OWASP Top 10 report

Sam Philipsen

Table of contents

Table 3
Reasoning 4
A1- Broken Access Control 4
A2- Cryptographic failures 4
A3- Injection 4
A4- Insecure design 4
A5- Security misconfiguration 5
A6- Vulnerable and outdated components 5
A7- Identification and authentication failures 5
A8- Software and data integrity failures 6
A9- Security logging and monitoring failures 6
A10- Server-side request forgery 6
Conclusion 7

Table

	Likelihood	Impact	Risk	Actions possible	Planned
A1: Broken access control	Likely	Severe	High	Roles added to users	Yes
A2: Cryptographic failures	Not likely	Moderate	Low	N/A	No
A3: Injection	Less likely	Severe	Low	Look into further protection against injection	No
A4: Insecure design	Less likely	Severe	Moderate	Look at how other similar programs are set up	No
A5: Security Misconfiguration	Not sure	Moderate	Moderate	Proper error handling and a hosted database without default accounts	Yes
A6: Vulnerable and outdated components	Not sure	Small	Low	Keeping up to date with new security updates and techniques, and applying it to the application	No
A7: Identification and authentication failures	Very likely	Small	Moderate	Adding secured password recovery, and password requirements should already help.	Yes
A8: Software and data integrity failures	Not very likely	Severe	Low	Frequently monitor updates to software in application	No
A9: Security logging and monitoring failures	Very likely	Moderate	High	Add IP or device information logging upon login or upon performing sensitive actions	No
A10: Server side request forgery (SSFR)	Less likely	Severe	Moderate	Further sanitize data sending out, because currently the DTO's are filled almost directly with data	No

				which can be tampered with	

Reasoning

A1- Broken Access Control

Broken access control means that users cannot act outside of their intended permissions. This means that a user with standard user permissions (such as getting their own username), cannot perform actions they are not permitted to. This includes but is not limited to, deleting, modifying or retrieving user data outside of the intended user's limits. Think of a user destroying another user's data by calling the appropriate endpoint without anything limiting them.

A way to prevent this is to implement a JWT authorization token. This makes it so a user has to log in first before accessing the rest of the website. The token has to be used for any future API calls, or a user cannot use the website further. This token expires after a while, so it cannot be re-used. The creating and handling of this token is a public library, however, which means it might be easier for hackers to create their own token.

While this does fix part of the issue, there is no role control. Anyone that is logged in has full access to the API. They can essentially edit everybody's information.

A2- Cryptographic failures

Cryptographic failures describes the failure to protect sensitive data like passwords, credit card numbers, health records and more. Sending sensitive data over the internet is a serious security risk. If any important information like passwords and credit card information is caught by malicious users, they can access restricted information.

The application prevents this by encrypting the password and sending it, encrypted, to the API and back. It is also stored encrypted in the database, so even if someone has access to it they cannot easily get any user's passwords. It uses the BCrypt password encoder for this.

A3- Injection

SQL injection is when you put a user's input directly into an SQL query. This means the user can also put their own SQL statements in there resulting in direct data manipulation from the database.

To prevent this, the application uses the Jpa repository interface whose methods already deal with SQL injection.

A4- Insecure design

Insecure design means that your application is not designed with secure and optimized design in mind.

The application has been designed while looking at other well designed and secure applications. It uses tokens for authorization, passwords, DTO models for passing data to and from the client and dependency injection/inversion.

A5- Security misconfiguration

The application does not have any default values enabled, and all the security systems in place have been configured and set up properly.

Furthermore, there is no unused code in the application so it should not be an entry point for malicious users.

Lastly, the application uses stable and secure software though not the newest version is installed.

The application has default accounts, but this is only because the database is in-memory and it slows development time down if a developer has to add a user every time the application has restarted. This will be removed once the application has been fully released.

On top of that, the application's error handling shows the full error message at the moment which is also a security risk. If a user triggers an error they can see a part of the code and where it took place, which is a security vulnerability.

A6- Vulnerable and outdated components

The application uses stable and secure versions of any libraries, components or other software as far as they are available. The component's are also properly configured as far as they need to be.

I am however not subscribed to any security bulletins, nor do I scan the software used for vulnerabilities. To confirm the validity of the security of the application I am going off of the most popular opinions of people on forums on the internet.

A7- Identification and authentication failures

Application has no non-default protection against automated- or brute fore attacks.

It also does not have any password requirements, so any weak password is permitted.

There is no weak or ineffective credential recovery, because there is no credential recovery at all.

The passwords are stored strongly encrypted on a database, and any password handling is handled on the backend. The password is never exposed to the user directly, or sent over the internet.

User data (without passwords) and authentication tokens are stored on the user's local storage and can be directly copied, edited or removed which is then saved on the database. This is of course a major security risk and will be adjusted.

A8- Software and data integrity failures

The application uses a lot of external libraries. All of these are from trusted sources that have millions of downloads and are almost universally acknowledged as trusted libraries. This does not completely mean there can't be malicious code uploaded by a user on the other end. There are no tools or software to monitor these changes.

A9- Security logging and monitoring failures

The application currently does not have any type of security logging in place that can detect security breaches. No logs are stored anywhere apart from the default that Spring provides, which do not produce any logs that would help in a security failure situation.

It also does not detect for any active attacks that can occur on the application.

A10- Server-side request forgery

To prevent SSRF, the application sends and receives sanitized data in the form of DTO's. No raw data is sent to and from the client, as it is all stuffed into a DTO object that only contains the information that it has to receive/send.

Conclusion

The application this report is about is very small at the moment. While it's small, it still has a ton of security strapped on it. This security is currently at a good level, though there are some major security issues. It is important to fix these big issues as soon as possible to avoid important security leaking or data tampering from happening. It's especially important to keep upping this security as the application grows bigger and handles more secure data.

The major security risks should be fixed, but for the rest the application is well protected against most of the OWASP Top 10 items.