Module: CMP-7038B – Developing Secure Software  
Assignment: R002 – Secure Development Project and  
Presentation (Individual)  
Set by: Debbie Taylor Debbie.taylor@uea.ac.uk  
Date set: 14 July 2023  
Value: 70%  
Date due: 08 August 2023 15:00  
Returned by: 05 September 2023  
Submission: Blackboard  
Checked by: Eleanor Leist E.leist@uea.ac.uk

**Learning outcomes**  
• Understand the importance of designing software with the security needs of  
an end user in mind  
• Develop a secure and usable website that meets the needs of the user  
• Analyse the effectiveness of a range of security methods and tools  
• Analyse the evolving threats associated with the internet

**Specification** **Overview**  
The aim of this R002 assignment is for you to code a secure usable and accessible web-based movie blog system that mitigates, at a minimum, the five most common security vulnerabilities of account enumeration, session hijacking, SQL injection, cross-site scripting and cross-site request forgery.  
You will work individually to code and secure the web-based blog using JavaScript and Node.js, with a MySQL database.  
At minimum, the movie blog system will require registration and login authentication (via 2FA of username/password and email One-Time-Passwords (OTP)), search functionality, and the ability to add, edit and delete posts. You can use pre-built security libraries, but you must clearly  
and concisely explain how they work and how they improve security for your movie blog system.  
To evidence your system’s security mitigations working, you need to create a maximum 15-minute MP4 (max 720P) video demonstration, showing both the front-end (user website view) and back-end (code and database) elements of your system and try attacking the system yourself to evidence you have protected your system from a threat actor attacking each vulnerability/element.

**Description**  
You are required to individually develop a small, secure, usable and accessible, web-based move blog site that mitigates various security vulnerabilities.

**Development coding of web-based movie blog:**  
At a minimum, your code should defend against the five most common vulnerabilities of:  
• Account enumeration  
• Session hijacking  
• SQL injection  
• Cross-site scripting  
• Cross-site request forgery

You need to concentrate on coding the security, usability and accessibility aspects of the web-based movie blog and not on web development, as you only need to produce a basic usable and accessible  
front-end. This will be used to evidence your security processes and mitigations during a 15-minute MP4 video demonstration. Functionality of the front-end should be prioritised over the aesthetics, but you still need to consider usability and accessibility.  
You must code your website using JavaScript and Node.js, with a MySQL database. Any Node framework, such as Express, is acceptable but you cannot use any other types of SQL databases, as you are restricted to using MySQL.  
To secure your movie blog you must include hashing and/or salting, encryption and a 2FA authentication of username/password and email One Time Password (OTP).  
The movie blog should not fully sacrifice security or usability and accessibility, and there will be some trade-offs needed. You must discuss and justify any trade-offs you have chosen, during your video  
demonstration. You can use any pre-built security libraries you believe will be useful, but you must be clearly and concisely explain how they work, what they secure against and exactly how they provide security protection specifically for this movie blog. If you cannot or do not fully explain your library use, you will not attain any marks for that mitigation. You should also consider coding some of your own processes, as extra marks are available for self-coded mitigations.  
Each mitigation must be valid across the whole web-blog site, e.g., you cannot mitigate SQL injection and then break it later when mitigating another vulnerability.  
The code you produce must be fully tested, using think-aloud user testing and system unit testing. Evidence of these tests (and results) should be recorded via separate test plans and must be shown during the MP4 video demonstration

Below is an outline of the key components and steps you'll need to consider when developing the web-based movie blog system:

1. **Setup and Configuration:**
   * Set up a Node.js project with a package.json file to manage dependencies.
   * Install and configure the necessary packages, including Express for the server and MySQL for the database.

**sudo mysql\_secure\_installation**

* + Create a MySQL database and set up the necessary tables for users, posts, etc.

1. **Authentication and User Management:**
   * Implement user registration, login, and 2FA (Two-Factor Authentication) using username/password and email OTP.
   * Hash and salt user passwords before storing them in the database.
   * Generate and verify email OTPs for 2FA authentication.
   * Create user sessions and implement secure session management.
2. **Input Validation and Sanitization:**
   * Validate and sanitize all user inputs to prevent SQL injection and XSS attacks.
   * Use parameterized queries to mitigate SQL injection risks.
3. **Cross-Site Request Forgery (CSRF) Protection:**
   * Implement CSRF tokens to prevent CSRF attacks on sensitive actions.
4. **Secure Communication:**
   * Enforce HTTPS to secure communication between the client and server.
5. **Post Management:**
   * Allow users to add, edit, and delete posts.
   * Implement proper authorization checks to ensure users can only modify their own posts.
6. **Search Functionality:**
   * Implement search functionality to search for movie blog posts.
7. **Front-end Development:**
   * Develop a basic front-end interface using HTML, CSS, and JavaScript.
   * Ensure the front-end is accessible and usable for all users.
8. **Security Libraries and Self-Coded Mitigations:**
   * Clearly and concisely explain how any pre-built security libraries used improve security for the movie blog system.
   * Consider adding some self-coded security mitigations for additional marks.
9. **Testing and Demonstration:**

* Perform think-aloud user testing to ensure usability and accessibility.
* Conduct system unit testing to verify the functionality and security of the application.
* Record evidence of the tests and results through separate test plans.
* Create a 15-minute MP4 video demonstration showcasing both the front-end and back-end elements of your system, including how you protect against each vulnerability/element.

Remember that security is an ongoing process, and it's essential to stay updated with the latest best practices and security considerations. Additionally, ensure that you thoroughly document your code, explaining the purpose and security features of each component.

This outline should provide you with a starting point for developing the web-based movie blog system. As you proceed with the implementation, make sure to refer to official documentation, security resources, and best practices for each security mitigation you're implementing. Good luck with your project!

Based on this part of the information above, let’s satisfy it first by the relevant files to be created and the codes to be filled in those files based on the requirements of the website. I have started by creating the mysql db (movie\_blog.db) and set up the necessary tables for users and posts. I want you to modify it and create other tables that you think are going to be used throughout the entire project and in the website.

CREATE DATABASE movie\_blog;

USE movie\_blog;

-- Table to stor users

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL,

password VARCHAR(100) NOT NULL,

otp\_secret VARCHAR(16) NULL,

otp\_enabled BOOLEAN DEFAULT 0

);

-- Table to store

CREATE TABLE posts (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100) NOT NULL,

content TEXT NOT NULL,

author\_id INT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (author\_id) REFERENCES users(id) ON DELETE CASCADE

);

I also have a html login page, modify it to be stylish and workable.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Movie Blog - Login</title>

</head>

<body>

<h1>Movie Blog Login</h1>

<form id="loginForm" method="POST" action="/login">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required><br>

<button type="submit">Login</button>

</form>

</body>

</html>

Then follow the remaining steps and write the relevant code.

* Authentication and User Management:
* Implement user registration, login, and 2FA (Two-Factor Authentication) using username/password and email OTP.
* Hash and salt user passwords before storing them in the database.
* Generate and verify email OTPs for 2FA authentication.
* Create user sessions and implement secure session management.

**Serverside.js**

- app.js (Main application file)

controllers/

- authController.js (User authentication related functions)

- postController.js (Post management related functions)

middleware/

- authenticationMiddleware.js (Handles user authentication checks)

- sessionMiddleware.js (Handles session management)

utils/

- otpUtils.js (Utility functions for generating and verifying OTPs)

- emailUtils.js (Utility functions for sending emails)

routes/

- authRoutes.js (User authentication routes)

- postRoutes.js (Post management routes)

- authMiddleware.js

models/

- OTPverification.js

- userModel.js (Database model for users)

- postModel.js (Database model for posts)

config/

- database.js (Database configuration)

views

- index.html

- login.html

styles

css

- index.css

-login.css

js

-index.js

-login.js

movie\_blog.sql

package\_lock.json

package.json

node\_modules

view folder:

Inside the view folder, you'll have different views (HTML files) that will be rendered to the users based on the routes and actions. For simplicity, let's assume you have the following views:

a. login.html - This view will display the login form.

b. dashboard.html - This view will display the user's dashboard after successful login.

c. create-post.html - This view will display the form to create a new post.

d. edit-post.html - This view will display the form to edit an existing post.

e. search-results.html - This view will display the search results for posts.

Now, in my system, I have these code folders and files.

config

- database.js