**LINK: https://github.com/RoniNiklas/cybersecs-project**

**How to install:**

Clone the github repo and open it as a project in netbeans and press play / run. It should open the program on localhost:8080.

It does not use the starter template provided, since that was too secure out of the box, but this is a maven + spring etc. project just like that.

**App description:**

This is a credit card details storing app. You create an account with a username, a password and your credit card details. Then you can later log in to check your card details instead of having to remember them or read them off your card.

You can try logging in with username: make and password: salasana to see what it looks like.

Hosted in localhost:8080 by default I think.

Localhost:8080/login is the login / account creation screen.

Localhost:8080/users/{username} is where you can check your accounts data.

Any other path redirects to the login screen.

**FLAWS:**

**FLAW 1: INJECTION VULNERABILITY**

**Description:**

The login and create account forms allow SQL injection.

**How to abuse:**

For example, you can log in as anyone by putting their username in and using the following snippet as your password.

‘ or ‘1’=’1

You can also run SQL commands by inserting snippets like the following into pretty much any field in the forms.

‘; DROP TABLE accounts; --

**How to fix:**

1. Clean any user input by blacklisting dangerous characters like quotation marks and the semi-colon etc. from being passed into SQL queries.

2. Use parameterized statements.

3. Use ORM tools like Hibernate, although they also are not completely invulnerable to injection attacks. But they do block off some of the more common ones as far as I know.

**FLAW 2: BROKEN AUTHENTICATION.**

**Description:**

The site has no protection against credential stuffing attacks such as spamming a bunch of the most common passwords with a fuzzer. You can send as many requests you like and as often as you like.

**How to abuse it:**

Grab a list of the most common passwords, grab a fuzzer like the OWASP ZAP’s one, guess a user’s username and start spamming login requests until you get through.

**How to fix it:**

Set a limit on the amount of wrong guesses that can be sent. Start enforcing a cooldown between requests, require the user to complete a google captcha or block their IP completely if the amount of wrong guesses from one address is too high or the frequency of requests is too high.

**FLAW 3: SENSITIVE DATA EXPOSURE.**

**Description:**

The site does not encrypt data saved or data transferred. Clients send their passwords and credit card details in cleartext over HTTP, and they are saved in cleartext on the server.

**How to abuse it:**

Somehow get your hands on the database (such as going to localhost:8080/h2-console, which does not require authentication) and voila; all the user data is readily available and readable.

**How to fix it:**

Encrypt the sensitive data before transferring it. Encrypt it before storing it on disk. Use strong encryption algorithms.

**FLAW 4: BROKEN ACCESS CONTROL**

**Description:**

The site does not have any sort of authentication check to make sure you are not viewing data you should not be authorized to see. While normal use does not lead you to see other people’s data, you can quite easily view other users’ credit card details.

**How to abuse it:**

In normal use, you open the front page, log in and it redirects you to localhost:8080/users/{yourusername} so that you can check your data.

However, you can just change the username in the url path to someone else’s username and you can read their data as you would your own. There’s no check to see if you’re actually the user whose data is behind that path, and it doesn’t even check that you’re logged in.

**How to fix it:**

1. Make the server’s controllers check that the user is properly authenticated and has the correct authorization to access the data behind a url path.

2. Since the app’s idea is to not be able to browse other people’s data, it would be smarter that the data would be passed based on the login credentials of your session, rather than an url path. Everything could be done in the root path (localhost:8080) and the data the controller passes would depend on who’s logged in to the session, rather than the url path.

**FLAW 5: INSUFFICIENT LOGGING**

**Description:**

The site does not log any requests sent to it or any queries it makes to the database or any responses it sends to the client.

**How to abuse it:**

Spam all sorts of attacks at it; none of them will be logged or create an alert of any sort. If something bad happens (say, someone uses SQL injection to delete one of your tables), no message of this is passed anywhere. No person will be alerted. The lack of the table in the DB just breaks the site; the login screen just stops letting people log in, and the users/{username} starts redirecting people to localhost:8080/error path (which also doesn’t work for some reason, but that is thematic in this “create a crappy website” project).

**How to fix it:**Start logging requests and SQL queries, especially failed ones and suspicious ones. Don’t put the logs in the database that can be injected. Create alerts to system administrators if suspicious events occur or the server is in a faulty state.