Research document

Security Principles & Considerations

Thomas van der Molen
IPS3-DB03

Project Information	
Project members	Thomas van der Molen (4168003)
Project name	Text Adventure
Version	1.2

Table of Contents

Version History

Version	Date	Change
1.0	11-01-2022	Created File
1.1	16-01-2022	Added the OWASP top 10
1.2	17-01-2022	Grammar checks

Introduction

During the development of my project, I have looked into many different systems for security and I have implemented some of them such as JWT and refresh tokens for authentication and authorization.

Within this document I will be going over the top 10 web application security risks supplied by the OWASP foundation.
I will be looking at the 2021 edition of this list.

OWASP top 10

The Open Web Application Security Project (OWASP) Foundation is a foundation aimed at improving the security of software while being non-profit. One of the ways OWASP achieves this is by creating a top 10 list of the most critical security concerns for web applications, this list documents all the risks and explains why the security risk is so important and how it works / can possibly be avoided.

https://owasp.org/www-project-top-ten/

A01 – Broken Access Control

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Thankfully I am very aware of this security risk. Broken access controls can cause mayor issues such as a normal user having access to admin controls, a user accessing other users' data, or even an API sending through unnecessary private information.

The first step in preventing this risk is by setting up a proper CORS policy, as I am using ASP .NET Core this step is surprisingly easy, however sometimes the setup can be confusing to new users of the system.

I currently have my CORS policy set to allow any origin access, this of course is not how I would set it up in the 'real world', however due to some testing and 'free' production hosts I am not able to set the CORS policy to allow a static set of origins.

Furthermore, I also secure my endpoints via JWT or access tokens depending on the purpose of the endpoint. This allows me to validate the user in many ways. Almost all endpoints are only allowed access to with a valid JWT token, when for example a user tries to resume a play session with an adventurer that is not connected to his account (checks via JWT claims) the access will still be denied even if the JWT itself is valid.

A02 – Cryptographic Failures

https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

All API connections inside my project are done using HTTPS connections, this means that the data sent will be encrypted and unreadable by any wrongdoers while the data is in transit.

When any private data such as passwords get stored within my project, they will be hashed, for hashing I use a C# package called Brypt.net – Next, this package is actively supported and highly recommended by many users. This package also allows for hashing with salts to make the hashed passwords even more secure.

Furthermore, within my models I use the Newtonsoft.Json annotations supplied by ASP .NET Core to block sensitive data such as password from being parsed into JSON objects in the first place, this will stop any possible accidental sending of sensitive information.

A03 – Injection

https://owasp.org/Top10/A03_2021-Injection/

Thankfully ASP .NET Core has a lot of implemented systems preventing most common forms of injection from happening.

all incoming data is sanitized before being used anywhere, such as within the ORM or sent to other players via the chat system. All incoming data from API's is also heavily controlled, not allowing for any dynamic or unexpected data to be sent through.

A04 – Insecure Design

https://owasp.org/Top10/A04_2021-Insecure_Design/

For this project I started of by doing research on security principles and what the proper way was to setup a secure web application, because of this I have spent a lot of time integrating already established security designs into my project.

A05 – Security Misconfiguration

https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

As explained in some previous parts. Most of my project is setup with security in mind, this means configuring the normal systems properly to not allow for any unexpected interactions.

However, at some points I knowingly do fail this risk, such as with setting up the most secure CORS policy, however I do this because I know how to configure it properly but for testing and educational purposes I have continuously decided not to.

For most other parts of my project however, security is configured properly. For example, I make sure never to send through any unnecessary data when it comes to exception messages sent to the user.

A06 - Vulnerable and Outdated Components

https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

This issue is one that I have kept a close eye on, especially for my frontend application. This is because when you start using a lot of dependencies (which can happen very fast) the chance of having one dependency fail the needed security becomes very high.

I have kept all my dependencies as up to date as possible, even going as far to fork one dependency and publish my own version which fixes certain user interactions that caused mayor issues.

A07 - Identification and Authentication Failures

https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

This issue I have spent some time on with certain parts, such as said with A02 – Cryptographic Failures.

However, I have not secured my application as much as I could and therefore do fail this security risk. For example I have not implemented any form of brute force prevention, such as reCAPTCHA or implemented any checks for easy passwords (however for the same reasoning as explained in A05 – Security Misconfiguration I have purposefully turned off the registration checks to allow play testers / teachers an easier experience to test out my project during presentations).

A08 – Software and Data Integrity Failures

https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/

Within my project I have made sure that my CI/CD is very secure and does not rely on any other sources then its own.

Within my CI/CD I build and test my application before possibly deploying the changes.

All my tests have been setup in a way where they don't rely on any other application / services to be running, I have also made sure within my project to not rely on any CDN's where possible. For example, my frontend uses bootstrap, however I do not get the bootstrap styles from an outside source (however this might reduce performance slightly because the browser could cache this data from other website, I think it is worth the performance for increased security) I have the bootstrap files locally within my project.

A09 – Security Logging and Monitoring Failures

https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/

Sadly, I have not kept track of this security risk as much as the others.

Besides the security logging that my host provider (in this case Azure free AppService) gives, I have not implemented any security messages or extra checks within this project.

A10 – Server-Side Request Forgery (SSRF)

https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

Within my project I have taken a couple steps to mitigate this risk from OWASP's list of possible prevention techniques, such as sanitizing and validating input data, not sending raw responses and disabling HTTP redirects.

Sources

OWASP:

- https://owasp.org/
- https://www.cloudflare.com/learning/security/threats/owasp-top-10/

External systems mentioned

- https://github.com/BcryptNet/bcrypt.net
- https://azure.microsoft.com/en-us/services/app-service/