OWASP TOP 10 REPORT SHARERECIPE

1. Injection

The preferred option is to use a safe API or by making use of an ORM. Sharerecipe makes use of an ORM so it adheres to the OWASP reccomendations. By making use of an ORM it becomes impossible to use sql injection since we’re not making use of sql directly.

2. Broken Authentication

These are the signs that there is an broken authentication vulnerability.

* Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.

* Permits brute force or other automated attacks.

By making use of keycloak these types of attacks get prevented by enabling a couple of settings within the admin console such as how many times they may try to login before a timeout.

* Permits default, weak, or well-known passwords, such as “Password1” or “admin/admin“.

This is not allowed by the restrictions that are setup within the keycloak register.

* Uses weak or ineffective credential recovery and forgot-password processes, such as “knowledge-based answers”, which cannot be made safe.

We do not make use of this

* Has missing or ineffective multi-factor authentication.

* Does not rotate Session IDs after successful login.

It does rotate session IDs

* Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren’t properly invalidated during logout or a period of inactivity.

After 15 minutes your session will be invalided this is why there is a refresh token to help with this.

3. Sensitive Data Exposure

ShareRecipe does not store any sensitive data. The only data it stores are images, username and email.

Any data is send over HTTPS that is secured by an SSL.

4. XML External Entities

ShareRecipe makes use of json for all communication and object transfers so therer is no chance of this vulnerability appearing.

5. Broken Access Control

ShareRecipe uses an ingress gateway for handling all of the API requests within the project.

* With the exception of public resources, deny by default.

This is standardly how the api reacts since if you don’t have an access token you will be rejected by default.

* Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.

The only mechanism that is being used is the access token that is given by keycloak.

* Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.

By making use of CQRS we are already seperating this logic. Since you create write commands that will already restrict how much they can add. By doing this you are seperating the records from the API and only allow changes you want.

* Unique application business limit requirements should be enforced by domain models.

This is already being handled in my application this is done in the API itself and the domain layer where the final checks are done incase something passed through.

* Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.

By making use of docker only the production environment will be pushed to production nothing else.

* Log access control failures, alert admins when appropriate (e.g. repeated failures).

Currently everything is being logged in the console.

* Rate limit API and controller access to minimize the harm from automated attack tooling.

This is currently not something I’m doing since the authorization already resolves most of these issues for me. Altough I will be warned if people are maliciously spamming my API.

6. Security Misconfigurations

ShareRecipe is setup using environment variables these can be setup using the appsettings.json files or they can be setup within a kubernetes secrets file. Since we’re using kubernetes for the deployment of our application we do not need to fear that these credentials get leaked since all of this authorization is

done within the internal network of kubernetes. This means that they’re impossible to be reached by external attacks.

7. Cross-Site Scripting

By making use of vue and vuetify everything that is inputted into the api will be escaped. This makes it so that no faulty request can be sent.

8. Insecure Deserialization

* Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.

In ShareRecipe we do not implement signatures be we do use validation on all of the messages we have received to prevent tampering.

* Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable

By using our predefined commands they cannot pass through more data than we would allow since they would never be passed through.

* Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.

The project logs any exception that has been caught within the project and will log them to the log.

9. Using Components with Known Vulnerabilities

Within the project ShareRecip there are only a couple of dependencies to third parties and these are all known and trusted sources. If there was a malicious component dependabot from github would warn me of them.

Within the frontend of the project we use npm and vue these have a lot of dependencies to modules but they’re al being checked by dependabot and npm for security vulnerabilities and will help resolve them for you.

10. Insufficient Loggin and Monitoring

Currently the project has insufficient monitoring since this is al being logged to the console instead of a project that can handle all of these logs and display them in one central place.