

Security report

DAVINES PROJECT



December 16, 2022

INDIVIDUAL PROJECT S3

DYULGERYAN, V.

Contents

[OWASP 2](#_Toc122688355)

[Top 10 OWASP 2](#_Toc122688356)

[A01:2021 – Broken Access Control: 2](#_Toc122688357)

[A02:2021 – Cryptographic Failures: 2](#_Toc122688358)

[A03:2021 – Injection: 2](#_Toc122688359)

[A04:2021 – Insecure Design: 2](#_Toc122688360)

[A05:2021 – Security Misconfiguration: 3](#_Toc122688361)

[A06:2021 – Vulnerable and Outdated Components: 3](#_Toc122688362)

[A07:2021 – Identification and Authentication Failures: 3](#_Toc122688363)

[A08:2021 – Software and Data Integrity Failures: 4](#_Toc122688364)

[A09:2021 – Security Logging and Monitoring Failures: 4](#_Toc122688365)

[A10:2021 – Server-Side Request Forgery (SSRF): 4](#_Toc122688366)

[Security Risks Analyze 5](#_Toc122688367)

[Conclusion & Reasoning 6](#_Toc122688368)

[Tops: 6](#_Toc122688369)

[Downs 6](#_Toc122688370)

# OWASP

The Open Web Application Security Project is an online community that produces freely available articles, methodologies, documentation, tools, and technologies in the field of web application security. The OWASP provides free and open resources. It is led by a non-profit called The OWASP Foundation.

# Top 10 OWASP

## A01:2021 – Broken Access Control:

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 the user's limits.

A02:2021 – Cryptographic Failures:  
 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, mainly 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).

## A03:2021 – Injection:

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

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, 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. Automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs is strongly encouraged. Organizations can include static (SAST), dynamic (DAST), and interactive (IAST) application security testing tools into the CI/CD pipeline to identify introduced injection flaws before production deployment.

## A04:2021 – Insecure Design:

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.

## A05:2021 – Security Misconfiguration:

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 A06:2021-Vulnerable and Outdated Components).

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

## A06:2021 – Vulnerable and Outdated Components:

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, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
* If software developers do not test the compatibility of updated, upgraded, or patched libraries.
* If you do not secure the components’ configurations (see A05:2021-Security Misconfiguration).

## A07:2021 – Identification and Authentication Failures:

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

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

## A08:2021 – Software and Data Integrity Failures:

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

## A09:2021 – Security Logging and Monitoring Failures:

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).

## A10:2021 – Server-Side Request Forgery (SSRF):

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.

# Security Risks Analyze

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OWASP | Likelihood | Impact | Risk | Actions possible | Planned |
| A01: Broken access control | Very unlikely | Severe | Low | Implementing filter chain and ant matchers for security/access token is encoded which provides security to this matter | Yes |
| A02: Cryptographic  failure | Very unlikely | Severe | Low | Passwords are encrypted and any user information is encrypted in the tokens | N/A |
| A03: Injection | Very unlikely | Severe | Low | More validations in front-end before sending data to backend | N/A |
| A04: Insecure Design | Medium | Severe | Moderate | More unit and integration testing on bad scenarios | Yes |
| A05: Security Misconfiguration | Very unlikely | Severe | Low | N/A | N/A |
| A06: Vulnerable and Outdated Components | Very unlikely | Severe | Low | Using components which are not outdated and still updated | Yes |
| A07: Identification and Authentication Failures | Very unlikely | Severe | Low | Using encrypted access token | Yes |
| A08: Software and Data Integrity Failures | Very unlikely | Severe | Low | Using only trusted plugins and nodes, with good documentation. | Yes |
| A09: Security Logging and Monitoring Failures | Moderate | Moderate/  Severe | High | N/A | N/A out of scope |
| A10: Server-Side Request Forgery (SSRF) | Very likely | Severe | High | To not send raw data and to be aware of the URL consistency. Sanitize and validate all client-supplied input data. | N/A out of scope |

# Conclusion & Reasoning

### Ups:

The application is sufficiently secured in ways that users that are not authorized can’t break the access control (to perform an action that they are not allowed, or access a page that they are not authorized to), sensitive information such as passwords are encrypted, client-input is validated before sending it to the back end, using new and updated components which are not vulnerable, users take access tokens to authenticate themselves so that they can perform the actions that they are authorized to.

Dependencies in the front-end are trusted and pipeline is implemented so that it notifies the status of the new commits project if something breaks that it’s going to be repaired as fast as possible before release. No data is sent raw to the client side and the URL filter is configured so that only authorized users can access those URLS. Access Token is kept in local storage, and it’s implemented with JWT tokens which prevents CSRF (Cross Site Request Forgery) attacks that can be used on any user.

Downs:

For this application we should do more Unit, Integration and bad scenarios tests and increase the overall coverage of the tests on the functionality of the back end, and cypress for E2E (end to end tests) for the front-end. Not everywhere the client-input is validated which brings security risks of injections etc. The Access Token is kept in local storage which make the application vulnerable to XSS (Cross Site Scripting) attacks that can steal the Access Token of a user and the attacker can take the identity of that user and perform actions on the user’s behalf or access pages/actions that he is not authorized to.