GLUB TRAINING MANAGER

**APPLICATION SECURITY**

Cristin Rusnac | Fontys University of Applied Science

# Injection

Threat

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.

Application Case

Preventing injection requires keeping data separate from commands and queries. In the case of this specific application the preferred option is applied by using a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface.

# Broken Authentication

Threat

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.

Application Case

In the application case the token generated during authorization is stored locally on the users machine and is removed after sign out or after the user is inactive for a specific period of time.

# Sensitive Data Exposure

Threat

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.

Application Case

This risk is not applicable for the application since any API call requires a token that is generated during authorization. Any API calls without a token will just return a 401 Unauthorized Status.

# XML External Entities

Threat

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

Application Case

Not applicable for the application, JSON is being used.

# Broken Access Control

Threat

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.

Application Case

The application is not at risk since you cant bypass the access control, any API call must contain an authorization token that is generated user wise and is being used as header otherwise 401 Status is returned.

# Security Misconfiguration

Threat

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.

Application Case

Not applicable for the application, no unnecessary ports, services, pages, accounts are enabled or installed and the default credentials were modified.

# Cross-Site Scripting

Threat

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Application Case

May be a risk for the application but only on the clients side and as long as his web browser tab is opened or the user is authenticated in the application.

# Insecure Deserialization

Threat

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

Application Case

Deserialization is not due to the same reasons as for the Access Control case, any API call must contain an authorization token that is generated user wise and is being used as header otherwise 401 Status is returned.

# Using Components with Known Vulnerabilities

Threat

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.

Application Case

Not applicable for the application due to the fact that the app is meant to store only short-term data with secured components and access defined by authorization.

# Insufficient Logging & Monitoring

Threat

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.

Application Case

The logging can be implemented at a later stage of development when the application has more implementations and functionalities, and has a considerable amount of public users.