My application can be found from https://github.com/woltsu/owasp. It is a simple web application that is used to publish news and read them. Only the people who have received credentials should be able to publish news, but everyone has an access to read them. The easiest way to launch the application is to clone it, open it with NetBeans, build it, run it and go to localhost:8080. Furthermore, if one wants to replicate the issues, Postman is an easy tool to use. The application contains 5 flaws from the OWASP top ten list: 1. Cross-Site Request Forgery, 2. Missing function level access control, 3. Injection, 4. Cross-Site Scripting and 5. Sensitive data exposure.

Issue 1: Cross-Site Request Forgery. Steps to reproduce: 1. Go to login page 2. Insert value “user” into the username field 3. Insert value “password” into the password field 4. Press Login 5. Go to http://localhost:8080/csrf-test without logging out 6. Logout 7. Browse to the news page 8. You can now see that a new piece of news has been submitted by the account “user”. This means that if one forgets to logout and they end up in a hostile website, someone else can post news in their name even with a proper function level access control. CSRF issues can be fixed by enabling csrf in the Spring security configuration. With csrf enabled, a randomized, impossible to guess token that matches with the one on server-side must be added into the request when f.ex. trying to post something to the server. This makes the exploitation of forged requests very close to impossible.

Issue 2: Missing function level access control. Steps to reproduce: 1. Open Postman 2. Set the method to POST 3. Insert http://localhost:8080/news?content=test to the url field 4. Press Send 5. Browse to the news page 6. You can see a new piece of news added into the application. This means that one can publish news without ever actually logging into the application. This issue can be prevented with a proper function level access control, such as checking authentication when one tries to create a new piece of news. Furthermore, in Spring security configuration, it is possible to set up roles for different users and configure which methods can be used by which roles.

Issue 3: Injection. Steps to reproduce: 1. Open Postman 2. Set the method to POST 3. Insert http://localhost:8080/news?content=', '');INSERT INTO Account VALUES ('hacker', 'hacker');INSERT INTO News(content, publisher) VALUES ('', ' 4. Press Send 5. Browse to the login page 6. Insert value “hacker” into the username field 7. Insert value “hacker” into the password field 8. Press Login 9. You are now logged in with an account that you have injected into the database via a forged request. With this flaw an attacker has an access to the entire database and can do whatever they please to do with it. Injection can be prevented by keeping untrusted data separated from commands and queries: one can use prepared statements in which inputs are parameterized and escaped when creating SQL queries.

Issue 4: Cross-Site Scripting (XXS). Steps to reproduce: 1. Go to login page 2. Login with the credentials that you just created or with “user” “password” combination 3. Insert value <script>alert("XXS");</script> into the text area 4. Press submit 5. Logout 6. Browse to the news page 7. You will see that the script which you entered was executed. With XXS attacks an attacker can f.ex. insert a key logger into a website or get an access to a victim’s session id. XXS attacks can be prevented by escaping data, extra validating and auto-sanization.

These methods ensure that if f.ex. a script tag is inserted into an application via user input, it will be handled as a string instead of a script html tag.

Issue 5: Sensitive data exposure. Steps to reproduce: 1. Open Postman 2. Set the method to POST 3. Insert http://localhost:8080/news?content=', "');INSERT INTO News(content) SELECT (username, password) FROM Account;INSERT INTO News VALUES ("', ' into the url field 4. Press send 5. Browse to the news page 6. You can now see all of the database's accounts' usernames and passwords in clear text. This issue can be prevented by encrypting all sensitive data with proper algorithms and keys designed for password protection, such as BCrypt.

In the end, there isn't a one silver bullet that could fix all of the application's flaws magically. F.ex. if I add in a csrf protection, but not proper access control, one could still post news into the database with forged requests. To fix all of the application's problems, I would need to add csrf protection, proper function level access control, encrypt sensitive data properly and make sure that user input is validated, escaped and parameterized.