Web application can be found at https://github.com/EliasHaaralahti/vulnerable-web-application

This project is a simple web application that allows you to log in and create a note that logged in users can see. This project includes the following vulnerabilities from the OWASP top 10 vulnerabilities list.

A1-Injection.
This is a SQL injection vulnerability in the login form. The parameters name and password are directly added into the SQL query without using proper queries with parameters or escaping user input. This vulnerability also allows you to create, remove and get data and such. This is a very powerful and dangerous vulnerability, as even user passwords are stored in plain-text. To fix this vulnerability one should use parameterized SQL queries or escape all user input. When using Spring, Spring repositories are a great way of solving this problem and every other query is actually created using them. See Owasp Zap steps at the bottom.

Steps to reproduce:
1. Launch the application and go to http://localhost:8080
3. In the name field put "Name" or "User" and in the password field put "NotRealPass'OR'1'='1". and submit.

A2-Broken Authentication and Session Management.
This vulnerability is caused by multiple other vulnerabilities, including user credentials not being stored safely, as passwords are stored in plain text and credentials can be accessed using login form SQL injection vulnerability. Also as Cross-Site Scripting attacks are possible the session can be hijacked. To reproduce and fix this we check details under respective vulnerabilities. Also viewing individual notes does not require authentication.

Steps to reproduce:
1. Launch the application, go to http://localhost:8080, log in using credentials "User" and "Password" and create a note.
2. Log out and go to http://localhost:8080/note/1

A3-Cross-Site Scripting (XSS).
After successfully logging in, a note can be created on the home page. The content of this note is not sanitized, so things like JavaScript can be included in the note and as they are sent to the browser of whoever is viewing the note, it will be executed as JavaScript. To manually do this the note content must be for an example "MyNote<script>console.log("message");</script>". This attack could be for an example used to hijack the session. To prevent Cross-Site Scripting attacks, the input contents should be sanitized and all the user input should be escaped. However when using Thymeleaf all that has to be done is to change "th:utext" to "th:text".

Steps to reproduce:
1. Launch the application, go to http://localhost:8080 and log in using credentials "User" and "Password".
2. Create a new note with content "MyNote<script>console.log("message");</script>".

A4-Insecure Direct Object References
This application allows users to create both hidden and public notes, hidden will only be shown to the user who created it. However notes are accessed using path /note/id and it does not implement any access control checks or other protection, so by guessing the ID number users are able to see hidden notes they are not supposed to. To fix this access control should be implemented to the page to check if the user is authenticated to view the note.

Steps to reproduce:
1. Launch the application, go to http://localhost:8080 and log in using credentials "User" and "Password".
2. Create two notes, one hidden and one visible.
3. Log in as "Name" and "Password", to view the hidden note go to /note/id.

A5-Security Misconfiguration.
The first part of this vulnerability is the fact that CSRF is disabled, which makes this application vulnerable to A8-Cross-Site Request Forgery attacks. However the tricky part is that as get request security is not handled using the framework's security, but instead by checking the session attribute, each request has to be verified manually. This will likely cause issues during development To fix this, by default all requests should require authentication and the pages that do not require it, such as login, should be specifically mentioned. Also it would be best to use the security framework provided by Spring.

A6-Sensitive Data Exposure.
This vulnerability is very simple, all the user data is stored as plain text, including passwords. Using the SQL injection vulnerability all data can be retrieved using SQL queries. This means all the accounts can be stolen easily. To fix this problem the data should be encrypted, one method could be using the Spring security crypto module.

A8-Cross-Site Request Forgery (CSRF).
A CSRF attack can be done by having a user who is logged in visit an another site that includes malicious code using the same browser. To test this you can log in and open the csrf.html file in the external-resources folder included in this project. Create a note and click submit, the note will be displayed and the creator is the user logged. To prevent this attack a CSRF token has to be added. In Spring this can be done automatically using the Spring security framework by adding line "http.csrf();" inside the security configurate function. It will then automatically add a CSRF token when needed. Outside Spring the token can be stored manually on the client.

Steps to reproduce:
1. Launch the application, go to http://localhost:8080 and log in using credentials "User" and "Password".
2. Open the file csrf.html in external-resources folder using the same browser. Submit the form and you will see the created note.

Owasp Zap steps for A1-Injection:
1. Start both this application and Owasp Zap.
2. In the quick start section, click Launch Browser, go to localhost:8080, add values username and password respectively.
3. In Owasp Zap click the post request, select the password value, which should be "password", right click and select "Fuzz".
4. Note: This part requires you have to have the FuzzDB tools installed. More information at the top of this file.
5. Select payloads, click add, select type "File Fuzzers", from the fuzzdb selection go to attack and select sql-injection. Click add and ok.
6. Start the attack. From the results you can see if the application is vulnerable.

Owasp Zap steps for XSS attack:
1. Start both this application and Owasp Zap.

2. In the quick start section, click Launch Browser, go to localhost:8080, login with the default credentials, set note field value to "test" and submit the form.
3. In Owasp Zap click the post request, select the content value, which should be "test", right click and select "Fuzz".
4. Note: This part requires you have to have the FuzzDB tools installed. More information at the top of this file.
5. Select payloads, click add, select type "File Fuzzers", from the fuzzdb selection go to attack and select xss. Click add and ok.
6. Start the attack. From the results you can see if the application is vulnerable.

Owasp Zap steps to detect if CSRF attack can work:
1. Start both this application and Owasp Zap.
2. In the quick start section, click Launch Browser and go to localhost:8080.
3. Click submit in the login form.
4. Inspect the request in Owasp Zap to find that there is no CRSF token included.