Aston University
BIRMINGHAM UK

Department of Computer Science
College of Engineering & Physical Sciences

CS3SP Coursework

Jonah Reader & Khaled Qasim

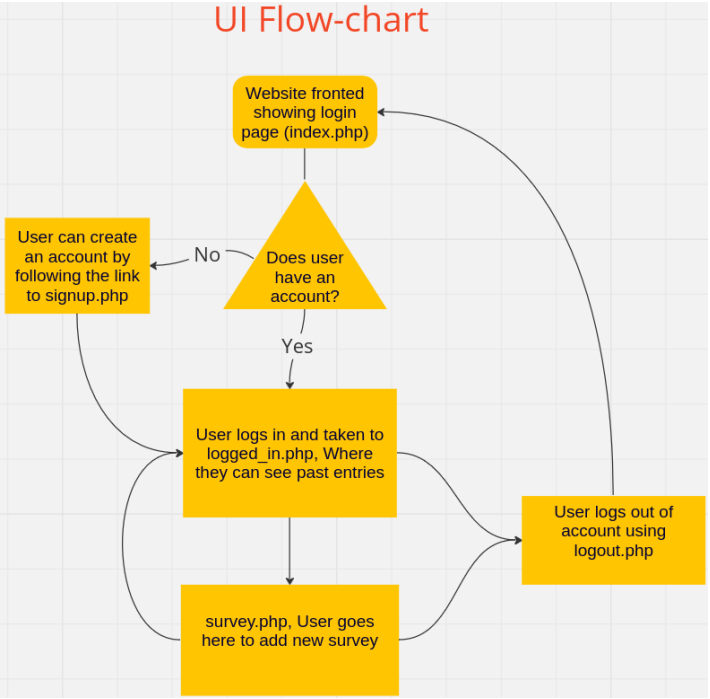
ClarityCheck

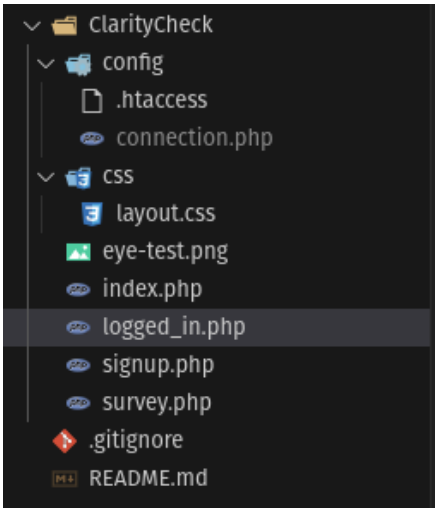
Table of Contents

Introduction to our Application

Clarity Check is a web application where users can take a survey to determine their eye prescriptions and view their past eye prescriptions allowing them to monitor their eye health throughout life.

Clarity Check is fronted by a login page requiring the user to provide a username and password. Upon logging in, the user is taken to their personal prescription portal where they can view their past prescriptions. Within this portal, a link can be found inviting the user to fill out a new survey and then it is added to a persistent storage that is unique to the user that filled it out.





User Security

One of the most important aspects of application security is making sure users and their accounts are protected. To achieve this, several controls were implemented during the signup and login process.

The first control implemented during the signup process was password validation. This follows best practice by requiring the user to provide a password with at least eight characters including an uppercase and lowercase letter, a special character, and a number. Without this mitigation, a user would be able to input a weak password (CWE-521) such as “123” which is easily vulnerable to brute force attacks. This was implemented on both the client-side and server-side as an attacker can manipulate POST requests to bypass the validation if there is no server-side validation.

```
<!-- Mitigation: Client-side password validation -->
<label for="password"><b>Password</b></label>
<input
type="password"
placeholder="Enter Password"
name="password"
pattern="^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
oninvalid="this.setCustomValidity('Password must contain at least 8 characters including one uppercase and lowercase letter, one number, and one special character')"
oninput ="setCustomValidity('')"
required>

<button name="submit" type="submit">Create Account</button>
<a href="index.php"> Already have an account? </a>
</div>
</form>
```

```
// Mitigation: Server-side password validation (https://www.codexworld.com/how-to/validate-password-strength-in-php/)
$uppercase = preg_match('@[A-Z]@', $password);
$lowercase = preg_match('@[a-z]@', $password);
$number    = preg_match('@[0-9]@', $password);
$specialChars = preg_match('@[^\w]@', $password);

if(!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 8) {

    echo 'Password must contain at least 8 characters including one uppercase and lowercase letter, one number, and one special character';

} else {
```

The second control implemented was password hashing. Password hashing is the practice of algorithmically turning a password into ciphertext, or an irreversibly obfuscated version of itself (Stytch, 2022). Password hashing helps to protect against packet sniffing attacks and is especially important to have on HTTP websites where passwords are sent in plaintext (CWE-256), and packets are not encrypted. It also protects the user in the case where the database table is breached, and their credentials are exposed as the attacker would have to crack the hashed password requiring additional time and effort.

```
# Mitigation - Password hashing
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

```
if (password_verify($password, $hashed_password)) {
```

```
MariaDB [coursework]> select * from users;
+----------+-------------------------------------------------------------+
| username | password                                                    |
+----------+-------------------------------------------------------------+
| jonah    | $2y$10$W1k3upf.JDMk6uYaf7RNMe4.rMQSyCXY5/szf8zX6o1U0OryS2ULW |
```

The final control implemented was the use of prepared statements to prevent SQL injection (CWE-89). This attack falls under Injection within the OWASP Top 10 and is ranked the third highest security risk for web applications. Prepared statements are best practice for preventing SQL injection as they separate the data and the query so that the query cannot be maliciously manipulated and only data can be inputted.

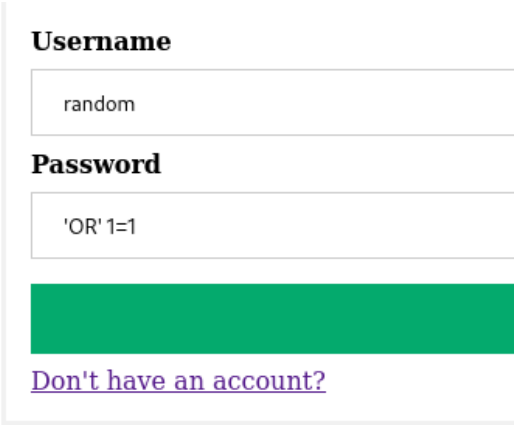
```
# Insecure code - Query vulnerbale to SQL Injection
$sql = "INSERT INTO users VALUES('". $username . "', '" . $password . "');";
$result = mysqli_query($con, $sql);

# Mitigation: Prepared Statement
$params = array($username, $hashed_password);
$result = $con->execute_query("INSERT INTO users VALUES(?, ?)", $params);
```

```
// Insecure code - Query vulnerbale to SQL Injection
$sql = "SELECT * FROM users WHERE username='". $username . "' AND password='" . $password . "';";
$result = mysqli_query($con, $sql);

// Mitigation: Prepared statement
$params = array($username);
$result = $con->execute_query("SELECT * FROM users WHERE username=?", $params);
```

Upon penetration testing the insecure code, I was able to login as any user by entering a random username and 'OR' 1=1 as the password value. I discovered that it was also possible to inject multiple, and more dangerous statements such as 'OR' 1=1; DROP TABLE users if the PHP programming supports multiple statements. For example, mysqli_multi_query().



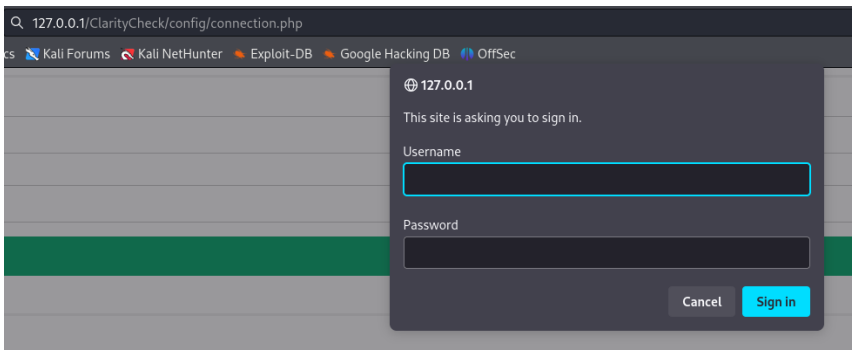
random Successfully logged in!
View your prescriptions here.

Apache Server Security

It is important to consider security-in-depth when developing an application and as many security controls should be implemented as possible whether it is the configuration of the server or secure programming.

The first Apache control implemented was separating the MySQL connection details from the root directory and restricting access to it through an .htaccess file. An attacker would need to login with credentials hidden on the server in the .htpasswd file to access it.

```
AuthType Basic
AuthName "Restricted Content"
AuthUserFile /etc/apache2/.htpasswd
Require valid-user
```



The second Apache control implemented was disabling Apache2 modules such as autoindex, status, info, and userdir. These modules give an attacker sensitive information like the version, IP, and port of the Apache2 server as well as the architecture of the application, which can be used to form attack vectors.

```
┌──(kali㉿kali)-[~]
└─$ sudo a2dismod autoindex
```

Index of /ClarityCheck

Name	Last modified	Size	Description
Parent Directory		-	
ClarityCheck/	2023-11-08 15:56	-	
README.md	2023-11-08 15:42	41	
Report/	2023-11-08 15:42	-	

Apache/2.4.58 (Debian) Server at 127.0.0.1 Port 80

There are many other Apache hardening techniques that we would implement before launching this application such as firewalls, HTTPS, security headers and content-security-policy (CSP), multi-factor authentication (MFA), and disabling unused network services like FTP to help prevent an attacker from injecting a web shell (Tutorialspoint, 2023).

Prescription Portal Security

Once a user successfully creates an account, they will be taken to their own personal content page located in file `logged_in.php`

This page uses prepared statements to prevent SQL injection (CWE-89), as mentioned above in the report. This is needed because the code uses '$_SESSION["username"]' global variable as input to search for the matching username in the database and retrieve the user's survey information.

```
// Insecure code - Query vulnerbale to SQL Injection    "vulnerbale": Unknown word.
//$sql = "SELECT * FROM users WHERE username='". $username."';
//$result = mysqli_query($con, $sql);

// Mitigation: Prepared statement
$username = $_SESSION['username'];
$params = array($username);
$result = $con->execute_query("SELECT data FROM user_prescriptions WHERE username=?", $params);
```

Prevention Against XSS and Stored XSS attacks

Cross Site Scripting (XSS) is client-side code injection attack. (Acunetix) (2017)
The attacker embeds malicious code into a genuine website, so when the user visits this site, the user's browser will execute the malicious code and exfiltrate the user's current session / cookies to the attacker.
A web page or web application is vulnerable to XSS if it uses Un-sanitized user input in the output that it generates (Acunetix) (2017)

That is why there is input validation on all data that is inputted into our WebApp.

The 'survey.php' contains the survey which takes in the user's input and saves it into the database. Below is a screenshot of our code that runs input and type validation to make sure that the correct data is stored in our database.

```
$stored_xss_attack = "<script type='text/javascript' >
            alert('Your Are Hacked!!');
            </script>";
$html =
"
 $stored_xss_attack

 Age: " . $new_value1[1] . ",
 Do you feel any irritation in your eyes?: " . $value2 . ",
 Do you currently wear glasses?: " . $value3 . ",
 Right Eye: +" . $value4 . ",
 Left Eye: +" . $new_value5 . "

";
// code vulnerable to Stored XSS attack, when moving the variable $stored_xss_attack inside the  $html variable, the script is executed
echo $html;      You, 4 hours ago • Code Functionality Completed, still needs securin…
```

```php
// do some server side validation before inserting into database
$age = $_POST["age"];
$irritation = $_POST["irritation"];
$glasses = $_POST["glasses"];
$right_eye = $_POST["right-eye"];
$left_eye = $_POST["left-eye"];

if (!filter_var($age, FILTER_VALIDATE_INT, ["options" => ["max_range" <= 100],["min_range" => 1]]) !== false) {
    echo "<h1 style='color: red'>Invalid age range, must be between 0 to 100</h1>";
    exit();
}
if ($irritation !== "Yes" && $irritation !== "No") {
    echo "<h1 style='color: red'>Invalid irritation value, must be Yes or No</h1>";
    exit();
}
if ($glasses !== "Yes" && $glasses !== "No") {
    echo "<h1 style='color: red'>Invalid glasses value, must be Yes or No</h1>";
    exit();
}
if (!filter_var($right_eye, FILTER_VALIDATE_FLOAT, ["options" => ["max_range" => 5],["min_range" > 0]]) !== false) {
    echo "<h1 style='color: red'>Invalid right eye range value, must be below 5 and above 0</h1>";
    exit();
}
if (!filter_var($left_eye, FILTER_VALIDATE_FLOAT, ["options" => ["max_range" => 5],["min_range" > 0],["decimal" < 2]]) !== false) {
    echo "<h1 style='color: red'>Invalid left eye range value, must be below 5 and above 0</h1>";
    exit();
}      You, 30 seconds ago • Uncommitted changes

$left_eye = number_format($left_eye, 2);
$right_eye = number_format($right_eye, 2);
```

There are also Stored XSS attacks, which is when malicious code finds its way inside persistence storage inside our WebApp. For this reason, it is important to sanitize code when displaying it back to the user, especially if the data came from the user to begin with.

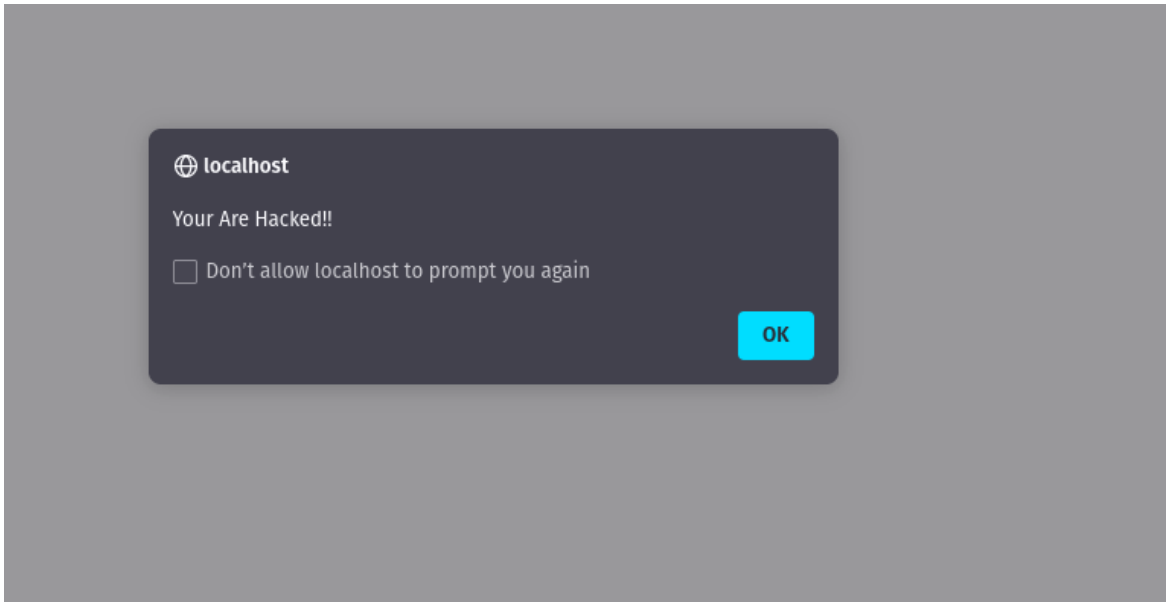
Below is a screenshot of our code inside the file ‘logged_in.php’ using the function ‘htmlspecialchars()’ this is needed because we are displaying html data with user input, and it must be sanitized using the mentioned function.

```php
$html =
"


 Age: " . $new_value1[1] . ",
 Do you feel any irritation in your eyes?: " . $value2 . ",
 Do you currently wear glasses?: " . $value3 . ",
 Right Eye: +" . $value4 . ",
 Left Eye: +" . $new_value5 . "

";
// code vulnerable to Stored XSS attack, when moving the variable $stored_xss_attack inside the  $html variable, the script is executed
// echo $html;
// MITIGATION, htmlspecialchars() function will not execute the script , instead will just print it out as a string
echo '<h3><br>' .htmlspecialchars($html, ENT_QUOTES, 'UTF-8'). '</h3>';
```

Now instead of using the ’htmlspecialchars()’ function and just echoing the data we could fall victim to a stored XSS attack. As shown below, I have a variable containing a script, I will add this variable to be executed by the ‘echo $html’ which will then run the script in my browser.


localhost
Your Are Hacked!!
Don't allow localhost to prompt you again
OK

References

Code:

- CodexWorld (2022), How to Validate Password Strength in PHP, [online] Available at https://www.codexworld.com/how-to/validate-password-strength-in-php/ [Accessed 7 November 2023]
- Dreamhost (2023), Password protecting your site with an .htaccess file, [online] Available at https://help.dreamhost.com/hc/en-us/articles/216363187-Password-protecting-your-site-with-an-htaccess-file [Accessed 2 November 2023]
- W3Schools (2023), How to Create a Login Form, [online] Available at: https://www.w3schools.com/howto/howto_css_login_form.asp [Accessed 2 November 2023]

Report:

- Tutorialspoint (2023), 10 Apache Web Server Security and Hardening Tips, [online] Available at https://www.tutorialspoint.com/10-apache-web-server-security-and-hardening-tips [Accessed 8 November 2023]
- Sytch (2023), What is password hashing?, [online] Available at: https://stytch.com/blog/what-is-password-hashing/#:~:text=Password%20hashing%20is%20the%20practice,the%20threat%20of%20password%20breaches [Accessed 7 November 2023]
- Acunetix (2017). What is Cross-site Scripting and How Can You Fix it? [online] Acunetix. Available at: https://www.acunetix.com/websitesecurity/cross-site-scripting/.

Source Code

Connection.php:

```php
<?php

# Mitigation - Hide MySQL Connection details in seperate directory/file and restrict access through .htaccess file
# Reference 4.0 & 4.2 - https://www.baeldung.com/linux/apache-restrict-specific-directory#:~:text=Password%2Dprotect%20a%20Directory&text=We%20can%20configure%20the%20Apache2,restrict%20access%20to%20any%20directory.&text=If%20it's%20the%20first%20time,a%20password%20for%20the%20user.


# Connect to database

$dbhost = "localhost";
$dbuser = "root";
$dbpass = "SuperStrongPassword^123?!?!";
$dbname = "coursework";

if(!$con = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname))
{
    die("failed to connect!");
}

?>
```

.htaccess:

```
# Reference - https://help.dreamhost.com/hc/en-us/articles/216363187-Password-protecting-your-site-with-an-htaccess-file

AuthType Basic
AuthName "Restricted Content"
AuthUserFile /etc/apache2/.htpasswd
Require valid-user
```

Index.php:

index.php

ClarityCheck > index.php

```php
<?php
// Imports the mysql database connection settings then checks for the existence of the users table and user_prescriptions table.
// If they do not exist, they are created.
session_destroy();
include("config/connection.php");

$table_users = $con->query("SHOW TABLES LIKE 'users'");


if ($table_users->num_rows <= 0) {

    $sql_users = "CREATE TABLE users (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        username VARCHAR(255) NOT NULL,
        password TEXT NOT NULL,
        reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    )";

    $con->query($sql_users);
    $table_users->free();
}
$table_user_prescriptions = $con->query("SHOW TABLES LIKE 'user_prescriptions'");
if ($table_user_prescriptions->num_rows <= 0) {
    $sql_user_prescriptions = "CREATE TABLE user_prescriptions (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        username VARCHAR(255) NOT NULL,
        data TEXT NOT NULL,
        reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    )";

    $con->query($sql_user_prescriptions);
    $table_user_prescriptions->free();
}


if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['logout.php'])){
    session_destroy();


}
}


?>
```

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title> Clarity Check Login </title>
    <link href="css/layout.css" rel="stylesheet" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>

    <!-- Form and CSS styling copied from https://www.w3schools.com/howto/howto_css_login_form.asp -->
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <div class="container">
            <label for="username"><b>Username</b></label>
            <input type="text" placeholder="Enter Username" name="username" required>

            <!-- Mitigation: Type password hides passwords from being shown in clear text and prevents shoulder surfing -->
            <label for="password"><b>Password</b></label>
            <input type="password" placeholder="Enter Password" name="password" required>

            <button class="default-button" name="submit" type="submit">Login</button>
            <a href="signup.php"> Don't have an account? </a>
        </div>
    </form>

    <?php

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        if (isset($_POST['submit'])) {

            $username = $_POST["username"];
            $password = $_POST["password"];

            // Insecure code - Query vulnerbale to SQL Injection
            //$sql = "SELECT * FROM users WHERE username='". $username . "' AND password='" . $password . "';";
            //$result = mysqli_query($con, $sql);

            // Mitigation: Prepared statement
            $params = array($username);
            $result = $con->execute_query("SELECT * FROM users WHERE username=?", $params);
```

```
90                  if (mysqli_num_rows($result) > 0) {
91                      $user_data = mysqli_fetch_assoc($result);
92                      $hashed_password = $user_data["password"];
93
94                      if (password_verify($password, $hashed_password)) {
95
96                          session_start();
97                          $_SESSION['login'] = true;
98                          $_SESSION['username'] = $username;
99                          // header("Location: logged_in.php");
100                         header("Location: logged_in.php");
101
102                     } else {
103                         echo "<h3 style='color: red;'>Password is Invalid</h3>";
104                     }
105
106                 } else {
107                     echo "<h3 style='color: red;'>Username or Password is invalid</h3>";
108                 }
109             }
110
111         }
112         ?>
113
114     </body>
115
116     </html>
```

Singup.php:

signup.php

ClarityCheck > signup.php

```php
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title> Clarity Check Signup </title>
    <link href="css/layout.css" rel="stylesheet" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
    <!-- Form and CSS styling copied from https://www.w3schools.com/howto/howto_css_login_form.asp -->
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <div class="container">

            <!-- Validate username -->
            <label for="username"><b>Username</b></label>
            <input type="text" placeholder="Enter Username" name="username" pattern="^[a-zA-Z0-9]*$"
                oninvalid="this.setCustomValidity('Username must only contain letters and numbers')"
                oninput="setCustomValidity('')" required>

            <!-- Mitigation: Client-side password validation -->
            <label for="password"><b>Password</b></label>
            <input type="password" placeholder="Enter Password" name="password"
                pattern="^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
                oninvalid="this.setCustomValidity('Password must contain at least 8 characters including one uppercase and lowercase letter,
                one number, and one special character')"
                oninput="setCustomValidity('')" required>

            <button class="default-button" name="submit" type="submit">Create Account</button>
            <a href="index.php"> Already have an account? </a>
        </div>
    </form>

    <?php
    include("config/connection.php");

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        if (isset($_POST['submit'])) {
            $username = $_POST["username"];
            $password = $_POST["password"];

            $params_check_unique_username = array($username);
            $result = $con->execute_query("SELECT * FROM users WHERE username=?", $params_check_unique_username);
```

```
45                  if (mysqli_num_rows($result) > 0) {
46
47                      echo "<h3 style='color: red;'>Username already taken, Please try again with a different username</h3>";
48
49
50                  } else {
51                      // Mitigation: Server-side password validation (https://www.codexworld.com/how-to/validate-password-strength-in-php/)
52                      $uppercase = preg_match('@[A-Z]@', $password);
53                      $lowercase = preg_match('@[a-z]@', $password);
54                      $number = preg_match('@[0-9]@', $password);
55                      $specialChars = preg_match('@[^\w]@', $password);
56
57                      if (!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 8) {
58
59                          echo 'Password must contain at least 8 characters including one uppercase and lowercase letter, one number, and one special
                            character';
60
61                      } else {
62                          # Mitigation - Password hashing
63                          $hashed_password = password_hash($password, PASSWORD_DEFAULT);
64
65
66
67                          # Insecure code - Query vulnerbale to SQL Injection
68                          # $sql = "INSERT INTO users VALUES('". $username . "', '" . $password . "');";
69                          # $result = mysqli_query($con, $sql);
70
71                          # Mitigation - Prepares, binds parameters, and executes SQL statement (https://www.php.net/manual/en/mysqli.execute-query.
                            php)
72                          $params = array($username, $hashed_password);
73                          $query = "INSERT INTO users (username, password) VALUES(?, ?)";
74                          $con->execute_query($query, $params);
75
76
77
78                          session_start();
79                          $_SESSION['login'] = true;
80                          $_SESSION['username'] = $username;
81                          header("Location: logged_in.php");
82                      }
83                  }
84
85
86              }
87          }
88          ?>
89      </body>
90      </html>
```

Logged_in.php:

logged_in.php

ClarityCheck > logged_in.php

```php
<?php
// Imports the mysql database connection settings then checks for the existence of the users table and user_prescriptions table.
// If they do not exist, they are created.
session_start();
include("config/connection.php");



?>

<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title> Clarity Check Login </title>
    <link href="css/layout.css" rel="stylesheet" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
    <form action="index.php" method="POST">
        <button type="submit" name="logout.php" class="round-button-top-right">LogOut</button>
    </form>


    <!-- Form and CSS styling copied from https://www.w3schools.com/howto/howto_css_login_form.asp -->


    <?php
    if ($_SESSION['login'] === true) {
        echo "<h1> Welcome: " . $_SESSION['username'] . "</h1>";
        $html =
            "


        <a href='survey.php'><h3>Take A New Eye Prescription Survey</h3></a>

        "

        ;
        echo $html;
        // Insecure code - Query vulnerbale to SQL Injection
        //$sql = "SELECT * FROM users WHERE username='". $username."';
        //$result = mysqli_query($con, $sql);
```

```php
        // Mitigation: Prepared statement
        $username = $_SESSION['username'];
        $params = array($username);
        $result = $con->execute_query("SELECT data FROM user_prescriptions WHERE username=?", $params);

        if ($result->num_rows > 0) {
            echo "<h3> Your Previous Survey Results: </h3>";

            while ($row["data"] = $result->fetch_assoc()) {

                $code_usable_array = var_export($row["data"], true);
                $dataArray = explode(",", $code_usable_array);


                $value1 = $dataArray[0];
                $value2 = $dataArray[1];
                $value3 = $dataArray[2];
                $value4 = $dataArray[3];
                $value5 = $dataArray[4];

                // fixing unwanted final values formatting
                $new_value1 = explode("=> '", $value1);
                $new_value5 = trim($value5, "'");

                // secure against this xss attack
                $stored_xss_attack = "<script type='text/javascript' >
                            alert('Your Are Hacked!!');
                            </script>";
                $html =
                "



                 Age: " . $new_value1[1] . ",
                 Do you feel any irritation in your eyes?: " . $value2 . ",
                 Do you currently wear glasses?: " . $value3 . ",
                 Right Eye: +" . $value4 . ",
                 Left Eye: +" . $new_value5 . "

                ";
                // code vulnerable to Stored XSS attack, when moving the variable $stored_xss_attack inside the  $html variable, the script is executed
                // echo $html;
                // MITIGATION, htmlspecialchars() function will not execute the script , instead will just print it out as a string
                echo '<h3><br>' .htmlspecialchars($html, ENT_QUOTES, 'UTF-8'). '</h3>';
            }
```

```
            // $user_data_array = explode(",",$user_data["data"]);
            // var_dump($user_data_array);
            // $html =
            //     "
            //     <h3> Your Previous Survey Results: </h3>
            //     <h4> Age: " . $user_data['data'] . "</h4>
            //     <h4> Do you feel any irritation in your eyes?: " . $user_data['irritation'] . "</h4>
            //     <h4> Do you currently wear glasses?: " . $user_data['glasses'] . "</h4>

            //     "
            // ;
            // echo $html;

        } else {
            echo "<h3> You have not taken a survey yet. </h3>";
        }



    } else {
        header("Location: index.php");
    }


    ?>



</body>

</html>
```

Survey.php:

signup.php M

ClarityCheck > signup.php

```php
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title> Clarity Check Signup </title>
    <link href="css/layout.css" rel="stylesheet" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
    <!-- Form and CSS styling copied from https://www.w3schools.com/howto/howto_css_login_form.asp -->
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <div class="container">

            <!-- Validate username -->
            <label for="username"><b>Username</b></label>
            <input type="text" placeholder="Enter Username" name="username" pattern="^[a-zA-Z0-9]*$"
                oninvalid="this.setCustomValidity('Username must only contain letters and numbers')"
                oninput="setCustomValidity('')" required>

            <!-- Mitigation: Client-side password validation -->
            <label for="password"><b>Password</b></label>
            <input type="password" placeholder="Enter Password" name="password"
                pattern="^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
                oninvalid="this.setCustomValidity('Password must contain at least 8 characters including one uppercase and lowercase letter,
                one number, and one special character')"
                oninput="setCustomValidity('')" required>

            <button class="default-button" name="submit" type="submit">Create Account</button>
            <a href="index.php"> Already have an account? </a>
        </div>
    </form>

    <?php
    include("config/connection.php");

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        if (isset($_POST['submit'])) {
            $username = $_POST["username"];
            $password = $_POST["password"];

            $params_check_unique_username = array($username);
            $result = $con->execute_query("SELECT * FROM users WHERE username=?", $params_check_unique_username);

            if (mysqli_num_rows($result) > 0) {

                echo "<h3 style='color: red;'>Username already taken, Please try again with a different username</h3>";
```

```
50              } else {
51                  // Mitigation: Server-side password validation (https://www.codexworld.com/how-to/validate-password-strength-in-php/)
52                  $uppercase = preg_match('@[A-Z]@', $password);
53                  $lowercase = preg_match('@[a-z]@', $password);
54                  $number = preg_match('@[0-9]@', $password);
55                  $specialChars = preg_match('@[^\w]@', $password);
56
57                  if (!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 8) {
58
59                      echo 'Password must contain at least 8 characters including one uppercase and lowercase letter, one number, and one special
                        character';
60
61                  } else {
62                      # Mitigation - Password hashing
63                      $hashed_password = password_hash($password, PASSWORD_DEFAULT);
64
65
66
67                      # Insecure code - Query vulnerbale to SQL Injection
68                      # $sql = "INSERT INTO users VALUES('". $username . "', '" . $password . "');";
69                      # $result = mysqli_query($con, $sql);
70
71                      # Mitigation - Prepares, binds parameters, and executes SQL statement (https://www.php.net/manual/en/mysqli.execute-query.
                        php)
72                      $params = array($username, $hashed_password);
73                      $query = "INSERT INTO users (username, password) VALUES(?, ?)";
74                      $con->execute_query($query, $params);
75
76
77
78                      session_start();
79                      $_SESSION['login'] = true;
80                      $_SESSION['username'] = $username;
81                      header("Location: logged_in.php");
82                  }
83              }
84
85
86          }
87      }
88      ?>
89  </body>
90  </html>
```

Layout.css:

layout.css

ClarityCheck > css > layout.css > ...

```
/* ~~~~~~~~~~~~~~~ LOGIN & SIGNUP PAGE (CSS styling taken from https://www.w3schools.com/howto/howto_css_login_form.asp) ~~~~~~~~~~~~~~ */

/* Bordered form */
form {
    border: 3px solid #f1f1f1;
}

  /* Full-width inputs */
input[type=text], input[type=password] {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  box-sizing: border-box;
}

  /* Set a style for all buttons */
.default-button {
  background-color: #04AA6D;
  color: white;
  padding: 14px 20px;
  margin: 8px 0;
  border: none;
  cursor: pointer;
  width: 100%;
}

  /* Add a hover effect for buttons */
.default-button:hover {
  opacity: 0.8;
}

  /* Add padding to containers */
.container {
  padding: 16px;
}



.round-button-top-right {
  position: fixed;
  top: 20px; /* Adjust this value to change the vertical position */
  right: 20px; /* Adjust this value to change the horizontal position */
  width: 100px;
  height: 100px;
  background-color: red;
  border: none;
  border-radius: 50%;
  color: white;
  text-align: center;
  font-size: 20px;
  cursor: pointer;
}

  /* Style for button hover effect (optional) */
.round-button-top-right:hover {
  background-color: darkred;
}
```

Contribution

Jonah Reader:

- Index.php & Signup.php
- .htaccess
- Layout.css
- Report pages – 3, 4-5, 6

Khaled Qasim:

- survey.php
- logged_in.php
- Layout.css
- Report pages – 3, 7-9