OWASP Analysis

Smoke-it web shop

Maarten Hormes

S3-CB03

2021

Contents

Introduction 3

OWASP analysis table 3

Table reasoning 4

Introduction

The OWASP (Open Web Application Security Project) top 10 are the 10 most common web application security risks of each year. To ensure building a secure web application, it is important to look into these security risks and analyze how the smoke-it application is protected against these risks.

The file will contain a table that indicates how likely these risks are to occur in the application. Combined with the impact it would have, we can indicate how risky these topics are for our situation.

After the table there will be a reasoning behind the values inserted in the above mentioned table

OWASP analysis table

	Likelihood	Impact	Risk	Actions possible	Planned
A1: Broken access control	Very unlikely	Significant	Moderate	N/A, fixed	N/A
A2: Cryptographic failures	Likely	Moderate	Moderate	Encrypt all stored personal information. Encrypt data send over the wire	No
A3: Injection	Possible	Severe	High	Validate all user input	Yes
A4: Insecure design	Unlikely	Minor	Low	Test front-end. Get more knowledge about spring security	Yes, testing is planned
A5: Security misconfiguration	Possible	Significant	Moderate	Setup CRSF in security. Setup error handling in front-end	No, risk accepted
A6: Vulnerable and outdated components	Very unlikely	Moderate	Low	N/A, fixed	N/A
A7: Identification and Authentication failures	Possible	Significant	Moderate	Setup password standards, multi-step authentication, remove default credentials	Yes, default credentials will be removed
A8: Software and data integrity failures	Unlikely	Moderate	Moderate	N/A, fixes	No, fixed
A9: Security logging and monitoring failures	Very likely	Severe	High	Add logging to the application. Setup monitoring tool to detect unexpected logging behavior	Yes
A10: Server side request forgery	Possible	Moderate	Moderate	Validate user input, stop sending raw data over the wire	No, risk accepted (low priority)

Table reasoning

A1: Broken access control

Access control forces policy such that users cannot act outside of their intended permissions. To ensure this is not a big risk in my application, there has been filtering setup on the router of the FE. This prevents the user from accessing a URL, which would include a page normally not available for them, that they do not have the right to visit.

On the BE side if the application, I have setup a authentication and authorization process using Spring security. This checks if the user has permission to access the called endpoint. This prevents users from performing admin actions, which they obviously should not be able to perform.

A2: Cryptographic failures

Certain API calls can expose personal and sensitive information to users not intended to have this information. In the smoke-it application there is not a lot of sensitive information being stored. Login details and the shipping information is the most vulnerable information currently stored in the application. The passwords are being encrypted by the Spring security, making them less vulnerable for a leak.

The shipping information for orders is being send from BE to FE without any type of encryption of security. These details are still at risk for a potential attack or data leak.

A3: Injection

Unvalidated user input can be dangerous. This is because people with malicious intent could inject a SQL query or JavaScript method into your application (There are way more types of injection not mentioned).

The application does not use any query's that include user input. This makes me conclude that the database is secured against SQL or ORM injection.

The BE does not check all user input before using it to perform the requested actions. This can be dangerous since there are way more types of injection that can cause the application to behave in an unintended way.

A4: Insecure design

Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. To ensure my application has a secure design, there have been multiple steps taken.

The first one being the tests, that can be run to ensure that the application works. Of course we cannot discover every bug with testing, but this should cover most of the code. Testing can be taken to a very far point. This means that my application can always use an improvement when it comes to testing.

Next to testing, I have used the spring security library to setup my authentication and authorization process. This should be a secure library.

Currently there is still a risk of insecure design being present in my application, since I do not have full knowledge of how spring security works. Next to that, my application is not yet fully tested to ensure that the application is resistant to all threats.

A5: Security misconfiguration

Configuring the security of any application is an important step. This is both ensuring that stuff like CORS and CRSF are setup correctly and ensuring the application does not share information not meant to share.

In the BE, the CORS has been setup to only accept request coming from my local host, since this is the only place where my front-end will be hosted now. The CRSF is not yet configured, and will be an accepted risk at this moment.

For error handling and sharing secure details, the application still needs some improvement. There is no error handling for when the FE crashes, meaning that all information will be shared with the user. Next to that, there still are ‘’admin, admin’’ accounts. This of course is not secure.

A6: Vulnerable and outdated components

This can become a risk when components being used in your application are outdated or no longer supported. This includes the OS, server, database and a lot more.

My web application does not use any third party software or dependencies outside of the Spring and React framework. These are both up to date and well supported.

Next to these frameworks, I made sure to remove all unused dependencies and documents.

A7: Identification and Authentication failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks.

There would be multiple possible options to prevent this risk. Being implementing weak password check, aligning password length, implementing multi-factor authentication, or not shipping the product with default credentials.

The application at this point is still very vulnerable to this risk. None of the above mentioned fixes have been implemented yet, and probably will not be implemented in the future. Of course the default credentials would be removed before shipping the product.

Luckily the application is pretty decently protected against the broken access control risk, resulting in this risk being shoved down the priority list.

A8: Software and data integrity failures

This causes issues when the application would rely on plugins or libraries from untrusted sources. Many of these have an auto update functionality, where malicious updates can be downloaded without being aware of it.

The smoke-it application does not rely on untrusted plugins or library's. The BE is build in the Spring boot framework, which is widely used and trusted in the Java community. The FE is build in the React framework which is trusted as well.

A9: Security logging and monitoring failures

Without logging and monitoring it would be very hard to detect breaches and attacks on the system. Decent logging and monitoring will help to detect, deescalate and respond to active breaches. Besides not logging at all, it is dangerous to only keep the logs locally or if the logs are not monitored for suspicious activity.

My application currently does not log anything. This of course is a big issue, since I would be unable to track an attack on the system. Once a logging system will be setup, it is important to ensure this logging is done in a secure manner, to prevent injection or attacks on the logging or monitoring system.

A10: Server-side request forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. This can be dangerous since the attacker could send a crafted request to an unexpected location, resulting in issues.

The smoke-it application is not really protected against this. Like mentioned before, the application does not test al user data. Next to this, you could protect the application by not sending raw responses to clients. This is currently happening.

Conclusion

Current the Smoke-it application is not that well protected against the top security risks of this year. As seen in the table, there are only 2 risks that have a risk of 'low'. There is the need to improve the security since it seems like at this moment an attacker would not have to go trough a lot of trouble to cause issue in the application. The priority would be on setting up a logging and monitoring system, since this would allows me to see when one of the other risks is coming into play.

The main focus of the last sprint will be on improving the UX and security of the application. I expect the app to be way more secure at the end of the semester.