Security report individual project



# Introduction

The purpose of this report is to go over the top 10 OWASP vulnerabilities and point how the individual project was affected and how (or if) any actions were taken to amend the vulnerabilities.

Please keep in mind that the OWASP top 10 list constantly changes and by the time you read this the order of the vulnerabilities might have changed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Impact if occurs** | **Risk of appearance in new code** | **Status** | **Action taken** |
| A1: broken access control | High | Moderate | App was affected. Users could delete ongoing listings, therefore invalidating any bids placed | Fixed in code. Users cannot modify a listing once it has begun. Only administrators can do so. |
| A2: Cryptographic failure | Severe | Low | Passwords were already hashed. Email could be hashed, but since the app does not feature 1:1 messaging, emails are used as a form of communication | None, no risks found |
| A3: Injection | Severe | Low | Mongoose handles character escaping well. Security tests show no vulnerabilities | None, no risks found |
| A4: Insecure design | High | Low | Some small issues existed with content management flow. | Extensive testing to reveal insecurities |
| A5: Security misconfiguration | Severe | Moderate | Express applications had a cors origin of “\*” set, while also missing some important headers | Properly configured backend headers |
| A6: Vulnerable and outdated components | Moderate | High | Some vulnerable packages always appear when updates are pushed. | Node package manager has a special command that updates to fix vulnerabilities |
| A7: Identification and Authentication Failures | Moderate | Moderate | No MFA and password recovery | Actions are clear, however they are outside of the scope of this semester |
| A8: Software and Data Integrity Failures | High | Moderate | Not affected | None, no risks found |
| A9: Security Logging and Monitoring Failures | Medium | Low | Application had logging on a console level, which is not ideal | Implemented logging through Sentry |
| A10: Server side request forgery | Moderate | Moderate | Microservices are configured for REST and can be accessed from any server-side running application. | Implemented deny-by-default, input sanitizing and DTOs |

# Explanations for the OWASP top 10

**A1**: This is a pretty nasty one, because it happens when code is not written up to quality. The application had some flaws with broken access control, such as users being able to edit/delete listings once they have begun. Issue was fixed in code by disabling this functionality and only allowing users with admin privileges to edit/delete live listings.

**A2** Application was not affected by A2, because sensitive information like user passwords are kept hashed in the system. Although emails are saved in clear text so far, that risk is accepted, as someone would have to leak the whole database for someone to brute force the hashed passwords, which is very unlikely. JWT secrets are also signed with a public/private key and are impossible to forge without them.

**A3** Application was not affected because it uses Mongoose as its ORM, which takes care of injection by default. No actions needed to be taken. The microservices also use body parser, which escapes characters, therefore disallowing injection. ZAP’s security tests showed no injections succeeded.

**A4** is a bit of an abstract vulnerability, but basically involves exploits in a way that the system is intended to function, not how it is implemented. Such exploits are basically a pitfall in the core design of the application. In the case of this project it is entirely possible for someone to bid on an auction and back out of it, refusing to buy it. This is fixable with placing a hold on the bidder’s credit card, which is unfortunately out of the scope of this project.

**A5** Is a combination of small pitfalls that can lead to a big problem, such as error handling revealing stack traces. In this project’s case the ZAP tool revealed that the microservices were leaking some information in the headers such as the “x-powered-by” header, as well as having a CORS origin of “\*” and not having some other security related headers. All of these things were fixed.

A screen shot of a computer

Description automatically generated

**A6:** Node packages get updated constantly and vulnerabilities are always found. For this reason, when installing NPM packages, the user is warned about outdated ones. This issue can be fixed by running “npm audit” which is a regular flow in developing the project.

A black background with white text

Description automatically generated

**A7** Basically has to do with improper treating of user credentials. While this application does not feature MFA and password recovery, these things are out of scope due to more pressing learning outcomes like scalability and dristrubuted data.

**A8:** 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. This application only uses very popular and trusted libraries, and credentials to cloud services are securely stored on the developer’s machine and never committed.

**A9:** Logging is pretty important for applications in production, because it monitors things that are going on. Application was logging errors in the terminal, however that is not sufficient. For this purpose, sentry was implemented as a 3rd party logging system, since the group project this semester already features a custom implementation of logging.

**A10:** This application is mostly safe from SSRF. As a meast to prevent SSRF, all user input was sanitized and, DTOs were implemented and deny-by-default mechanism was enforced in the microservices, where any errors with validating the authentication token lead to a refused request.

# Conclusion

The OWASP top 10 principles definitely revealed some security vulnerabilities with the application that were addressed, since most of them were in scope of this project and the “security by design” learning outcome. As a result, OWASP’s ZAP scanner that runs active security tests and passive scanning revealed no high or severe vulnerabilities. The only warnings found were because of the frontend’s development server, which is not an issue in production.

A screenshot of a computer

Description automatically generated

The full scan in the CI/CD pipeline also shows over 100 passed security tests with 8 warnings, which can be ignored due to the development server that Vite uses for React

A computer screen shot of a black screen

Description automatically generated