Radolina Petrova

Security report

Table of Contents

[Analyzation of project 2](#_Toc122610467)

[Reasoning 3](#_Toc122610468)

[A1: Broken access control 3](#_Toc122610469)

[A2: Cryptographic failures 3](#_Toc122610470)

[A3: Injection 3](#_Toc122610471)

[A4: Insecure design 4](#_Toc122610472)

[A6: Vulnerable and outdated components 4](#_Toc122610473)

[A7: Identification and Authentication failures 4](#_Toc122610474)

# Analyzation of project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Likelihood | Impact | Risk | Action possible |
| A1: Broken access control | Unlikely | Severe | Low | Double-checking the identity of the user before executing actions |
| A2: Cryptographic failures | Unlikely | Severe | Low | Hashing and decoding of sensitive information |
| A3: Injection | Unlikely | Severe | Low | Usage of prepared statements only |
| A4: Insecure design | Likely | Severe | High |  |
| A5: Security Misconfiguration |  |  |  |  |
| A6: Vulnerable and outdated components | Unlikely | Moderate | Low | Removal of unused elements and continuous inventory of the components |
| A7: Identification and Authorization failures | Unlikely | Severe | Low | Using secure way of authorization and authentication |
| A8: Software and data integrity failures | Likely | Severe | High | Usage of digital signatures |
| A9: Security logging and monitoring failures | Likely | Moderate | High | Input validation |
| A10: Severe side request forgery | Likely | Severe | High | Implementing defence in depth |

# Reasoning

## A1: Broken access control

The broken access control principle enforces, that the user cannot act outside of their limits. They are authorized to perform certain actions and should not be able to force browse to a page, where they are not authorized for. Bypassing control checks by parameter tampering or metadata manipulation should also be prevented, in order for unauthorized users to act as authorized ones.

The impact on my application would not be that severe, because the data, that I am saving does not have that great of an importance and the most damage that can be done is deleting or modifying certain user’s musical content. The chance of this affecting my application is very unlikely, because before executing certain actions, the identity of the user is verified by the JWT secret, which cannot be tempered. Furthermore, force browsing is prevented by checking the identity of the user from the token, before displaying the information.

## A2: Cryptographic failures

The cryptographic failures principle enforces, that sensitive data should not be transferred in plain text. Not only the transmitting of data, but also its storage requires extra protection.

The impact on my application would again be not that severe, because the data does not have that great of an importance. The likeliness of it affecting my application is also not that big, because all sensitive data is decoded, when being transferred or hashed, when being saved.

## A3: Injection

The injections principle enforces, that data an application is vulnerable to these attacks, when user-supplied data is not validated, dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter and hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

The impact on my application could be severe because whole sets of data can be deleted. The likeliness of this affecting my application is small, because I am using JPA Repository methods and for the more complex queries, which depend on the user input, I use named queries.

## A4: Insecure design

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.

## A6: Vulnerable and outdated components

Usage of older versions of components, where there are deprecated methods. Not scanning for vulnerabilities regularly. Not testing the compatibility of updated, upgraded, or patched libraries.

## A7: Identification and Authentication failures

The usage of plain text for saving sensitive data can lead to this kind of failure. Missing or ineffective multi-factor authentication or exposing session identifier in the URL.

## A8: Software and data integrity

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. An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.

## A10: Server-side request forgery

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. 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.