Table of Contents`

[Introduction 2](#_Toc160736552)

[Technical Description of Major website Features: 3](#_Toc160736553)

[Security Modelling & Testing: 3](#_Toc160736554)

[Two Unfixed Vulnerabilities 4](#_Toc160736555)

[LOGIN PAGE VULNERABILITY 4](#_Toc160736556)

[SQL Injection Vulnerability: 4](#_Toc160736557)

[Description of Vulnerability: 4](#_Toc160736558)

[Potential Exploitation: 4](#_Toc160736559)

[Solution To SQL Injection Vulnerability: 5](#_Toc160736560)

[FILE UPLOAD FUNCTIONALITY: 5](#_Toc160736561)

[Two Fixed Vulnerabilities 7](#_Toc160736562)

[Reset Password: 7](#_Toc160736563)

[Vulnerable code: 7](#_Toc160736564)

[Fixed code: 7](#_Toc160736565)

[Post Page: 8](#_Toc160736566)

[Injecting Dynamic Content: 8](#_Toc160736567)

[Vulnerable code: 8](#_Toc160736568)

[Fixed code: 8](#_Toc160736569)

**NOTE: Figures refered to are found Code Diff screenshot file.**

Introduction

This document describes a website's technical aspects and security testing, including information on the register, login, and post pages, as well as file upload and token generating functionalities. It lists the vulnerabilities found in these domains, emphasising file upload and SQL injection threats, along with an explanation of how they were fixed. It also discusses the security and development methods put in place to improve the integrity of the system and user safety.

# Technical Description of Major website Features:

* Login Page: To create a responsive layout, HTML and CSS were used in the design of the login page. It has a form where users can enter their password and username. The form securely submits data using the POST technique. There are also stylistic components, such a button and links for recovering your password and creating an account.

* Registration Page: Users can fill out a form on this page with their username, password, and email address. The PHP code makes sure that user inputs—like the length of the password, the format of the email, and the username—are validated at the very least. After being safely hashed, passwords are kept in the database. To avoid duplicate registrations, the process also includes a check for usernames and emails that already exist in the database.
* Profile Picture Upload on profile page: The file upload functionality featured on the website is part of a profile page available to the user upon logging in. Once entered onto the profile page, the user can view key details about themselves and add or change their profile picture.

* Token generation: upon being transferred to the page in which users would be able to enter their authentication token, The PHP code generates a unique token made from their email, old password, and the time of day they enter their email and request to change password. This page was created to validate that the user trying to reset the password for an email was the actual proposed user.

* Post Page: Using HTML and CSS, a responsive layout was created to arrange the posts created by multiple users in an organised manner. Once, the user enters the post page from the index page, they can make a post, submit into the database and see what they posted, as well as what other users have posted. It has a form element that securely submits the data using the POST method. It checks if the user is logged in, and if it returns as false, it assigns the user to a random number between 1 – 999.

# Security Modelling & Testing:

|  |  |  |
| --- | --- | --- |
| Features | Tests Done | Changes/Results |
| Login and Register Functionality (vulnerable) | BurpSuite Intruder attack using pitchfork styled payload delivery with SQL injection authentication bypass payloads. Used 144 payloads in 77 requests | 19/77 requests caused SQL errors |
| Profile Picture Upload (vulnerable) | Attempted to upload PHP file and navigated to the uploaded PHP file. | PHP file rendered indicating PHP reverse shell upload is possible. |
| Reset Password Page (secure) | Confirmed expiry time of token of 1 minute, Burpsuite Intruder attack using Sniper styled payload using 144 SQL injection payloads. | All pages unaffected by SQL payloads. |
| Posts Page and functionality (secure) | Burpsuite Intruder attack using pitchfork styled payload delivery with 220 XSS payloads. | All requests were correctly sanitized leading to no XSS. |

# Two Unfixed Vulnerabilities

## LOGIN PAGE VULNERABILITY

### SQL Injection Vulnerability:

### To achieve the instruction of the coursework, the login page vulnerable to SQL INJECTION. Below is a screenshot of the SQL injection vulnerability present in the code:

A black background with text

Description automatically generated

### Description of Vulnerability:

The vulnerability is in the way user input is directly incorporated into the SQL query without proper validation. The **$username** variable, which is derived from the user input, is directly concatenated into the SQL query string without any consideration for potential SQL injection attempts. This can be considered a failure to follow good practice.

### Potential Exploitation:

An attacker can exploit this vulnerability by manipulating the $\_POST['username'] input field. For example, they could input a malicious string like " OR 1=1; --, which would modify the SQL query to always return true, allowing unauthorized access to the system.

### Solution To SQL Injection Vulnerability:

It is important to use prepared statements. These ensure that user input is treated as data and not executable SQL code. Here's an example of how the vulnerable code can be secured:

A screen shot of a computer

Description automatically generated

## FILE UPLOAD FUNCTIONALITY:

For a file upload feature to be vulnerable it would need to be improperly validating file types that are uploaded. A file upload vulnerability can only be exploited if the attacker can call the malicious file, they had uploaded either directly by navigating to the image's directory through the URL or by chaining a Local File Inclusion (LFI)/Path Traversal Vulnerability to navigate to the said malicious file.

The page then proceeds to call a function named ‘find\_pfp’ which is responsible for locating a pre-existing profile picture for the user based of the pre-defined file naming and if one is not found it would use a default profile image.

A computer screen with colorful text

Description automatically generated

Using a predictable file naming convention as shown above enables the attacker to easily locate and call the uploaded file. A secure alternative would be generating a random string of bytes and using that as a file name in combination with the user's username as such 'Alice- 82ec1355cfb3fed.jpg'. This method can be shown in the snippet below.



The page continues then to query the SQL database for key information on the user to be displayed such as the user ID, their username and email they used to register. Afterwards processing and handling files uploaded by the user.

A computer screen shot of text

Description automatically generated

Although the code checks for a file size restriction, it does not check whether the uploaded file is an image, leaving the page exposed to malicious file uploads. As specified in [CWE-434](https://cwe.mitre.org/data/definitions/434.html), this page allows for unrestricted file uploads. Unrestricted file upload is widely used to upload a PHP reverse shell script, which is then executed by the PHP interpreter and spawns a reverse shell on the web server. To address this issue, we can implement whitelisting and file extension checking in conjunction with MIME type checking, as demonstrated below.

A computer screen with text

Description automatically generated

Furthermore, we can implement Magic Byte/File Signature checking to ensure the files binary data matches with the expected structure of a PNG/JPEG/JPG image file. For example, the first 8 bytes in a PNG's file header should start with ’89 50 4E 47 0D 0A 1A 0A’.

# Two Fixed Vulnerabilities:

## Reset Password:

Improper generation of token vulnerability:

The vulnerability in this is that when trying to generate a one-time authentication token we want it to be unique as it makes is harder to guess.

### Vulnerable code:

A simple function called Random Bytes in PHP, this would generate a 32-character long byte randomly and while it sounds good as a hacker would have trouble entering a string created randomly it wouldn’t take away the possibility of A) the hacker just doing the same thing and/or B) Brute force attack which just enables a hacker to run a program that just tries every random possibility. To defeat this, the token needed to be generated in a unique way. Refer to Figure 1.1.

### Fixed code:

In order to increase the uniqueness of this, a seed was established for the token. The user's email address, the password that is now in use (before it is changed), and the time that they were brought to the page were all combined into a string in a variable. After that, just hash the string using the built-in hash function this would make the token generated unique and random enough that a brute force attack couldn’t happen but also not to random that a hacker could simply do the same. An added benefit is the string that concludes the token is generated within the time issued which essentially means that the token is only valid for that minute. Refer to Figure 1.2.

## Post Page:

### Injecting Dynamic Content:

One of the vulnerabilities of this page is that when making a post that would be saved in the database, the code will assign the user if they are not registered, to a random generated username. Below are two different screenshots of the code, that shows its vulnerability and what has been adjusted to secure the vulnerability.

### Vulnerable code:

In the code, someone who is not a registered user can access the post page and make a post, that saves into the database, and they will see all the posts that have been displayed by previous users, which makes the page vulnerable. Refer to Figure 2.7.

### Fixed code:

To secure this vulnerability, an if check statement was implemented. The first thing that was done, was to add a username and userid column to the database, so that when the posts are displayed, it shows who made the posts. The if statement checks if the user is registered, if they are not registered, as shown by the ‘loggedInStatus’ !== true, the code generates a guest name with a random id for them as shown by the ‘$username = "Guest-" . rand(1, 999);’ section of the code. However, if the loggedInStatus returns as true, the code displays their username and userid. Refer to Figure 2.8

Group Members’ Contributions Table:

|  |  |  |
| --- | --- | --- |
| Group Members | Completed Feature/Vulnerability | Contribution Percentage (%) |
| Adunoluwa Oguntuga | Dynamic Content Injection | 100% |
| Dara Omiwale | One-time authentication token | 100% |
| El Madhoun, Mohammad | File Upload | 100% |
| Oluwatoyin Chinell Olotu | SQL Injection and Login-Bypass | 100% |

References:

* Codex, A.C. (2023) *Securing PHP applications: Input Sanitization and Best Practices*, *Reintech media*. Available at: https://reintech.io/blog/securing-php-applications-input-sanitization-best-practices (Accessed: 07 March 2024).
* OWASP (no date a) *File upload cheat sheet¶*, *File Upload - OWASP Cheat Sheet Series*. Available at: <https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html>(Accessed: 08 March 2024).
* OWASP (no date) *Multifactor authentication cheat sheet¶*, *Multifactor Authentication - OWASP Cheat Sheet Series*. Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html>(Accessed: 08 March 2024).
* OWASP (n.d.). *SQL Injection*. [online] OWASP. Available at: <https://owasp.org/www-community/attacks/SQL_Injection>(Accessed: 07 March 2024).
* Rizwan, M. and Hussien, Dr.M. (2023) *What is the best way to prevent XSS attacks with HTTP-only cookies?*, *How HTTP-only Cookies Can Prevent XSS Attacks*. Available at: <https://www.linkedin.com/advice/0/what-best-way-prevent-xss-attacks-http-only-mdsme#:~:text=HTTP%2Donly%20cookies%20prevent%20XSS%20attacks%20by%20limiting%20the%20attacker’s,not%20included%20in%20the%20document>.(Accessed: 05 March 2024).