OWASP Report

Job Offers Website

Name: Mohammed Aljader

Student number: 4219090

Semester 3 individual project

Contents

	Likelyhood	*Impact*	*Risk*	*Actions*	*Planned*
A01:2021-Broken Access Control	Likely	Severe	High	Authorization is required for all non-public resources, access to such pages is denied by default for non-authenticated users.	Make sure to check the authorizatio n and thus changing the content of the page accordingly.
A02:2021-Cryptographi c Failures	Low	Low	Low	Usage of proper key management. (UUID's)	Using UUID as id instead of using the auto increment id
A03:2021-Injection	Very unlikely	Moderate	Very Low	By using Hibernate ORM and Spring Data JPA methods with parameterized queries prevents SQL injections.	Already used Spring Data JPA in the project.
A04:2021-Insecure Design	Low	Moderate	Low	Write unit and integration tests to validate all critical flows.	Unit tests, Integration tests

A05:2021-Security Misconfiguration	Very unlikely	Severe	Low	Sending security directives to clients	make sure that the stack trace is not returned to the user, but instead an error message or alert will be shown.
A06:2021-Vulnerable and Outdated Components	Very unlikely	Moderate	Low	Remove unused dependencies continuously inventory versions of applications	N/A
A07:2021-Identification and Authentication Failures	Likely	Moderate	Moderate	Multi-factor authentication	N/A
A08:2021-Software and Data Integrity Failures	Unlikely	Severe	Moderate	Ensure that CI/CD pipeline has proper segregation, configuration, and access control to ensure integrity of the code.	N/A

A09:2021-Security Logging and Monitoring Failures	Very likely	Severe	High	For example, when an attacker uses a scan for trying common passwords. They can thus takeover all accounts using this password.	Effective monitoring and alerting should be established.
A10:2021-Server-Side Request Forgery	Very likely	Severe	High	In the application layer, all client-supplied data should be sanitized and validated. The URL schema, port and destination be enforced	N/A

Explanation

1- Broken Access Control:

A broken access control is a type of vulnerability that allows attackers to gain access to restricted resources. These resources mostly fall into two categories: sensitive data, which should only be accessed by select users, and functions that can modify data on the webserver, or even modify the server's functionality.

Examples of broken access control:

- **Insecure ID's:** To prevent that kind of attacks. JobOffers website uses UUID. A UUID is more difficult to remember than a simple number. So, someone passing by, having a glance at your screen would not be able to know what file number you are working on.

Hacking into a system to retrieve information is more difficult if you don't know what you're looking for or where to look.

- **Forced browsing:** I'll make sure to check the authorization and thus changing the content of the page accordingly.

2- Cryptographic Failures:

Cryptographic Failures are when the cause is not described, leading to data exposure and the inability to find the source of the problem. It is not a risk that applies to my web application.

This is because no personal data is stored. Only the password, name and email to JWT/later using Auth0. Which makes it more secure and thus prevents cryptographic Failures, such as health records and credit card information.

3- Injection:

Injection defects such as SQL, NoSQL, or Command occur when untrusted data is sent to a server/database, as part of a command or query. An attacker's data is capable of causing the interpreter to perform unwanted commands, or even access unauthorized data.

I am using Hibernate ORM and Spring Data JPA methods with parameterized queries prevents SQL injections.

4- Insecure Design:

Due to weak use of secure design patterns, principles, and reference architectures, serious weaknesses and flaws stay under the surface no matter how perfectly we implement a software.

To prevent that kind of risks I am using SonarQube which reduces the risk of software development within a very short amount of time. It detects bugs in the code automatically and alerts me to fix them before rolling it out for production. SonarQube also highlights the complex areas of code that are less covered by unit tests.

Additionally, to ensure the quality of the code I wrote, I tested all layers of my code using unit tests, and after that, I tested the entire system (integration testing) using Postman/Newman. finally. I tested all Frontend with Cypress.

5- Security Misconfiguration:

Security misconfiguration is a risk that will give unauthorized access to system data. In order to prevent this risk, I make sure that the stack trace is not returned to the user, but instead an error message or alert will be shown.

6- Vulnerable and Outdated Components:

The platforms, frameworks and dependencies I use are upgraded regularly. Also no vulnerable software is used. All the unused dependencies, features, files and documentation are removed from the folder which prevents this risk.

7- Identification and Authentication Failures:

Identification and Authentication Failures imply that for example a person who is not authorized can gain access to an admin account. My application does make use of different roles (Admin and User), so this risk is applicable. But using 0Auth and UUID will fix this problem. Also, the password of the user should be strong and to check that I'll make sure that the length of the password should be at least 6 in both frontend and backend.

8- Software and Data Integrity Failures:

Software and Data Integrity Failures can happen when data is turned into an object, for example by using JSON. This risk does apply to the application. To safely prevent this from happening, untrusted sources should not be accepted to use serialization mediums.

9- Security Logging and Monitoring Failures:

For example when an attackers uses a scan for trying common passwords. They can thus takeover all accounts using this password. For the other users, this scan leaves only one false login behind. To prevent this from happening effective monitoring and alerting should be established.

10 - Server-Side Request Forgery:

Making unauthorized HTTP request (potentially posting data that may break into the security architecture, etc.) is a very common attack that application have to deal with. JobOffers uses JWT's to avoid such attacks, however, due to their storge in the local cache of the user, this could be easily altered.