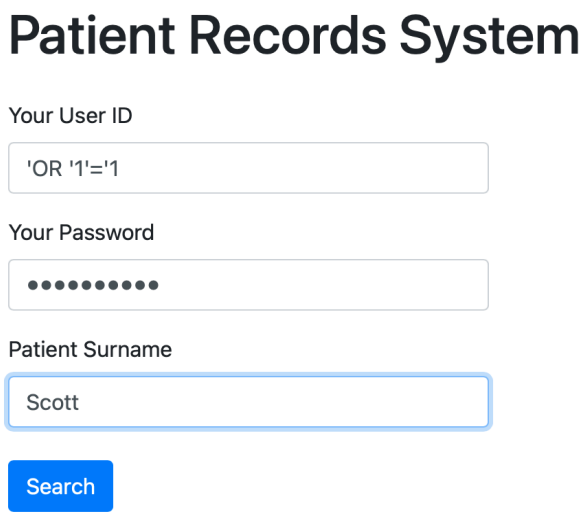
COMP3911 Coursework 2

Analysis of Flaws: The application exhibits a critical SQL Injection Vulnerability due to the lack of input sanitization, enabling attackers to manipulate SQL queries and potentially access or alter sensitive data. Furthermore, there is an exposure to Cross-Site Request Forgery (CSRF) attacks because user-generated content is rendered without proper escaping, allowing malicious scripts to be executed in other users' browsers. Lastly, the security of user accounts is jeopardized by the Insecure Password Storage flaw; passwords are stored in plain text within the database, making them easily accessible in the event of a data breach.

1. SQL Injection Vulnerability: Unsanitized Input Handling



SQL Injection is a prevalent attack vector where an adversary manipulates a standard SQL query to control the database. It allows unauthorized viewing, alteration, or retrieval of data by injecting malicious SQL commands through application inputs. The SQL Injection Vulnerability in our application was evidenced through the application's mishandling of user input. By inserting the string 'OR '1'='1 into both the User ID and Password fields, the system improperly processed the input as part of the SQL command, effectively bypassing authentication mechanisms and granting access to sensitive patient data. This flaw was confirmed by the application's response, which displayed the details of a patient without requiring legitimate credentials, as shown in the first image (refer to Image 1). The resulting patient details page, which should be restricted, was displayed with the patient's full information, including surname, forename, date of birth, GP identifier, and diagnosis, as evidenced in the second image. This critical security issue underscores the urgent need for input sanitization and the use of parameterized queries to prevent malicious data from being executed within SQL statements.

2. Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) [1] is a malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts. If the resource is vulnerable to CSRF, it will have no way of knowing which request was legitimate or malicious. The provided application is vulnerable to CSRF. This was tested by creating a malicious replica of the login page called bad.html, which on submission gives a valid post request and is able to get sensitive patient data. If correct login credentials are entered from this page, a successful POST request is sent to the server which retrieves user data even though the source of the request was from an unauthorized page,This critical vulnerability needs to be addressed with the use of CSRF session tokens.

3. Insecure Password Storage: Lack of Data Encryption

The vulnerability stemmed from the initial practice of storing user passwords in plain text, exposing a critical security risk. A security assessment, which included executing SQL commands such as `SELECT * FROM user` to examine the database, revealed the ease with which unauthorized parties could access and exploit these plaintext passwords. The output of the SQL query confirmed the presence of unencrypted passwords:

Before Encryption:

```
sqlite> SELECT * FROM user;
1|Nick Efford|nde|wysiwyg0
2|Mary Jones|mjones|marymary
3|Andrew Smith|aps|abcd1234
```

After Encryption:

```
sqlite> SELECT * FROM user;
1|Nick  Efford|nde|$2a$10$1yGmcJmhP25jqJIzc5u3GuxJHYxMLKL55fUbrZZlDia6XY9zmeYtO
2|Mary Jones|mjones|$2a$10$ydWs7iEeb.ynBCAvc030NuU9vfqXD7HZCiDEfOPA/kX0exkPskw.S
3|Andrew Smith|aps|$2a$10$w1.iX1lOgTvPjO9VA3lFtevabol6nzQdFhJbRM7n0v4Nn0KTFobYi
```

```
sqlite>
```

Fixes Implemented

Fixes Implemented for SQL Injection Vulnerability: To remedy the SQL Injection vulnerability, critical modifications were made to the AppServlet.java file. The authenticated and searchResults methods were refactored to utilize PreparedStatement objects, thereby preventing the injection of malicious SQL code via user input. Specifically, the authenticated method now securely processes the username and password, and the searchResults method properly handles the surname input. By parameterizing inputs and using prepared statements, the application robustly mitigates the risk of SQL injection. Post-implementation testing confirmed the vulnerability was fixed, with the application now correctly displaying "no record found" for injection attempts, reflecting the successful safeguarding of the database interactions.

Fixes Implemented for CSRF To fix the CSRF vulnerability. Modifications are made to AppServlet.java. doGet() and doPost() methods are modified to include CSRF tokens. The doGet() method has the CSRF token initialized and passed on to the login_page.html file. When the User attempts to log in, a POST request is made which is when the doPost() method will extract the CSRF token and compare it to the token stored in the user's session. If the token is invalid or does not exist, a 403 Forbidden error message is supplied to the user. Post-implementation testing confirmed that bad requests were no longer allowed from the bad.html file, with accurate error messages being displayed to the user.These tests confirmed that unauthorized requests were effectively blocked and appropriate error messages were displayed to the user, validating the effectiveness of the CSRF protection implemented.

Fixes Implemented for Lack of Data Encryption: In addressing the lack of data encryption, the application was updated to utilize **bcrypt** hashing, ensuring enhanced password security. This involved modifying the database schema to store 60-character **bcrypt** hashes, a significant increase from the original 8-character size, while still allowing the original passwords to function seamlessly. Additionally, the **bcrypt** hashing algorithm was integrated into the AppServlet.java for existing user passwords. The authentication logic underwent revision to validate passwords against the **bcrypt** hashes, providing robust protection. Comprehensive testing confirmed the secure handling of passwords and the maintenance of the application's functionality, thus significantly bolstering security against potential brute-force attacks.

References:

[1] OWASP n.d. Cross Site Request Forgery (CSRF). Owasp.org. [Online]. [Accessed 4th December 2023]. Available from: https://owasp.org/www-community/attacks/csrf.

[2] Class BCrypt. Spring.IO. [Online]. [Accessed 6th December 2023]. Available from: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html