As discussed throughout the last part of the course the OWASP organization issued recently a list containing the most common vulnerabilities in websites. In this project a simple web application is developed to demonstrate some of these vulnerabilities and how to modify the security configuration as well as the code to avoid them. Web applications with these vulnerabilities can be a victim to attacks that exploit the website through these vulnerabilities.

The web application used in this demonstration is implemented on spring framework version 1.4.1 and h2 database, which is implemented in memory. Data is lost when the application is terminated, which is not practical, however the application is used for demonstration purposes. The web application has the following functionalities. It implements basic http form login which authenticate the user. Afterwards the user is presented with a form, which can be used to store contacts. There are two fields two submit and save a name and his phone number.

In the following section some vulnerabilities that exist in the application will be identified. The vulnerability will be presented in the form of a set of steps if followed correctly the vulnerability can be reproduced. Afterwards, a short explanation of the vulnerability will be stated and briefly explained. Finally, suggestions to fix these vulnerabilities will be presented to the reader.

First vulnerability identified in the application is "Sensitive data exposure" as described in the OWASP top 10 list "Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser."

Steps to reproduce the vulnerability:

1- Open the "CustomUserDetailsService" under src.main.java.sec.project.config
2- In the init function the data for the username and password is stored

Although, the password is stored as encrypted text the username is stored in plain text which is still considered a bad practice. A malicious actor can do fuzz attack using this user name while modifying only the password which reduces the complexity of such an attack. It is recommended to encrypt the username as well as the password.

The second vulnerability is "Using components with known vulnerabilities" as defined in the OWASP list "Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts". To reproduce the vulnerability the following steps should be followed:

1- Open the terminal to the location of the maven project
2- Run the following command "mvn dependency-check:check"
3- The dependency check report should be present under the target folder

The vulnerability report highlights the known and published vulnerabilities related to all the components in the project. The mentioned vulnerabilities should be considered while designing the software, and the affected components should be updated to the latest versions.

The third vulnerability is "Cross-Site request forgery" defined as "A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This

allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim." To reproduce this vulnerability, the following steps should be followed:

1- Log in to the application using the following credentials username :"ted" , password: "president"
2- Create the following html file:

```
<form action="http://localhost:8080/addcontact" method="post">
<input type="hidden"
        name="name"
        value="ahmed"/>
<input type="hidden"
        name="number"
        value="23355"/>
<input type="submit"
        value="Win Money!"/>
</form>
```

3- Open this webpage in the same web browser
4- Click the "win money!" button
5- A new contact will be added to the logged user

To fix this vulnerability go to : src.main.java.sec.project.config.securityConfiguration , then in the configure function comment the http.csrf.disable() line. This ensures that the requests handeled by the application have the same csrf token as the page that performed the log in.

The fourth vulnerability is "Cross-Site scripting (XSS)" which is defined as "XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites." To reproduce the vulnerability, the following steps should be followed:

1- Log in to the application using the following credentials username :"ted" , password: "president"
2- In the bottom of the page a button that represents an input from another user titled" click me!" is present.
3- Hovering over the button will redirect you to another website, namely google.

The most common technique to secure the application against this vulnerability is output sanitization and escaping. There are libraries that can be used to sanitize output like "OWASP Java HTML Sanitizer Project". The instructions to utilize this library can be found at https://github.com/OWASP/java-html-sanitizer.

Finally the last vulnerability is insecure direct object reference, described by the OWASP website as "A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data." To reproduce the vulnerability, the following steps should be followed:

1- Under src.main.java.sec.project.controller.contactController uncomment the init function
2- Now a new contact is added to the contacts repository which belongs to an account with the username jack

3- Log in to the application using the following credentials username :”ted” , password: “president”
4- The contact is printed on the contact list

Here there is no output validation implemented in the application. Under src.main.java.sec.project.controller. contactController, in the view function instead of adding the whole list to the model using the contactRepository,findAll(), a for loop should be implemented to validate that the contact object is identical to the username in the application cookie.