Security Report

On OWASP top 10 security risks

Onlineyearbook

Table of Contents

Introduction

This document contains the information about how I handle or do not handle the top 10 OWASP vulnerabilities. Each vulnerability will be stated, and a description of its handling is also be included.

OWASP Vulnerabilities

Injection:

In my application sql injections have been handled with the use of JPA and also ensuring no manually written queries have variables as parameters.
JPA helped solve sql injection in the case of adding new data in the database with the use of repo.save(<entity name>) which directly saves an object of the entity

Broken Authentication:

My application backend which manages Authentication, I have handled this issue by including in my JWTRequestfilter configure method a piece of code which prevents my backend from managing the sessions

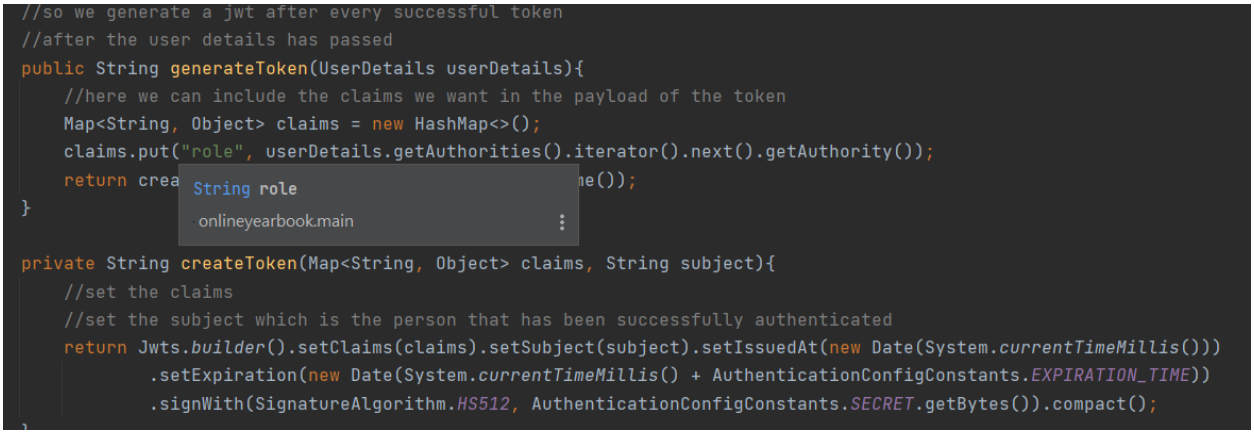
```
        //here we are also saying jwt should not manage sessions therefore we make it stateless
    .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
//here we then add the newly created filter
```

This piece of code states my session policy should be stateless which means my backend will not store any information about the logged in user in its memory.

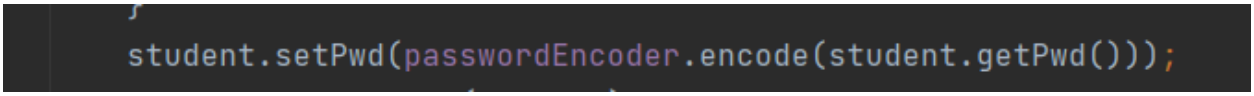
Sensitive Data Exposure

My application handles this by using both password encryption when the user is being signed up and tokenization (JWT) for log in security and securing all api calls

JWT code



Password encryption code

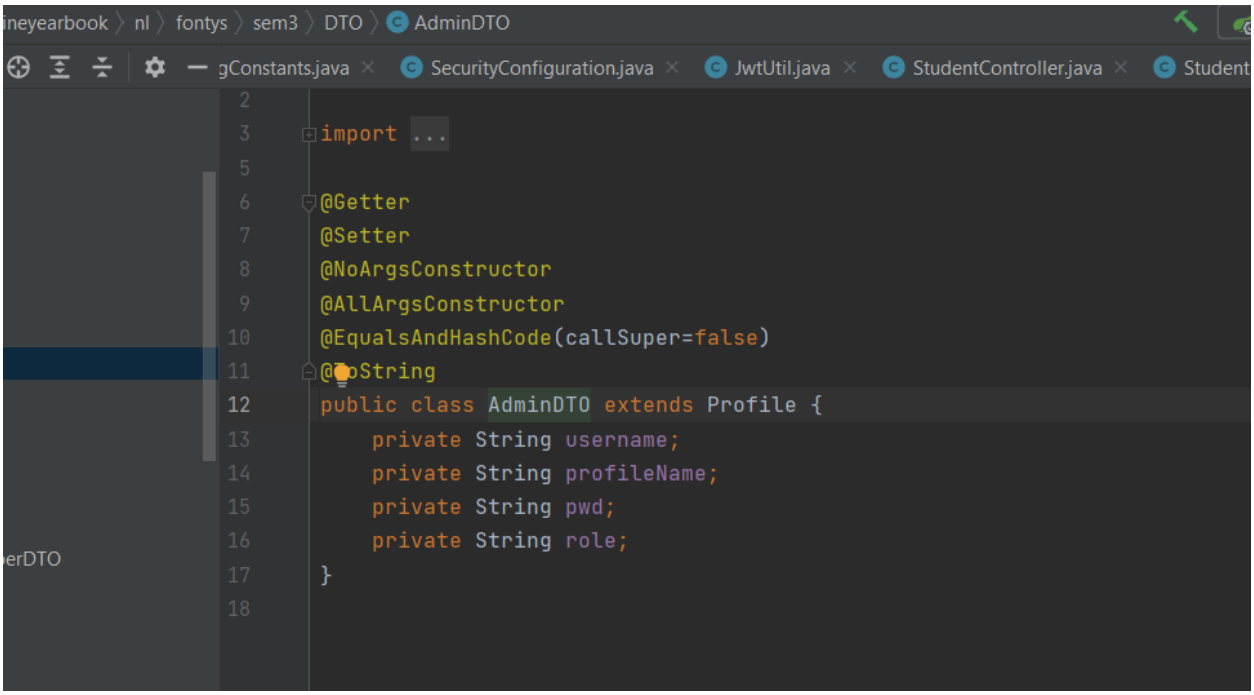


XML External Entities

My application handles this using a Data transfer object, I use a DTO in parts of the controller which require an object's insertion.

Controller making use of the DTO

DTO actual class



Broken Access Controls

My application handles this using Authentication and Authorization with the integration of spring security.

With Authorization added, api functionality calls which belong to the admin are blocked from other account roles and can only be accessed by an admin

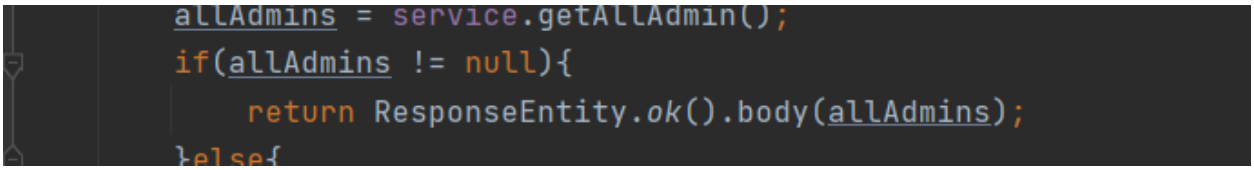
Cross-Site Scripting

My application does not properly handle this vulnerability, this is because when I try to specify the web application origin on controllers, I receive an error.

Therefore, I will try to research more on how to fix this error but for now It doesn't properly handle this.

Insecure Deserialization

My application handles this by not sending primitive datatypes directly as a return value for my controllers, for returning value I make use of ResponseEnitty and this can hold the data in its payload, thus in my frontend I can access the data I required to be returned by the call of res.data.



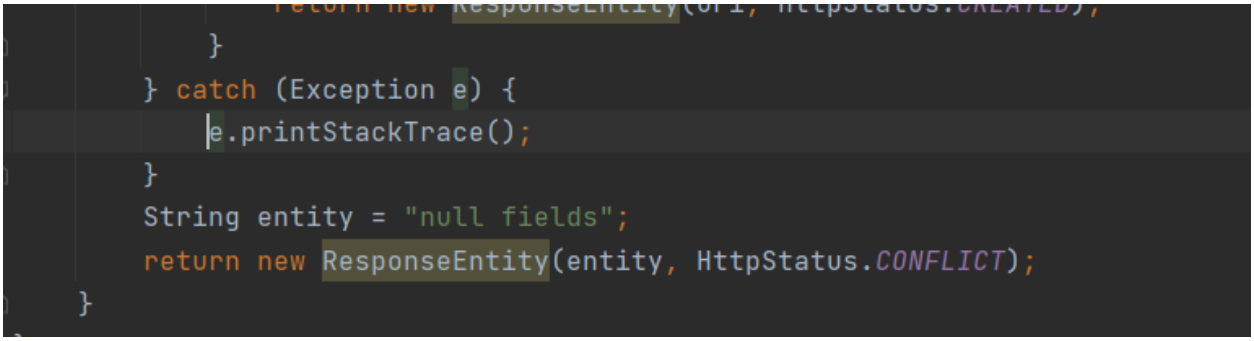
Using Components with Known Vulnerabilities

Currently my application does not make use of vulnerable 3rd party components, a third-party component which I do use is phpMyAdmin database and the security does not have any known vulnerability so far.

Insufficient Logging and Monitoring

My application handles this vulnerability using try catch method which handles detecting all errors.

Every error noticed will be returned both to the backend and frontend console and the user will also be given feedback.



Conclusion

After reviewing all the OWASP vulnerabilities, any security hazard which my application still contains will be fixed before the final delivery.