Cyber Security Base: First Project

The project can be found at https://github.com/antreun/cybersecuritybase_firstproject

The project consists of a simple application which enables users that have an account to sign up people into some unknown event. The users with an account could be for example the people that are organizing the event, and the signups are the guests.

The application has a login page, which is the only page available before logging in. The application has two hard-coded user accounts, "john" with password "pw123" and "jack" with password "supersecret". The database file is re-created on startup if it's not already present.

After a successful login, the user is redirected to a main page displaying the username of the logged in user, a link for signing up a new person, a link for logging out of the application and the list of persons that have been signed up. The list also has a remove link, but only for the persons that this particular user has signed up.

Clicking the sign up link displays a page with a form. Using this for a new person can be signed up to the event.

The logout link does what it says, logs out the current user.

The application works correctly, but has several serious security flaws in it:

Issue 1: Security Misconfiguration

The default administrator account has username "admin" and password "admin". This can be found easily by fuzzing the login form with for example OWASP Zap using a list of common default usernames and passwords.

Steps to reproduce:

1. Attack the login form with a fuzzer or by manually entering most common login and password combinations for administrative accounts.

2. Upon successful login, the attacker can get access to the application.

How to fix the issue:

Change the default password for the username admin. Force users to use better passwords by enabling password strength checking when changing the password. For keeping things simple, this particular application has no password change feature and the changes need to be made manually to the database. This can be achieved by removing the .db-files from the project directory and editing the database import file in /sql -folder. The database is then recreated on application restart.

Issue 2: Injection

The login form has SQL injection flaw in the username field. The contents entered to the field are directly used in a SQL query without sanitizing. This vulnerability allows many different kinds of attacks against the application. An unauthenticated attacker can create a new user, change the password of an existing user, remove an user or clear the entire database which effectively disables the application causing a denial of service situation.

Detecting the vulnerability is difficult using automated tools, as directly logging in using injection is not possible and the login page displays the same "Invalid user name or password" error even after successful SQL commands making it a blind SQL injection vulnerability. However, by using a time-consuming SQL statement and monitoring the time it takes to process the login for the application, the injection can be confirmed. if the attacker knows an username and password, the injection can be confirmed by adding a SQL statement which has a TRUE value to the username.

Steps to reproduce:

1. To confirm the blind SQL injection using a time-consuming statement, navigate to the login form and enter the username "'; call hash('SHA256', '', 10000000); --" without the double quotes and anything for the password. There should be significant delay before the "Invalid username or password" message is displayed.

2. To confirm the injection with visible results, a real username and password is required. Enter the username and append "' AND 1=1 --" without the double quotes. For the password, enter the real password.

3. You are now logged in as user "username".

4. Log out and try to login again with same username but now append "' AND 1=0 --" without the double quotes. This time the login doesn't work, confirming that there really is a blind injection vulnerability present.

5. Several SQL statements can be used attack the application without being logged in: By entering an username "'; INSERT INTO User (name, password) VALUES ('johndoe', 'hacked'); --" and anything to the password field, a new user is created in the application with username "johndoe" and password "hacked" while the login page displays a generic login error. This new user can then be used to log into the application. Similarly an existing user's password can be changed by replacing "johndoe" with an existing username. Using an username "'; DROP TABLE Users; --" removes the entire login information table disabling the login for all users.

How to fix the issue:

The input for the username field needs to be sanitized before using it in a SQL query. Using a prepared statement resolves this vulnerability. The fix is in comments around line 61 in the CustomUserDetailsService.java file.

Issue 3: Broken Authentication and Session Management

The signup form has a problem which allows users to sign up people on behalf or other users. The current username is passed in the form in a hidden field which can be manipulated.

Steps to reproduce:

1. Log into the application and navigate to the signup form.

2. Enter the name and address for a new person.

2. Using for example Chrome Development Tools, press F12 to open the console and view the source code for the form.

3. Change the username to something else in the line "<input type="hidden" name="username" value="username">"

4. Press submit.

5. A new person is signed up by the username you changed. Also notice that if there is no such username in the application, the signed up person cannot be removed from the list (except using the next vulnerability!).

How to fix the issue:

Passing the value of the currently logged user via the form's hidden field is not necessary, as it is already available on the server side. The file SignupController.java has the "submitForm" method responsible for adding the signup. The fix for this security issue is in the comments. Also the "loadForm" method should be changed not to include the current username in the model, and the "form" template not to include it in a hidden field.

Issue 4: Missing Function Level Access Control

There is no access control in removing a signup, so any logged in user can remove any signup including the ones created by another user.

Steps to reproduce:

1. Log into the application with an username and password

2. Any signup can be removed by directly accessing the remove method via navigating to URL: http://localhost:8080/remove?id=1 where id has the value of the signup to remove.

How to fix the issue:

The controller responsible for deleting a signup has to check that the currently logged in user is actually the creator of the signup. The "SignupController.java" has comments in the method "removeSignup" which shows how to do that.

Issue 5: Cross-Site Scripting (XSS)

The signup form allows HTML and JavaScript to be entered into the "Name" and "Address" fields, which is then stored to database and displayed on the signup page for all logged in users. This enables a malicious user to enter for example scripts that will be run by other logged in users when displaying the sign up list.

Steps to reproduce:

1. Log into the application with a valid username and password.

2. Navigate to the signup form.

3. For the name, enter "<script>alert("XSS!");</script>" without the double quotes. For the address, enter anything.

4. Submit the form.

5. A JavaScript alert displaying "XSS!" is displayed.

How to fix the issue:

A quick fix is to convert the "th:utext" tags in "signups.html" template to "th:text". This disables the rendering of HTML tags in the signups list. Another way would be to remove the tags in the controller before adding the text into the database using for example a regular expression: name = name.replaceAll("\\<.*?\\>", "");

Issue 6: Cross-Site Request Forgery (CSRF)

The application is vulnerable to cross-site requests in deleting a signup and creating a new signup, as there is no CSRF tokens used.

Steps to reproduce:

1. Create a page on attacker-controlled site with JavaScript code:

```
<script>
   for (var i=1; i<100; i++) {
      document.write("<img src='http://localhost:8080/remove?id="+i+"'>");
   }
</script>
```

Replace the address localhost:8080 with the address of the target application.

2. Log into the application using an username and password

3. Without logging out, visit the page created in step #1.

4. Signups with id from 1 to 99 are removed from the application.

How to fix the issue:

Removing a signup cannot be safely performed using a GET request. The remove functionality has to be implemented using a form that uses a POST or DELETE method to access the remove function. This allows embedding a CSRF token into the form, which prevents Cross-Site Request Forgeries.

The CSRF token can be enabled for the forms by commenting out the "http.csrf().disable()" command in the file "SecurityConfiguration.java".

More issues...

The application stores user passwords in plain text without encryption!