9 Week 9: Web Security, Authentication and Authorisation

9.1 *Prac Overview*

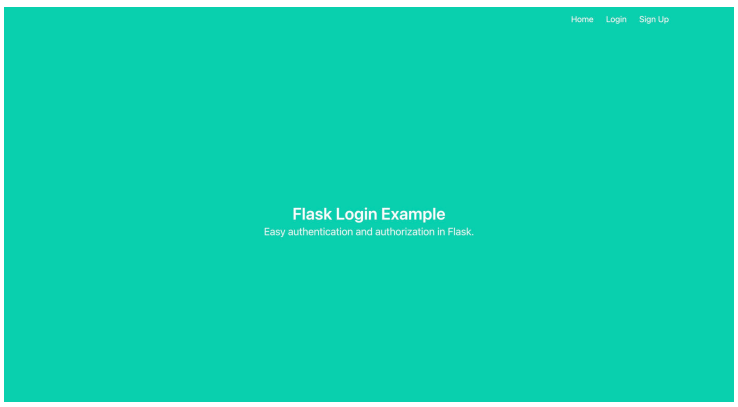
The code for this week is written in Python using the Flask framework. Flask is a simple web application framework for Python which implements many standard security features. This workshop will guide you through a simple login application using Flask and Python. This week to get marks for your work you need to complete the SQL injection vulnerability exercise.

9.2 *Web Authentication Application*

Your code for this week is a simple web application which allows users to signup and login using basic password authentication. The session management is handled by Flask using cookies. You may work on this task in groups or individually.

Exercise 9.2.1. Your first task is to download the code from github classroom using the following link: https://classroom.github.com/a/4WKJa3H_ The repository contains a README.md file with instructions on what to install to get the website running.

If you follow the instructions on the README.md you should end up with a webserver running on port 8001 which you can view by browsing to the url http://localhost:8001/. You should see a web page looking like this:



Click around and make sure that the signup, login and logout

functionality works as you expect.

9.2.1 *Overview of the codebase*

NOTE: Python code uses tabs for indentation rather than braces. Be careful about adding spaces into the code as this could cause errors when you try to run the code.

You will see the following files in the main directory in github:

- *run.py* - initialises the Flask application and runs it on port 8001.
- *instance* - contains the database called *db.sqlite* which contains the user login details.
- *project* - contains the main application logic

Looking inside the project directory, there are 4 python scripts containing most of the application logic. These are:

- *__init__.py* - initialises the flask application, including configuration, setting up the database handle and setting up the login manager which handles login redirection.
- *auth.py* - defines the endpoints (urls) for login, logout and signup as well as the required application logic.
- *main.py* - defines the endpoint (urls) for the homepage and the profile page.
- *models.py* - defines classes which map to database tables. In this case there is only one - the User class, corresponding to the user table in the sqlite database.

Under the project directory there is also a templates directory which contains the HTML fragments which are used to render the web pages. These files contain scripting directives into which Flask variables can be injected before being rendered.

9.2.2 *Walkthrough of the signup workflow*

To give you an understanding of how the code fits together, let's walk through the application logic for signup.

Step 1. Navigate to http://localhost:8001/ and click on Sign Up. Notice that the url for the Sign up page is */signup*.

Step 2. The signup logic is handled in the *project/auth.py* file. In that file you will see on lines 32-34 the following code:

```
1 @auth.route('/signup')
2 def signup():
3     return render_template('signup.html')
```

This is the code which handles the logic when you navigate to the url *'/signup'*. It defines the function signup() which just returns the template signup.html (in the templates directory) after rendering it (ie. processing any Flask variables referenced in the file).

Notice that line 36 also begins:

```
1 @auth.route('/signup', methods=['POST'])
```

This code is executed if the *'/signup'* url is accessed using the POST method of HTTP. Since clicking on the Sign Up link produces a GET request, this code is not executed.

Step 3. Add some details into the Sign Up form and click submit. Notice that you get redirected to the Login page (assuming you did not enter credentials which already exist!). If you look at the *templates/signup.html* file you will notice it contains the Sign up form with a POST method on url *'/signup'*. So when you clicked Sign Up, the code which was called was the code in line 36 of auth.py:

```
1 @auth.route('/signup', methods=['POST'])
```

You can see that this code fetches the *email*, *name* and *password* fields from the form (lines 38-40), then runs an SQL command to check if a user with that email address exists in the database. If so, an error message is sent to the template via the *flash* command (line 44), the application writes to the debug log (line 45) , and the user is redirected to the url *'signup'* in the auth file (line 46).

If the user does not exist, a new User class is instantiated (line 49) using the form fields, the user is added to the database (lines 52-53) and the user is redirected to the login page (line 55).

Exercise 9.2.2. You may have noticed some security issues and/or design flaws in the Sign Up process. Write down any problems you noticed by looking at the code, particularly in lines 36-55 of the *auth.py* file. Identify any security principles which the code violates.
Exercise 9.2.3. Run the code through codeQL using github and see if it picks up any of the security issues you identified.

You should find that codeQL discovers an SQL injection vulnerability. Notice that in *auth.py* on line 19 there is a database query which makes use of the *filter_by* function to run a query on the database without using SQL directly. This function is provided by the SQLAlchemy module, and it is designed to protect against SQL injection attacks by performing input filtering.

✓ Task of the Week

Exercise 9.2.4. Using the code on line 19 of auth.py as a template, see if you can fix the SQL injection vulnerability. (Remember to be careful with tab-spacing in Python). Run the code through codeQL to verify it has been fixed. How else might you verify that the SQL injection vulnerability no longer exists? Think about what sort of tests you could run.

9.2.3 *Other workflows*

The other workflows in this web application are the login workflow and the logout workflow. The logout workflow has a couple of interesting features.

Notice firstly that the Logout link only appears at the top of the page if you are logged in. The logic for this occurs in the *templates/base.html* file. There is a line of template code checking whether the current user is authenticated. This check just determines whether or not to display the link (you can verify this by removing the template check and observing what happens when you click on the Logout link).

The main logic for the logout workflow occurs at the end of the *auth.py* file (lines 57-61). Notice that there is an annotation *@login_required*. This is what the Flask application uses to verify whether the user is logged in before displaying the page. If not, the user is redirected to the login page - the logic that determines this is in the *__init__.py* file (lines 16-25) where the login manager is setup.

Exercise 9.2.5. Write down the logic for the login workflow as implemented in the code. Make sure at each step you check what happens if a user is logged in or logged out. At what point is the profile page made visible?

9.2.4 *Designing a more secure authentication system*

The authentication system implemented in this code is very basic and has a number of fundamental security flaws including:

- Use of plaintext passwords which could leave users vulnerable should the database be compromised.
- Vulnerability to brute-forcing attacks since it uses purely password-based authentication with no additional factors.

A more secure authentication system should use either multi-factor authentication, or a 3rd party authentication framework such as OAuth/OpenID.

Exercise 9.2.6. Propose one of the above authentication systems as an alternative to the simple password-based system used in this code. Write down a design of the login and signup workflows for your system. Identify potential security vulnerabilities and how your design and/or implementation would protect against these attacks.

9.2.5 *Practice with SecDim*

After completing the above tasks, you should now continue working on SecDim. See last week's workshop instructions for how to login.

Exercise 9.2.7. Complete the *Privilege Escalation* task in the Python Secure Coding game. You may work in groups or individually to solve this task.