF-1089]. See the reference for full details of this issue.

Parity is error detecting but not error correcting.

Example Language: Other

(Bad)

Due to single-event upsets, bits are flipped in memories. As a result, memory-parity checks fail, which results in restart and a temporary denial of service of two to three minutes.

Example Language: Other

(Good)





Using error-correcting codes could have avoided the restart caused by SEUs.

Example 2:

In 2016, a security researcher, who was also a patient using a pacemaker, was on an airplane when a bit flip occurred in the pacemaker, likely due to the higher prevalence of cosmic radiation at such heights. The pacemaker was designed to account for bit flips and went into a default safe mode, which still forced the patient to go to a hospital to get it reset. The bit flip also inadvertently enabled the researcher to access the crash file, perform reverse engineering, and detect a hard-coded key. [REF-1101]

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf		1365	ICS Communications: Unreliability	1358	2502
MemberOf		1388	Physical Access Issues and Concerns	1194	2518
MemberOf		1405	Comprehensive Categorization: Improper Check or Handling of Exceptional Conditions	1400	2531

References

[REF-1086]Fan Wang and Vishwani D. Agrawal. "Single Event Upset: An Embedded Tutorial". < https://www.eng.auburn.edu/~agrawvd/TALKS/tutorial_6pg.pdf >.

[REF-1087]P. D. Bradley and E. Normand. "Single Event Upsets in Implantable Cardioverter Defibrillators". < <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=736549&tag=1> >.2023-04-07.

[REF-1088]Melanie Berg, Kenneth LaBel and Jonathan Pellish. "Single Event Effects in FPGA Devices 2015-2016". < <https://ntrs.nasa.gov/search.jsp?R=20160007754> >.

[REF-1089]Cisco. "Cisco 12000 Single Event Upset Failures Overview and Work Around Summary". < <https://www.cisco.com/c/en/us/support/docs/field-notices/200/fn25994.html> >.

[REF-1090]Cypress. "Different Ways to Mitigate Soft Errors in Asynchronous SRAMs - KBA90939". < <https://community.infineon.com/t5/Knowledge-Base-Articles/Different-Ways-to-Mitigate-Soft-Errors-in-Asynchronous-SRAMs-KBA90939/ta-p/257944> >.2023-04-07.

[REF-1091]Ian Johnston. "Cosmic particles can change elections and cause plans to fall through the sky, scientists warn". < <https://www.independent.co.uk/news/science/subatomic-particles-cosmic-rays-computers-change-elections-planes-autopilot-a7584616.html> >.

[REF-1101]Anders B. Wilhelmsen, Eivind S. Kristiansen and Marie Moe. "The Hard-coded Key to my Heart - Hacking a Pacemaker Programmer". 2019 August 0. < <https://anderbw.github.io/2019-08-10-DC27-Biohacking-pacemaker-programmer.pdf> >.

CWE-1262: Improper Access Control for Register Interface

Weakness ID : 1262

Structure : Simple

Abstraction : Base

Description

The product uses memory-mapped I/O registers that act as an interface to hardware functionality from software, but there is improper access control to those registers.

Extended Description

Software commonly accesses peripherals in a System-on-Chip (SoC) or other device through a memory-mapped register interface. Malicious software could tamper with any security-critical hardware data that is accessible directly or indirectly through the register interface, which could lead to a loss of confidentiality and integrity.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Memory Read Application Data Modify Memory Modify Application Data Gain Privileges or Assume Identity Bypass Protection Mechanism Unexpected State Alter Execution Logic <i>Confidentiality of hardware assets may be violated if the protected information can be read out by software through the register interface. Registers storing security state, settings, other security-critical data may be corruptible by software without correctly implemented protections.</i>	

Detection Methods

Manual Analysis

This is applicable in the Architecture phase before implementation started. Make sure access policy is specified for the entire memory map. Manual analysis may not ensure the implementation is correct.

Effectiveness = Moderate

Manual Analysis

Registers controlling hardware should have access control implemented. This access control may be checked manually for correct implementation. Items to check consist of how are trusted parties set, how are trusted parties verified, how are accesses verified, etc. Effectiveness of a manual analysis will vary depending upon how complicated the interface is constructed.

Effectiveness = Moderate

Simulation / Emulation

Functional simulation is applicable during the Implementation Phase. Testcases must be created and executed for memory mapped registers to verify adherence to the access control policy. This method can be effective, since functional verification needs to be performed on the design, and verification for this weakness will be included. There can be difficulty covering the entire memory space during the test.

Effectiveness = Moderate

Formal Verification

Formal verification is applicable during the Implementation phase. Assertions need to be created in order to capture illegal register access scenarios and prove that they cannot occur. Formal methods are exhaustive and can be very effective, but creating the cases for large designs may be complex and difficult.

Effectiveness = High

Automated Analysis

Information flow tracking can be applicable during the Implementation phase. Security sensitive data (assets) - for example, as stored in registers - is automatically tracked over time through the design to verify the data doesn't reach illegal destinations that violate the access policies for the memory map. This method can be very effective when used together with simulation and emulation, since detecting violations doesn't rely on specific scenarios or data values. This method does rely on simulation and emulation, so testcases must exist in order to use this method.

Effectiveness = High

Architecture or Design Review

Manual documentation review of the system memory map, register specification, and permissions associated with accessing security-relevant functionality exposed via memory-mapped registers.

Effectiveness = Moderate

Fuzzing

Perform penetration testing (either manual or semi-automated with fuzzing) to verify that access control mechanisms such as the memory protection units or on-chip bus firewall settings adequately protect critical hardware registers from software access.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Design proper policies for hardware register access from software.

Phase: Implementation

Ensure that access control policies for register access are implemented in accordance with the specified design.

Demonstrative Examples

Example 1:

The register interface provides software access to hardware functionality. This functionality is an attack surface. This attack surface may be used to run untrusted code on the system through the register interface. As an example, cryptographic accelerators require a mechanism for software to select modes of operation and to provide plaintext or ciphertext data to be encrypted or decrypted as well as other functions. This functionality is commonly provided through registers.

Example Language:

(Bad)

Cryptographic key material stored in registers inside the cryptographic accelerator can be accessed by software.

Example Language:

(Good)

Key material stored in registers should never be accessible to software. Even if software can provide a key, all read-back paths to software should be disabled.

Example 2:

The example code is taken from the Control/Status Register (CSR) module inside the processor core of the HACK@DAC'19 buggy CVA6 SoC [REF-1340]. In RISC-V ISA [REF-1341], the CSR file contains different sets of registers with different privilege levels, e.g., user mode (U), supervisor mode (S), hypervisor mode (H), machine mode (M), and debug mode (D), with different read-write policies, read-only (RO) and read-write (RW). For example, machine mode, which is the highest privilege mode in a RISC-V system, registers should not be accessible in user, supervisor, or hypervisor modes.

Example Language: Verilog

(Bad)

```
if (csr_we || csr_read) begin
  if ((riscv::priv_lvl_t'(priv_lvl_o & csr_addr.csr_decode.priv_lvl) != csr_addr.csr_decode.priv_lvl) && !
    (csr_addr.address==riscv::CSR_MEPC)) begin
    csr_exception_o.cause = riscv::ILLEGAL_INSTR;
    csr_exception_o.valid = 1'b1;
  end
  // check access to debug mode only CSRs
  if (csr_addr_i[11:4] == 8'h7b && !debug_mode_q) begin
    csr_exception_o.cause = riscv::ILLEGAL_INSTR;
  end
end
```

```
csr_exception_o.valid = 1'b1;  
end  
end
```

The vulnerable example code allows the machine exception program counter (MEPC) register to be accessed from a user mode program by excluding the MEPC from the access control check. MEPC as per the RISC-V specification can be only written or read by machine mode code. Thus, the attacker in the user mode can run code in machine mode privilege (privilege escalation).

To mitigate the issue, fix the privilege check so that it throws an Illegal Instruction Exception for user mode accesses to the MEPC register. [REF-1345]

Example Language: Verilog

(Good)



```
if (csr_we || csr_read) begin  
  if ((riscv::priv_lvl_t'(priv_lvl_o & csr_addr.csr_decode.priv_lvl) != csr_addr.csr_decode.priv_lvl)) begin  
    csr_exception_o.cause = riscv::ILLEGAL_INSTR;  
    csr_exception_o.valid = 1'b1;  
  end  
  // check access to debug mode only CSRs  
  if (csr_addr_i[11:4] == 8'h7b && !debug_mode_q) begin  
    csr_exception_o.cause = riscv::ILLEGAL_INSTR;  
    csr_exception_o.valid = 1'b1;  
  end  
end  
end
```

Observed Examples

Reference	Description
CVE-2014-2915	virtualization product does not restrict access to debug and other processor registers in the hardware, allowing a crash of the host or guest OS https://www.cve.org/CVERecord?id=CVE-2014-2915
CVE-2021-3011	virtual interrupt controller in a virtualization product allows crash of host by writing a certain invalid value to a register, which triggers a fatal error instead of returning an error code https://www.cve.org/CVERecord?id=CVE-2021-3011
CVE-2020-12446	Driver exposes access to Model Specific Register (MSR) registers, allowing admin privileges. https://www.cve.org/CVERecord?id=CVE-2020-12446
CVE-2015-2150	Virtualization product does not restrict access to PCI command registers, allowing host crash from the guest. https://www.cve.org/CVERecord?id=CVE-2015-2150

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
680	Exploitation of Improperly Controlled Registers

References

[REF-1340]"Hackatdac19 csr_regfile.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/619e9fb0ef32ee1e01ad76b8732a156572c65700/src/csr_regfile.sv#L854:L857 >.2023-06-21.

[REF-1341]Andrew Waterman, Yunsup Lee, Rimas Avižienis, David Patterson and Krste Asanovi#. "The RISC-V Instruction Set Manual". Volume II: Privileged Architecture. 2016 November 4. <
<https://people.eecs.berkeley.edu/~krste/papers/riscv-privileged-v1.9.1.pdf> >.2023-06-21.

[REF-1345]Florian Zaruba, Michael Schaffner and Andreas Traber. "csr_regfile.sv". 2019. <
https://github.com/openhwgroup/cva6/blob/7951802a0147aedb21e8f2f6dc1e1e9c4ee857a2/src/csr_regfile.sv#L868:L871 >.2023-06-21.

CWE-1263: Improper Physical Access Control

Weakness ID : 1263

Structure : Simple

Abstraction : Class

Description

The product is designed with access restricted to certain information, but it does not sufficiently protect against an unauthorized actor with physical access to these areas.

Extended Description

Sections of a product intended to have restricted access may be inadvertently or intentionally rendered accessible when the implemented physical protections are insufficient. The specific requirements around how robust the design of the physical protection mechanism needs to be depends on the type of product being protected. Selecting the correct physical protection mechanism and properly enforcing it through implementation and manufacturing are critical to the overall physical security of the product.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680
ParentOf	ⓑ	1243	Sensitive Non-Volatile Information Not Protected During Debug	2035
PeerOf	ⓑ	1191	On-Chip Debug and Test Interface With Improper Access Control	1980

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Specific protection requirements depend strongly on contextual factors including the level of acceptable risk associated with compromise to the product's protection mechanism. Designers could incorporate anti-tampering measures that protect against or detect when the product has been tampered with.

Phase: Testing

The testing phase of the lifecycle should establish a method for determining whether the protection mechanism is sufficient to prevent unauthorized access.

Phase: Manufacturing

Ensure that all protection mechanisms are fully activated at the time of manufacturing and distribution.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2474
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
401	Physically Hacking Hardware

CWE-1264: Hardware Logic with Insecure De-Synchronization between Control and Data Channels

Weakness ID : 1264

Structure : Simple

Abstraction : Base

Description

The hardware logic for error handling and security checks can incorrectly forward data before the security check is complete.

Extended Description

Many high-performance on-chip bus protocols and processor data-paths employ separate channels for control and data to increase parallelism and maximize throughput. Bugs in the hardware logic that handle errors and security checks can make it possible for data to be forwarded before the completion of the security checks. If the data can propagate to a location in the hardware observable to an attacker, loss of data confidentiality can occur. 'Meltdown' is a concrete example of how de-synchronization between data and permissions checking logic can violate confidentiality requirements. Data loaded from a page marked as privileged was returned to the cpu regardless of current privilege level for performance reasons. The assumption was that the cpu could later remove all traces of this data during the handling of the illegal memory access exception, but this assumption was proven false as traces of the secret data were not removed from the microarchitectural state.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	821	Incorrect Synchronization	1722
PeerOf	B	1037	Processor Optimization Removal or Modification of Security-critical Code	1870

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Thoroughly verify the data routing logic to ensure that any error handling or security checks effectively block illegal dataflows.

Demonstrative Examples

Example 1:

There are several standard on-chip bus protocols used in modern SoCs to allow communication between components. There are a wide variety of commercially available hardware IP implementing the interconnect logic for these protocols. A bus connects components which initiate/request communications such as processors and DMA controllers (bus masters) with peripherals which respond to requests. In a typical system, the privilege level or security designation of the bus master along with the intended functionality of each peripheral determine the security policy specifying which specific bus masters can access specific peripherals. This security policy (commonly referred to as a bus firewall) can be enforced using separate IP/logic from the actual interconnect responsible for the data routing.

Example Language: Other

(Bad)

The firewall and data routing logic becomes de-synchronized due to a hardware logic bug allowing components that should not be allowed to communicate to share data. For example, consider an SoC with two processors. One is being used as a root of trust and can access a cryptographic key storage peripheral. The other processor (application cpu) may run potentially untrusted code and should not access the key store. If the application cpu can issue a read request to the key store which is not blocked due to de-synchronization of data routing and the bus firewall, disclosure of cryptographic keys is possible.

Example Language: Other

(Good)


All data is correctly buffered inside the interconnect until the firewall has determined that the endpoint is allowed to receive the data.

Observed Examples

Reference	Description
CVE-2017-5754	Systems with microprocessors utilizing speculative execution and indirect branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis of the data cache. https://www.cve.org/CVERecord?id=CVE-2017-5754

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1196	Security Flow Issues	1194	2469
MemberOf		1401	Comprehensive Categorization: Concurrency	1400	2526

Notes

Maintenance

As of CWE 4.9, members of the CWE Hardware SIG are closely analyzing this entry and others to improve CWE's coverage of transient execution weaknesses, which include issues related to Spectre, Meltdown, and other attacks. Additional investigation may include other weaknesses related to microarchitectural state. As a result, this entry might change significantly in CWE 4.10.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
233	Privilege Escalation
663	Exploitation of Transient Instruction Execution

CWE-1265: Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls

Weakness ID : 1265

Structure : Simple

Abstraction : Base

Description

During execution of non-reentrant code, the product performs a call that unintentionally produces a nested invocation of the non-reentrant code.




Extended Description

In a complex product, a single function call may lead to many different possible code paths, some of which may involve deeply nested calls. It may be difficult to foresee all possible code paths that could emanate from a given function call. In some systems, an external actor can manipulate inputs to the system and thereby achieve a wide range of possible control flows. This is frequently a concern in products that execute scripts from untrusted sources. Examples of such products are web browsers and PDF readers. A weakness is present when one of the possible code paths resulting from a function call alters program state that the original caller assumes to be unchanged during the call.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		691	Insufficient Control Flow Management	1517
PeerOf		663	Use of a Non-reentrant Function in a Concurrent Context	1452
CanPrecede		416	Use After Free	1012

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

Nature	Type	ID	Name	Page
MemberOf		371	State Issues	2321

Weakness Ordinalities

Primary :

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State <i>Exploitation of this weakness can leave the application in an unexpected state and cause variables to be reassigned before the first invocation has completed. This may eventually result in memory corruption or unexpected code execution.</i>	Unknown

Potential Mitigations

Phase: Architecture and Design

When architecting a system that will execute untrusted code in response to events, consider executing the untrusted event handlers asynchronously (asynchronous message passing) as opposed to executing them synchronously at the time each event fires. The untrusted code should execute at the start of the next iteration of the thread's message loop. In this way, calls into non-reentrant code are strictly serialized, so that each operation completes fully before the next operation begins. Special attention must be paid to all places where type coercion may result in script execution. Performing all needed coercions at the very beginning of an operation can help reduce the chance of operations executing at unexpected junctures.

Effectiveness = High

Phase: Implementation

Make sure the code (e.g., function or class) in question is reentrant by not leveraging non-local data, not modifying its own code, and not calling other non-reentrant code.

Effectiveness = High

Demonstrative Examples

Example 1:

The implementation of the Widget class in the following C++ code is an example of code that is not designed to be reentrant. If an invocation of a method of Widget inadvertently produces a second nested invocation of a method of Widget, then data member backgroundImage may unexpectedly change during execution of the outer call.

Example Language: C++

(Bad)

```

class Widget
{
private:
    Image* backgroundImage;
public:
    void click()
    {
        if (backgroundImage)
        {
            backgroundImage->click();
        }
    }
    void changeBackgroundImage(Image* newImage)
    {
        if (backgroundImage)
        {
            delete backgroundImage;
        }
        backgroundImage = newImage;
    }
}
class Image
{
public:
    void click()
    {
        scriptEngine->fireOnImageClick();
        /* perform some operations using "this" pointer */
    }
}

```

Looking closer at this example, `Widget::click()` calls `backgroundImage->click()`, which in turn calls `scriptEngine->fireOnImageClick()`. The code within `fireOnImageClick()` invokes the appropriate script handler routine as defined by the document being rendered. In this scenario this script routine is supplied by an adversary and this malicious script makes a call to `Widget::changeBackgroundImage()`, deleting the `Image` object pointed to by `backgroundImage`. When control returns to `Image::click`, the function's `backgroundImage` "this" pointer (which is the former value of `backgroundImage`) is a dangling pointer. The root of this weakness is that while one operation on `Widget` (`click`) is in the midst of executing, a second operation on the `Widget` object may be invoked (in this case, the second invocation is a call to different method, namely `changeBackgroundImage`) that modifies the non-local variable.

Example 2:

This is another example of C++ code that is not designed to be reentrant.

Example Language: C++

(Bad)

```

class Request
{
private:
    std::string uri;
    /* ... */
public:
    void setup(ScriptObject* _uri)
    {
        this->uri = scriptEngine->coerceToString(_uri);
        /* ... */
    }
    void send(ScriptObject* _data)
    {
        Credentials credentials = GetCredentials(uri);
        std::string data = scriptEngine->coerceToString(_data);
        doSend(uri, credentials, data);
    }
}

```

```

    }
}

```


The expected order of operations is a call to Request::setup(), followed by a call to Request::send(). Request::send() calls scriptEngine->coerceToString(_data) to coerce a script-provided parameter into a string. This operation may produce script execution. For example, if the script language is ECMAScript, arbitrary script execution may result if _data is an adversary-supplied ECMAScript object having a custom toString method. If the adversary's script makes a new call to Request::setup, then when control returns to Request::send, the field uri and the local variable credentials will no longer be consistent with one another. As a result, credentials for one resource will be shared improperly with a different resource. The root of this weakness is that while one operation on Request (send) is in the midst of executing, a second operation may be invoked (setup).

Observed Examples

Reference	Description
CVE-2014-1772	In this vulnerability, by registering a malicious onerror handler, an adversary can produce unexpected re-entrance of a CDOMRange object. [REF-1098] https://www.cve.org/CVERecord?id=CVE-2014-1772
CVE-2018-8174	This CVE covers several vulnerable scenarios enabled by abuse of the Class_Terminate feature in Microsoft VBScript. In one scenario, Class_Terminate is used to produce an undesirable re-entrance of ScriptingDictionary during execution of that object's destructor. In another scenario, a vulnerable condition results from a recursive entrance of a property setter method. This recursive invocation produces a second, spurious call to the Release method of a reference-counted object, causing a UAF when that object is freed prematurely. This vulnerability pattern has been popularized as "Double Kill". [REF-1099] https://www.cve.org/CVERecord?id=CVE-2018-8174

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	<input checked="" type="checkbox"/>	1400 2536

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State

References

[REF-1098]Jack Tang. "Root Cause Analysis of CVE-2014-1772 - An Internet Explorer Use After Free Vulnerability". 2014 November 5. < https://www.trendmicro.com/en_us/research.html >.2023-04-07.

[REF-1099]Simon Zuckerbraun. "It's Time To Terminate The Terminator". 2018 May 5. < <https://www.zerodayinitiative.com/blog/2018/5/15/its-time-to-terminate-the-terminator> >.

CWE-1266: Improper Scrubbing of Sensitive Data from Decommissioned Device

Weakness ID : 1266
Structure : Simple

Abstraction : Base**Description**

The product does not properly provide a capability for the product administrator to remove sensitive data at the time the product is decommissioned. A scrubbing capability could be missing, insufficient, or incorrect.

Extended Description

When a product is decommissioned - i.e., taken out of service - best practices or regulatory requirements may require the administrator to remove or overwrite sensitive data first, i.e. "scrubbing." Improper scrubbing of sensitive data from a decommissioned device leaves that data vulnerable to acquisition by a malicious actor. Sensitive data may include, but is not limited to, device/manufacturer proprietary information, user/device credentials, network configurations, and other forms of sensitive 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	980

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Potential Mitigations**Phase: Architecture and Design**

Functionality to completely scrub data from a product at the conclusion of its lifecycle should be part of the design phase. Trying to add this function on top of an existing architecture could lead to incomplete removal of sensitive information/data.

Phase: Policy



The manufacturer should describe the location(s) where sensitive data is stored and the policies and procedures for its removal. This information may be conveyed, for example, in an Administrators Guide or a Statement of Volatility.

Phase: Implementation

If the capability to wipe sensitive data isn't built-in, the manufacturer may need to provide a utility to scrub sensitive data from storage if that data is located in a place which is non-accessible by the administrator. One example of this could be when sensitive data is stored on an EEPROM for which there is no user/admin interface provided by the system.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1195	Manufacturing and Life Cycle Management Concerns	1194	2469
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data
150	Collect Data from Common Resource Locations
545	Pull Data from System Resources
546	Incomplete Data Deletion in a Multi-Tenant Environment
675	Retrieve Data from Decommissioned Devices

References

[REF-1080]Christopher Tarnovsky. "Security Failures in Secure Devices". < <https://www.blackhat.com/presentations/bh-europe-08/Tarnovsky/Presentation/bh-eu-08-tarnovsky.pdf> >.

CWE-1267: Policy Uses Obsolete Encoding

Weakness ID : 1267
Structure : Simple
Abstraction : Base

Description

The product uses an obsolete encoding mechanism to implement access controls.

Extended Description

Within a System-On-a-Chip (SoC), various circuits and hardware engines generate transactions for the purpose of accessing (read/write) assets or performing various actions (e.g., reset, fetch, compute, etc.). Among various types of message information, a typical transaction is comprised of source identity (identifying the originator of the transaction) and a destination identity (routing the transaction to the respective entity). Sometimes the transactions are qualified with a Security Token. This Security Token helps the destination agent decide on the set of allowed actions (e.g., access to an asset for reads and writes). A policy encoder is used to map the bus transactions to Security Tokens that in turn are used as access-controls/protection mechanisms. A common weakness involves using an encoding which is no longer trusted, i.e., an obsolete encoding.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	Modify Files or Directories	
Access Control	Read Files or Directories	
	DoS: Resource Consumption (Other)	
	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	
	Reduce Reliability	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Security Token Decoders should be reviewed for design inconsistency and common weaknesses. Access and programming flows should be tested in both pre-silicon and post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider a system that has four bus masters. The table below provides bus masters, their Security Tokens, and trust assumptions.

The policy encoding is to be defined such that Security Token will be used in implemented access-controls. The bits in the bus transaction that contain Security-Token information are Bus_transaction [15:11]. The assets are the AES-Key registers for encryption or decryption. The key of 128 bits is implemented as a set of four, 32-bit registers.

Below is an example of a policy encoding scheme inherited from a previous project where all "ODD" numbered Security Tokens are trusted.

Example Language:

(Bad)

```
If (Bus_transaction[14] == "1")
  Trusted = "1"
Else
  Trusted = "0"
If (trusted)
  Allow access to AES-Key registers
Else
  Deny access to AES-Key registers
```

The inherited policy encoding is obsolete and does not work for the new system where an untrusted bus master with an odd Security Token exists in the system, i.e., Master_3 whose Security Token is "11". Based on the old policy, the untrusted bus master (Master_3) has access to

the AES-Key registers. To resolve this, a register AES_KEY_ACCESS_POLICY can be defined to provide necessary, access controls:

New Policy:

The AES_KEY_ACCESS_POLICY register defines which agents with a Security Token in the transaction can access the AES-key registers. Each bit in this 32-bit register defines a Security Token. There could be a maximum of 32 security Tokens that are allowed access to the AES-key registers. The number of the bit when set (i.e., "1") allows respective action from an agent whose identity matches the number of the bit and, if "0" (i.e., Clear), disallows the respective action to that corresponding agent. Thus, any bus master with Security Token "01" is allowed access to the AES-Key registers. Below is the Pseudo Code for policy encoding:



Example Language:

(Good)

```
Security_Token[4:0] = Bus_transaction[15:11]
If (AES_KEY_ACCESS_POLICY[Security_Token] == "1")
    Allow access to AES-Key registers
Else
    Deny access to AES-Key registers
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces
681	Exploitation of Improperly Controlled Hardware Security Identifiers

References

[REF-1093]Brandon Hill. "Huge Intel CPU Bug Allegedly Causes Kernel Memory Vulnerability With Up To 30% Performance Hit In Windows And Linux". 2018 January 2. < <https://hothardware.com/news/intel-cpu-bug-kernel-memory-isolation-linux-windows-macos> >.2023-04-07.

CWE-1268: Policy Privileges are not Assigned Consistently Between Control and Data Agents

Weakness ID : 1268

Structure : Simple

Abstraction : Base

Description

The product's hardware-enforced access control for a particular resource improperly accounts for privilege discrepancies between control and write policies.

Extended Description

Integrated circuits and hardware engines may provide access to resources (device-configuration, encryption keys, etc.) belonging to trusted firmware or software modules (commonly set by a BIOS or a bootloader). These accesses are typically controlled and limited by the hardware. Hardware design access control is sometimes implemented using a policy. A policy defines which entity or

agent may or may not be allowed to perform an action. When a system implements multiple levels of policies, a control policy may allow direct access to a resource as well as changes to the policies themselves.

Resources that include agents in their control policy but not in their write policy could unintentionally allow an untrusted agent to insert itself in the write policy register. Inclusion in the write policy register could allow a malicious or misbehaving agent write access to resources. This action could result in security compromises including leaked information, leaked encryption keys, or modification of device configuration.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Crash, Exit, or Restart	
Access Control	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	
	Read Files or Directories	
	Reduce Reliability	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Access-control-policy definition and programming flow must be sufficiently tested in pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

Consider a system of seven registers for storing and configuring an AES key for encryption or decryption.

Four 32-bit registers are used to store a 128-bit AES key. The names of those registers are AES_ENC_DEC_KEY_0, AES_ENC_DEC_KEY_1, AES_ENC_DEC_KEY_2, and AES_ENC_DEC_KEY_3. Collectively these are referred to as the AES Key registers.

Three 32-bit registers are used to define access control for the AES-key registers. The names of those registers are AES_KEY_CONTROL_POLICY, AES_KEY_READ_POLICY, and AES_KEY_WRITE_POLICY. Collectively these registers are referred to as the Policy registers, and their functions are explained next.

- The AES_KEY_CONTROL_POLICY register defines which agents can write to the AES_KEY_READ_POLICY or AES_KEY_WRITE_POLICY registers.
- The AES_KEY_READ_POLICY register defines which agents can read the AES-key registers.
- The AES_KEY_WRITE_POLICY register defines which agents can write the AES key registers.

The preceding three policy registers encode access control at the bit level. Therefore a maximum of 32 agents can be defined (1 bit per agent). The value of the bit when set (i.e., "1") allows the respective action from an agent whose identity corresponds to the number of the bit. If clear (i.e., "0"), it disallows the respective action to that corresponding agent. For example, if bit 0 is set to "1" in the AES_KEY_READ_POLICY register, then agent 0 has permission to read the AES-key registers.

Consider that there are 4 agents named Agent 1, Agent 2, Agent 3, and Agent 4. For access control purposes Agent 1 is assigned to bit 1, Agent 2 to bit 2, Agent 3 to bit 3, and Agent 4 to bit 4. All agents are trusted except for Agent 3 who is untrusted. Also consider the register values in the below table.

Example Language:

(Bad)

The AES_KEY_CONTROL_POLICY register value is 0x00000018. In binary, the lower 8 bits will be 0001 1000, meaning that:

- Bits 3 and 4 are set, thus Agents 3 and 4 will have write access to AES_KEY_READ_POLICY or AES_KEY_WRITE_POLICY.
- All other bits are clear, hence agents other than 3 and 4 will not have access to write to AES_KEY_READ_POLICY or AES_KEY_WRITE_POLICY.

The AES_KEY_READ_POLICY register value is 0x00000002. In binary, the lower 8 bits will be 0000 0010, meaning that:

- Bit 1 is set, thus Agent 1 will be able to read the AES key registers.

The AES_KEY_WRITE_POLICY register value is 0x00000004. In binary, the lower 8 bits will be 0000 0100, meaning that:

- Bit 2 is set, thus Agent 2 will be able to write the AES Key registers.

The configured access control policy for Agents 1,2,3,4 is summarized in table below.

At this point Agents 3 and 4 can only configure which agents can read AES keys and which agents can write AES keys. Agents 3 and 4 cannot read or write AES keys - just configure access control.

Now, recall Agent 3 is untrusted. As explained above, the value of the AES_KEY_CONTROL_POLICY register gives agent 3 access to write to the AES_KEY_WRITE_POLICY register. Agent 3 can use this write access to add themselves to the AES_KEY_WRITE_POLICY register. This is accomplished by Agent 3 writing the value 0x00000006. In binary, the lower 8 bits are 0000 0110, meaning that bit 3 will be set. Thus, giving Agent 3 having the ability to write to the AES Key registers.

If the AES_KEY_CONTROL_POLICY register value is 0x00000010, the lower 8 bits will be 0001 0000. This will give Agent 4, a trusted agent, write access to AES_KEY_WRITE_POLICY, but Agent 3, who is untrusted, will not have write access. The Policy register values should therefore be as follows:

Example Language:

(Good)

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
180	Exploiting Incorrectly Configured Access Control Security Levels

CWE-1269: Product Released in Non-Release Configuration

Weakness ID : 1269

Structure : Simple

Abstraction : Base

Description

The product released to market is released in pre-production or manufacturing configuration.

Extended Description

Products in the pre-production or manufacturing stages are configured to have many debug hooks and debug capabilities, including but not limited to:

- Ability to override/bypass various cryptographic checks (including authentication, authorization, and integrity)
- Ability to read/write/modify/dump internal state (including registers and memory)
- Ability to change system configurations
- Ability to run hidden or private commands that are not allowed during production (as they expose IP).

The above is by no means an exhaustive list, but it alludes to the greater capability and the greater state of vulnerability of a product during its preproduction or manufacturing state.

Complexity increases when multiple parties are involved in executing the tests before the final production version. For example, a chipmaker might fabricate a chip and run its own preproduction tests, following which the chip would be delivered to the Original Equipment Manufacturer (OEM), who would now run a second set of different preproduction tests on the same chip. Only after both of these sets of activities are complete, can the overall manufacturing phase be called "complete" and have the "Manufacturing Complete" fuse blown. However, if the OEM forgets to blow the

Manufacturing Complete fuse, then the system remains in the manufacturing stage, rendering the system both exposed and vulnerable.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1520

Applicable Platforms

Language : VHDL (*Prevalence = Undetermined*)

Language : Verilog (*Prevalence = Undetermined*)

Language : Compiled (*Prevalence = Undetermined*)

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Other (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Other	High

Potential Mitigations

Phase: Implementation

Ensure that there exists a marker for denoting the Manufacturing Complete stage and that the Manufacturing Complete marker gets updated at the Manufacturing Complete stage (i.e., the Manufacturing Complete fuse gets blown).

Phase: Integration

Ensure that there exists a marker for denoting the Manufacturing Complete stage and that the Manufacturing Complete marker gets updated at the Manufacturing Complete stage (i.e., the Manufacturing Complete fuse gets blown).

Phase: Manufacturing

Ensure that there exists a marker for denoting the Manufacturing Complete stage and that the Manufacturing Complete marker gets updated at the Manufacturing Complete stage (i.e., the Manufacturing Complete fuse gets blown).

Demonstrative Examples

Example 1:

This example shows what happens when a preproduction system is made available for production.

Example Language:

(Bad)

Suppose the chipmaker has a way of scanning all the internal memory (containing chipmaker-level secrets) during the manufacturing phase, and the way the chipmaker or the Original Equipment Manufacturer (OEM) marks the end of the manufacturing phase is by blowing a Manufacturing Complete fuse. Now, suppose that whoever blows the Manufacturing Complete fuse inadvertently forgets to execute the step to blow the fuse.

An attacker will now be able to scan all the internal memory (containing chipmaker-level secrets).

Example Language:

(Good)



Blow the Manufacturing Complete fuse.

Observed Examples

Reference	Description
CVE-2019-13945	Regarding SSA-686531, a hardware based manufacturing access on S7-1200 and S7-200 SMART has occurred. A vulnerability has been identified in SIMATIC S7-1200 CPU family (incl. SIPLUS variants) (All versions), SIMATIC S7-200 SMART CPU family (All versions). There is an access mode used during manufacturing of S7-1200 CPUs that allows additional diagnostic functionality. The security vulnerability could be exploited by an attacker with physical access to the UART interface during boot process. At the time of advisory publication, no public exploitation of this security vulnerability was known. https://www.cve.org/CVERecord?id=CVE-2019-13945
CVE-2018-4251	Laptops with Intel chipsets were found to be running in Manufacturing Mode. After this information was reported to the OEM, the vulnerability (CVE-2018-4251) was patched disallowing access to the interface. https://www.cve.org/CVERecord?id=CVE-2018-4251

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1195	Manufacturing and Life Cycle Management Concerns	1194	2469
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
439	Manipulation During Distribution

References

[REF-1103]Lucian Armasu. "Intel ME's Undocumented Manufacturing Mode Suggests CPU Hacking Risks". 2018 October 3. < <https://www.tomshardware.com/news/intel-me-cpu-undocumented-manufacturing-mode,37883.html> >.

CWE-1270: Generation of Incorrect Security Tokens

Weakness ID : 1270

Structure : Simple

Abstraction : Base

Description

The product implements a Security Token mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. However, the Security Tokens generated in the system are incorrect.

Extended Description

Systems-On-a-Chip (SoC) (Integrated circuits and hardware engines) implement Security Tokens to differentiate and identify actions originated from various agents. These actions could be "read", "write", "program", "reset", "fetch", "compute", etc. Security Tokens are generated and assigned to every agent on the SoC that is either capable of generating an action or receiving an action from another agent. Every agent could be assigned a unique, Security Token based on its trust level or privileges. Incorrectly generated Security Tokens could result in the same token used for multiple agents or multiple tokens being used for the same agent. This condition could result in a Denial-of-Service (DoS) or the execution of an action that in turn could result in privilege escalation or unintended 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf	⊕	1294	Insecure Security Identifier Mechanism	2150

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Files or Directories	High
Integrity	Execute Unauthorized Code or Commands	
Availability	Bypass Protection Mechanism	
Access Control	Gain Privileges or Assume Identity	
	Read Memory	
	Modify Memory	
	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Generation of Security Tokens should be reviewed for design inconsistency and common weaknesses. Security-Token definition and programming flow should be tested in pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

Consider a system with a register for storing an AES key for encryption or decryption. The key is 128 bits long implemented as a set of four 32-bit registers. The key registers are assets, and register, AES_KEY_ACCESS_POLICY, is defined to provide necessary access controls. The access-policy register defines which agents, using a Security Token, may access the AES-key registers. Each bit in this 32-bit register is used to define a Security Token. There could be a maximum of 32 Security Tokens that are allowed access to the AES-key registers. When set (bit = "1") bit number allows action from an agent whose identity matches that bit number. If Clear (bit = "0") the action is disallowed for the corresponding agent.

Assume the system has two agents: a Main-controller and an Aux-controller. The respective Security Tokens are "1" and "2".

An agent with a Security Token "1" has access to AES_ENC_DEC_KEY_0 through AES_ENC_DEC_KEY_3 registers. As per the above access policy, the AES-Key-access policy allows access to the AES-key registers if the security Token is "1".

Example Language: Other (Bad)

The SoC incorrectly generates Security Token "1" for every agent. In other words, both Main-controller and Aux-controller are assigned Security Token "1".

Both agents have access to the AES-key registers.

Example Language: Other (Good)

The SoC should correctly generate Security Tokens, assigning "1" to the Main-controller and "2" to the Aux-controller

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces
633	Token Impersonation
681	Exploitation of Improperly Controlled Hardware Security Identifiers

CWE-1271: Uninitialized Value on Reset for Registers Holding Security Settings

Weakness ID : 1271
Structure : Simple
Abstraction : Base

Description

Security-critical logic is not set to a known value on reset.

Extended Description

When the device is first brought out of reset, the state of registers will be indeterminate if they have not been initialized by the logic. Before the registers are initialized, there will be a window during which the device is in an insecure state and may be vulnerable to attack.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		909	Missing Initialization of Resource	1797

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1304	Improperly Preserved Integrity of Hardware Configuration State During a Power Save/Restore Operation	2176

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control Authentication Authorization	Varies by Context	

Potential Mitigations

Phase: Implementation

Design checks should be performed to identify any uninitialized flip-flops used for security-critical functions.

Phase: Architecture and Design

All registers holding security-critical information should be set to a specific value on reset.

Demonstrative Examples

Example 1:

Shown below is a positive clock edge triggered flip-flop used to implement a lock bit for test and debug interface. When the circuit is first brought out of reset, the state of the flip-flop will be unknown until the enable input and D-input signals update the flip-flop state. In this example, an attacker can reset the device until the test and debug interface is unlocked and access the test interface until the lock signal is driven to a known state by the logic.

Example Language: Verilog

(Bad)

```
always @(posedge clk) begin
    if (en) lock_jtag <= d;
end
```

The flip-flop can be set to a known value (0 or 1) on reset, but requires that the logic explicitly update the output of the flip-flop if the reset signal is active.

Example Language: Verilog

(Good)

```
always @(posedge clk) begin
  if (~reset) lock_jtag <= 0;
  else if (en) lock_jtag <= d;
end
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating State

CWE-1272: Sensitive Information Uncleared Before Debug/Power State Transition

Weakness ID : 1272

Structure : Simple

Abstraction : Base

Description

The product performs a power or debug state transition, but it does not clear sensitive information that should no longer be accessible due to changes to information access restrictions.

Extended Description

A device or system frequently employs many power and sleep states during its normal operation (e.g., normal power, additional power, low power, hibernate, deep sleep, etc.). A device also may be operating within a debug condition. State transitions can happen from one power or debug state to another. If there is information available in the previous state which should not be available in the next state and is not properly removed before the transition into the next state, sensitive information may leak from the system.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	226	Sensitive Information in Resource Not Removed Before Reuse	562

Nature	Type	ID	Name	Page
CanPrecede		200	Exposure of Sensitive Information to an Unauthorized Actor	504

Weakness Ordinalities

Primary :

Applicable Platforms

Language : VHDL (*Prevalence = Undetermined*)

Language : Verilog (*Prevalence = Undetermined*)

Language : Hardware Description Language (*Prevalence = Undetermined*)

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	High
Integrity	Read Application Data	
Availability	<i>Sensitive information may be used to unlock additional capabilities of the device and take advantage of hidden functionalities which could be used to compromise device security.</i>	
Access Control		
Accountability		
Authentication		
Authorization		
Non-Repudiation		

Detection Methods

Manual Analysis

Write a known pattern into each sensitive location. Enter the power/debug state in question. Read data back from the sensitive locations. If the reads are successful, and the data is the same as the pattern that was originally written, the test fails and the device needs to be fixed. Note that this test can likely be automated.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

During state transitions, information not needed in the next state should be removed before the transition to the next state.

Demonstrative Examples

Example 1:

This example shows how an attacker can take advantage of an incorrect state transition.

Suppose a device is transitioning from state A to state B. During state A, it can read certain private keys from the hidden fuses that are only accessible in state A but not in state B. The device reads the keys, performs operations using those keys, then transitions to state B, where those private keys should no longer be accessible.

Example Language:

(Bad)

During the transition from A to B, the device does not scrub the memory.

After the transition to state B, even though the private keys are no longer accessible directly from the fuses in state B, they can be accessed indirectly by reading the memory that contains the private keys.

Example Language:

(Good)

For transition from state A to state B, remove information which should not be available once the transition is complete.

Observed Examples




Reference	Description
CVE-2020-12926	Product software does not set a flag as per TPM specifications, thereby preventing a failed authorization attempt from being recorded after a loss of power. https://www.cve.org/CVERecord?id=CVE-2020-12926

Functional Areas

- Power

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data
150	Collect Data from Common Resource Locations
545	Pull Data from System Resources
546	Incomplete Data Deletion in a Multi-Tenant Environment

References

[REF-1220]Zhenyu Ning and Fengwei Zhang. "Understanding the Security of ARM Debugging Features". 2019 IEEE Symposium on Security and Privacy (SP). 2019 May 2. < <https://www.computer.org/csdl/proceedings-article/sp/2019/666000b156/19skgcwSgsE> >.2023-04-07.

CWE-1273: Device Unlock Credential Sharing

Weakness ID : 1273

Structure : Simple

Abstraction : Base

Description

The credentials necessary for unlocking a device are shared across multiple parties and may expose sensitive information.

Extended Description


"Unlocking a device" often means activating certain unadvertised debug and manufacturer-specific capabilities of a device using sensitive credentials. Unlocking a device might be necessary for the

purpose of troubleshooting device problems. For example, suppose a device contains the ability to dump the content of the full system memory by disabling the memory-protection mechanisms. Since this is a highly security-sensitive capability, this capability is "locked" in the production part. Unless the device gets unlocked by supplying the proper credentials, the debug capabilities are not available. For cases where the chip designer, chip manufacturer (fabricator), and manufacturing and assembly testers are all employed by the same company, the risk of compromise of the credentials is greatly reduced. However, the risk is greater when the chip designer is employed by one company, the chip manufacturer is employed by another company (a foundry), and the assemblers and testers are employed by yet a third company. Since these different companies will need to perform various tests on the device to verify correct device function, they all need to share the unlock key. Unfortunately, the level of secrecy and policy might be quite different at each company, greatly increasing the risk of sensitive credentials being compromised.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		200	Exposure of Sensitive Information to an Unauthorized Actor	504

Applicable Platforms

Language : VHDL (*Prevalence = Undetermined*)

Language : Verilog (*Prevalence = Undetermined*)

Language : Compiled (*Prevalence = Undetermined*)

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Other (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Integrity	Read Memory	
Availability	Modify Files or Directories	
Access Control	Read Files or Directories	
Accountability	Modify Application Data	
Authentication	Execute Unauthorized Code or Commands	
Authorization	Gain Privileges or Assume Identity	
Non-Repudiation	Bypass Protection Mechanism	
	Once unlock credentials are compromised, an attacker can use the credentials to unlock the device and gain unauthorized access to the hidden functionalities protected by those credentials.	

Potential Mitigations

Phase: Integration

Ensure the unlock credentials are shared with the minimum number of parties and with utmost secrecy. To limit the risk associated with compromised credentials, where possible, the credentials should be part-specific.

Phase: Manufacturing

Ensure the unlock credentials are shared with the minimum number of parties and with utmost secrecy. To limit the risk associated with compromised credentials, where possible, the credentials should be part-specific.

Demonstrative Examples**Example 1:**

This example shows how an attacker can take advantage of compromised credentials.

Example Language:

(Bad)

Suppose a semiconductor chipmaker, "C", uses the foundry "F" for fabricating its chips. Now, F has many other customers in addition to C, and some of the other customers are much smaller companies. F has dedicated teams for each of its customers, but somehow it mixes up the unlock credentials and sends the unlock credentials of C to the wrong team. This other team does not take adequate precautions to protect the credentials that have nothing to do with them, and eventually the unlock credentials of C get leaked.

When the credentials of multiple organizations are stored together, exposure to third parties occurs frequently.

Example Language:

(Good)

Vertical integration of a production company is one effective method of protecting sensitive credentials. Where vertical integration is not possible, strict access control and need-to-know are methods which can be implemented to reduce these risks.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2469
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

Notes**Maintenance**

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
560	Use of Known Domain Credentials

CWE-1274: Improper Access Control for Volatile Memory Containing Boot Code

Weakness ID : 1274

Structure : Simple

Abstraction : Base

Description

The product conducts a secure-boot process that transfers bootloader code from Non-Volatile Memory (NVM) into Volatile Memory (VM), but it does not have sufficient access control or other protections for the Volatile Memory.

Extended Description

Adversaries could bypass the secure-boot process and execute their own untrusted, malicious boot code.

As a part of a secure-boot process, the read-only-memory (ROM) code for a System-on-Chip (SoC) or other system fetches bootloader code from Non-Volatile Memory (NVM) and stores the code in Volatile Memory (VM), such as dynamic, random-access memory (DRAM) or static, random-access memory (SRAM). The NVM is usually external to the SoC, while the VM is internal to the SoC. As the code is transferred from NVM to VM, it is authenticated by the SoC's ROM code.

If the volatile-memory-region protections or access controls are insufficient to prevent modifications from an adversary or untrusted agent, the secure boot may be bypassed or replaced with the execution of an adversary's code.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory	High
Integrity	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	

Detection Methods

Manual Analysis

Ensure the volatile memory is lockable or has locks. Ensure the volatile memory is locked for writes from untrusted agents or adversaries. Try modifying the volatile memory from an untrusted agent, and ensure these writes are dropped.

Effectiveness = High

Manual Analysis

Analyze the device using the following steps: Identify all fabric master agents that are active during system Boot Flow when initial code is loaded from Non-volatile storage to volatile memory. Identify the volatile memory regions that are used for storing loaded system executable program. During system boot, test programming the identified memory regions in step 2 from all the masters identified in step 1. Only trusted masters should be allowed to write to the memory

regions. For example, pluggable device peripherals should not have write access to program load memory regions.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Ensure that the design of volatile-memory protections is enough to prevent modification from an adversary or untrusted code.

Phase: Testing

Test the volatile-memory protections to ensure they are safe from modification or untrusted code.

Demonstrative Examples

Example 1:

A typical SoC secure boot's flow includes fetching the next piece of code (i.e., the boot loader) from NVM (e.g., serial, peripheral interface (SPI) flash), and transferring it to DRAM/SRAM volatile, internal memory, which is more efficient.

Example Language:

(Bad)

The volatile-memory protections or access controls are insufficient.

The memory from where the boot loader executes can be modified by an adversary.

Example Language:

(Good)

A good architecture should define appropriate protections or access controls to prevent modification by an adversary or untrusted agent, once the bootloader is authenticated.

Observed Examples

Reference	Description
CVE-2019-2267	Locked memory regions may be modified through other interfaces in a secure-boot-loader image due to improper access control. https://www.cve.org/CVERecord?id=CVE-2019-2267

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2469
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
456	Infected Memory
679	Exploitation of Improperly Configured or Implemented Memory Protections

CWE-1275: Sensitive Cookie with Improper SameSite Attribute

Weakness ID : 1275

Structure : Simple

Abstraction : Variant

Description

The SameSite attribute for sensitive cookies is not set, or an insecure value is used.


Extended Description

The SameSite attribute controls how cookies are sent for cross-domain requests. This attribute may have three values: 'Lax', 'Strict', or 'None'. If the 'None' value is used, a website may create a cross-domain POST HTTP request to another website, and the browser automatically adds cookies to this request. This may lead to Cross-Site-Request-Forgery (CSRF) attacks if there are no additional protections in place (such as Anti-CSRF tokens).

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		923	Improper Restriction of Communication Channel to Intended Endpoints	1827
CanPrecede		352	Cross-Site Request Forgery (CSRF)	868

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Web Based (*Prevalence = Undetermined*)

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Non-Repudiation Access Control	Modify Application Data <i>If the website does not impose additional defense against CSRF attacks, failing to use the 'Lax' or 'Strict' values could increase the risk of exposure to CSRF attacks. The likelihood of the integrity breach is Low because a successful attack does not only depend on an insecure SameSite attribute. In order to perform a CSRF attack there are many conditions that must be met, such as the lack of CSRF tokens, no confirmations for sensitive actions on the website, a "simple" "Content-Type" header in the HTTP request and many more.</i>	Low

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 the SameSite attribute of a sensitive cookie to 'Lax' or 'Strict'. This instructs the browser to apply this cookie only to same-domain requests, which provides a good Defense in Depth against CSRF attacks. When the 'Lax' value is in use, cookies are also sent for top-level cross-domain navigation via HTTP GET, HEAD, OPTIONS, and TRACE methods, but not for other HTTP methods that are more like to cause side-effects of state mutation.

Effectiveness = High

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

Demonstrative Examples

Example 1:

In this example, a cookie is used to store a session ID for a client's interaction with a website. The snippet of code below establishes a new cookie to hold the sessionId.

Example Language: JavaScript

(Bad)

```
let sessionId = generateSessionId()
let cookieOptions = { domain: 'example.com' }
response.cookie('sessionId', sessionId, cookieOptions)
```

Since the sameSite attribute is not specified, the cookie will be sent to the website with each request made by the client. An attacker can potentially perform a CSRF attack by using the following malicious page:

Example Language: HTML

(Attack)

```
<html>
  <form id=evil action="http://local:3002/setEmail" method="POST">
    <input type="hidden" name="newEmail" value="abc@example.com" />
  </form>
  <script>evil.submit()</script>
</html>
```

When the client visits this malicious web page, it submits a '/setEmail' POST HTTP request to the vulnerable website. Since the browser automatically appends the 'sessionId' cookie to the request, the website automatically performs a 'setEmail' action on behalf of the client.

To mitigate the risk, use the sameSite attribute of the 'sessionId' cookie set to 'Strict'.

Example Language: JavaScript

(Good)



```
let sessionId = generateSessionId()
let cookieOptions = { domain: 'example.com', sameSite: 'Strict' }
response.cookie('sessionId', sessionId, cookieOptions)
```

Observed Examples

Reference	Description
CVE-2022-24045	Web application for a room automation system has client-side JavaScript that sets a sensitive cookie without the SameSite security attribute, allowing the cookie to be sniffed https://www.cve.org/CVERecord?id=CVE-2022-24045

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control	1344	2487
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
62	Cross Site Request Forgery

References

[REF-1104]M. West and M. Goodwin. "SameSite attribute specification draft". 2016 April 6. < <https://datatracker.ietf.org/doc/html/draft-west-first-party-cookies-07> >.2023-04-07.

[REF-1105]Mozilla. "SameSite attribute description on MDN Web Docs". 2020 June 0. < <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite> >.

[REF-1106]The Chromium Projects. "Chromium support for SameSite attribute". 2019 September 6. < <https://www.chromium.org/updates/same-site/> >.2023-04-07.

CWE-1276: Hardware Child Block Incorrectly Connected to Parent System

Weakness ID : 1276

Structure : Simple

Abstraction : Base

Description

Signals between a hardware IP and the parent system design are incorrectly connected causing security risks.

Extended Description

Individual hardware IP must communicate with the parent system in order for the product to function correctly and as intended. If implemented incorrectly, while not causing any apparent functional issues, may cause security issues. For example, if the IP should only be reset by a system-wide hard reset, but instead the reset input is connected to a software-triggered debug mode reset (which is also asserted during a hard reset), integrity of data inside the IP can be violated.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	IP	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Varies by Context	

Potential Mitigations

Phase: Testing

System-level verification may be used to ensure that components are correctly connected and that design security requirements are not violated due to interactions between various IP blocks.

Demonstrative Examples

Example 1:

Many SoCs use hardware to partition system resources between trusted and un-trusted entities. One example of this concept is the Arm TrustZone, in which the processor and all security-aware IP attempt to isolate resources based on the status of a privilege bit. This privilege bit is part of the input interface in all TrustZone-aware IP. If this privilege bit is accidentally grounded or left unconnected when the IP is instantiated, privilege escalation of all input data may occur.

Example Language: Verilog

(Bad)

```
// IP definition
module tz_peripheral(clk, reset, data_in, data_in_security_level, ...);
    input clk, reset;
    input [31:0] data_in;
    input data_in_security_level;
    ...
endmodule
// Instantiation of IP in a parent system
module soc(...)
    ...
    tz_peripheral u_tz_peripheral(
        .clk(clk),
        .rst(rst),
        .data_in(rdata),
        //Copy-and-paste error or typo grounds data_in_security_level (in this example 0=secure, 1=non-secure) effectively
        //promoting all data to "secure"
        .data_in_security_level(1'b0),
    );
    ...
endmodule
```

In the Verilog code below, the security level input to the TrustZone aware peripheral is correctly driven by an appropriate signal instead of being grounded.

Example Language: Verilog

(Good)

```
// Instantiation of IP in a parent system
module soc(...)
    ...
```

```

tz_peripheral u_tz_peripheral(
    .clk(clk),
    .rst(rst),
    .data_in(rdata),
    // This port is no longer grounded, but instead driven by the appropriate signal
    .data_in_security_level(rdata_security_level),
);
...
endmodule

```

Example 2:

Here is a code snippet from the Ariane core module in the HACK@DAC'21 Openpiton SoC [REF-1362]. To ensure full functional correctness, developers connect the ports with names. However, in some cases developers forget to connect some of these ports to the desired signals in the parent module. These mistakes by developers can lead to incorrect functional behavior or, in some cases, introduce security vulnerabilities.

Example Language: Verilog

(Bad)

```

...
csr_regfile #(
    ...
) csr_regfile_i (
    .flush_o ( flush_csr_ctrl ),
    .halt_csr_o ( halt_csr_ctrl ),
    ...
    .irq_i(),
    .time_irq_i(),
    *
);
...

```

In the above example from HACK@DAC'21, since interrupt signals are not properly connected, the CSR module will fail to send notifications in the event of interrupts. Consequently, critical information in CSR registers that should be flushed or modified in response to an interrupt won't be updated. These vulnerabilities can potentially result in information leakage across various privilege levels.

To address the aforementioned vulnerability, developers must follow a two-step approach. First, they should ensure that all module signals are properly connected. This can often be facilitated using automated tools, and many simulators and sanitizer tools issue warnings when a signal remains unconnected or floats. Second, it is imperative to validate that the signals connected to a module align with the specifications. In the provided example, the developer should establish the correct connection of interrupt signals from the parent module (Ariane core) to the child module (csr_regfile) [REF-1363].

Example Language: Verilog

(Good)

```

...
csr_regfile #(
    ...
) csr_regfile_i (
    .flush_o ( flush_csr_ctrl ),
    .halt_csr_o ( halt_csr_ctrl ),
    ...
    .irq_i (irq_i),
    .time_irq_i (time_irq_i),
    *
);
...

```


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

Nature	Type	ID	Name	V	Page
MemberOf	C	1197	Integration Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

References

[REF-1362]"ariane.sv". 2021. < <https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fedef7a9d9b/piton/design/chip/tile/ariane/src/ariane.sv#L539:L540> >.2023-07-15.

[REF-1363]"Fix CWE-1276". 2021. < <https://github.com/HACK-EVENT/hackatdac21/blob/9a796ee83e21f59476d4b0a68ec3d8e8d5148214/piton/design/chip/tile/ariane/src/ariane.sv#L539:L540> >.2023-09-01.

CWE-1277: Firmware Not Updateable

Weakness ID : 1277

Structure : Simple

Abstraction : Base

Description

The product does not provide its users with the ability to update or patch its firmware to address any vulnerabilities or weaknesses that may be present.

Extended Description

Without the ability to patch or update firmware, consumers will be left vulnerable to exploitation of any known vulnerabilities, or any vulnerabilities that are discovered in the future. This can expose consumers to permanent risk throughout the entire lifetime of the device, which could be years or decades. Some external protective measures and mitigations might be employed to aid in preventing or reducing the risk of malicious attack, but the root weakness cannot be corrected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	1329	Reliance on Component That is Not Updateable	2219

Weakness Ordinalities

Primary :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Gain Privileges or Assume Identity	Medium
Integrity	Bypass Protection Mechanism	
Access Control	Execute Unauthorized Code or Commands	
Authentication	DoS: Crash, Exit, or Restart	
Authorization	<i>If an attacker can identify an exploitable vulnerability in one device that has no means of patching, the attack may be used against an entire class of devices.</i>	

Detection Methods

Manual Analysis

Create a new installable boot image of the current build with a minor version number change. Use the standard installation method to update the boot image. Verify that the minor version number has changed. Create a fake image. Verify that the boot updater will not install the fake image and generates an "invalid image" error message or equivalent.

Effectiveness = High

Architecture or Design Review

Check the consumer or maintainer documentation, the architecture/design documentation, or the original requirements to ensure that the documentation includes details for how to update the firmware.

Effectiveness = Moderate

Manual Dynamic Analysis

Determine if there is a lack of a capability to update read-only memory (ROM) structure. This could manifest as a difference between the latest firmware version and the current version within the device.

Effectiveness = High

Potential Mitigations

Phase: Requirements

Specify requirements to include the ability to update the firmware. Include integrity checks and authentication to ensure that untrusted firmware cannot be installed.

Phase: Architecture and Design

Design the device to allow for updating the firmware. Ensure that the design specifies how to distribute the updates and ensure their integrity and authentication.

Phase: Implementation

Implement the necessary functionality to allow the firmware to be updated.

Demonstrative Examples

Example 1:

A refrigerator has an Internet interface for the official purpose of alerting the manufacturer when that refrigerator detects a fault. Because the device is attached to the Internet, the refrigerator is a target for hackers who may wish to use the device other potentially more nefarious purposes.

Example Language: Other

(Bad)

The refrigerator has no means of patching and is hacked becoming a spewer of email spam.

Example Language: Other

(Good)

The device automatically patches itself and provides considerable more protection against being hacked.

Observed Examples

Reference	Description
CVE-2020-9054	Chain: network-attached storage (NAS) device has a critical OS command injection (CWE-78) vulnerability that is actively exploited to place IoT devices into a botnet, but some products are "end-of-support" and cannot be patched (CWE-1277). [REF-1097] https://www.cve.org/CVERecord?id=CVE-2020-9054
[REF-1095]	A hardware "smart lock" has weak key generation that allows attackers to steal the key by BLE sniffing, but the device's firmware cannot be upgraded and hence remains vulnerable [REF-1095]. https://www.theregister.com/2019/12/11/f_secure_keywe/

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2474
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2544

Notes

Terminology

The "firmware" term does not have a single commonly-shared definition, so there may be variations in how this CWE entry is interpreted during mapping.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
682	Exploitation of Firmware or ROM Code with Unpatchable Vulnerabilities

References

[REF-1095]Matthew Hughes. "Bad news: KeyWe Smart Lock is easily bypassed and can't be fixed". 2019 December 1. < https://www.theregister.com/2019/12/11/f_secure_keywe/ >.2023-04-07.

[REF-1096]Alex Scroxton. "Alarm bells ring, the IoT is listening". < <https://www.computerweekly.com/news/252475324/Alarm-bells-ring-the-IoT-is-listening> >.

[REF-1097]Brian Krebs. "Zyxel Flaw Powers New Mirai IoT Botnet Strain". 2020 March 0. < <https://krebsonsecurity.com/2020/03/zyxel-flaw-powers-new-mirai-iot-botnet-strain/> >.

CWE-1278: Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques

Weakness ID : 1278

Structure : Simple

Abstraction : Base

Description

Information stored in hardware may be recovered by an attacker with the capability to capture and analyze images of the integrated circuit using techniques such as scanning electron microscopy.

Extended Description

The physical structure of a device, viewed at high enough magnification, can reveal the information stored inside. Typical steps in IC reverse engineering involve removing the chip packaging (decapsulation) then using various imaging techniques ranging from high resolution x-ray microscopy to invasive techniques involving removing IC layers and imaging each layer using a scanning electron microscope.

The goal of such activities is to recover secret keys, unique device identifiers, and proprietary code and circuit designs embedded in hardware that the attacker has been unsuccessful at accessing through other means. These secrets may be stored in non-volatile memory or in the circuit netlist. Memory technologies such as masked ROM allow easier to extraction of secrets than One-time Programmable (OTP) memory.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	693	Protection Mechanism Failure	1520

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context <i>A common goal of malicious actors who reverse engineer ICs is to produce and sell counterfeit versions of the IC.</i>	

Potential Mitigations

Phase: Architecture and Design

The cost of secret extraction via IC reverse engineering should outweigh the potential value of the secrets being extracted. Threat model and value of secrets should be used to choose the technology used to safeguard those secrets. Examples include IC camouflaging and obfuscation, tamper-proof packaging, active shielding, and physical tampering detection information erasure.

Demonstrative Examples

Example 1:

Consider an SoC design that embeds a secret key in read-only memory (ROM). The key is baked into the design logic and may not be modified after fabrication causing the key to be identical for all devices. An attacker in possession of the IC can decapsulate and delayer the device. After imaging the layers, computer vision algorithms or manual inspection of the circuit features locate the ROM and reveal the value of the key bits as encoded in the visible circuit structure of the ROM.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1372	ICS Supply Chain: OT Counterfeit and Malicious Corruption	1358	2509
MemberOf	C	1377	ICS Engineering (Construction/Deployment): Inherent Predictability in Design	1358	2513
MemberOf	C	1388	Physical Access Issues and Concerns	1194	2518
MemberOf	C	1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements. It is more attack-oriented, so it might be more suited for CAPEC.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data
188	Reverse Engineering
545	Pull Data from System Resources

References

[REF-1092]Shahed E. Quadir, Junlin Chen, Domenic Forte, Navid Asadizanjani, Sina Shahbazmohamadi, Lei Wang, John Chandy and Mark Tehranipoor. "A Survey on Chip to System Reverse Engineering". < <https://dl.acm.org/doi/pdf/10.1145/2755563> >.2023-04-07.

[REF-1129]Christopher Tarnovsky. "Security Failures In Secure Devices". 2008 February 1. < <https://www.blackhat.com/presentations/bh-dc-08/Tarnovsky/Presentation/bh-dc-08-tarnovsky.pdf> >.

CWE-1279: Cryptographic Operations are run Before Supporting Units are Ready

Weakness ID : 1279

Structure : Simple

Abstraction : Base

Description

Performing cryptographic operations without ensuring that the supporting inputs are ready to supply valid data may compromise the cryptographic result.

Extended Description

Many cryptographic hardware units depend upon other hardware units to supply information to them to produce a securely encrypted result. For example, a cryptographic unit that depends on an external random-number-generator (RNG) unit for entropy must wait until the RNG unit is producing random numbers. If a cryptographic unit retrieves a private encryption key from a fuse unit, the fuse unit must be up and running before a key may be supplied.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1456
ChildOf		691	Insufficient Control Flow Management	1517

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Varies by Context	
Confidentiality		
Integrity		
Availability		
Accountability		
Authentication		
Authorization		
Non-Repudiation		

Potential Mitigations

Phase: Architecture and Design

Best practices should be used to design cryptographic systems.

Phase: Implementation

Continuously ensuring that cryptographic inputs are supplying valid information is necessary to ensure that the encrypted output is secure.

Demonstrative Examples

Example 1:

The following pseudocode illustrates the weak encryption resulting from the use of a pseudo-random-number generator output.

Example Language: Pseudocode

(Bad)

```
If random_number_generator_self_test_passed() == TRUE
then Seed = get_random_number_from_RNG()
else Seed = hardcoded_number
```

In the example above, first a check of RNG ready is performed. If the check fails, the RNG is ignored and a hard coded value is used instead. The hard coded value severely weakens the encrypted output.



Example Language: Pseudocode

(Good)

```
If random_number_generator_self_test_passed() == TRUE
then Seed = get_random_number_from_RNG()
else enter_error_state()
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1205	Security Primitives and Cryptography Issues	1194	2473
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
97	Cryptanalysis

CWE-1280: Access Control Check Implemented After Asset is Accessed

Weakness ID : 1280

Structure : Simple

Abstraction : Base

Description

A product's hardware-based access control check occurs after the asset has been accessed.


Extended Description

The product implements a hardware-based access control check. The asset should be accessible only after the check is successful. If, however, this operation is not atomic and the asset is accessed before the check is complete, the security of the system may be compromised.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680
ChildOf		696	Incorrect Behavior Order	1527

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory	
Confidentiality	Read Memory	
Integrity	Modify Application Data	
	Read Application Data	

Scope	Impact	Likelihood
	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Implement the access control check first. Access should only be given to asset if agent is authorized.

Demonstrative Examples

Example 1:

Assume that the module `foo_bar` implements a protected register. The register content is the asset. Only transactions made by user id (indicated by signal `usr_id`) 0x4 are allowed to modify the register contents. The signal `grant_access` is used to provide access.

Example Language: Verilog

(Bad)

```
module foo_bar(data_out, usr_id, data_in, clk, rst_n);
output reg [7:0] data_out;
input wire [2:0] usr_id;
input wire [7:0] data_in;
input wire clk, rst_n;
wire grant_access;
always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        data_out = 0;
    else
        data_out = (grant_access) ? data_in : data_out;
        assign grant_access = (usr_id == 3'h4) ? 1'b1 : 1'b0;
end
endmodule
```

This code uses Verilog blocking assignments for `data_out` and `grant_access`. Therefore, these assignments happen sequentially (i.e., `data_out` is updated to new value first, and `grant_access` is updated the next cycle) and not in parallel. Therefore, the asset `data_out` is allowed to be modified even before the access control check is complete and `grant_access` signal is set. Since `grant_access` does not have a reset value, it will be meta-stable and will randomly go to either 0 or 1.

Flipping the order of the assignment of `data_out` and `grant_access` should solve the problem. The correct snippet of code is shown below.

Example Language: Verilog

(Good)

```
always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        data_out = 0;
    else
        assign grant_access = (usr_id == 3'h4) ? 1'b1 : 1'b0;
        data_out = (grant_access) ? data_in : data_out;
end
endmodule
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
180	Exploiting Incorrectly Configured Access Control Security Levels

CWE-1281: Sequence of Processor Instructions Leads to Unexpected Behavior

Weakness ID : 1281

Structure : Simple

Abstraction : Base

Description

Specific combinations of processor instructions lead to undesirable behavior such as locking the processor until a hard reset performed.

Extended Description

If the instruction set architecture (ISA) and processor logic are not designed carefully and tested thoroughly, certain combinations of instructions may lead to locking the processor or other unexpected and undesirable behavior. Upon encountering unimplemented instruction opcodes or illegal instruction operands, the processor should throw an exception and carry on without negatively impacting security. However, specific combinations of legal and illegal instructions may cause unexpected behavior with security implications such as allowing unprivileged programs to completely lock the CPU.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1517

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Availability		

Potential Mitigations

Phase: Testing

Implement a rigorous testing strategy that incorporates randomization to explore instruction sequences that are unlikely to appear in normal workloads in order to identify halt and catch fire instruction sequences.

Phase: Patching and Maintenance

Patch operating system to avoid running Halt and Catch Fire type sequences or to mitigate the damage caused by unexpected behavior. See [REF-1108].

Demonstrative Examples**Example 1:**

The Pentium F00F bug is a real-world example of how a sequence of instructions can lock a processor. The "cmpxchg8b" instruction compares contents of registers with a memory location. The operand is expected to be a memory location, but in the bad code snippet it is the eax register. Because the specified operand is illegal, an exception is generated, which is the correct behavior and not a security issue in itself. However, when prefixed with the "lock" instruction, the processor deadlocks because locked memory transactions require a read and write pair of transactions to occur before the lock on the memory bus is released. The exception causes a read to occur but there is no corresponding write, as there would have been if a legal operand had been supplied to the cmpxchg8b instruction. [REF-1331]

Example Language: x86 Assembly

(Bad)

```
lock cmpxchg8b eax
```

Example 2:

The Cyrix Coma bug was capable of trapping a Cyrix 6x86, 6x86L, or 6x86MX processor in an infinite loop. An infinite loop on a processor is not necessarily an issue on its own, as interrupts could stop the loop. However, on select Cyrix processors, the x86 Assembly 'xchg' instruction was designed to prevent interrupts. On these processors, if the loop was such that a new 'xchg' instruction entered the instruction pipeline before the previous one exited, the processor would become deadlocked. [REF-1323]

Example 3:

The Motorola MC6800 microprocessor contained the first documented instance of a Halt and Catch Fire instruction - an instruction that causes the normal function of a processor to stop. If the MC6800 was given the opcode 0x9D or 0xDD, the processor would begin to read all memory very quickly, in sequence, and without executing any other instructions. This will cause the processor to become unresponsive to anything but a hard reset. [REF-1324]

Example 4:

The example code is taken from the commit stage inside the processor core of the HACK@DAC'19 buggy CVA6 SoC [REF-1342]. To ensure the correct execution of atomic instructions, the CPU must guarantee atomicity: no other device overwrites the memory location between the atomic read starts and the atomic write finishes. Another device may overwrite the memory location only before the read operation or after the write operation, but never between them, and finally, the content will still be consistent.

Atomicity is especially critical when the variable to be modified is a mutex, counting semaphore, or similar piece of data that controls access to shared resources. Failure to ensure atomicity may result in two processors accessing a shared resource simultaneously, permanent lock-up, or similar disastrous behavior.

Example Language: Verilog

(Bad)

```
if (csr_exception_i.valid && csr_exception_i.cause[63] && commit_instr_i[0].fu != CSR) begin
    exception_o = csr_exception_i;
```

```
exception_o.tval = commit_instr_i[0].ex.tval;  
end
```

The above vulnerable code checks for CSR interrupts and gives them precedence over any other exception. However, the interrupts should not occur when the processor runs a series of atomic instructions. In the above vulnerable code, the required check must be included to ensure the processor is not in the middle of a series of atomic instructions.

Refrain from interrupting if the intention is to commit an atomic instruction that should not be interrupted. This can be done by adding a condition to check whether the current committing instruction is atomic. [REF-1343]

Example Language: Verilog

(Good)



```
if (csr_exception_i.valid && csr_exception_i.cause[63] && !amo_valid_commit_o && commit_instr_i[0].fu != CSR) begin  
    exception_o = csr_exception_i;  
    exception_o.tval = commit_instr_i[0].ex.tval;  
end
```

Observed Examples

Reference	Description
CVE-2021-26339	A bug in AMD CPU's core logic allows a potential DoS by using a specific x86 instruction sequence to hang the processor https://www.cve.org/CVERecord?id=CVE-2021-26339
CVE-1999-1476	A bug in some Intel Pentium processors allow DoS (hang) via an invalid "CMPXCHG8B" instruction, causing a deadlock https://www.cve.org/CVERecord?id=CVE-1999-1476

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1201	Core and Compute Issues	1194	2471
MemberOf		1410	Comprehensive Categorization: Insufficient Control Flow Management	1400	2536

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
212	Functionality Misuse

References

[REF-1094]Christopher Domas. "Breaking the x86 ISA". < https://github.com/xoreaxeaxeax/sandsifter/blob/master/references/domas_breaking_the_x86_isa_wp.pdf >.

[REF-1108]Intel Corporation. "Deep Dive: Retpoline: A Branch Target Injection Mitigation". < <https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/overview.html> >.2023-04-07.

[REF-1323]"Cyril coma bug". 2006 March 2. Wikipedia. < https://en.wikipedia.org/wiki/Cyril_coma_bug >.

[REF-1324]Gary Wheeler. "Undocumented M6800 Instructions". 1977 December. < <https://spivey.oriel.ox.ac.uk/wiki/images-corner/1/1a/Undoc6800.pdf> >.2023-04-20.

[REF-1331]Robert R. Collins. "The Pentium F00F Bug". 1998 May 1. < <https://www.drdoobs.com/embedded-systems/the-pentium-f00f-bug/184410555> >.2023-04-25.

[REF-1342]"Hackatdac19 commit_stage.sv". 2019. < https://github.com/HACK-EVENT/hackatdac19/blob/619e9fb0ef32ee1e01ad76b8732a156572c65700/src/commit_stage.sv#L287:L290 >.2023-06-21.

[REF-1343]Florian Zaruba, Michael Schaffner, Stefan Mach and Andreas Traber. "commit_stage.sv". 2018. < https://github.com/openhwgroup/cva6/blob/7951802a0147aedb21e8f2f6dc1e1e9c4ee857a2/src/commit_stage.sv#L296:L301 >.2023-06-21.

CWE-1282: Assumed-Immutable Data is Stored in Writable Memory

Weakness ID : 1282

Structure : Simple

Abstraction : Base

Description

Immutable data, such as a first-stage bootloader, device identifiers, and "write-once" configuration settings are stored in writable memory that can be re-programmed or updated in the field.



Extended Description

Security services such as secure boot, authentication of code and data, and device attestation all require assets such as the first stage bootloader, public keys, golden hash digests, etc. which are implicitly trusted. Storing these assets in read-only memory (ROM), fuses, or one-time programmable (OTP) memory provides strong integrity guarantees and provides a root of trust for securing the rest of the system. Security is lost if assets assumed to be immutable can be modified.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1469
CanPrecede		471	Modification of Assumed-Immutable Data (MAID)	1121

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	

Potential Mitigations

Phase: Implementation

All immutable code or data should be programmed into ROM or write-once memory.



Demonstrative Examples

Example 1:

Cryptographic hash functions are commonly used to create unique fixed-length digests used to ensure the integrity of code and keys. A golden digest is stored on the device and compared to the digest computed from the data to be verified. If the digests match, the data has not been maliciously modified. If an attacker can modify the golden digest they then have the ability to store arbitrary data that passes the verification check. Hash digests used to verify public keys and early stage boot code should be immutable, with the strongest protection offered by hardware immutability.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1202	Memory and Storage Issues	1194	2472
MemberOf		1403	Comprehensive Categorization: Exposed Resource	1400	2528

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Maintenance

As of CWE 4.3, CWE-1282 and CWE-1233 are being investigated for potential duplication or overlap.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
458	Flash Memory Attacks
679	Exploitation of Improperly Configured or Implemented Memory Protections

CWE-1283: Mutable Attestation or Measurement Reporting Data

Weakness ID : 1283

Structure : Simple

Abstraction : Base

Description

The register contents used for attestation or measurement reporting data to verify boot flow are modifiable by an adversary.

Extended Description

A System-on-Chip (SoC) implements secure boot or verified boot. During this boot flow, the SoC often measures the code that it authenticates. The measurement is usually done by calculating the one-way hash of the code binary and extending it to the previous hash. The hashing algorithm should be a Secure One-Way hash function. The final hash, i.e., the value obtained after the completion of the boot flow, serves as the measurement data used in reporting or in attestation. The calculated hash is often stored in registers that can later be read by the party of interest to determine tampering of the boot flow. A common weakness is that the contents in these registers are modifiable by an adversary, thus spoofing the measurement.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf	I P	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Measurement data should be stored in registers that are read-only or otherwise have access controls that prevent modification by an untrusted agent.

Demonstrative Examples

Example 1:

The SoC extends the hash and stores the results in registers. Without protection, an adversary can write their chosen hash values to these registers. Thus, the attacker controls the reported results.

To prevent the above scenario, the registers should have one or more of the following properties:

- Should be Read-Only with respect to an adversary
- Cannot be extended or modifiable either directly or indirectly (using a trusted agent as proxy) by an adversary
- Should have appropriate access controls or protections

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2469
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry is still in development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
680	Exploitation of Improperly Controlled Registers

References

[REF-1107]Intel Corporation. "PCIe Device Measurement Requirements". 2018 September. < <https://www.intel.com/content/dam/www/public/us/en/documents/reference-guides/pcie-device-security-enhancements.pdf> >.

[REF-1131]John Butterworth, Cory Kallenberg and Xeno Kovah. "BIOS Chronomancy: Fixing the Core Root of Trust for Measurement". 2013 July 1. < <https://media.blackhat.com/us-13/US-13-Butterworth-BIOS-Security-Slides.pdf> >.

CWE-1284: Improper Validation of Specified Quantity in Input

Weakness ID : 1284

Structure : Simple

Abstraction : Base

Description

The product receives input that is expected to specify a quantity (such as size or length), but it does not validate or incorrectly validates that the quantity has the required properties.




Extended Description

Specified quantities include size, length, frequency, price, rate, number of operations, time, and others. Code may rely on specified quantities to allocate resources, perform calculations, control iteration, etc. When the quantity is not properly validated, then attackers can specify malicious quantities to cause excessive resource allocation, trigger unexpected failures, enable buffer overflows, etc.

Relationships

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



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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
ParentOf		606	Unchecked Input for Loop Condition	1357
CanPrecede		789	Memory Allocation with Excessive Size Value	1674

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478
MemberOf		1218	Memory Buffer Errors	2479

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Scope	Impact	Likelihood
	<i>Since quantities are used so often to affect resource allocation or process financial data, they are often present in many places in the code.</i>	

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.

Effectiveness = High

Demonstrative Examples

Example 1:

This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.

Example Language: Java

(Bad)

```
...
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
...
```

The user has no control over the price variable, however the code does not prevent a negative value from being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.

Example 2:

This example asks the user for a height and width of an m X n game board with a maximum dimension of 100 squares.

Example Language: C

(Bad)

```
...
#define MAX_DIM 100
...
/* board dimensions */
int m,n, error;
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
printf("Please specify the board width: \n");
error = scanf("%d", &n);
```

```
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
if ( m > MAX_DIM || n > MAX_DIM ) {
    die("Value too large: Die evil hacker!\n");
}
board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
...
```

While this code checks to make sure the user cannot specify large, positive integers and consume too much memory, it does not check for negative values supplied by the user. As a result, an attacker can perform a resource consumption (CWE-400) attack against this program by specifying two, large negative values that will not overflow, resulting in a very large memory allocation (CWE-789) and possibly a system crash. Alternatively, an attacker can provide very large negative values which will cause an integer overflow (CWE-190) and unexpected behavior will follow depending on how the values are treated in the remainder of the program.

Observed Examples

Reference	Description
CVE-2022-21668	Chain: Python library does not limit the resources used to process images that specify a very large number of bands (CWE-1284), leading to excessive memory consumption (CWE-789) or an integer overflow (CWE-190). https://www.cve.org/CVERecord?id=CVE-2022-21668
CVE-2008-1440	lack of validation of length field leads to infinite loop https://www.cve.org/CVERecord?id=CVE-2008-1440
CVE-2008-2374	lack of validation of string length fields allows memory consumption or buffer over-read https://www.cve.org/CVERecord?id=CVE-2008-2374

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1285: Improper Validation of Specified Index, Position, or Offset in Input

Weakness ID : 1285

Structure : Simple

Abstraction : Base

Description

The product receives input that is expected to specify an index, position, or offset into an indexable resource such as a buffer or file, but it does not validate or incorrectly validates that the specified index/position/offset has the required properties.

Extended Description




Often, indexable resources such as memory buffers or files can be accessed using a specific position, index, or offset, such as an index for an array or a position for a file. When untrusted input

is not properly validated before it is used as an index, attackers could access (or attempt to access) unauthorized portions of these resources. This could be used to cause buffer overflows, excessive resource allocation, or trigger unexpected failures.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
ParentOf		129	Improper Validation of Array Index	341
ParentOf		781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1646

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

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.

Effectiveness = High

Demonstrative Examples

Example 1:

The following example retrieves the sizes of messages for a pop3 mail server. The message sizes are retrieved from a socket that returns in a buffer the message number and the message size, the message number (num) and size (size) are extracted from the buffer and the message size is placed into an array using the message number for the array index.

Example Language: C

(Bad)

```

/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    ...
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_rcv(sock, buf, sizeof(buf))) == 0)
    {
        // continue read from socket until buf only contains '.'
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2)
            sizes[num - 1] = size;
    }
    ...
}

```

In this example the message number retrieved from the buffer could be a value that is outside the allowable range of indices for the array and could possibly be a negative number. Without proper validation of the value to be used for the array index an array overflow could occur and could potentially lead to unauthorized access to memory addresses and system crashes. The value of the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

Example Language: C

(Good)

```

/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    ...
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_rcv(sock, buf, sizeof(buf))) == 0)
    {
        // continue read from socket until buf only contains '.'
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2) {
            if (num > 0 && num <= (unsigned)count)
                sizes[num - 1] = size;
            else
                /* warn about possible attempt to induce buffer overflow */
                report(stderr, "Warning: ignoring bogus data for message sizes returned by server.\n");
        }
    }
    ...
}

```

Example 2:

In the following example the method `displayProductSummary` is called from a Web service servlet to retrieve product summary information for display to the user. The servlet obtains the integer value of the product number from the user and passes it to the `displayProductSummary` method. The `displayProductSummary` method passes the integer value of the product number to the `getProductSummary` method which obtains the product summary from the array object containing the project summaries using the integer value of the product number as the array index.

Example Language: Java

(Bad)

```

// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("");
    try {

```

```

    String productSummary = getProductSummary(index);
  } catch (Exception ex) {...}
  return productSummary;
}
public String getProductSummary(int index) {
  return products[index];
}

```

In this example the integer value used as the array index that is provided by the user may be outside the allowable range of indices for the array which may provide unexpected results or cause the application to fail. The integer value used for the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

Example Language: Java

(Good)

```

// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
  String productSummary = new String("");
  try {
    String productSummary = getProductSummary(index);
  } catch (Exception ex) {...}
  return productSummary;
}
public String getProductSummary(int index) {
  String productSummary = "";
  if ((index >= 0) && (index < MAX_PRODUCTS)) {
    productSummary = products[index];
  }
  else {
    System.err.println("index is out of bounds");
    throw new IndexOutOfBoundsException();
  }
  return productSummary;
}

```

An alternative in Java would be to use one of the collection objects such as `ArrayList` that will automatically generate an exception if an attempt is made to access an array index that is out of bounds.

Example Language: Java

(Good)

```

ArrayList productArray = new ArrayList(MAX_PRODUCTS);
...
try {
  productSummary = (String) productArray.get(index);
} catch (IndexOutOfBoundsException ex) {...}

```

Example 3:

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

Example Language: C

(Bad)

```

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

```

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

Observed Examples

Reference	Description
CVE-2005-0369	large ID in packet used as array index https://www.cve.org/CVERecord?id=CVE-2005-0369
CVE-2001-1009	negative array index as argument to POP LIST command https://www.cve.org/CVERecord?id=CVE-2001-1009

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1286: Improper Validation of Syntactic Correctness of Input

Weakness ID : 1286

Structure : Simple

Abstraction : Base

Description

The product receives input that is expected to be well-formed - i.e., to comply with a certain syntax - but it does not validate or incorrectly validates that the input complies with the syntax.



Extended Description

Often, complex inputs are expected to follow a particular syntax, which is either assumed by the input itself, or declared within metadata such as headers. The syntax could be for data exchange formats, markup languages, or even programming languages. When untrusted input is not properly validated for the expected syntax, attackers could cause parsing failures, trigger unexpected errors, or expose latent vulnerabilities that might not be directly exploitable if the input had conformed to the syntax.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
ParentOf		112	Missing XML Validation	269

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

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.

Effectiveness = High

Demonstrative Examples

Example 1:

The following code loads and parses an XML file.

Example Language: Java

(Bad)

```
// Read DOM
try {
    ...
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating( false );
    ....
    c_dom = factory.newDocumentBuilder().parse( xmlFile );
} catch(Exception ex) {
    ...
}
```

The XML file is loaded without validating it against a known XML Schema or DTD.

Observed Examples

Reference	Description
CVE-2016-4029	Chain: incorrect validation of intended decimal-based IP address format (CWE-1286) enables parsing of octal or hexadecimal formats (CWE-1389), allowing bypass of an SSRF protection mechanism (CWE-918). https://www.cve.org/CVERecord?id=CVE-2016-4029
CVE-2007-5893	HTTP request with missing protocol version number leads to crash https://www.cve.org/CVERecord?id=CVE-2007-5893

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
66	SQL Injection
676	NoSQL Injection

CWE-1287: Improper Validation of Specified Type of Input

Weakness ID : 1287

Structure : Simple

Abstraction : Base

Description

The product receives input that is expected to be of a certain type, but it does not validate or incorrectly validates that the input is actually of the expected type.

Extended Description



When input does not comply with the expected type, attackers could trigger unexpected errors, cause incorrect actions to take place, or exploit latent vulnerabilities that would not be possible if the input conformed with the expected type.

This weakness can appear in type-unsafe programming languages, or in programming languages that support casting or conversion of an input to another type.



Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
PeerOf		843	Access of Resource Using Incompatible Type ('Type Confusion')	1776

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478
MemberOf		136	Type Errors	2310

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Scope	Impact	Likelihood
-------	--------	------------

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.


Effectiveness = High

Observed Examples

Reference	Description
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric. https://www.cve.org/CVERecord?id=CVE-2008-2223

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation		1400 2531

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1288: Improper Validation of Consistency within Input

Weakness ID : 1288

Structure : Simple

Abstraction : Base

Description

The product receives a complex input with multiple elements or fields that must be consistent with each other, but it does not validate or incorrectly validates that the input is actually consistent.

Extended Description

Some input data can be structured with multiple elements or fields that must be consistent with each other, e.g. a number-of-items field that is followed by the expected number of elements. When such complex inputs are inconsistent, attackers could trigger unexpected errors, cause incorrect actions to take place, or exploit latent vulnerabilities.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

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.

Effectiveness = High

Observed Examples

Reference	Description
CVE-2018-16733	product does not validate that the start block appears before the end block https://www.cve.org/CVERecord?id=CVE-2018-16733
CVE-2006-3790	size field that is inconsistent with packet size leads to buffer over-read https://www.cve.org/CVERecord?id=CVE-2006-3790
CVE-2008-4114	system crash with offset value that is inconsistent with packet size https://www.cve.org/CVERecord?id=CVE-2008-4114

MemberOf Relationships

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

Nature	Type	ID	Name		Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Notes**Maintenance**

This entry is still under development and will continue to see updates and content improvements.

CWE-1289: Improper Validation of Unsafe Equivalence in Input

Weakness ID : 1289

Structure : Simple

Abstraction : Base

Description

The product receives an input value that is used as a resource identifier or other type of reference, but it does not validate or incorrectly validates that the input is equivalent to a potentially-unsafe value.




Extended Description

Attackers can sometimes bypass input validation schemes by finding inputs that appear to be safe, but will be dangerous when processed at a lower layer or by a downstream component. For example, a simple XSS protection mechanism might try to validate that an input has no "<script>" tags using case-sensitive matching, but since HTML is case-insensitive when processed by web browsers, an attacker could inject "<ScRiPt>" and trigger XSS.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	20
PeerOf		41	Improper Resolution of Path Equivalence	86
PeerOf		178	Improper Handling of Case Sensitivity	445

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

Nature	Type	ID	Name	Page
MemberOf		1215	Data Validation Issues	2478

Applicable Platforms

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

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

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.

Effectiveness = High

Observed Examples

Reference	Description
CVE-2021-39155	Chain: A microservice integration and management platform compares the hostname in the HTTP Host header in a case-sensitive way (CWE-178, CWE-1289), allowing bypass of the authorization policy (CWE-863) using a hostname with mixed case or other variations. https://www.cve.org/CVERecord?id=CVE-2021-39155
CVE-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-2005-0269	File extension check in forum software only verifies extensions that contain all lowercase letters, which allows remote attackers to upload arbitrary files via file extensions that include uppercase letters. https://www.cve.org/CVERecord?id=CVE-2005-0269
CVE-2001-1238	Task Manager does not allow local users to end processes with uppercase letters named (1) winlogon.exe, (2) csrss.exe, (3) smss.exe and (4) services.exe via the Process tab which could allow local users to install Trojan horses that cannot be stopped. https://www.cve.org/CVERecord?id=CVE-2001-1238
CVE-2004-2214	HTTP server allows bypass of access restrictions using URIs with mixed case. https://www.cve.org/CVERecord?id=CVE-2004-2214

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1406	Comprehensive Categorization: Improper Input Validation	1400	2531

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1290: Incorrect Decoding of Security Identifiers

Weakness ID : 1290

Structure : Simple

Abstraction : Base

Description

The product implements a decoding mechanism to decode certain bus-transaction signals to security identifiers. If the decoding is implemented incorrectly, then untrusted agents can now gain unauthorized access to the asset.

Extended Description

In a System-On-Chip (SoC), various integrated circuits and hardware engines generate transactions such as to access (reads/writes) assets or perform certain actions (e.g., reset, fetch, compute, etc.). Among various types of message information, a typical transaction is comprised of source identity (to identify the originator of the transaction) and a destination identity (to route the transaction to the respective entity). Sometimes the transactions are qualified with a security identifier. The security identifier helps the destination agent decide on the set of allowed actions (e.g., access an asset for read and writes). A decoder decodes the bus transactions to map security identifiers into necessary access-controls/protectations.

A common weakness that can exist in this scenario is incorrect decoding because an untrusted agent's security identifier is decoded into a trusted agent's security identifier. Thus, an untrusted agent previously without access to an asset can now gain access to the asset.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf		1294	Insecure Security Identifier Mechanism	2150

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Bus/Interface Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Resource Consumption (Other)	
Access Control	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Security identifier decoders must be reviewed for design consistency and common weaknesses.

Phase: Implementation

Access and programming flows must be tested in pre-silicon and post-silicon testing in order to check for this weakness.

Demonstrative Examples

Example 1:

Consider a system that has four bus masters and a decoder. The decoder is supposed to decode every bus transaction and assign a corresponding security identifier. The security identifier is used to determine accesses to the assets. The bus transaction that contains the security information is Bus_transaction [15:14], and the bits 15 through 14 contain the security identifier information. The table below provides bus masters as well as their security identifiers and trust assumptions:

The assets are the AES-Key registers for encryption or decryption. The key is 128 bits implemented as a set of four 32-bit registers. The AES_KEY_ACCESS_POLICY is used to define which agents with a security identifier in the transaction can access the AES-key registers. The size of the security identifier is 4 bits (i.e., bit 3 through 0). Each bit in these 4 bits defines a security identifier. There are only 4 security identifiers that are allowed accesses to the AES-key registers. The number of the bit when set (i.e., "1") allows respective action from an agent whose identity matches the number of the bit. If clear (i.e., "0"), disallows the respective action to that corresponding agent.

The following Pseudo code outlines the process of checking the value of the Security Identifier within the AES_KEY_ACCESS_POLICY register:

Example Language: Other

(Informative)

```
If (AES_KEY_ACCESS_POLICY[Security_Identifier] == "1")
    Allow access to AES-Key registers
Else
    Deny access to AES-Key registers
```

Below is a decoder's Pseudo code that only checks for bit [14] of the bus transaction to determine what Security Identifier it must assign.

Example Language: Other

(Bad)

```
If (Bus_transaction[14] == "1")
    Security_Identifier == "1"
Else
    Security_Identifier == "0"
```

The security identifier is two bits, but the decoder code above only checks the value of one bit. Two Masters have their bit 0 set to "1" - Master_1 and Master_3. Master_1 is trusted, while Master_3 is not. The code above would therefore allow an untrusted agent, Master_3, access to the AES-Key registers in addition to intended trusted Master_1.

The decoder should check for the entire size of the security identifier in the bus-transaction signal to assign a corresponding security identifier. The following is good Pseudo code:


Example Language: Other

(Good)

```
If (Bus_transaction[15:14] == "00")
    Security_Identifier == "0"
If (Bus_transaction[15:14] == "01")
    Security_Identifier == "1"
If (Bus_transaction[15:14] == "10")
    Security_Identifier == "2"
If (Bus_transaction[15:14] == "11")
    Security_Identifier == "3"
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

CWE-1291: Public Key Re-Use for Signing both Debug and Production Code

Weakness ID : 1291

Structure : Simple

Abstraction : Base

Description

The same public key is used for signing both debug and production code.

Extended Description

A common usage of public-key cryptography is to verify the integrity and authenticity of another entity (for example a firmware binary). If a company wants to ensure that its firmware runs only on its own hardware, before the firmware runs, an encrypted hash of the firmware image will be decrypted with the public key and then verified against the now-computed hash of the firmware image. This means that the public key forms the root of trust, which necessitates that the public key itself must be protected and used properly.

During the development phase, debug firmware enables many hardware debug hooks, debug modes, and debug messages for testing. Those debug facilities provide significant, additional views about the firmware's capability and, in some cases, additional capability into the chip or SoC. If compromised, these capabilities could be exploited by an attacker to take full control of the system.



Once the product exits the manufacturing stage and enters production, it is good practice to use a different public key. Debug firmware images are known to leak. With the debug key being reused as the production key, the debug image will also work on the production image. Thus, it will open all the internal, debug capabilities to the attacker.

If a different public key is used for the production image, even if the attacker gains access to the debug firmware image, they will not be able to run it on a production machine. Thus, damage will be limited to the intellectual property leakage resulting from the debug image.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		693	Protection Mechanism Failure	1520
PeerOf		321	Use of Hard-coded Cryptographic Key	785

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	High
Integrity	Modify Memory	
Availability	Execute Unauthorized Code or Commands	
Access Control	Gain Privileges or Assume Identity	
Accountability	Varies by Context	
Authentication		
Authorization		
Non-Repudiation		
Other		

Detection Methods

Architecture or Design Review

Compare the debug key with the production key to make sure that they are not the same.

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

Compare the debug key with the production key to make sure that they are not the same.

Effectiveness = High

Potential Mitigations

Phase: Implementation

Use different keys for Production and Debug

Demonstrative Examples

Example 1:

This example illustrates the danger of using the same public key for debug and production.

Example Language: Other

(Bad)

Suppose the product design requires frugality of silicon real estate. Assume that originally the architecture allows just enough storage for two 2048-bit RSA keys in the fuse: one to be used for debug and the other for production. However, in the meantime, a business decision is taken to make the security future-proof beyond 2030, which means the architecture needs to use the NIST-recommended 3072-bit keys instead of the originally-planned 2048-bit keys. This means that, at most, one key can be fully stored in the fuses, not two. So the product design team decides to use the same public key for debug and production.


Example Language: Other

(Informative)

Increase the storage so that two different keys of the required size can be stored.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474

Nature	Type	ID	Name	V	Page
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

CWE-1292: Incorrect Conversion of Security Identifiers

Weakness ID : 1292

Structure : Simple

Abstraction : Base

Description

The product implements a conversion mechanism to map certain bus-transaction signals to security identifiers. However, if the conversion is incorrectly implemented, untrusted agents can gain unauthorized access to the asset.

Extended Description

In a System-On-Chip (SoC), various integrated circuits and hardware engines generate transactions such as to access (reads/writes) assets or perform certain actions (e.g., reset, fetch, compute, etc.). Among various types of message information, a typical transaction is comprised of source identity (to identify the originator of the transaction) and a destination identity (to route the transaction to the respective entity). Sometimes the transactions are qualified with a security identifier. This security identifier helps the destination agent decide on the set of allowed actions (e.g., access an asset for read and writes).

A typical bus connects several leader and follower agents. Some follower agents implement bus protocols differently from leader agents. A protocol conversion happens at a bridge to seamlessly connect different protocols on the bus. One example is a system that implements a leader with the Advanced High-performance Bus (AHB) protocol and a follower with the Open-Core Protocol (OCP). A bridge AHB-to-OCP is needed to translate the transaction from one form to the other.

A common weakness that can exist in this scenario is that this conversion between protocols is implemented incorrectly, whereupon an untrusted agent may gain unauthorized access to an asset.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		284	Improper Access Control	680

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf		1294	Insecure Security Identifier Mechanism	2150

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Bus/Interface Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Resource Consumption (Other)	
Access Control	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Security identifier decoders must be reviewed for design inconsistency and common weaknesses.

Phase: Implementation

Access and programming flows must be tested in pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

Consider a system that supports AHB. Let us assume we have a follower agent that only understands OCP. To connect this follower to the leader, a bridge is introduced, i.e., AHB to OCP.

The follower has assets to protect accesses from untrusted leaders, and it employs access controls based on policy, (e.g., AES-Key registers for encryption or decryption). The key is 128 bits implemented as a set of four 32-bit registers. The key registers are assets, and register AES_KEY_ACCESS_POLICY is defined to provide the necessary access controls.

The AES_KEY_ACCESS_POLICY access-policy register defines which agents with a security identifier in the transaction can access the AES-key registers. The implemented AES_KEY_ACCESS_POLICY has 4 bits where each bit when "Set" allows access to the AES-Key registers to the corresponding agent that has the security identifier. The other bits from 31 through 4 are reserved and not used.

During conversion of the AHB-to-OCP transaction, the security identifier information must be preserved and passed on to the follower correctly.

Example Language: Other

(Bad)

In AHB-to-OCP bridge, the security identifier information conversion is done incorrectly.

Because of the incorrect conversion, the security identifier information is either lost or could be modified in such a way that an untrusted leader can access the AES-Key registers.

Example Language: Other

(Good)

The conversion of the signals from one protocol (AHB) to another (OCP) must be done while preserving the security identifier correctly.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

CWE-1293: Missing Source Correlation of Multiple Independent Data

Weakness ID : 1293

Structure : Simple

Abstraction : Base

Description

The product relies on one source of data, preventing the ability to detect if an adversary has compromised a data source.

Extended Description

To operate successfully, a product sometimes has to implicitly trust the integrity of an information source. When information is implicitly signed, one can ensure that the data was not tampered in transit. This does not ensure that the information source was not compromised when responding to a request. By requesting information from multiple sources, one can check if all of the data is the same. If they are not, the system should report the information sources that respond with a different or minority value as potentially compromised. If there are not enough answers to provide a majority or plurality of responses, the system should report all of the sources as potentially compromised. As the seriousness of the impact of incorrect integrity increases, so should the number of independent information sources that would need to be queried.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	851
PeerOf		654	Reliance on a Single Factor in a Security Decision	1439

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data Gain Privileges or Assume Identity <i>An attacker that may be able to execute a single Person-in-the-Middle attack can subvert a check of an external oracle (e.g. the ACME protocol check for a file on a website), and thus inject an arbitrary reply to the single perspective request to the external oracle.</i>	

Potential Mitigations

Phase: Requirements

Design system to use a Practical Byzantine fault method, to request information from multiple sources to verify the data and report on potentially compromised information sources.

Phase: Implementation

Failure to use a Practical Byzantine fault method when requesting data. Lack of place to report potentially compromised information sources. Relying on non-independent information sources for integrity checking. Failure to report information sources that respond in the minority to incident response procedures.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1411	Comprehensive Categorization: Insufficient Verification of Data Authenticity	1400	2538

References

[REF-1125]moparisthebest. "Validation Vulnerabilities". 2015 June 5. < https://mailarchive.ietf.org/arch/msg/acme/s6Q5PdJP48LEUwgzrVuw_XPKCsM/ >.

[REF-1126]Josh Aas, Daniel McCarney and Roland Shoemaker. "Multi-Perspective Validation Improves Domain Validation Security". 2020 February 9. < <https://letsencrypt.org/2020/02/19/multi-perspective-validation.html> >.

[REF-1127]Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". 2002 November 4. < <https://dl.acm.org/doi/pdf/10.1145/571637.571640> >.2023-04-07.

CWE-1294: Insecure Security Identifier Mechanism

Weakness ID : 1294

Structure : Simple

Abstraction : Class

Description

The System-on-Chip (SoC) implements a Security Identifier mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. However, the Security Identifiers are not correctly implemented.

Extended Description

Systems-On-Chip (Integrated circuits and hardware engines) implement Security Identifiers to differentiate/identify actions originated from various agents. These actions could be 'read', 'write', 'program', 'reset', 'fetch', 'compute', etc. Security identifiers are generated and assigned to every agent in the System (SoC) that is either capable of generating an action or receiving an action from another agent. Every agent could be assigned a unique, Security Identifier based on its trust level or privileges.

A broad class of flaws can exist in the Security Identifier process, including but not limited to missing security identifiers, improper conversion of security identifiers, incorrect generation of security identifiers, etc.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	I	284	Improper Access Control	680
ParentOf	B	1302	Missing Source Identifier in Entity Transactions on a System-On-Chip (SOC)	2172

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ParentOf	B	1259	Improper Restriction of Security Token Assignment	2073
ParentOf	B	1270	Generation of Incorrect Security Tokens	2100
ParentOf	B	1290	Incorrect Decoding of Security Identifiers	2142
ParentOf	B	1292	Incorrect Conversion of Security Identifiers	2147

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Bus/Interface Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Resource Consumption (Other)	
Access Control	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Security Identifier Decoders must be reviewed for design inconsistency and common weaknesses.

Phase: Implementation

Access and programming flows must be tested in pre-silicon and post-silicon testing.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces

CAPEC-ID Attack Pattern Name

681 Exploitation of Improperly Controlled Hardware Security Identifiers

CWE-1295: Debug Messages Revealing Unnecessary Information**Weakness ID :** 1295**Structure :** Simple**Abstraction :** Base**Description**

The product fails to adequately prevent the revealing of unnecessary and potentially sensitive system information within debugging messages.



Extended Description

Debug messages are messages that help troubleshoot an issue by revealing the internal state of the system. For example, debug data in design can be exposed through internal memory array dumps or boot logs through interfaces like UART via TAP commands, scan chain, etc. Thus, the more information contained in a debug message, the easier it is to debug. However, there is also the risk of revealing information that could help an attacker either decipher a vulnerability, and/or gain a better understanding of the system. Thus, this extra information could lower the "security by obscurity" factor. While "security by obscurity" alone is insufficient, it can help as a part of "Defense-in-depth".

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		200	Exposure of Sensitive Information to an Unauthorized Actor	504
PeerOf		209	Generation of Error Message Containing Sensitive Information	533

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	Medium
Integrity	Bypass Protection Mechanism	
Availability	Gain Privileges or Assume Identity	
Access Control	Varies by Context	
Accountability		
Authentication		
Authorization		
Non-Repudiation		

Potential Mitigations

Phase: Implementation

Ensure that a debug message does not reveal any unnecessary information during the debug process for the intended response.

Demonstrative Examples**Example 1:**

This example here shows how an attacker can take advantage of unnecessary information in debug messages.

Example 1: Suppose in response to a Test Access Port (TAP) chaining request the debug message also reveals the current TAP hierarchy (the full topology) in addition to the success/failure message.

Example 2: In response to a password-filling request, the debug message, instead of a simple Granted/Denied response, prints an elaborate message, "The user-entered password does not match the actual password stored in <directory name>."

The result of the above examples is that the user is able to gather additional unauthorized information about the system from the debug messages.

The solution is to ensure that Debug messages do not reveal additional details.

Observed Examples

Reference	Description
CVE-2021-25476	Digital Rights Management (DRM) capability for mobile platform leaks pointer information, simplifying ASLR bypass https://www.cve.org/CVERecord?id=CVE-2021-25476
CVE-2020-24491	Processor generates debug message that contains sensitive information ("addresses of memory transactions"). https://www.cve.org/CVERecord?id=CVE-2020-24491
CVE-2017-18326	modem debug messages include cryptographic keys https://www.cve.org/CVERecord?id=CVE-2017-18326

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2474
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces

References

[REF-1112]"Android Security Bulletin - December 2018". < <https://source.android.com/security/bulletin/2018-12-01.html> >.2023-04-07.

CWE-1296: Incorrect Chaining or Granularity of Debug Components

Weakness ID : 1296
Structure : Simple
Abstraction : Base

Description

The product's debug components contain incorrect chaining or granularity of debug components.

Extended Description

For debugging and troubleshooting a chip, several hardware design elements are often implemented, including:

- Various Test Access Ports (TAPs) allow boundary scan commands to be executed.
- For scanning the internal components of a chip, there are scan cells that allow the chip to be used as a "stimulus and response" mechanism.
- Chipmakers might create custom methods to observe the internal components of their chips by placing various tracing hubs within their chip and creating hierarchical or interconnected structures among those hubs.

Logic errors during design or synthesis could misconfigure the interconnection of the debug components, which could allow unintended access permissions.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	284	Improper Access Control	680

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Gain Privileges or Assume Identity	Medium
Integrity	Bypass Protection Mechanism	
Access Control	Execute Unauthorized Code or Commands	
Authentication	Modify Memory	
Authorization	Modify Files or Directories	
Availability	<i>Depending on the access to debug component(s) erroneously granted, an attacker could use the debug component to gain additional understanding about the system to further an attack and/or execute other commands. This could compromise any security property, including the ones listed above.</i>	
Accountability		

Detection Methods

Architecture or Design Review

Appropriate Post-Si tests should be carried out at various authorization levels to ensure that debug components are properly chained and accessible only to users with appropriate credentials.

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

Appropriate Post-Si tests should be carried out at various authorization levels to ensure that debug components are properly chained and accessible only to users with appropriate credentials.

Effectiveness = High

Potential Mitigations

Phase: Implementation

Ensure that debug components are properly chained and their granularity is maintained at different authentication levels.

Demonstrative Examples

Example 1:

The following example shows how an attacker can take advantage of incorrect chaining or missing granularity of debug components.

In a System-on-Chip (SoC), the user might be able to access the SoC-level TAP with a certain level of authorization. However, this access should not also grant access to all of the internal TAPs (e.g., Core). Separately, if any of the internal TAPs is also stitched to the TAP chain when it should not be because of a logic error, then an attacker can access the internal TAPs as well and execute commands there.


As a related example, suppose there is a hierarchy of TAPs (TAP_A is connected to TAP_B and TAP_C, then TAP_B is connected to TAP_D and TAP_E, then TAP_C is connected to TAP_F and TAP_G, etc.). Architecture mandates that the user have one set of credentials for just accessing TAP_A, another set of credentials for accessing TAP_B and TAP_C, etc. However, if, during implementation, the designer mistakenly implements a daisy-chained TAP where all the TAPs are connected in a single TAP chain without the hierarchical structure, the correct granularity of debug components is not implemented and the attacker can gain unauthorized access.

Observed Examples

Reference	Description
CVE-2017-18347	Incorrect access control in RDP Level 1 on STMicroelectronics STM32F0 series devices allows physically present attackers to extract the device's protected firmware via a special sequence of Serial Wire Debug (SWD) commands because there is a race condition between full initialization of the SWD interface and the setup of flash protection. https://www.cve.org/CVERecord?id=CVE-2017-18347
CVE-2020-1791	There is an improper authorization vulnerability in several smartphones. The system has a logic-judging error, and, under certain scenarios, a successful exploit could allow the attacker to switch to third desktop after a series of operations in ADB mode. (Vulnerability ID: HWPSIRT-2019-10114). https://www.cve.org/CVERecord?id=CVE-2020-1791

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474

Nature	Type	ID	Name	V	Page
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces
702	Exploiting Incorrect Chaining or Granularity of Hardware Debug Components

CWE-1297: Unprotected Confidential Information on Device is Accessible by OSAT Vendors

Weakness ID : 1297

Structure : Simple

Abstraction : Base

Description

The product does not adequately protect confidential information on the device from being accessed by Outsourced Semiconductor Assembly and Test (OSAT) vendors.

Extended Description

In contrast to complete vertical integration of architecting, designing, manufacturing, assembling, and testing chips all within a single organization, an organization can choose to simply architect and design a chip before outsourcing the rest of the process to OSAT entities (e.g., external foundries and test houses). In the latter example, the device enters an OSAT facility in a much more vulnerable pre-production stage where many debug and test modes are accessible. Therefore, the chipmaker must place a certain level of trust with the OSAT. To counter this, the chipmaker often requires the OSAT partner to enter into restrictive non-disclosure agreements (NDAs). Nonetheless, OSAT vendors likely have many customers, which increases the risk of accidental sharing of information. There may also be a security vulnerability in the information technology (IT) system of the OSAT facility. Alternatively, a malicious insider at the OSAT facility may carry out an insider attack. Considering these factors, it behooves the chipmaker to minimize any confidential information in the device that may be accessible to the OSAT vendor.

Logic errors during design or synthesis could misconfigure the interconnection of the debug components, which could provide improper authorization to sensitive information.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		285	Improper Authorization	684

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood	
Confidentiality	Gain Privileges or Assume Identity	Medium	
Integrity	Bypass Protection Mechanism		
Access Control	Execute Unauthorized Code or Commands		
Authentication	Modify Memory		
Authorization	Modify Files or Directories		
Availability	<i>The impact depends on the confidential information itself and who is inadvertently granted access. For example, if the confidential information is a key that can unlock all the parts of a generation, the impact could be severe.</i>		
Accountability			
Non-Repudiation			

Detection Methods

Architecture or Design Review

Appropriate Post-Si tests should be carried out to ensure that residual confidential information is not left on parts leaving one facility for another facility.

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

Appropriate Post-Si tests should be carried out to ensure that residual confidential information is not left on parts leaving one facility for another facility.

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Ensure that when an OSAT vendor is allowed to access test interfaces necessary for preproduction and returned parts, the vendor only pulls the minimal information necessary. Also, architect the product in such a way that, when an "unlock device" request comes, it only unlocks that specific part and not all the parts for that product line. Ensure that the product's non-volatile memory (NVM) is scrubbed of all confidential information and secrets before handing it over to an OSAT. Arrange to secure all communication between an OSAT facility and the chipmaker.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

The following example shows how an attacker can take advantage of a piece of confidential information that has not been protected from the OSAT.

Suppose the preproduction device contains NVM (a storage medium that by definition/design can retain its data without power), and this NVM contains a key that can unlock all the parts for that generation. An OSAT facility accidentally leaks the key.

Compromising a key that can unlock all the parts of a generation can be devastating to a chipmaker.

The likelihood of such a compromise can be reduced by ensuring all memories on the preproduction device are properly scrubbed.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2469
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

This entry might be subject to CWE Scope Exclusion SCOPE.SITUATIONS (Focus on situations in which weaknesses may appear); SCOPE.HUMANPROC (Human/organizational process; and/or SCOPE.CUSTREL (Not customer-relevant).

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1113]Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran and Ozgur Sinanoglu. "Provably-Secure Logic Locking: From Theory To Practice". < <https://dl.acm.org/doi/10.1145/3133956.3133985> >.2023-04-07.

[REF-1114]Muhammad Yasin, Jeyavijayan (JV) Rajendran and Ozgur Sinanoglu. "Trustworthy Hardware Design: Combinational Logic Locking Techniques". < <https://link.springer.com/book/10.1007/978-3-030-15334-2> >.2023-04-07.

CWE-1298: Hardware Logic Contains Race Conditions

Weakness ID : 1298

Structure : Simple

Abstraction : Base

Description

A race condition in the hardware logic results in undermining security guarantees of the system.

Extended Description


A race condition in logic circuits typically occurs when a logic gate gets inputs from signals that have traversed different paths while originating from the same source. Such inputs to the gate can change at slightly different times in response to a change in the source signal. This results in a timing error or a glitch (temporary or permanent) that causes the output to change to an unwanted state before settling back to the desired state. If such timing errors occur in access control logic or finite state machines that are implemented in security sensitive flows, an attacker might exploit them to circumvent existing protections.

Relationships

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

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

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

Nature	Type	ID	Name	Page
ChildOf		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity Alter Execution Logic	

Potential Mitigations

Phase: Architecture and Design

Adopting design practices that encourage designers to recognize and eliminate race conditions, such as Karnaugh maps, could result in the decrease in occurrences of race conditions.

Phase: Implementation

Logic redundancy can be implemented along security critical paths to prevent race conditions. To avoid metastability, it is a good practice in general to default to a secure state in which access is not given to untrusted agents.

Demonstrative Examples

Example 1:

The code below shows a 2x1 multiplexor using logic gates. Though the code shown below results in the minimum gate solution, it is disjoint and causes glitches.

Example Language: Verilog

(Bad)

```
// 2x1 Multiplexor using logic-gates
module glitchEx(
    input wire in0, in1, sel,
    output wire z
);
    wire not_sel;
    wire and_out1, and_out2;
    assign not_sel = ~sel;
    assign and_out1 = not_sel & in0;
    assign and_out2 = sel & in1;
    // Buggy line of code:
    assign z = and_out1 | and_out2; // glitch in signal z
endmodule
```

The buggy line of code, commented above, results in signal 'z' periodically changing to an unwanted state. Thus, any logic that references signal 'z' may access it at a time when it is in this unwanted state. This line should be replaced with the line shown below in the Good Code Snippet which results in signal 'z' remaining in a continuous, known, state. Reference for the above code, along with waveforms for simulation can be found in the references below.

*Example Language: Verilog**(Good)*

```
assign z <= and_out1 or and_out2 or (in0 and in1);
```

This line of code removes the glitch in signal z.

Example 2:

The example code is taken from the DMA (Direct Memory Access) module of the buggy OpenPiton SoC of HACK@DAC'21. The DMA contains a finite-state machine (FSM) for accessing the permissions using the physical memory protection (PMP) unit.

PMP provides secure regions of physical memory against unauthorized access. It allows an operating system or a hypervisor to define a series of physical memory regions and then set permissions for those regions, such as read, write, and execute permissions. When a user tries to access a protected memory area (e.g., through DMA), PMP checks the access of a PMP address (e.g., `pmpaddr_i`) against its configuration (`pmpcfg_i`). If the access violates the defined permissions (e.g., `CTRL_ABORT`), the PMP can trigger a fault or an interrupt. This access check is implemented in the `pmp` parametrized module in the below code snippet. The below code assumes that the state of the `pmpaddr_i` and `pmpcfg_i` signals will not change during the different DMA states (i.e., `CTRL_IDLE` to `CTRL_DONE`) while processing a DMA request (via `dma_ctrl_reg`). The DMA state machine is implemented using a case statement (not shown in the code snippet).

*Example Language: Verilog**(Bad)*

```
module dma # (...)(...);
...
input [7:0] [16-1:0] pmpcfg_i;
input logic [16-1:0][53:0] pmpaddr_i;
...
//// Save the input command
always @ (posedge clk_i or negedge rst_ni)
begin: save_inputs
    if (!rst_ni)
        begin
            ...
        end
    else
        begin
            if (dma_ctrl_reg == CTRL_IDLE || dma_ctrl_reg == CTRL_DONE)
                begin
                    ...
                end
            end
        end
end // save_inputs
...
// Load/store PMP check
pmp #(
    .XLEN ( 64 ),
    .PMP_LEN ( 54 ),
    .NR_ENTRIES ( 16 )
) i_pmp_data (
    .addr_i ( pmp_addr_reg ),
    .priv_lvl_i ( riscv::PRIV_LVL_U ),
    .access_type_i ( pmp_access_type_reg ),
    // Configuration
    .conf_addr_i ( pmpaddr_i ),
    .conf_i ( pmpcfg_i ),
    .allow_o ( pmp_data_allow )
);
endmodule
```

However, the above code [REF-1394] allows the values of `pmpaddr_i` and `pmpcfg_i` to be changed through DMA's input ports. This causes a race condition and will enable attackers to access sensitive addresses that the configuration is not associated with.

Attackers can initialize the DMA access process (`CTRL_IDLE`) using `pmpcfg_i` for a non-privileged PMP address (`pmpaddr_i`). Then during the loading state (`CTRL_LOAD`), attackers can replace the non-privileged address in `pmpaddr_i` with a privileged address without the requisite authorized access configuration.

To fix this issue (see [REF-1395]), the value of the `pmpaddr_i` and `pmpcfg_i` signals should be stored in local registers (`pmpaddr_reg` and `pmpcfg_reg` at the start of the DMA access process and the pmp module should reference those registers instead of the signals directly. The values of the registers can only be updated at the start (`CTRL_IDLE`) or the end (`CTRL_DONE`) of the DMA access process, which prevents attackers from changing the PMP address in the middle of the DMA access process.

Example Language: Verilog

(Good)

```
module dma # (...)(...);
...
  input [7:0] [16-1:0] pmpcfg_i;
  input logic [16-1:0][53:0] pmpaddr_i;
  ...
  reg [7:0] [16-1:0] pmpcfg_reg;
  reg [16-1:0][53:0] pmpaddr_reg;
  ...
  /// Save the input command
  always @ (posedge clk_i or negedge rst_ni)
    begin: save_inputs
      if (!rst_ni)
        begin
          ...
          pmpaddr_reg <= 'b0 ;
          pmpcfg_reg <= 'b0 ;
          end
        else
          begin
            if (dma_ctrl_reg == CTRL_IDLE || dma_ctrl_reg == CTRL_DONE)
              begin
                ...
                pmpaddr_reg <= pmpaddr_i;
                pmpcfg_reg <= pmpcfg_i;
                end
              end
            end // save_inputs
          ...
          // Load/store PMP check
          pmp #(
            .XLEN ( 64 ),
            .PMP_LEN ( 54 ),
            .NR_ENTRIES ( 16 )
          ) i_pmp_data (
            .addr_i ( pmp_addr_reg ),
            .priv_lvl_i ( riscv::PRIV_LVL_U ), // we intend to apply filter on
            // DMA always, so choose the least privilege .access_type_i ( pmp_access_type_reg ),
            // Configuration
            .conf_addr_i ( pmpaddr_reg ),
            .conf_i ( pmpcfg_reg ),
            .allow_o ( pmp_data_allow )
          );
        endmodule
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2471
MemberOf	C	1401	Comprehensive Categorization: Concurrency	1400	2526

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions

References

[REF-1115]Meher Krishna Patel. "FPGA designs with Verilog (section 7.4 Glitches)". < <https://verilogguide.readthedocs.io/en/latest/verilog/fsm.html> >.

[REF-1116]Clifford E. Cummings. "Non-Blocking Assignments in Verilog Synthesis, Coding Styles that Kill!". 2000. < http://www.sunburst-design.com/papers/CummingsSNUG2000SJ_NBA.pdf >.

[REF-1394]"dma.sv". 2021. < <https://github.com/HACK-EVENT/hackatdac21/blob/main/piton/design/chip/tile/ariane/src/dma/dma.sv> >.2024-02-09.

[REF-1395]"Fix for dma.sv". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/cwe_1298_in_dma/piton/design/chip/tile/ariane/src/dma/dma.sv >.2024-02-09.

CWE-1299: Missing Protection Mechanism for Alternate Hardware Interface

Weakness ID : 1299

Structure : Simple

Abstraction : Base

Description

The lack of protections on alternate paths to access control-protected assets (such as unprotected shadow registers and other external facing unguarded interfaces) allows an attacker to bypass existing protections to the asset that are only performed against the primary path.

Extended Description

An asset inside a chip might have access-control protections through one interface. However, if all paths to the asset are not protected, an attacker might compromise the asset through alternate paths. These alternate paths could be through shadow or mirror registers inside the IP core, or could be paths from other external-facing interfaces to the IP core or SoC.

Consider an SoC with various interfaces such as UART, SMBUS, PCIe, USB, etc. If access control is implemented for SoC internal registers only over the PCIe interface, then an attacker could still modify the SoC internal registers through alternate paths by coming through interfaces such as UART, SMBUS, USB, etc.

Alternatively, attackers might be able to bypass existing protections by exploiting unprotected, shadow registers. Shadow registers and mirror registers typically refer to registers that can be accessed from multiple addresses. Writing to or reading from the aliased/mirrored address has the same effect as writing to the address of the main register. They are typically implemented within an IP core or SoC to temporarily hold certain data. These data will later be updated to the main register, and both registers will be in synch. If the shadow registers are not access-protected, attackers could simply initiate transactions to the shadow registers and compromise system 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	288	Authentication Bypass Using an Alternate Path or Channel	700
ChildOf	B	420	Unprotected Alternate Channel	1018

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf	B	1191	On-Chip Debug and Test Interface With Improper Access Control	1980
PeerOf	B	1314	Missing Write Protection for Parametric Data Values	2187

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Microcontroller Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Technology : Bus/Interface Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Resource Consumption (Other)	
Access Control	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Alter Execution Logic	
	Bypass Protection Mechanism	
	Quality Degradation	

Potential Mitigations

Phase: Requirements

Protect assets from accesses against all potential interfaces and alternate paths.

Effectiveness = Defense in Depth

Phase: Architecture and Design

Protect assets from accesses against all potential interfaces and alternate paths.

Effectiveness = Defense in Depth

Phase: Implementation

Protect assets from accesses against all potential interfaces and alternate paths.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

Register SECURE_ME is located at address 0xF00. A mirror of this register called COPY_OF_SECURE_ME is at location 0x800F00. The register SECURE_ME is protected from malicious agents and only allows access to select, while COPY_OF_SECURE_ME is not.

Access control is implemented using an allowlist (as indicated by acl_oh_allowlist). The identity of the initiator of the transaction is indicated by the one hot input, incoming_id. This is checked against the acl_oh_allowlist (which contains a list of initiators that are allowed to access the asset).

Though this example is shown in Verilog, it will apply to VHDL as well.

Example Language: Verilog (Informative)

```
module foo_bar(data_out, data_in, incoming_id, address, clk, rst_n);
output [31:0] data_out;
input [31:0] data_in, incoming_id, address;
input clk, rst_n;
wire write_auth, addr_auth;
reg [31:0] data_out, acl_oh_allowlist, q;
assign write_auth = | (incoming_id & acl_oh_allowlist) ? 1 : 0;
always @*
    acl_oh_allowlist <= 32'h8312;
assign addr_auth = (address == 32'hF00) ? 1 : 0;
always @ (posedge clk or negedge rst_n)
    if (!rst_n)
        begin
            q <= 32'h0;
            data_out <= 32'h0;
        end
    else
        begin
            q <= (addr_auth & write_auth) ? data_in : q;
            data_out <= q;
        end
    end
endmodule
```

Example Language: Verilog (Bad)

```
assign addr_auth = (address == 32'hF00) ? 1 : 0;
```

The bugged line of code is repeated in the Bad example above. The weakness arises from the fact that the SECURE_ME register can be modified by writing to the shadow register COPY_OF_SECURE_ME. The address of COPY_OF_SECURE_ME should also be included in the check. That buggy line of code should instead be replaced as shown in the Good Code Snippet below.

Example Language: Verilog (Good)

```
assign addr_auth = (address == 32'hF00 || address == 32'h800F00) ? 1 : 0;
```

Observed Examples

Reference	Description
CVE-2022-38399	Missing protection mechanism on serial connection allows for arbitrary OS command execution. https://www.cve.org/CVERecord?id=CVE-2022-38399
CVE-2020-9285	Mini-PCI Express slot does not restrict direct memory access. https://www.cve.org/CVERecord?id=CVE-2020-9285
CVE-2020-8004	When the internal flash is protected by blocking access on the Data Bus (DBUS), it can still be indirectly accessed through the Instruction Bus (IBUS). https://www.cve.org/CVERecord?id=CVE-2020-8004

Reference	Description
CVE-2017-18293	When GPIO is protected by blocking access to corresponding GPIO resource registers, protection can be bypassed by writing to the corresponding banked GPIO registers instead. https://www.cve.org/CVERecord?id=CVE-2017-18293
CVE-2020-15483	monitor device allows access to physical UART debug port without authentication https://www.cve.org/CVERecord?id=CVE-2020-15483

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
457	USB Memory Attacks
554	Functionality Bypass

CWE-1300: Improper Protection of Physical Side Channels

Weakness ID : 1300

Structure : Simple

Abstraction : Base

Description

The device does not contain sufficient protection mechanisms to prevent physical side channels from exposing sensitive information due to patterns in physically observable phenomena such as variations in power consumption, electromagnetic emissions (EME), or acoustic emissions.

Extended Description

An adversary could monitor and measure physical phenomena to detect patterns and make inferences, even if it is not possible to extract the information in the digital domain.


Physical side channels have been well-studied for decades in the context of breaking implementations of cryptographic algorithms or other attacks against security features. These side channels may be easily observed by an adversary with physical access to the device, or using a tool that is in close proximity. If the adversary can monitor hardware operation and correlate its data processing with power, EME, and acoustic measurements, the adversary might be able to recover of secret keys and 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	203	Observable Discrepancy	518

Nature	Type	ID	Name	Page
ParentOf		1255	Comparison Logic is Vulnerable to Power Side-Channel Attacks	2062

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ChildOf		203	Observable Discrepancy	518

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Detection Methods

Manual Analysis

Perform a set of leakage detection tests such as the procedure outlined in the Test Vector Leakage Assessment (TVLA) test requirements for AES [REF-1230]. TVLA is the basis for the ISO standard 17825 [REF-1229]. A separate methodology is provided by [REF-1228]. Note that sole reliance on this method might not yield expected results [REF-1239] [REF-1240].

Effectiveness = Moderate

Manual Analysis

Post-silicon, perform full side-channel attacks (penetration testing) covering as many known leakage models as possible against test code.

Effectiveness = Moderate

Manual Analysis

Pre-silicon - while the aforementioned TVLA methods can be performed post-silicon, models of device power consumption or other physical emanations can be built from information present at various stages of the hardware design process before fabrication. TVLA or known side-channel attacks can be applied to these simulated traces and countermeasures applied before tape-out. Academic research in this field includes [REF-1231] [REF-1232] [REF-1233].

Effectiveness = Moderate

Potential Mitigations

Phase: Architecture and Design

Apply blinding or masking techniques to implementations of cryptographic algorithms.

Phase: Implementation

Add shielding or tamper-resistant protections to the device to increase the difficulty of obtaining measurements of the side-channel.

Demonstrative Examples

Example 1:

Consider a device that checks a passcode to unlock the screen.

Example Language:

(Bad)

As each character of the PIN number is entered, a correct character exhibits one current pulse shape while an incorrect character exhibits a different current pulse shape.

PIN numbers used to unlock a cell phone should not exhibit any characteristics about themselves. This creates a side channel. An attacker could monitor the pulses using an oscilloscope or other method. Once the first character is correctly guessed (based on the oscilloscope readings), they can then move to the next character, which is much more efficient than the brute force method of guessing every possible sequence of characters.

Example Language:

(Good)

Rather than comparing each character to the correct PIN value as it is entered, the device could accumulate the PIN in a register, and do the comparison all at once at the end. Alternatively, the components for the comparison could be modified so that the current pulse shape is the same regardless of the correctness of the entered character.

Example 2:

Consider the device vulnerability CVE-2021-3011, which affects certain microcontrollers [REF-1221]. The Google Titan Security Key is used for two-factor authentication using cryptographic algorithms. The device uses an internal secret key for this purpose and exchanges information based on this key for the authentication. If this internal secret key and the encryption algorithm were known to an adversary, the key function could be duplicated, allowing the adversary to masquerade as the legitimate user.

Example Language:

(Bad)

The local method of extracting the secret key consists of plugging the key into a USB port and using electromagnetic (EM) sniffing tools and computers.

Example Language:

(Good)

Several solutions could have been considered by the manufacturer. For example, the manufacturer could shield the circuitry in the key or add randomized delays, indirect calculations with random values involved, or randomly ordered calculations to make extraction much more difficult or a combination of these techniques.

Example 3:

The code snippet provided here is part of the modular exponentiation module found in the HACK@DAC'21 Openpiton System-on-Chip (SoC), specifically within the RSA peripheral [REF-1368]. Modular exponentiation, denoted as " $a^b \bmod n$," is a crucial operation in the RSA public/private key encryption. In RSA encryption, where 'c' represents ciphertext, 'm' stands for a message, and 'd' corresponds to the private key, the decryption process is carried out using this modular exponentiation as follows: $m = c^d \bmod n$, where 'n' is the result of multiplying two large prime numbers.

Example Language: Verilog

(Bad)

```
...
module mod_exp
...
  `UPDATE: begin
    if (exponent_reg != 'd0) begin
      if (exponent_reg[0])
        result_reg <= result_next;
      base_reg <= base_next;
      exponent_reg <= exponent_next;
      state <= `UPDATE;
    end
  end
endmodule
```

```
...
endmodule
```

The vulnerable code shows a buggy implementation of binary exponentiation where it updates the result register (result_reg) only when the corresponding exponent bit (exponent_reg[0]) is set to 1. However, when this exponent bit is 0, the output register is not updated. It's important to note that this implementation introduces a physical power side-channel vulnerability within the RSA core. This vulnerability could expose the private exponent to a determined physical attacker. Such exposure of the private exponent could lead to a complete compromise of the private key.

To address mitigation requirements, the developer can develop the module by minimizing dependency on conditions, particularly those reliant on secret keys. In situations where branching is unavoidable, developers can implement masking mechanisms to obfuscate the power consumption patterns exhibited by the module (see good code example). Additionally, certain algorithms, such as the Karatsuba algorithm, can be implemented as illustrative examples of side-channel resistant algorithms, as they necessitate only a limited number of branch conditions [REF-1369].

Example Language: Verilog (Good)

```
...
module mod_exp
...
  `UPDATE: begin
    if (exponent_reg != 'd0) begin
      if (exponent_reg[0]) begin
        result_reg <= result_next;
      end else begin
        mask_reg <= result_next;
      end
      base_reg <= base_next;
      exponent_reg <= exponent_next;
      state <= `UPDATE;
    end
  end
endmodule
```

Observed Examples

Reference	Description
CVE-2022-35888	Power side-channels leak secret information from processor https://www.cve.org/CVERecord?id=CVE-2022-35888
CVE-2021-3011	electromagnetic-wave side-channel in security-related microcontrollers allows extraction of private key https://www.cve.org/CVERecord?id=CVE-2021-3011
CVE-2019-14353	Crypto hardware wallet's power consumption relates to total number of pixels illuminated, creating a side channel in the USB connection that allows attackers to determine secrets displayed such as PIN numbers and passwords https://www.cve.org/CVERecord?id=CVE-2019-14353
CVE-2020-27211	Chain: microcontroller system-on-chip contains uses a register value stored in flash to set product protection state on the memory bus but does not contain protection against fault injection (CWE-1319), which leads to an incorrect initialization of the memory bus (CWE-1419) leading the product to be in an unprotected state. https://www.cve.org/CVERecord?id=CVE-2020-27211
CVE-2013-4576	message encryption software uses certain instruction sequences that allows RSA key extraction using a chosen-ciphertext attack and acoustic cryptanalysis https://www.cve.org/CVERecord?id=CVE-2013-4576

Reference	Description
CVE-2020-28368	virtualization product allows recovery of AES keys from the guest OS using a side channel attack against a power/energy monitoring interface. https://www.cve.org/CVERecord?id=CVE-2020-28368
CVE-2019-18673	power consumption varies based on number of pixels being illuminated in a display, allowing reading of secrets such as the PIN by using the USB interface to measure power consumption https://www.cve.org/CVERecord?id=CVE-2019-18673

Functional Areas

- Power

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	V	1343	Weaknesses in the 2021 CWE Most Important Hardware Weaknesses List	1343	2592
MemberOf	C	1388	Physical Access Issues and Concerns	1194	2518
MemberOf	C	1417	Comprehensive Categorization: Sensitive Information Exposure	1400	2548

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
189	Black Box Reverse Engineering
699	Eavesdropping on a Monitor

References

[REF-1117]Paul Kocher, Joshua Jaffe and Benjamin Jun. "Introduction to differential power analysis and related attacks". 1998. < <https://www.rambus.com/wp-content/uploads/2015/08/DPATechInfo.pdf> >.

[REF-1118]Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao and Pankaj Rohatgi. "The EM Side-Channel(s)". 2007 August 4. < https://link.springer.com/content/pdf/10.1007/3-540-36400-5_4.pdf >.2023-04-07.

[REF-1119]Daniel Genkin, Adi Shamir and Eran Tromer. "RSA key extraction via low-bandwidth acoustic cryptanalysis". 2014 June 3. < <https://www.iacr.org/archive/crypto2014/86160149/86160149.pdf> >.

[REF-1120]Colin O'Flynn. "Power Analysis for Cheapskates". 2013 January 4. < <https://media.blackhat.com/eu-13/briefings/OFlynn/bh-eu-13-for-cheapstakes-oflynn-wp.pdf> >.

[REF-1055]Peter Gutmann. "Data Remanence in Semiconductor Devices". 10th USENIX Security Symposium. 2001 August. < https://www.usenix.org/legacy/events/sec01/full_papers/gutmann/gutmann.pdf >.

[REF-1218]Graham Cluley. "This Black Box Can Brute Force Crack iPhone PIN Passcodes". The Mac Security Blog. 2015 March 6. < <https://www.intego.com/mac-security-blog/iphone-pin-passcode/> >.

[REF-1221]Victor Lomne and Thomas Roche. "A Side Journey to Titan". 2021 January 7. < https://web.archive.org/web/20210107182441/https://ninjalab.io/wp-content/uploads/2021/01/a_side_journey_to_titan.pdf >.2023-04-07.

[REF-1228]Gilbert Goodwill, Benjamin Jun, Josh Jaffe and Pankaj Rohatgi. "A testing methodology for side-channel resistance validation". 2011. < https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf >.

[REF-1229]ISO/IEC. "ISO/IEC 17825:2016: Testing methods for the mitigation of non-invasive attack classes against cryptographic modules". 2016. < <https://www.iso.org/standard/60612.html> >.

[REF-1230]Cryptography Research Inc.. "Test Vector Leakage Assessment (TVLA) Derived Test Requirements (DTR) with AES". 2015 August. < <https://www.rambus.com/wp-content/uploads/2015/08/TVLA-DTR-with-AES.pdf> >.

[REF-1231]Danilo Šija#ci', Josep Balasch, Bohan Yang, Santosh Ghosh and Ingrid Verbauwhede. "Towards efficient and automated side-channel evaluations at design time". Journal of Cryptographic Engineering, 10(4). 2020. < <https://www.esat.kuleuven.be/cosic/publications/article-3204.pdf> >.

[REF-1232]Amit Kumar, Cody Scarborough, Ali Yilmaz and Michael Orshansky. "Efficient simulation of EM side-channel attack resilience". IEEE/ACM International Conference on Computer-Aided Design (ICCAD). 2017. < <https://dl.acm.org/doi/pdf/10.5555/3199700.3199717> >.2023-04-07.

[REF-1233]Yuan Yao, Tuna Tufan, Tarun Kathuria, Baris Ege, Ulkuhan Guler and Patrick Schaumont. "Pre-silicon Architecture Correlation Analysis (PACA): Identifying and Mitigating the Source of Side-channel Leakage at Gate-level". 2021 April 1. IACR Cryptology ePrint Archive. < <https://eprint.iacr.org/2021/530.pdf> >.

[REF-1234]Elisabeth Oswald, Thomas Popp and Stefan Mangard. "Power Analysis Attacks - Revealing the Secrets of Smart Cards". 2007. < <https://link.springer.com/book/10.1007/978-0-387-38162-6> >.2023-04-07.

[REF-1235]David Oswald, Bastian Richter and Christof Paar. "Side-Channel Attacks on the Yubikey 2 One-Time Password Generator". 2013 June 4. < https://www.emsec.ruhr-uni-bochum.de/media/crypto/veroeffentlichungen/2014/02/04/paper_yubikey_sca.pdf >.

[REF-1239]François-Xavier Standaert. "How (not) to Use Welch's T-test in Side-Channel Security Evaluations". 2017 February 5. IACR Cryptology ePrint Archive. < <https://eprint.iacr.org/2017/138.pdf> >.

[REF-1240]Carolyn Whitnall and Elisabeth Oswald. "A Critical Analysis of ISO 17825 ('Testing methods for the mitigation of non-invasive attack classes against cryptographic modules')". 2019 September 0. IACR Cryptology ePrint Archive. < <https://eprint.iacr.org/2019/1013.pdf> >.

[REF-1285]Texas Instruments. "Physical Security Attacks Against Silicon Devices". 2022 January 1. < <https://www.ti.com/lit/an/swra739/swra739.pdf?ts=1644234570420> >.

[REF-1286]Lennert Wouters, Benedikt Gierlichs and Bart Preneel. "On The Susceptibility of Texas Instruments SimpleLink Platform Microcontrollers to Non-Invasive Physical Attacks". 2022 March 4. < <https://eprint.iacr.org/2022/328.pdf> >.

[REF-1368]"mod_exp.v". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/rsa/mod_exp.v#L46:L47 >.2023-07-15.

[REF-1369]"Fix CWE-1300". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/37e42f724c14b8e4cc8f6e13462c12a492778219/piton/design/chip/tile/ariane/src/rsa/mod_exp.v#L47:L51 >.2023-09-29.

CWE-1301: Insufficient or Incomplete Data Removal within Hardware Component

Weakness ID : 1301

Structure : Simple

Abstraction : Base

Description

The product's data removal process does not completely delete all data and potentially sensitive information within hardware components.

Extended Description



Physical properties of hardware devices, such as remanence of magnetic media, residual charge of ROMs/RAMs, or screen burn-in may still retain sensitive data after a data removal process has taken place and power is removed.

Recovering data after erasure or overwriting is possible due to a phenomenon called data remanence. For example, if the same value is written repeatedly to a memory location, the corresponding memory cells can become physically altered to a degree such that even after the original data is erased that data can still be recovered through physical characterization of the memory cells.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		226	Sensitive Information in Resource Not Removed Before Reuse	562
ParentOf		1330	Remanent Data Readable after Memory Erase	2222

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
ParentOf		1330	Remanent Data Readable after Memory Erase	2222

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Apply blinding or masking techniques to implementations of cryptographic algorithms.

Phase: Implementation

Alter the method of erasure, add protection of media, or destroy the media to protect the data.

Observed Examples

Reference	Description
CVE-2019-8575	Firmware Data Deletion Vulnerability in which a base station factory reset might not delete all user information. The impact of this enables a new owner of a used device that has been "factory-default reset" with a vulnerable

Reference	Description
	firmware version can still retrieve, at least, the previous owner's wireless network name, and the previous owner's wireless security (such as WPA2) key. This issue was addressed with improved, data deletion. https://www.cve.org/CVERecord?id=CVE-2019-8575

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2474
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data

References

[REF-1117]Paul Kocher, Joshua Jaffe and Benjamin Jun. "Introduction to differential power analysis and related attacks". 1998. < <https://www.rambus.com/wp-content/uploads/2015/08/DPATechInfo.pdf> >.

[REF-1118]Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao and Pankaj Rohatgi. "The EM Side-Channel(s)". 2007 August 4. < https://link.springer.com/content/pdf/10.1007/3-540-36400-5_4.pdf >.2023-04-07.

[REF-1119]Daniel Genkin, Adi Shamir and Eran Tromer. "RSA key extraction via low-bandwidth acoustic cryptanalysis". 2014 June 3. < <https://www.iacr.org/archive/crypto2014/86160149/86160149.pdf> >.

[REF-1120]Colin O'Flynn. "Power Analysis for Cheapskates". 2013 January 4. < <https://media.blackhat.com/eu-13/briefings/O'Flynn/bh-eu-13-for-cheapstakes-oflynn-wp.pdf> >.

[REF-1055]Peter Gutmann. "Data Remanence in Semiconductor Devices". 10th USENIX Security Symposium. 2001 August. < https://www.usenix.org/legacy/events/sec01/full_papers/gutmann/gutmann.pdf >.

CWE-1302: Missing Source Identifier in Entity Transactions on a System-On-Chip (SOC)

Weakness ID : 1302

Structure : Simple

Abstraction : Base

Description

The product implements a security identifier mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. A transaction is sent without a security identifier.

Extended Description

In a System-On-Chip (SoC), various integrated circuits and hardware engines generate transactions such as to access (reads/writes) assets or perform certain actions (e.g., reset, fetch, compute). A typical transaction is comprised of source identity (to identify the originator of the transaction) and a destination identity (to route the transaction to the respective entity) in addition to much more information in the message. Sometimes the transactions are qualified with a Security Identifier. This Security Identifier helps the destination agent decide on the set of allowed or disallowed actions.

A weakness that can exist in such transaction schemes is that the source agent does not consistently include the necessary Security Identifier with the transaction. If the Security Identifier is missing, the destination agent might drop the message (resulting in an inadvertent Denial-of-Service (DoS)) or take inappropriate action by default in its attempt to execute the transaction, resulting in privilege escalation or provision of unintended 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf		1294	Insecure Security Identifier Mechanism	2150

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	High
Integrity	Read Memory	
Availability	DoS: Crash, Exit, or Restart	
Access Control	Bypass Protection Mechanism Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Transaction details must be reviewed for design inconsistency and common weaknesses.

Phase: Implementation

Security identifier definition and programming flow must be tested in pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

Consider a system with a register for storing AES key for encryption or decryption. The key is of 128 bits implemented as a set of four 32-bit registers. The key registers are assets, and the register AES_KEY_ACCESS_POLICY is defined to provide the necessary access controls.

The access-policy register defines which agents with a security identifier in the transaction can access the AES-key registers. Each bit in this 32-bit register defines a security identifier. There could be a maximum of 32 security identifiers that are allowed accesses to the AES-key registers. The number of the bit when set (i.e., "1") allows for a respective action from an agent whose identity matches the number of the bit; if set to "0" (i.e., Clear), it disallows the respective action to that corresponding agent.

Example Language:

(Bad)

The originator sends a transaction with no security identifier, i.e., meaning the value is "0" or NULL. The AES-Key-access register does not allow the necessary action and drops the transaction because the originator failed to include the required security identifier.

Example Language:

(Good)

The originator should send a transaction with Security Identifier "2" which will allow access to the AES-Key-access register and allow encryption and decryption operations.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces
681	Exploitation of Improperly Controlled Hardware Security Identifiers

CWE-1303: Non-Transparent Sharing of Microarchitectural Resources

Weakness ID : 1303

Structure : Simple

Abstraction : Base

Description

Hardware structures shared across execution contexts (e.g., caches and branch predictors) can violate the expected architecture isolation between contexts.

Extended Description

Modern processors use techniques such as out-of-order execution, speculation, prefetching, data forwarding, and caching to increase performance. Details about the implementation of these techniques are hidden from the programmer's view. This is problematic when the hardware implementation of these techniques results in resources being shared across supposedly isolated contexts. Contention for shared resources between different contexts opens covert channels that allow malicious programs executing in one context to recover information from another context.

Some examples of shared micro-architectural resources that have been used to leak information between contexts are caches, branch prediction logic, and load or store buffers. Speculative

and out-of-order execution provides an attacker with increased control over which data is leaked through the covert channel.

If the extent of resource sharing between contexts in the design microarchitecture is undocumented, it is extremely difficult to ensure system assets are protected against disclosure.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	203	Observable Discrepancy	518
ChildOf	B	1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)	1976

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Memory <i>Microarchitectural side-channels have been used to leak specific information such as cryptographic keys, and Address Space Layout Randomization (ASLR) offsets as well as arbitrary memory.</i>	

Potential Mitigations

Phase: Architecture and Design

Microarchitectural covert channels can be addressed using a mixture of hardware and software mitigation techniques. These include partitioned caches, new barrier and flush instructions, and disabling high resolution performance counters and timers.

Phase: Requirements

Microarchitectural covert channels can be addressed using a mixture of hardware and software mitigation techniques. These include partitioned caches, new barrier and flush instructions, and disabling high resolution performance counters and timers.

Demonstrative Examples

Example 1:

On some processors the hardware indirect branch predictor is shared between execution contexts, for example, between sibling SMT threads. When SMT thread A executes an indirect branch to a target address X, this target may be temporarily stored by the indirect branch predictor. A subsequent indirect branch mis-prediction for SMT thread B could speculatively execute instructions at X (or at a location in B's address space that partially aliases X). Even though the processor rolls back the architectural effects of the mis-predicted indirect branch, the memory accesses alter data cache state, which is not rolled back after the indirect branch is resolved.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1364	ICS Communications: Zone Boundary Failures	1358	2501
MemberOf	C	1366	ICS Communications: Frail Security in Protocols	1358	2503
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles	1400	2549

Notes

Maintenance

As of CWE 4.9, members of the CWE Hardware SIG are closely analyzing this entry and others to improve CWE's coverage of transient execution weaknesses, which include issues related to Spectre, Meltdown, and other attacks. Additional investigation may include other weaknesses related to microarchitectural state. Finally, this entry's demonstrative example might not be appropriate. As a result, this entry might change significantly in CWE 4.10.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
663	Exploitation of Transient Instruction Execution

References

- [REF-1121]Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stegfan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom and Mike Hamberg. "Meltdown: Reading Kernel Memory from User Space". 2018 January 3. < <https://meltdownattack.com/meltdown.pdf> >.
- [REF-1122]Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stegfan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom and Mike Hamberg. "Spectre Attacks: Exploiting Speculative Execution". 2018 January 3. < <https://spectreattack.com/spectre.pdf> >.
- [REF-1123]Dmitry Evtushkin, Dmitry Ponomarev and Nael Abu-Ghazaleh. "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR". 2016 October 9. < <https://ieeexplore.ieee.org/abstract/document/7783743/> >.
- [REF-1124]Qian Ge, Yuval Yarom, David Cock and Gernot Heiser. "A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware". 2016 October 4. < <https://eprint.iacr.org/2016/613.pdf> >.

CWE-1304: Improperly Preserved Integrity of Hardware Configuration State During a Power Save/Restore Operation

Weakness ID : 1304

Structure : Simple

Abstraction : Base

Description

The product performs a power save/restore operation, but it does not ensure that the integrity of the configuration state is maintained and/or verified between the beginning and ending of the operation.

Extended Description

Before powering down, the Intellectual Property (IP) saves current state (S) to persistent storage such as flash or always-on memory in order to optimize the restore operation. During this process, an attacker with access to the persistent storage may alter (S) to a configuration that could potentially modify privileges, disable protections, and/or cause damage to the hardware. If the IP does not validate the configuration state stored in persistent memory, upon regaining power or becoming operational again, the IP could be compromised through the activation of an unwanted/harmful configuration.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680
PeerOf	Ⓢ	345	Insufficient Verification of Data Authenticity	851

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf	Ⓢ	1271	Uninitialized Value on Reset for Registers Holding Security Settings	2102

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	DoS: Instability DoS: Crash, Exit, or Restart DoS: Resource Consumption (Other) Gain Privileges or Assume Identity Bypass Protection Mechanism Alter Execution Logic Quality Degradation Unexpected State Reduce Maintainability Reduce Performance Reduce Reliability	High

Potential Mitigations

Phase: Architecture and Design

Inside the IP, incorporate integrity checking on the configuration state via a cryptographic hash. The hash can be protected inside the IP such as by storing it in internal registers which never lose power. Before powering down, the IP performs a hash of the configuration and saves it in these persistent registers. Upon restore, the IP performs a hash of the saved configuration and compares it with the saved hash. If they do not match, then the IP should not trust the configuration.

Phase: Integration

Outside the IP, incorporate integrity checking of the configuration state via a trusted agent. Before powering down, the trusted agent performs a hash of the configuration and saves the hash in persistent storage. Upon restore, the IP requests the trusted agent validate its current configuration. If the configuration hash is invalid, then the IP should not trust the configuration.

Phase: Integration

Outside the IP, incorporate a protected environment that prevents undetected modification of the configuration state by untrusted agents. Before powering down, a trusted agent saves the IP's configuration state in this protected location that only it is privileged to. Upon restore, the trusted agent loads the saved state into the IP.

Demonstrative Examples

Example 1:

The following pseudo code demonstrates the power save/restore workflow which may lead to weakness through a lack of validation of the config state after restore.

Example Language: C

(Bad)

```
void save_config_state()
{
    void* cfg;
    cfg = get_config_state();
    save_config_state(cfg);
    go_to_sleep();
}
void restore_config_state()
{
    void* cfg;
    cfg = get_config_file();
    load_config_file(cfg);
}
```

The following pseudo-code is the proper workflow for the integrity checking mitigation:

Example Language: C

(Good)

```
void save_config_state()
{
    void* cfg;
    void* sha;
    cfg = get_config_state();
    save_config_state(cfg);
    // save hash(cfg) to trusted location
    sha = get_hash_of_config_state(cfg);
    save_hash(sha);
    go_to_sleep();
}
void restore_config_state()
{
    void* cfg;
    void* sha_1, sha_2;
    cfg = get_config_file();
    // restore hash of config from trusted memory
    sha_1 = get_persisted_sha_value();
    sha_2 = get_hash_of_config_state(cfg);
    if (sha_1 != sha_2)
        assert_error_and_halt();
    load_config_file(cfg);
}
```

It must be noted that in the previous example of good pseudo code, the memory (where the hash of the config state is stored) must be trustworthy while the hardware is between the power save and restore states.

Functional Areas

- Power

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
176	Configuration/Environment Manipulation

CWE-1310: Missing Ability to Patch ROM Code

Weakness ID : 1310

Structure : Simple

Abstraction : Base

Description

Missing an ability to patch ROM code may leave a System or System-on-Chip (SoC) in a vulnerable state.

Extended Description

A System or System-on-Chip (SoC) that implements a boot process utilizing security mechanisms such as Root-of-Trust (RoT) typically starts by executing code from a Read-only-Memory (ROM) component. The code in ROM is immutable, hence any security vulnerabilities discovered in the ROM code can never be fixed for the systems that are already in use.

A common weakness is that the ROM does not have the ability to patch if security vulnerabilities are uncovered after the system gets shipped. This leaves the system in a vulnerable state where an adversary can compromise the SoC.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined 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	1329	Reliance on Component That is Not Updateable	2219

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context Reduce Maintainability <i>When the system is unable to be patched, it can be left in a vulnerable state.</i>	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Secure patch support to allow ROM code to be patched on the next boot.

Effectiveness = Moderate

Some parts of the hardware initialization or signature verification done to authenticate patches will always be "not patchable."

Phase: Architecture and Design

Phase: Implementation

Support patches that can be programmed in-field or during manufacturing through hardware fuses. This feature can be used for limited patching of devices after shipping, or for the next batch of silicon devices manufactured, without changing the full device ROM.

Effectiveness = Moderate

Patches that use hardware fuses will have limitations in terms of size and the number of patches that can be supported. Note that some parts of the hardware initialization or signature verification done to authenticate patches will always be "not patchable."

Demonstrative Examples

Example 1:

A System-on-Chip (SOC) implements a Root-of-Trust (RoT) in ROM to boot secure code. However, at times this ROM code might have security vulnerabilities and need to be patched. Since ROM is immutable, it can be impossible to patch.

ROM does not have built-in application-programming interfaces (APIs) to patch if the code is vulnerable. Implement mechanisms to patch the vulnerable ROM code.

Example 2:

The example code is taken from the SoC peripheral wrapper inside the buggy OpenPiton SoC of HACK@DAC'21. The wrapper is used for connecting the communications between SoC peripherals, such as crypto-engines, direct memory access (DMA), reset controllers, JTAG, etc. The secure implementation of the SoC wrapper should allow users to boot from a ROM for Linux (i_bootrom_linux) or from a patchable ROM (i_bootrom_patch) if the Linux bootrom has security or functional issues. The example code is taken from the SoC peripheral wrapper inside the buggy OpenPiton SoC of HACK@DAC'21. The wrapper is used for connecting the communications between SoC peripherals, such as crypto-engines, direct memory access (DMA), reset controllers, JTAG, etc. The secure implementation of the SoC wrapper should allow users to boot from a ROM for Linux (i_bootrom_linux) or from a patchable ROM (i_bootrom_patch) if the Linux bootrom has security or functional issues.

Example Language: Verilog

(Bad)

```
...
bootrom i_bootrom_patch (
    .clk_i ,
    .req_i ( rom_req ),
    .addr_i ( rom_addr ),
```

```
.rdata_o ( rom_rdata_patch )
);
bootrom_linux i_bootrom_linux (
    .clk_i ,
    .req_i ( rom_req ),
    .addr_i ( rom_addr ),
    .rdata_o ( rom_rdata_linux )
);
assign rom_rdata = (ariane_boot_sel_i) ? rom_rdata_linux : rom_rdata_patch;
...
```

The above implementation causes the ROM data to be hardcoded for the linux system (rom_rdata_linux) regardless of the value of ariane_boot_sel_i. Therefore, the data (rom_rdata_patch) from the patchable ROM code is never used [REF-1396].

This weakness disables the ROM's ability to be patched. If attackers uncover security vulnerabilities in the ROM, the users must replace the entire device. Otherwise, the weakness exposes the system to a vulnerable state forever.

A fix to this issue is to enable rom_rdata to be selected from the patchable rom (rom_rdata_patch) [REF-1397].

Example Language: Verilog

(Good)

```
...
bootrom i_bootrom_patch (
    .clk_i ,
    .req_i ( rom_req ),
    .addr_i ( rom_addr ),
    .rdata_o ( rom_rdata_patch )
);
bootrom_linux i_bootrom_linux (
    .clk_i ,
    .req_i ( rom_req ),
    .addr_i ( rom_addr ),
    .rdata_o ( rom_rdata_linux )
);
assign rom_rdata = (ariane_boot_sel_i) ? rom_rdata_patch : rom_rdata_linux;
...
```

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2469
MemberOf	C	1415	Comprehensive Categorization: Resource Control	1400	2544

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
682	Exploitation of Firmware or ROM Code with Unpatchable Vulnerabilities

References

[REF-1396]"riscv_peripherals.sv line 534". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/75e5c0700b5a02e744f006fe8a09ff3c2ccdd32d/piton/design/chip/tile/ariane/openpiton/riscv_peripherals.sv#L534 >.2024-02-12.

[REF-1397]"Fix for riscv_peripherals.sv line 534". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/cwe_1310_riscv_peripheral/piton/design/chip/tile/ariane/openpiton/riscv_peripherals.sv#L534 >.2024-02-12.

CWE-1311: Improper Translation of Security Attributes by Fabric Bridge

Weakness ID : 1311

Structure : Simple

Abstraction : Base

Description

The bridge incorrectly translates security attributes from either trusted to untrusted or from untrusted to trusted when converting from one fabric protocol to another.

Extended Description

A bridge allows IP blocks supporting different fabric protocols to be integrated into the system. Fabric end-points or interfaces usually have dedicated signals to transport security attributes. For example, HPROT signals in AHB, AxPROT signals in AXI, and MReqInfo and SRespInfo signals in OCP.

The values on these signals are used to indicate the security attributes of the transaction. These include the immutable hardware identity of the controller initiating the transaction, privilege level, and type of transaction (e.g., read/write, cacheable/non-cacheable, posted/non-posted).

A weakness can arise if the bridge IP block, which translates the signals from the protocol used in the IP block endpoint to the protocol used by the central bus, does not properly translate the security attributes. As a result, the identity of the initiator could be translated from untrusted to trusted or vice-versa. This could result in access-control bypass, privilege escalation, or denial of service.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Applicable Platforms

Language : Verilog (*Prevalence = Undetermined*)

Language : VHDL (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Integrity	Read Memory	
Access Control	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	
	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

The translation must map signals in such a way that untrusted agents cannot map to trusted agents or vice-versa.

Phase: Implementation

Ensure that the translation maps signals in such a way that untrusted agents cannot map to trusted agents or vice-versa.

Demonstrative Examples

Example 1:

The bridge interfaces between OCP and AHB end points. OCP uses MReqInfo signal to indicate security attributes, whereas AHB uses HPROT signal to indicate the security attributes. The width of MReqInfo can be customized as needed. In this example, MReqInfo is 5-bits wide and carries the privilege level of the OCP controller.

The values 5'h11, 5'h10, 5'h0F, 5'h0D, 5'h0C, 5'h0B, 5'h09, 5'h08, 5'h04, and 5'h02 in MReqInfo indicate that the request is coming from a privileged state of the OCP bus controller. Values 5'h1F, 5'h0E, and 5'h00 indicate untrusted, privilege state.

Though HPROT is a 5-bit signal, we only consider the lower, two bits in this example. HPROT values 2'b00 and 2'b10 are considered trusted, and 2'b01 and 2'b11 are considered untrusted.

The OCP2AHB bridge is expected to translate trusted identities on the controller side to trusted identities on the responder side. Similarly, it is expected to translate untrusted identities on the controller side to untrusted identities on the responder side.

Example Language: Verilog

(Bad)

```
module ocp2ahb
(
    ahb_hprot,
    ocp_mreqinfo
);
output [1:0] ahb_hprot; // output is 2 bit signal for AHB HPROT
input [4:0] ocp_mreqinfo; // input is 5 bit signal from OCP MReqInfo
wire [6:0] p0_mreqinfo_o_temp; // OCP signal that transmits hardware identity of bus controller
wire y;
reg [1:0] ahb_hprot;
// hardware identity of bus controller is in bits 5:1 of p0_mreqinfo_o_temp signal
assign p0_mreqinfo_o_temp[6:0] = {1'b0, ocp_mreqinfo[4:0], y};
always @*
begin
    case (p0_mreqinfo_o_temp[4:2])
        000: ahb_hprot = 2'b11; // OCP MReqInfo to AHB HPROT mapping
        001: ahb_hprot = 2'b00;
        010: ahb_hprot = 2'b00;
        011: ahb_hprot = 2'b01;
        100: ahb_hprot = 2'b00;
        101: ahb_hprot = 2'b00;
        110: ahb_hprot = 2'b10;
        111: ahb_hprot = 2'b00;
    endcase
end
endmodule
```

Logic in the case statement only checks for MReqInfo bits 4:2, i.e., hardware-identity bits 3:1. When ocp_mreqinfo is 5'h1F or 5'h0E, p0_mreqinfo_o_temp[2] will be 1. As a result, untrusted IDs from OCP 5'h1F and 5'h0E get translated to trusted ahb_hprot values 2'b00.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1203	Peripherals, On-chip Fabric, and Interface/IO Problems	1194	2472

Nature	Type	ID	Name	V	Page
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels
233	Privilege Escalation

CWE-1312: Missing Protection for Mirrored Regions in On-Chip Fabric Firewall

Weakness ID : 1312

Structure : Simple

Abstraction : Base

Description

The firewall in an on-chip fabric protects the main addressed region, but it does not protect any mirrored memory or memory-mapped-IO (MMIO) regions.

Extended Description

Few fabrics mirror memory and address ranges, where mirrored regions contain copies of the original data. This redundancy is used to achieve fault tolerance. Whatever protections the fabric firewall implements for the original region should also apply to the mirrored regions. If not, an attacker could bypass existing read/write protections by reading from/writing to the mirrored regions to leak or corrupt the original 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf	B	1251	Mirrored Regions with Different Values	2054

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Integrity	Read Memory	
Access Control	Bypass Protection Mechanism	

Detection Methods

Manual Dynamic Analysis

Using an external debugger, send write transactions to mirrored regions to test if original, write-protected regions are modified. Similarly, send read transactions to mirrored regions to test if the original, read-protected signals can be read.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

The fabric firewall should apply the same protections as the original region to the mirrored regions.

Phase: Implementation

The fabric firewall should apply the same protections as the original region to the mirrored regions.

Demonstrative Examples

Example 1:

A memory-controller IP block is connected to the on-chip fabric in a System on Chip (SoC). The memory controller is configured to divide the memory into four parts: one original and three mirrored regions inside the memory. The upper two bits of the address indicate which region is being addressed. 00 indicates the original region and 01, 10, and 11 are used to address the mirrored regions. All four regions operate in a lock-step manner and are always synchronized. The firewall in the on-chip fabric is programmed to protect the assets in the memory.

The firewall only protects the original range but not the mirrored regions.

The attacker (as an unprivileged user) sends a write transaction to the mirrored region. The mirrored region has an address with the upper two bits set to "10" and the remaining bits of the address pointing to an asset. The firewall does not block this write transaction. Once the write is successful, contents in the protected-memory region are also updated. Thus, the attacker can bypass existing, memory protections.

Firewall should protect mirrored regions.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1203	Peripherals, On-chip Fabric, and Interface/IO Problems	1194	2472
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
456	Infected Memory
679	Exploitation of Improperly Configured or Implemented Memory Protections

References

[REF-1134]Taku Izumi, Fujitsu Limited. "Address Range Memory Mirroring". 2016. < <https://www.fujitsu.com/jp/documents/products/software/os/linux/catalog/LinuxConJapan2016-Izumi.pdf> >.

CWE-1313: Hardware Allows Activation of Test or Debug Logic at Runtime

Weakness ID : 1313

Structure : Simple
Abstraction : Base

Description

During runtime, the hardware allows for test or debug logic (feature) to be activated, which allows for changing the state of the hardware. This feature can alter the intended behavior of the system and allow for alteration and leakage of sensitive data by an adversary.

Extended Description

An adversary can take advantage of test or debug logic that is made accessible through the hardware during normal operation to modify the intended behavior of the system. For example, an accessible Test/debug mode may allow read/write access to any system data. Using error injection (a common test/debug feature) during a transmit/receive operation on a bus, data may be modified to produce an unintended message. Similarly, confidentiality could be compromised by such features allowing access to secrets.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Integrity	Read Memory	
Availability	DoS: Crash, Exit, or Restart	
	DoS: Instability	
	DoS: Resource Consumption (CPU)	
	DoS: Resource Consumption (Memory)	
	DoS: Resource Consumption (Other)	
	Execute Unauthorized Code or Commands	
	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	
	Alter Execution Logic	
	Quality Degradation	
	Unexpected State	
	Reduce Performance	
	Reduce Reliability	

Potential Mitigations

Phase: Architecture and Design

Insert restrictions on when the hardware's test or debug features can be activated. For example, during normal operating modes, the hardware's privileged modes that allow access to such features cannot be activated. Configuring the hardware to only enter a test or debug mode within a window of opportunity such as during boot or configuration stage. The result is disablement of such test/debug features and associated modes during normal runtime operations.

Phase: Implementation

Insert restrictions on when the hardware's test or debug features can be activated. For example, during normal operating modes, the hardware's privileged modes that allow access to such features cannot be activated. Configuring the hardware to only enter a test or debug mode within a window of opportunity such as during boot or configuration stage. The result is disablement of such test/debug features and associated modes during normal runtime operations.

Phase: Integration



Insert restrictions on when the hardware's test or debug features can be activated. For example, during normal operating modes, the hardware's privileged modes that allow access to such features cannot be activated. Configuring the hardware to only enter a test or debug mode within a window of opportunity such as during boot or configuration stage. The result is disablement of such test/debug features and associated modes during normal runtime operations.

Observed Examples

Reference	Description
CVE-2021-33150	Hardware processor allows activation of test or debug logic at runtime. https://www.cve.org/CVERecord?id=CVE-2021-33150
CVE-2021-0146	Processor allows the activation of test or debug logic at runtime, allowing escalation of privileges https://www.cve.org/CVERecord?id=CVE-2021-0146

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1207	Debug and Test Problems	1194	2474
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Non-Production Interfaces

CWE-1314: Missing Write Protection for Parametric Data Values

Weakness ID : 1314

Structure : Simple

Abstraction : Base

Description

The device does not write-protect the parametric data values for sensors that scale the sensor value, allowing untrusted software to manipulate the apparent result and potentially damage hardware or cause operational failure.

Extended Description

Various sensors are used by hardware to detect any devices operating outside of the design limits. The threshold limit values are set by hardware fuses or trusted software such as the BIOS. These

limits may be related to thermal, power, voltage, current, and frequency. Hardware mechanisms may be used to protect against alteration of the threshold limit values by untrusted software.

The limit values are generally programmed in standard units for the type of value being read. However, the hardware-sensor blocks may report the settings in different units depending upon sensor design and operation. The raw sensor output value is converted to the desired units using a scale conversion based on the parametric data programmed into the sensor. The final converted value is then compared with the previously programmed limits.

While the limit values are usually protected, the sensor parametric data values may not be. By changing the parametric data, safe operational limits may be bypassed.


Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		862	Missing Authorization	1780

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name	Page
PeerOf		1299	Missing Protection Mechanism for Alternate Hardware Interface	2162

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Sensor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Availability	Quality Degradation DoS: Resource Consumption (Other) <i>Sensor value manipulation, particularly thermal or power, may allow physical damage to occur or disabling of the device by a false fault shutdown causing a Denial-Of-Service.</i>	High

Potential Mitigations

Phase: Architecture and Design

Access controls for sensor blocks should ensure that only trusted software is allowed to change threshold limits and sensor parametric data.

Effectiveness = High

Demonstrative Examples

Example 1:

Malicious software executes instructions to increase power consumption to the highest possible level while causing the clock frequency to increase to its maximum value. Such a program

executing for an extended period of time would likely overheat the device, possibly resulting in permanent damage to the device.

A ring, oscillator-based temperature sensor will generally report the sensed value as oscillator frequency rather than degrees centigrade. The temperature sensor will have calibration values that are used to convert the detected frequency into the corresponding temperature in degrees centigrade.

Consider a SoC design where the critical maximum temperature limit is set in fuse values to 100C and is not modifiable by software. If the scaled thermal sensor output equals or exceeds this limit, the system is commanded to shut itself down.

The thermal sensor calibration values are programmable through registers that are exposed to system software. These registers allow software to affect the converted temperature output such that the output will never exceed the maximum temperature limit.

Example Language: Other

(Bad)

The sensor frequency value is scaled by applying the function:

$$\text{Sensed Temp} = a + b * \text{Sensor Freq}$$

where a and b are the programmable calibration data coefficients. Software sets a and b to zero ensuring the sensed temperature is always zero.

This weakness may be addressed by preventing access to a and b.

Example Language: Other

(Good)

The sensor frequency value is scaled by applying the function:

$$\text{Sensed Temp} = a + b * \text{Sensor Freq}$$

where a and b are the programmable calibration data coefficients. Untrusted software is prevented from changing the values of either a or b, preventing this method of manipulating the temperature.

Observed Examples

Reference	Description
CVE-2017-8252	Kernel can inject faults in computations during the execution of TrustZone leading to information disclosure in Snapdragon Auto, Snapdragon Compute, Snapdragon Connectivity, Snapdragon Consumer Electronics Connectivity, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon IoT, Snapdragon Mobile, Snapdragon Voice and Music, Snapdragon Wearables, Snapdragon Wired Infrastructure and Networking. https://www.cve.org/CVERecord?id=CVE-2017-8252

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf	C	1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID Attack Pattern Name

1	Accessing Functionality Not Properly Constrained by ACLs
---	--

References

[REF-1082]Adrian Tang, Simha Sethumadhavan and Salvatore Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". < <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-tang.pdf> >.

CWE-1315: Improper Setting of Bus Controlling Capability in Fabric End-point**Weakness ID :** 1315**Structure :** Simple**Abstraction :** Base**Description**

The bus controller enables bits in the fabric end-point to allow responder devices to control transactions on the fabric.

Extended Description

To support reusability, certain fabric interfaces and end points provide a configurable register bit that allows IP blocks connected to the controller to access other peripherals connected to the fabric. This allows the end point to be used with devices that function as a controller or responder. If this bit is set by default in hardware, or if firmware incorrectly sets it later, a device intended to be a responder on a fabric is now capable of controlling transactions to other devices and might compromise system 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, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

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

Nature	Type	ID	Name	Page
ChildOf	IP	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

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

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory Read Memory Bypass Protection Mechanism	

Potential Mitigations**Phase: Architecture and Design**

For responder devices, the register bit in the fabric end-point that enables the bus controlling capability must be set to 0 by default. This bit should not be set during secure-boot flows. Also,

writes to this register must be access-protected to prevent malicious modifications to obtain bus-controlling capability.

Phase: Implementation

For responder devices, the register bit in the fabric end-point that enables the bus controlling capability must be set to 0 by default. This bit should not be set during secure-boot flows. Also, writes to this register must be access-protected to prevent malicious modifications to obtain bus-controlling capability.

Phase: System Configuration

For responder devices, the register bit in the fabric end-point that enables the bus controlling capability must be set to 0 by default. This bit should not be set during secure-boot flows. Also, writes to this register must be access-protected to prevent malicious modifications to obtain bus-controlling capability.

Demonstrative Examples

Example 1:

A typical, phone platform consists of the main, compute core or CPU, a DRAM-memory chip, an audio codec, a baseband modem, a power-management-integrated circuit ("PMIC"), a connectivity (WiFi and Bluetooth) modem, and several other analog/RF components. The main CPU is the only component that can control transactions, and all the other components are responder-only devices. All the components implement a PCIe end-point to interface with the rest of the platform. The responder devices should have the bus-control-enable bit in the PCIe-end-point register set to 0 in hardware to prevent the devices from controlling transactions to the CPU or other peripherals.

The audio-codec chip does not have the bus-controller-enable-register bit hardcoded to 0. There is no platform-firmware flow to verify that the bus-controller-enable bit is set to 0 in all responders.

Audio codec can now master transactions to the CPU and other platform components. Potentially, it can modify assets in other platform components to subvert system security.

Platform firmware includes a flow to check the configuration of bus-controller-enable bit in all responder devices. If this register bit is set on any of the responders, platform firmware sets it to 0. Ideally, the default value of this register bit should be hardcoded to 0 in RTL. It should also have access control to prevent untrusted entities from setting this bit to become bus controllers.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1203	Peripherals, On-chip Fabric, and Interface/IO Problems	1194	2472
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1135]Benoit Morgan, Eric Alata, Vincent Nicomette, Mohamed Kaaniche. "Bypassing IOMMU Protection against I/O Attacks". 2016. < <https://hal.archives-ouvertes.fr/hal-01419962/document> >.

[REF-1136]Colin L. Rothwell. "Exploitation from malicious PCI Express peripherals". 2019. < <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-934.pdf> >.

CWE-1316: Fabric-Address Map Allows Programming of Unwarranted Overlaps of Protected and Unprotected Ranges

Weakness ID : 1316

Structure : Simple

Abstraction : Base

Description

The address map of the on-chip fabric has protected and unprotected regions overlapping, allowing an attacker to bypass access control to the overlapping portion of the protected region.

Extended Description

Various ranges can be defined in the system-address map, either in the memory or in Memory-Mapped-IO (MMIO) space. These ranges are usually defined using special range registers that contain information, such as base address and size. Address decoding is the process of determining for which range the incoming transaction is destined. To ensure isolation, ranges containing secret data are access-control protected.

Occasionally, these ranges could overlap. The overlap could either be intentional (e.g. due to a limited number of range registers or limited choice in choosing size of the range) or unintentional (e.g. introduced by errors). Some hardware designs allow dynamic remapping of address ranges assigned to peripheral MMIO ranges. In such designs, intentional address overlaps can be created through misconfiguration by malicious software. When protected and unprotected ranges overlap, an attacker could send a transaction and potentially compromise the protections in place, violating the principle of least privilege.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Bus/Interface Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Bypass Protection Mechanism	Medium
Integrity	Read Memory	
Access Control	Modify Memory	
Authorization		

Detection Methods

Automated Dynamic Analysis

Review address map in specification to see if there are any overlapping ranges.

Effectiveness = High

Manual Static Analysis

Negative testing of access control on overlapped ranges.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

When architecting the address map of the chip, ensure that protected and unprotected ranges are isolated and do not overlap. When designing, ensure that ranges hardcoded in Register-Transfer Level (RTL) do not overlap.

Phase: Implementation

Ranges configured by firmware should not overlap. If overlaps are mandatory because of constraints such as a limited number of registers, then ensure that no assets are present in the overlapped portion.

Phase: Testing

Validate mitigation actions with robust testing.

Demonstrative Examples

Example 1:

An on-chip fabric supports a 64KB address space that is memory-mapped. The fabric has two range registers that support creation of two protected ranges with specific size constraints--4KB, 8KB, 16KB or 32KB. Assets that belong to user A require 4KB, and those of user B require 20KB. Registers and other assets that are not security-sensitive require 40KB. One range register is configured to program 4KB to protect user A's assets. Since a 20KB range cannot be created with the given size constraints, the range register for user B's assets is configured as 32KB. The rest of the address space is left as open. As a result, some part of untrusted and open-address space overlaps with user B range.

The fabric does not support least privilege, and an attacker can send a transaction to the overlapping region to tamper with user B data.

Since range B only requires 20KB but is allotted 32KB, there is 12KB of reserved space. Overlapping this region of user B data, where there are no assets, with the untrusted space will prevent an attacker from tampering with user B data.

Observed Examples

Reference	Description
CVE-2009-4419	Attacker can modify MCHBAR register to overlap with an attacker-controlled region, which modification prevents the SENTER instruction from properly applying VT-d protection while a Measured Launch Environment is being launched. https://www.cve.org/CVERecord?id=CVE-2009-4419

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1203	Peripherals, On-chip Fabric, and Interface/IO Problems	1194	2472
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Notes

Maintenance

As of CWE 4.6, CWE-1260 and CWE-1316 are siblings under view 1000, but CWE-1260 might be a parent of CWE-1316. More analysis is warranted.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
456	Infected Memory
679	Exploitation of Improperly Configured or Implemented Memory Protections

References

[REF-1137]Yuriy Bulygin, Oleksandr Bazhaniuk, Andrew Furtak, John Loucaides, Mikhail Gorobets. "BARing the System - New vulnerabilities in Coreboot & UEFI-based Systems". 2017. < https://www.c7zero.info/stuff/REConBrussels2017_BARing_the_system.pdf >.

CWE-1317: Improper Access Control in Fabric Bridge

Weakness ID : 1317

Structure : Simple

Abstraction : Base

Description

The product uses a fabric bridge for transactions between two Intellectual Property (IP) blocks, but the bridge does not properly perform the expected privilege, identity, or other access control checks between those IP blocks.

Extended Description

In hardware designs, different IP blocks are connected through interconnect-bus fabrics (e.g. AHB and OCP). Within a System on Chip (SoC), the IP block subsystems could be using different bus protocols. In such a case, the IP blocks are then linked to the central bus (and to other IP blocks) through a fabric bridge. Bridges are used as bus-interconnect-routing modules that link different protocols or separate, different segments of the overall SoC interconnect.

For overall system security, it is important that the access-control privileges associated with any fabric transaction are consistently maintained and applied, even when they are routed or translated by a fabric bridge. A bridge that is connected to a fabric without security features forwards transactions to the slave without checking the privilege level of the master and results in a weakness in SoC access-control security. The same weakness occurs if a bridge does not check the hardware identity of the transaction received from the slave interface of the bridge.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	IP	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	DoS: Crash, Exit, or Restart	Medium
Integrity	Bypass Protection Mechanism	
Access Control	Read Memory	
Availability	Modify Memory	

Detection Methods

Simulation / Emulation

RTL simulation to ensure that bridge-access controls are implemented properly.

Effectiveness = High

Formal Verification

Formal verification of bridge RTL to ensure that access control cannot be bypassed.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Ensure that the design includes provisions for access-control checks in the bridge for both upstream and downstream transactions.

Phase: Implementation

Implement access-control checks in the bridge for both upstream and downstream transactions.

Demonstrative Examples

Example 1:

This example is from CVE-2019-6260 [REF-1138]. The iLPC2AHB bridge connects a CPU (with multiple, privilege levels, such as user, super user, debug, etc.) over AHB interface to an LPC bus. Several peripherals are connected to the LPC bus. The bridge is expected to check the privilege level of the transactions initiated in the core before forwarding them to the peripherals on the LPC bus.

The bridge does not implement the checks and allows reads and writes from all privilege levels.

To address this, designers should implement hardware-based checks that are either hardcoded to block untrusted agents from accessing secure peripherals or implement firmware flows that configure the bridge to block untrusted agents from making arbitrary reads or writes.

Example 2:

The example code below is taken from the AES and core local interrupt (CLINT) peripherals of the HACK@DAC'21 buggy OpenPiton SoC. The access to all the peripherals for a given privilege level of the processor is controlled by an access control module in the SoC. This ensures that malicious users with insufficient privileges do not get access to sensitive data, such as the AES keys used by the operating system to encrypt and decrypt information. The security of the entire system will be compromised if the access controls are incorrectly enforced. The access controls are enforced through the interconnect-bus fabrics, where access requests with insufficient access control permissions will be rejected.

Example Language: Verilog

(Bad)

```
...
module aes0_wrapper #(...) (...);
...

```

```

input logic acct_ctrl_i;
...
axi_lite_interface #(
) axi_lite_interface_i (
...
.en_o ( en_acct ),
...
..);
assign en = en_acct && acct_ctrl_i;
...
endmodule
...
module clint #(...)(...);
...
axi_lite_interface #(
) axi_lite_interface_i (
...
.en_o ( en ),
...
);
...
endmodule

```

The previous code snippet [REF-1382] illustrates an instance of a vulnerable implementation of access control for the CLINT peripheral (see module clint). It also shows a correct implementation of access control for the AES peripheral (see module aes0_wrapper) [REF-1381]. An enable signal (en_o) from the fabric's AXI interface (present in both modules) is used to determine if an access request is made to the peripheral. In the case of the AES peripheral, this en_o signal is first received in a temporary signal en_acct. Then, the access request is enabled (by asserting the en signal) only if the request has sufficient access permissions (i.e., acct_ctrl_i signal should be enabled). However, in the case of the CLINT peripheral, the enable signal, en_o, from the AXI interface, is directly used to enable accesses. As a result, users with insufficient access permissions also get full access to the CLINT peripheral.

To fix this, enable access requests to CLINT [REF-1383] only if the user has sufficient access as indicated by the acct_ctrl_i signal in the boolean && with en_acct.

Example Language: Verilog (Good)

```

module clint #(
) (
...
input logic acct_ctrl_i,
...
);
logic en, en_acct;
...
axi_lite_interface #(
) axi_lite_interface_i (
...
.en_o ( en_acct ),
...
);
assign en = en_acct && acct_ctrl_i;
...
endmodule

```

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].

Reference	Description
	https://www.cve.org/CVERecord?id=CVE-2019-6260

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1203	Peripherals, On-chip Fabric, and Interface/IO Problems	1194	2472
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
122	Privilege Abuse

References

[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-1381]"aes0_wrapper.sv lines 72 - 78". 2021. < https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/aes0/aes0_wrapper.sv#L72-L78 >.2024-01-16.

[REF-1382]"clint.sv line 71". 2021. < <https://github.com/HACK-EVENT/hackatdac21/blob/b9ecdf6068445d76d6bee692d163fededf7a9d9b/piton/design/chip/tile/ariane/src/clint/clint.sv#L71C2-L71C36> >.2024-01-16.

[REF-1383]"Fix for clint.sv line 78". 2021. < <https://github.com/HACK-EVENT/hackatdac21/blob/45a004368b5a31857008834d9780536f0764f055/piton/design/chip/tile/ariane/src/clint/clint.sv#L78> >.2024-01-16.

CWE-1318: Missing Support for Security Features in On-chip Fabrics or Buses

Weakness ID : 1318

Structure : Simple

Abstraction : Base

Description

On-chip fabrics or buses either do not support or are not configured to support privilege separation or other security features, such as access control.

Extended Description

Certain on-chip fabrics and buses, especially simple and low-power buses, do not support security features. Apart from data transfer and addressing ports, some fabrics and buses do not have any interfaces to transfer privilege, immutable identity, or any other security attribute coming from the bus master. Similarly, they do not have dedicated signals to transport security-sensitive data from slave to master, such as completions for certain types of transactions. Few other on-chip fabrics and buses support security features and define specific interfaces/signals for transporting security attributes from master to slave or vice-versa. However, including these signals is not mandatory and could be left unconfigured when generating the register-transfer-level (RTL) description for the fabric. Such fabrics or buses should not be used to transport any security attribute coming from the bus master. In general, peripherals with security assets should not be connected to such buses before the transaction from the bus master reaches the bus, unless some form of access control is performed at a fabric bridge or another intermediate module.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	693	Protection Mechanism Failure	1520

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : Processor Hardware (*Prevalence = Undetermined*)

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

Common Consequences

Scope	Impact	Likelihood
Confidentiality	DoS: Crash, Exit, or Restart	Medium
Integrity	Read Memory	
Access Control	Modify Memory	
Availability		

Detection Methods

Architecture or Design Review

Review the fabric specification and ensure that it contains signals to transfer security-sensitive signals.

Effectiveness = High

Manual Static Analysis - Source Code

Lack of security features can also be confirmed through manual RTL review of the fabric RTL.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

If fabric does not support security features, implement security checks in a bridge or any component that is between the master and the fabric. Alternatively, connect all fabric slaves that do not have any security assets under one such fabric and connect peripherals with security assets to a different fabric that supports security features.

Demonstrative Examples

Example 1:

Several systems on chips (SoCs) use the Advanced-Microcontroller Bus Architecture (AMBA) Advanced-Peripheral Bus (APB) protocol. APB is a simple, low-power bus and uses the PPROT[2:0] bits to indicate the security state of the bus masters ;PPROT[0] indicates privilege, PPROT[1] indicates secure/non-secure transaction, and PPROT[2] indicates instruction/data. Assume that there is no fabric bridge in the SoC. One of the slaves, the power-management unit, contains registers that store the thermal-shutdown limits.

The APB bus is used to connect several bus masters, each with a unique and immutable hardware identity, to several slaves. For a CPU supporting 8 potential identities (each with varying privilege

levels), 16 types of outgoing transactions can be made--8 read transactions with each supported privilege level and 8 write transactions with each supported privilege level.

Since APB PPROT can only support up to 8 transaction types, access-control checks cannot be performed on transactions going to the slaves at the right granularity for all possible transaction types. Thus, potentially, user code running on the CPU could maliciously corrupt the thermal-shutdown-configuration registers to burn the device, resulting in permanent denial of service.

In this scenario, only peripherals that need access protection from 8 of the 16 possible transaction types can be connected to the APB bus. Peripherals that require protection from the remaining 8 transaction types can be connected to a different APB bus. Alternatively, a bridge could be implemented to handle such complex scenarios before forwarding traffic to the APB bus.

Example 2:

The Open-Core-Protocol (OCP) fabric supports two configurable, width-optional signals for transporting security attributes: MReqInfo and SRespInfo. MReqInfo is used to transport security attributes from bus master to slave, and SRespInfo is used to transport security attributes from slave to bus master. An SoC uses OCP to connect several bus masters, each with a unique and immutable hardware identity, to several slaves. One of the bus masters, the CPU, reports the privilege level (user or super user) in addition to the unique identity. One of the slaves, the power-management unit, contains registers that store the thermal-shutdown limits.



Since MReqInfo and SRespInfo are not mandatory, these signals are not configured when autogenerating RTL for the OCP fabric. Thus, the fabric cannot be used to transport security attributes from bus masters to slave.

Code running at user-privilege level on the CPU could maliciously corrupt the thermal-shutdown-configuration registers to burn the device and cause permanent denial of service.

To address this, configure the fabric to include MReqInfo and SRespInfo signals and use these to transport security identity and privilege level to perform access-control checks at the slave interface.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1198	Privilege Separation and Access Control Issues	1194	2470
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1139]ARM. "AMBA APB Protocol Specification, Version 2.0". 2010. < https://www.eecs.umich.edu/courses/eecs373/readings/IHI0024C_amba_apb_protocol_spec.pdf >.

[REF-1140]OCP-IP. "Open Core Protocol Specification, Release 2.2". 2006. < <http://read.pudn.com/downloads95/doc/388103/OCPSpecification%202.2.pdf> >.

CWE-1319: Improper Protection against Electromagnetic Fault Injection (EM-FI)

Weakness ID : 1319**Structure** : Simple**Abstraction** : Base

Description

The device is susceptible to electromagnetic fault injection attacks, causing device internal information to be compromised or security mechanisms to be bypassed.

Extended Description

Electromagnetic fault injection may allow an attacker to locally and dynamically modify the signals (both internal and external) of an integrated circuit. EM-FI attacks consist of producing a local, transient magnetic field near the device, inducing current in the device wires. A typical EMFI setup is made up of a pulse injection circuit that generates a high current transient in an EMI coil, producing an abrupt magnetic pulse which couples to the target producing faults in the device, which can lead to:

- Bypassing security mechanisms such as secure JTAG or Secure Boot
- Leaking device information
- Modifying program flow
- Perturbing secure hardware modules (e.g. random number generators)

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	[P]	693	Protection Mechanism Failure	1520

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Technology : Microcontroller Hardware (*Prevalence = Undetermined*)

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Power Management Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Technology : Test/Debug Hardware (*Prevalence = Undetermined*)

Technology : Sensor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Integrity	Read Memory	
Access Control	Gain Privileges or Assume Identity	
Availability	Bypass Protection Mechanism	
	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

1. Redundancy - By replicating critical operations and comparing the two outputs can help indicate whether a fault has been injected.
2. Error detection and correction codes - Gay, Mael, et al. proposed a new scheme that not only detects faults injected by a malicious adversary but also automatically corrects single nibble/byte errors introduced by low-multiplicity faults.
3. Fail by default coding - When checking conditions (switch or if) check all possible cases and fail by default because the default case in a switch (or the else part of a cascaded if-else-if construct) is used for dealing with the last possible (and valid) value without checking. This is prone to fault injection because this alternative is easily selected as a result of potential data manipulation [REF-1141].
4. Random Behavior - adding random delays before critical operations, so that timing is not predictable.
5. Program Flow Integrity Protection - The program flow can be secured by integrating run-time checking aiming at detecting control flow inconsistencies. One such example is tagging the source code to indicate the points not to be bypassed [REF-1147].
6. Sensors - Usage of sensors can detect variations in voltage and current.
7. Shields - physical barriers to protect the chips from malicious manipulation.

Demonstrative Examples

Example 1:

In many devices, security related information is stored in fuses. These fuses are loaded into shadow registers at boot time. Disturbing this transfer phase with EM-FI can lead to the shadow registers storing erroneous values potentially resulting in reduced security.



Colin O'Flynn has demonstrated an attack scenario which uses electro-magnetic glitching during booting to bypass security and gain read access to flash, read and erase access to shadow memory area (where the private password is stored). Most devices in the MPC55xx and MPC56xx series that include the Boot Assist Module (BAM) (a serial or CAN bootloader mode) are susceptible to this attack. In this paper, a GM ECU was used as a real life target. While the success rate appears low (less than 2 percent), in practice a success can be found within 1-5 minutes once the EMFI tool is setup. In a practical scenario, the author showed that success can be achieved within 30-60 minutes from a cold start.

Observed Examples

Reference	Description
CVE-2020-27211	Chain: microcontroller system-on-chip uses a register value stored in flash to set product protection state on the memory bus and does not contain protection against fault injection (CWE-1319) which leads to an incorrect initialization of the memory bus (CWE-1419) causing the product to be in an unprotected state. https://www.cve.org/CVERecord?id=CVE-2020-27211

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1388	Physical Access Issues and Concerns	1194	2518
MemberOf		1413	Comprehensive Categorization: Protection Mechanism Failure	1400	2542

Notes

Maintenance

This entry is attack-oriented and may require significant modification in future versions, or even deprecation. It is not clear whether there is really a design "mistake" that enables such attacks, so this is not necessarily a weakness and may be more appropriate for CAPEC.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
624	Hardware Fault Injection
625	Mobile Device Fault Injection

References

- [REF-1141]Marc Witteman. "Secure Application Programming in the presence of Side Channel Attacks". 2017. < https://riscureprodstorage.blob.core.windows.net/production/2017/08/Riscure_Whitepaper_Side_Channel_Patterns.pdf >.2023-04-07.
- [REF-1142]A. Dehbaoui, J. M. Dutertre, B. Robisson, P. Orsatelli, P. Maurine, A. Tria. "Injection of transient faults using electromagnetic pulses. Practical results on a cryptographic system". 2012. < <https://eprint.iacr.org/2012/123.pdf> >.
- [REF-1143]A. Menu, S. Bhasin, J. M. Dutertre, J. B. Rigaud, J. Danger. "Precise Spatio-Temporal Electromagnetic Fault Injections on Data Transfers". 2019. < <https://hal.telecom-paris.fr/hal-02338456/document> >.
- [REF-1144]Colin O'Flynn. "BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks". < <https://eprint.iacr.org/2020/937.pdf> >.
- [REF-1145]J. Balasch, D. Arumí, S. Manich. "Design and Validation of a Platform for Electromagnetic Fault Injection". < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8311630> >.
- [REF-1146]M. Gay, B. Karp, O. Keren, I. Polian. "Error control scheme for malicious and natural faults in cryptographic modules". 2019. < <https://link.springer.com/content/pdf/10.1007/s13389-020-00234-7.pdf> >.2023-04-07.
- [REF-1147]M. L. Akkar, L. Goubin, O. Ly. "Automatic Integration of Counter-Measures Against Fault Injection Attacks". < <https://www.labri.fr/perso/ly/publications/cfed.pdf> >.
- [REF-1285]Texas Instruments. "Physical Security Attacks Against Silicon Devices". 2022 January 1. < <https://www.ti.com/lit/an/swra739/swra739.pdf?ts=1644234570420> >.

CWE-1320: Improper Protection for Outbound Error Messages and Alert Signals

Weakness ID : 1320

Structure : Simple

Abstraction : Base

Description

Untrusted agents can disable alerts about signal conditions exceeding limits or the response mechanism that handles such alerts.

Extended Description

Hardware sensors are used to detect whether a device is operating within design limits. The threshold values for these limits are set by hardware fuses or trusted software such as a BIOS. Modification of these limits may be protected by hardware mechanisms.

When device sensors detect out of bound conditions, alert signals may be generated for remedial action, which may take the form of device shutdown or throttling.

Warning signals that are not properly secured may be disabled or used to generate spurious alerts, causing degraded performance or denial-of-service (DoS). These alerts may be masked by untrusted software. Examples of these alerts involve thermal and power sensor alerts.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Technology : Microcontroller Hardware (*Prevalence = Undetermined*)

Technology : Memory Hardware (*Prevalence = Undetermined*)

Technology : Power Management Hardware (*Prevalence = Undetermined*)

Technology : Processor Hardware (*Prevalence = Undetermined*)

Technology : Test/Debug Hardware (*Prevalence = Undetermined*)

Technology : Sensor Hardware (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Instability DoS: Crash, Exit, or Restart Reduce Reliability Unexpected State	High

Potential Mitigations

Phase: Architecture and Design

Alert signals generated by critical events should be protected from access by untrusted agents.

Only hardware or trusted firmware modules should be able to alter the alert configuration.

Demonstrative Examples

Example 1:

Consider a platform design where a Digital-Thermal Sensor (DTS) is used to monitor temperature and compare that output against a threshold value. If the temperature output equals or exceeds the threshold value, the DTS unit sends an alert signal to the processor.

The processor, upon getting the alert, input triggers system shutdown. The alert signal is handled as a General-Purpose-I/O (GPIO) pin in input mode.

Example Language:

(Bad)

The processor-GPIO controller exposes software-programmable controls that allow untrusted software to reprogram the state of the GPIO pin.

Reprogramming the state of the GPIO pin allows malicious software to trigger spurious alerts or to set the alert pin to a zero value so that thermal sensor alerts are not received by the processor.



Example Language:

(Good)

The GPIO alert-signal pin is blocked from untrusted software access and is controlled only by trusted software, such as the System BIOS.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1206	Power, Clock, Thermal, and Reset Concerns	1194	2473
MemberOf		1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
180	Exploiting Incorrectly Configured Access Control Security Levels

CWE-1321: Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution')

Weakness ID : 1321

Structure : Simple

Abstraction : Variant

Description

The product receives input from an upstream component that specifies attributes that are to be initialized or updated in an object, but it does not properly control modifications of attributes of the object prototype.

Extended Description

By adding or modifying attributes of an object prototype, it is possible to create attributes that exist on every object, or replace critical attributes with malicious ones. This can be problematic if the product depends on existence or non-existence of certain attributes, or uses pre-defined attributes of object prototype (such as `hasOwnProperty`, `toString` or `valueOf`).


This weakness is usually exploited by using a special attribute of objects called `proto`, `constructor` or `prototype`. Such attributes give access to the object prototype. This weakness is often found in code that assigns object attributes based on user input, or merges or clones objects recursively.

Relationships


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

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

Nature	Type	ID	Name	Page
ChildOf		915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1809

Nature	Type	ID	Name	Page
CanPrecede		471	Modification of Assumed-Immutable Data (MAID)	1121

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

Nature	Type	ID	Name	Page
ChildOf		913	Improper Control of Dynamically-Managed Code Resources	1805

Applicable Platforms

Language : JavaScript (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data <i>An attacker can inject attributes that are used in other components.</i>	High
Availability	DoS: Crash, Exit, or Restart <i>An attacker can override existing attributes with ones that have incompatible type, which may lead to a crash.</i>	High

Potential Mitigations

Phase: Implementation

By freezing the object prototype first (for example, `Object.freeze(Object.prototype)`), modification of the prototype becomes impossible.

Effectiveness = High

While this can mitigate this weakness completely, other methods are recommended when possible, especially in components used by upstream software ("libraries").

Phase: Architecture and Design

By blocking modifications of attributes that resolve to object prototype, such as `proto` or `prototype`, this weakness can be mitigated.

Effectiveness = High

Phase: Implementation

Strategy = Input Validation

When handling untrusted objects, validating using a schema can be used.

Effectiveness = Limited

Phase: Implementation

By using an object without prototypes (via `Object.create(null)`), adding object prototype attributes by accessing the prototype via the special attributes becomes impossible, mitigating this weakness.

Effectiveness = High

Phase: Implementation

Map can be used instead of objects in most cases. If Map methods are used instead of object attributes, it is not possible to access the object prototype or modify it.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

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

Example Language: JavaScript (Bad)

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

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

Example Language: JavaScript (Bad)

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

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

Example Language: JavaScript (Good)

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

Observed Examples

Reference	Description
CVE-2018-3721	Prototype pollution by merging objects. https://www.cve.org/CVERecord?id=CVE-2018-3721
CVE-2019-10744	Prototype pollution by setting default values to object attributes recursively. https://www.cve.org/CVERecord?id=CVE-2019-10744
CVE-2019-11358	Prototype pollution by merging objects recursively. https://www.cve.org/CVERecord?id=CVE-2019-11358
CVE-2020-8203	Prototype pollution by setting object attributes based on dot-separated path. https://www.cve.org/CVERecord?id=CVE-2020-8203

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1415	Comprehensive Categorization: Resource Control	1400	2544

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
77	Manipulating User-Controlled Variables
180	Exploiting Incorrectly Configured Access Control Security Levels

References

[REF-1148]Olivier Arteau. "Prototype pollution attack in NodeJS application". 2018 May 5. < https://github.com/HoLyVieR/prototype-pollution-nsec18/blob/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf >.

[REF-1149]Changhui Xu. "What is Prototype Pollution?". 2019 July 0. < <https://codeburst.io/what-is-prototype-pollution-49482fc4b638> >.

CWE-1322: Use of Blocking Code in Single-threaded, Non-blocking Context

Weakness ID : 1322

Structure : Simple

Abstraction : Base

Description

The product uses a non-blocking model that relies on a single threaded process for features such as scalability, but it contains code that can block when it is invoked.

Extended Description



When an attacker can directly invoke the blocking code, or the blocking code can be affected by environmental conditions that can be influenced by an attacker, then this can lead to a denial of service by causing unexpected hang or freeze of the code. Examples of blocking code might be an expensive computation or calling blocking library calls, such as those that perform exclusive file operations or require a successful network operation.

Due to limitations in multi-thread models, single-threaded models are used to overcome the resource constraints that are caused by using many threads. In such a model, all code should generally be non-blocking. If blocking code is called, then the event loop will effectively be stopped, which can be undesirable or dangerous. Such models are used in Python asyncio, Vert.x, and Node.js, or other custom event loop code.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf		834	Excessive Iteration	1754
CanPrecede		835	Loop with Unreachable Exit Condition ('Infinite Loop')	1757

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

Nature	Type	ID	Name	Page
MemberOf		557	Concurrency Issues	2329

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) <i>An unexpected call to blocking code can trigger an infinite loop, or a large loop that causes the software to pause and wait indefinitely.</i>	

Potential Mitigations

Phase: Implementation


Generally speaking, blocking calls should be replaced with non-blocking alternatives that can be used asynchronously. Expensive computations should be passed off to worker threads, although the correct approach depends on the framework being used.

Phase: Implementation

For expensive computations, consider breaking them up into multiple smaller computations. Refer to the documentation of the framework being used for guidance.

MemberOf Relationships

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

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

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

CWE-1323: Improper Management of Sensitive Trace Data

Weakness ID : 1323

Structure : Simple

Abstraction : Base

Description

Trace data collected from several sources on the System-on-Chip (SoC) is stored in unprotected locations or transported to untrusted agents.

Extended Description

To facilitate verification of complex System-on-Chip (SoC) designs, SoC integrators add specific IP blocks that trace the SoC's internal signals in real-time. This infrastructure enables observability of the SoC's internal behavior, validation of its functional design, and detection of hardware and software bugs. Such tracing IP blocks collect traces from several sources on the SoC including the CPU, crypto coprocessors, and on-chip fabrics. Traces collected from these sources are then aggregated inside trace IP block and forwarded to trace sinks, such as debug-trace ports that facilitate debugging by external hardware and software debuggers.

Since these traces are collected from several security-sensitive sources, they must be protected against untrusted debuggers. If they are stored in unprotected memory, an untrusted software

debugger can access these traces and extract secret information. Additionally, if security-sensitive traces are not tagged as secure, an untrusted hardware debugger might access them to extract confidential information.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	680

Applicable Platforms

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

Operating_System : Not OS-Specific (*Prevalence = Undetermined*)

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

Technology : System on Chip (*Prevalence = Undetermined*)

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory <i>An adversary can read secret values if they are captured in debug traces and stored unsafely.</i>	

Potential Mitigations

Phase: Implementation

Tag traces to indicate owner and debugging privilege level (designer, OEM, or end user) needed to access that trace.

Demonstrative Examples

Example 1:

In a SoC, traces generated from sources include security-sensitive IP blocks such as CPU (with tracing information such as instructions executed and memory operands), on-chip fabric (e.g., memory-transfer signals, transaction type and destination, and on-chip-firewall-error signals), power-management IP blocks (e.g., clock- and power-gating signals), and cryptographic coprocessors (e.g., cryptographic keys and intermediate values of crypto operations), among other non-security-sensitive IP blocks including timers and other functional blocks. The collected traces are then forwarded to the debug and trace interface used by the external hardware debugger.

Example Language: Other

(Bad)

The traces do not have any privilege level attached to them. All collected traces can be viewed by any debugger (i.e., SoC designer, OEM debugger, or end user).

Example Language: Other

(Good)

Some of the traces are SoC-design-house secrets, while some are OEM secrets. Few are end-user secrets and the rest are not security-sensitive. Tag all traces with the appropriate, privilege level at the source. The bits indicating the privilege level must be immutable in their transit from trace source to the final, trace sink. Debugger privilege level must be checked before providing access to traces.

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2474
MemberOf	C	1396	Comprehensive Categorization: Access Control	1400	2519

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
150	Collect Data from Common Resource Locations
167	White Box Reverse Engineering
545	Pull Data from System Resources

References

[REF-1150]Jerry Backer, David Hely and Ramesh Karri. "Secure design-for-debug for Systems-on-Chip". 2015 October 6. < <https://ieeexplore.ieee.org/document/7342418> >.

[REF-1151]Jerry Backer, David Hely and Ramesh Karri. "Secure and Flexible Trace-Based Debugging of Systems-on-Chip". 2016 December. < <https://dl.acm.org/doi/pdf/10.1145/2994601> >.2023-04-07.

CWE-1325: Improperly Controlled Sequential Memory Allocation

Weakness ID : 1325

Structure : Simple

Abstraction : Base

Description

The product manages a group of objects or resources and performs a separate memory allocation for each object, but it does not properly limit the total amount of memory that is consumed by all of the combined objects.

Extended Description

While the product might limit the amount of memory that is allocated in a single operation for a single object (such as a malloc of an array), if an attacker can cause multiple objects to be allocated in separate operations, then this might cause higher total memory consumption than the developer intended, leading to a denial of service.

Relationships

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

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

Nature	Type	ID	Name	Page
ChildOf	B	770	Allocation of Resources Without Limits or Throttling	1613
PeerOf	V	789	Memory Allocation with Excessive Size Value	1674
CanPrecede	B	476	NULL Pointer Dereference	1132

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C (Prevalence = Undetermined)

Language : C++ (Prevalence = Undetermined)

Language : Not Language-Specific (Prevalence = Undetermined)

Alternate Terms

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

Common Consequences

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

Potential Mitigations

Phase: Implementation

Ensure multiple allocations of the same kind of object are properly tracked - possibly across multiple sessions, requests, or messages. Define an appropriate strategy for handling requests that exceed the limit, and consider supporting a configuration option so that the administrator can extend the amount of memory to be used if necessary.

Phase: Operation

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

Demonstrative Examples

Example 1:

This example contains a small allocation of stack memory. When the program was first constructed, the number of times this memory was allocated was probably inconsequential and presented no problem. Over time, as the number of objects in the database grow, the number of allocations will grow - eventually consuming the available stack, i.e. "stack exhaustion." An attacker who is able to add elements to the database could cause stack exhaustion more rapidly than assumed by the developer.

Example Language: C

(Bad)

```
// Gets the size from the number of objects in a database, which over time can conceivably get very large
int end_limit = get_nmbr_obj_from_db();
int i;
int *base = NULL;
int *p = base;
for (i = 0; i < end_limit; i++)
{
    *p = alloca(sizeof(int *)); // Allocate memory on the stack
    p = *p; /// Point to the next location to be saved
}
```

Since this uses `alloca()`, it allocates memory directly on the stack. If `end_limit` is large enough, then the stack can be entirely consumed.

Observed Examples

Reference	Description
CVE-2020-36049	JavaScript-based packet decoder uses concatenation of many small strings, causing out-of-memory (OOM) condition https://www.cve.org/CVERecord?id=CVE-2020-36049
CVE-2019-20176	Product allocates a new buffer on the stack for each file in a directory, allowing stack exhaustion https://www.cve.org/CVERecord?id=CVE-2019-20176
CVE-2013-1591	Chain: an integer overflow (CWE-190) in the image size calculation causes an infinite loop (CWE-835) which sequentially allocates buffers without limits (CWE-1325) until the stack is full. https://www.cve.org/CVERecord?id=CVE-2013-1591

MemberOf Relationships

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

Nature	Type	ID	Name	V	Page
MemberOf		1416	Comprehensive Categorization: Resource Lifecycle Management	1400	2545

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
130	Excessive Allocation

CWE-1326: Missing Immutable Root of Trust in Hardware

Weakness ID : 1326

Structure : Simple

Abstraction : Base

Description

A missing immutable root of trust in the hardware results in the ability to bypass secure boot or execute untrusted or adversarial boot code.

Extended Description

A System-on-Chip (SoC) implements secure boot by verifying or authenticating signed boot code. The signing of the code is achieved by an entity that the SoC trusts. Before executing the boot code, the SoC verifies that the code or the public key with which the code has been signed has not been tampered with. The other data upon which the SoC depends are system-hardware settings in fuses such as whether "Secure Boot is enabled". These data play a crucial role in establishing a Root of Trust (RoT) to execute secure-boot flows.

One of the many ways RoT is achieved is by storing the code and data in memory or fuses. This memory should be immutable, i.e., once the RoT is programmed/provisioned in memory, that memory should be locked and prevented from further programming or writes. If the memory contents (i.e., RoT) are mutable, then an adversary can modify the RoT to execute their choice of code, resulting in a compromised secure boot.

Note that, for components like ROM, secure patching/update features should be supported to allow authenticated and authorized updates in the 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, MemberOf and give insight to