**Security Tests**

**Student Name Daniel Dinelli**

**Student Number C002427241**

XXS on login page

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | On the login screen of website I enter <script>alert(“hi”)</script> as a name | The result of the attack should give a JavaScript alert box to popup.  FINDING: Reflective XSS |
| **Mitigation** | | |

The mitigation to this is sanitize the inputs that would allow for an XSS attack

XXS on register page

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | On the register screen of website I enter <script>alert(“hi”)</script> as a name | The result of the attack should give a JavaScript alert box to popup when the name is accessed from the database and displayed to the screen.  FINDING: Persistent XSS |
| **Mitigation** | | |

The mitigation to this is sanitize the inputs before allowing the name to be entered into the database

XXS on register or login page

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | On the register or login screen of website I enter <script>alert(“hi”)</script> as a name | The result of the attack will insert a script into the log table.  FINDING: Persistent XSS |
| **Mitigation** | | |

The mitigation to this is sanitize the inputs before allowing the name to be entered into the database

Accessing a web page that they are not authenticated to access

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | On any user authenticated page, the user tries to access a admin page through URL. | The result of the attack would allow an authenticated user access to access an admin only page.  FINDING: Session Management |
| **Mitigation** | | |

The mitigation to this is to have a different session variable for a user and an admin

CRSF on change password

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | On the change password page, the user enters the details to change the password using a GET request to the server.  After changing the password, they try and run the same command again in the URL. | The result of the attack would be to change the password by sending a get request through URL instead of the change password form.  FINDING: CRSF |
| **Mitigation** | | |

The mitigation to this is to have a session variable and a hidden field in the form and if they don’t match it fails and unset that variable after the password is changed through the form.

Brute force login

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | A user is on the login page tries to guess the username and password combination. | The result of the attack is to try and gain access to the website through guessing the username and password.  FINDING: |
| **Mitigation** | | |

The mitigation to this to block the user from entering the username and password for a fixed amount of time. The IP address and user agent are used to identify the user.

Brute force register

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | A user is on the register page tries to guess the username. | The result of the attack is to try and guess an already verified user by trying to guess their username so they would only need to guess the password to gain access.  FINDING: |
| **Mitigation** | | |

The mitigation to this to give a general message if the username is already in use.

After pressing logout button

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | A user is on the menu page presses logout. | The result shouldn’t allow for a user to press the back button to gain access to the page they were just on.  FINDING: Session Management |
| **Mitigation** | | |

The mitigation to this is to destroy the user’s current session when they press the logout button.

SQL Login screen

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | A user is on the login page and tries to enter a query (Admin OR 1=1) as a username to try and gain access to the site | The result would fail if there was no username “Admin” but would succeed with 1=1 because that’s true.  FINDING: SQL Injection |
| **Mitigation** | | |

The mitigation to this is to use prepared statements.

Trying to access an authenticated web page

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 8/2/2022 | A user tries to access a web page that requires a user to be authenticated. | The result of the attack is to gain access to web pages they are not authorised to access.  FINDING: Session Management |
| **Mitigation** | | |

The mitigation to this is to only allow users who enter the username and password correctly to have an active session with the site if they don’t redirect back to login page.

Trying to time the password authentication

|  |  |  |
| --- | --- | --- |
| **Date** | **Attack** | **Result** |
| 27/2/2022 | A user tries to guess the password by seeing how long it takes after before it gives back a message | The result of the attack is to gain access to web pages they are not authorised to access.  FINDING: Time based attack |
| **Mitigation** | | |

The mitigation to this to give the same time it takes for a password to fail or succeed so there is no way to time the password check. I have a bool value that is set to true at the start and only becomes false in the loop if the passwords don’t match but will continue to run until all characters have been checked.