OWASP Document

Feeler

1 INDEX

2 INTRODUCTION

This document will define which of the security vulnerabilities from the OWASP top ten are relevant for this project, and in which way they can be prevented.

2.1 REFERENCES

Where needed, this document will reference any external documents, but all documentation and code can be seen here: documentation, frontend code, backend code.

3 VULNERABILITIES

3.1 INJECTION

3.1.1 Description

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

3.1.2 Preventing

Preventing injection requires keeping data separate from commands and queries.

A good example for this is Object Relational Mapping.

3.2 BROKEN AUTHENTICATION

3.2.1 Description

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

3.2.2 Preventing

- Implementing multi-factor authentication
- Do not ship or deploy with default credentials.
- Implement weak-password checks.
- Align password length.
- Prevent enumeration attacks by using the same message for all outcomes.
- Limit or increasingly delay on failed login attempts.
- Use server-side, secure, built-in session manager that generates a new random session ID after login. Session ID's should not be in the URL and should be invalidated after logout, idle, and absolute timeouts.

3.3 SENSITIVE DATA EXPOSURE

3.3.1 Description

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

3.3.2 Preventing

- Identify and classify sensitive data.
- Apply controls per classification.
- Don't store sensitive data when it's not needed, and discard it as soon as possible.

- Encrypt all sensitive data at rest.
- Ensure up-to-date strong algorithms.
- Encrypt all data in transit.
- Disable caching for response that contain sensitive data.
- Store passwords using salted hashing functions with a delay factor.
- Verify independently the effectiveness of configuration and settings.

3.4 Broken Access Control

3.4.1 Description

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

3.4.2 Preventing

- Deny by default unless a public resource.
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- WT tokens should be invalidated on the server after logout.

3.5 Security Misconfiguration

3.5.1 Description

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

3.5.2 Preventing

- 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 a new secure environment.
- 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.
- A segmented application architecture.
- Sending security directives to clients, e.g. Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

3.6 USING COMPONENTS WITH KNOWN VULNERABILITIES

3.6.1 Description

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

3.6.2 Preventing

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously Inventory the versions of components.
- Only obtain components from official sources over secure links.
- Monitor for unmaintained libraries.

3.7 INSUFFICIENT LOGGING & MONITORING

3.7.1 Description

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

3.7.2 Preventing

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.