G6077 COMPUTER SECURITY - REPORT

Application URL Sussex: http://users.sussex.ac.uk/~bcc28/G6077/LovejoyAntiques/index.html

Code: Zip file Location: https://universityofsussex-my.sharepoint.com/:f:/g/personal/bcc28_sussex_ac_uk/EkTiNIBsFLVLtWCDUrYG1oUBCOTRG__4YAvKJVRgMOaodw?e=PrQZb

CONTENTS

TASK 1 – USER REGISTRATION

REGISTRATION FORM CODE

```
<!DOCTYPE html>
<html lang='en'>
    <head>
        <meta charset="UTF-8">
        <!-- Stylesheet handles basic structure of website to ensure cvlean and structured look. Found on Bootstrap website -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <title>Lovejoy Antiques Registeration Form</title>
        <!-- My own style sheet which handles the look of the website to ensure a consistent look over all interactive pages, and
            to style objects such as nav bars, tables for a consistent look across all pages.-->
        <link rel="stylesheet" href="style1.css">
    </head>

    <body>
        <!-- Divs used for fomratyting/CSS styling-->
            <div class="container">
                <div class="row">
                    <div class = "col-md-12">
                        <!-- Register form with all elements to be entered by the users-->
                        <h2><strong>LoveJoy Antique Evaluations: </strong><br>Registration Form</h2><br>
                        <p><strong>Please fill in the entire form to register for an account.</strong></p>
                        <!-- On Submit, redirects to Register.php-->
                        <form action="register.php" method="post" autocomplete="off">
                            <div class="form-group">
                                <label for="username"></label>
                            </div>
                            <!-- Textboxes for user to enter username, fullname, email & telephone number. These have HTML5 validation using
                                the attribute 'type' along with 'pattern' for email, fullname & username-->
                            <div class="form-group">
                                <input type="text" name="username" minlength="3" placeholder="Username" id="username" pattern="[A-Za-z0-9]+" class="form-control" required>
                                <label for="fullname"></label>
                            </div>
                            <!-- Ensure name is no less than 6 characters, and pattern only allows for letter, spaces and hypens. These are needed in case of double barrel names-->
                            <div class="form-group">
                                <input type="text" name="fullname" placeholder="Full name" minlength="6" id="fullname" class="form-control" pattern="[A-Za-z -]+" required>
                                <label for="email"></label>
                            </div>
                            <!-- HTML5 'Type' for email ensures an @ symbol along with a 'abc.abc' afterwords -->
                            <div class="form-group">
                                <input type="email" name="email" placeholder="Email" class="form-control" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$" required>
                                <label for="telephone"></label>
                            </div>
                            <!-- HTML5 'Type' for tel i.e. telephone ensures input is a phone number. The pattern set requires the user to enter the UK 0 then 10 numerical digits.-->
                            <div class="form-group">
                                <input type="tel" name="telephone" placeholder="Telephone Number" pattern="[0]{1}[0-9]{10}" class="form-control" required>
                                <label for="securityQuestion"></label>
                                <small>Format: 01234567890</small>
                            </div>
                            <!-- Dropdown box allows for users to select a security question of their choice. These are needed for password resets.-->
                            <div class="form-group">
                                <select name="securityQuestion" class="form-control">
                                    <option value="What is your Mothers maiden name?" selected>What is your Mothers maiden name?</option>
                                    <option value="What is the name of your first pet?">What is the name of your first pet?</option>
                                    <option value="What is your favourite movie?">What is your favourite movie?</option>
                                    <option value="What is your least favourite cereal?">What is your least favourite cereal?</option>
                                    <option value="In what city where you born?">In what city where you born?</option>
                                    <option value="What was the name of your first school?">What was the name of your first school?</option>
                                </select>
                                <label for="securityAnswer"></label>
                            </div>
                            <!-- Answer for Security Question -->
                            <div class="form-group">
                                <input type="text" name="securityAnswer" placeholder="Answer to Security Question" id="securityAnswer" pattern="[A-Za-z0-9 -]+" class="form-control" required>
                                <label for="password"></label>
                            </div>
                            <!-- Password for user. Min length 10 characters. Type 'Password' hides inputs as *s. Further validation is done in PHP -->
                            <div class="form-group">
                                <input type="password" name="password" placeholder="Password" id="password" class="form-control" minlength="10" required>
                                <label for="confirm_password"></label>
                                <small>Must be atleast 10 characters long & contain atleast one uppercase, lowercase, number & special character</small>
                            </div>
                            <!-- Confirm Password field. Same settings as Password field-->
                            <div class="form-group">
                                <input type="password" name="confirm_password" placeholder="Confirm Password" minlength="10" id="confirm_password" class="form-control" required>
                            </div>
                            <!-- Submit Button -->
                            <div class="form-group">
                                <input type="submit" class="btn btn-primary" value="Register">
                            </div>
                            <!-- Link to Login form -->
                            <p>Already have an account? <a href="login.php">Login here</a>.</p>
                        </form>
                    </div>
                </div>
            </div>>
    </body>
</html>
```

Figure 1 – index.html (registration form).

CODE WHEN REGISTRATION FORM SUBMITTED

```php
<?php
// Main.php includes db connection
include 'main.php';

// Sanitizing user inputs - Protection against SQL Injection attacks by escaping special chars which include " & '
$username = mysqli_real_escape_string($con, $_POST['username']);
$fullname = mysqli_real_escape_string($con, $_POST['fullname']);
$email = mysqli_real_escape_string($con, $_POST['email']);
$telephone = mysqli_real_escape_string($con, $_POST['telephone']);
$password = mysqli_real_escape_string($con, $_POST['password']);
$confirm_password = mysqli_real_escape_string($con, $_POST['confirm_password']);
$securityQuestion = mysqli_real_escape_string($con, $_POST['securityQuestion']);
$securityAnswer = mysqli_real_escape_string($con, $_POST['securityAnswer']);
// Further sanitizing - Prevention of XSS by changing special chars like <> to html entities
$username = htmlspecialchars($username);
$fullname = htmlspecialchars($fullname);
$email = htmlspecialchars($email);
$telephone = htmlspecialchars($telephone);
$password = htmlspecialchars($password);
$confirm_password = htmlspecialchars($confirm_password);
$securityQuestion = htmlspecialchars($securityQuestion);
$securityAnswer = htmlspecialchars($securityAnswer);
/* Checks for invalid email, empty fields done through HTML. E.g. using type='email' in the input tag & using 'required' in input tag
 Username & fullname have also been checked using HTML 'pattern' tag & must contain only characters and numbers, and letters,
 spaces & hypens (for double barrel names) respectively.*/

//Password Strength Check. Ensures atleast 1 uppercase, lowercase, number & special char is present, along wiht a min length
$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) < 10) {
    exit("Password should be at least 10 characters in length and should include at least one upper case letter,
        one number, and one special character. <a href='index.html'>Click to return to the registration form</a>");
}
/* Ensuring no data entered is above the database maximum capacity. Telephone does not need to be checked as HTML
 pattern attribute within input tag is used to ensure 11 digits entered*/
if(strlen($username) > 49 || strlen($fullname) > 99 || strlen($email) > 99 || strlen($password) > 100 || strlen($securityQuestion) > 99)  {
    exit("One of the inputs you have entered has has exceeded the maximum charcter length.
        Please revise this! <a href='index.html'>Click to return to the registration form</a>");
}
// Check pwds match
if ($confirm_password != $password) {
    exit("Passwords do not match!<a href='index.html'> Click to return to the registration form</a>");
}
// Check to see if Password is based on username. This checks for palindrome & if user replaces letters like s,e,o & l with number/other charcters alike.
$lc_password = strtolower($password);
$lc_username = strtolower($username);
$denum_pass = strtr($lc_password,'5301!','seoll');
if (($lc_password == $lc_username) || ($lc_password == strrev($lc_username)) || ($denum_pass == $lc_username) || ($denum_pass == strrev($lc_username))) {
    exit("Password cannot be based on the username!<a href='index.html'> Click to return to the registration form</a>");
}
```

Figure 2 – register.php, called when user submits registration form.

ANNOTATION DESCRIPTIONS

1. Sanitizing all inputs entered by the user to protect against SQL injection attacks
2. Further sanitation of inputs against Cross-Site Scripting XSS.
3. Password Entropy/Strength check, ensuring desired format of at least 1 uppercase, lowercase, number & special character is present along with a minimum of 10 characters.
4. Ensure length of inputs is reasonable and would not overload database, and then ensure confirmation password is equal to initial entered password.
5. Additional password requirement tested, which ensures password is not based at all on username, checking for palindromes of each other, along with common number/special character replacement of letters.

```php
//Firstly Ensure no account has the same username & email since these must be unique in database design
//Prepared statements protect users against SQL injection attacks since the query and the data are sent to the database server separately.
$stmt = $con->prepare('SELECT id, password FROM accounts WHERE username = ? OR email = ?');
// Binding parameters protects against SQl injection attacks
$stmt->bind_param('ss', $username, $email);
$stmt->execute();
$stmt->store_result();
// Result is stored to check if the account exists in the database.
if ($stmt->num_rows > 0) {
    // If username and/or email already exist - exit
    echo "Username and/or email already exists! <a href='index.html'>Click to return to the registration form</a>";
} else {
    $stmt->close();
    // If Username doesnt exists, insert new account into db
    $stmt = $con->prepare('INSERT INTO accounts (username, fullname, password, email, telephone ,activation_code, securityQuestion, securityAnswer, ip)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)');
    // Hashing the password and use password_verify when a user attempts to logs in.
    $passwordenc = password_hash($password, PASSWORD_DEFAULT);
    //Creates unique activation code
    $activate = uniqid();
    //Gets IP
    $ip = $_SERVER['REMOTE_ADDR'];
    //Storing all user inputs, with password hashed, IP for CSRF check/2FA check & activation code
    $stmt->bind_param('sssssssss', $username, $fullname ,$passwordenc, $email, $telephone , $activate, $securityQuestion, $securityAnswer, $ip);
    $stmt->execute();
    $stmt->close();
    // User is sent the activation email.
    $subject = 'Account Activation Required';
    $activate_link = 'http://users.sussex.ac.uk/~bcc28/G6077/LovejoyAntiques/activate.php?email=' . $email . '&code=' . $activate;
    $message = '<p>Please click the following link to activate your account!: <a href="' . $activate_link . '">' . $activate_link . '</a></p>';

    //Using Email Template for nice formatting
    $email_template = str_replace('%link%', $message, file_get_contents('activation.html'));
    //Adding subject & body to email
    if (sendEmail($email, $email_template, $subject)) {
        echo 'Message has been sent. Please check your email (and Junk Mail)';
    } else {
        echo 'Message cannot be sent. Mail Error Occured';
        header("Refresh:5; url=login.php");
    }
}
?>
```


Figure 3 – Register.php, insertion of successful registration into database, along with activation email sending.

ANNOTATION DESCRIPTIONS

1. All data entered is valid, SQL Select statement is prepared, this is to protect against SQL Injection
2. Check if username and/or email already exist in database. If so, exit code.
3. If both username & email are unique to previous accounts, prepare another SQL statement to insert the entered data into the database.
4. Encrypt the password, create a unique code for account activation, and store the IP of the current session and insert these along with all entered data into the table ‘accounts’.
5. Using function sendEmail() defined in main.php (see Figure 26), activation email is sent to the user with a link to click on. The link will contain the email address of the email & the activation code to ensure its unique to the user. Emails sent successfully will notify user to check inbox (and junk mail).

ACCOUNTS DATABASE TABLE

```sql
CREATE TABLE IF NOT EXISTS `accounts` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `username` varchar(50) NOT NULL,
    `fullname` varchar(100) NOT NULL,
    `password` varchar(255) NOT NULL,
    `email` varchar(100) NOT NULL UNIQUE,
    `telephone` varchar(11) NOT NULL,
    `activation_code` varchar(50) NOT NULL,
    `securityQuestion` varchar(100) NOT NULL,
    `securityAnswer` varchar(100) NOT NULL,
    `role` enum('Member','Admin') NOT NULL DEFAULT 'Member',
    `reset_code` varchar(50) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Figure 4 – accounts database table.

WHY THIS IS SECURE

PASSWORD POLICY

- I believe I have implemented a very good password policy, since the registration form enforces a password policy of a minimum of 10 characters, with at least 1 of each of those being, 1 uppercase letter, 1 lowercase letter, 1 number and 1 special character. (Figure 2, Annotation 3)
- With this, if a user chooses the weakest password from this policy i.e. 6 lowercase, 1 uppercase, 1 number and 1 special character, allows for 59bits of entropy or 5.9873694e+19 possible combinations which cannot be brute forced.
- In addition, the policy that the password cannot be based on the username (Figure 2, Annotation 5), ensures a brute force attack with known usernames of users will not be effective.
- The passwords are also encrypted using a hashing function and then stored in the database (Figure 3 Annotation 4). The function password_hash() creates a new password hash using a strong one-way hashing algorithm along with a randomly generated salt.
- In addition with this function, the un-hashed password does not ever need to be stored or decrypted to, due to the effective use of the function password_verify() which checks if a password entered is the same as the hashed stored password.
- Finally regarding password policy, the user must select 1 of 6 security questions and provide an answer, which is used when users need to reset their password. This ensures if a user's email is compromised, an unauthorised presence cannot reset a user's password.

VULNERABILITIES

- Vulnerabilities such as SQL injection, Cross-site scripting (XSS) and duplicate email/username account creation is protected against very well on this section. In addition using HTML validation, all fields are giving the attribute 'required' ensuring a user has entered something in these fields, and some use regex patterns to ensure only valid data is entered (Figure 1).
- For example, all fields have been given a pattern so characters not in the regex are not excepted. This means users cannot input special characters like '=, & and ;' which are required for SQL injection statements or '<>' used for XSS.
- In addition, all inputs of the users are sanitized using the PHP functions mysql_real_escape_string() and htmlspecialchars().
- mysql_real_escape_string() provides protection against SQL injection by escaping special characters in a string which can be used in SQL statements (Figure 2 Annotation 1).
- Htmlspecialchars() function provides protection against XSS attacks by converting some predefined characters to HTML entities (Figure 2 Annotation 2). The character replacement are as follows:
    - & (ampersand) becomes &
    - " (double quote) becomes "
    - ' (single quote) becomes '
    - < (less than) becomes <
    - > (greater than) becomes >
- This means XSS attacks like '<script>...' are converted to '<script>' which will not do anything malicious and will most likely be caught by the HTML validation.
- SQL Injection attacks are further prevented using prepared SQL statements (Figure 3 Annotation 1 & 3). This is because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.
- A final vulnerability protected against is allowing users to have the same username & email. If this was allowed, a user could create an account and potentially log into another user's account. This is protected using a Prepared SQL statement shown in (Figure 3 Annotation 1 & 2)

AUTHENTICATION

- Finally, in the authentication requirements, email account activation is sent to users after a successfully registration (Figure 3 Annotation 5).
- The user is sent an email with an activation link which is required to be clicked and accepted before being able to log in (Figure 27).
- This ensures fake email address, or users using email address they do not own, cannot be used to create, and register activated accounts.

TASK 2 - DEVELOP A SECURE LOGIN FEATURE.

LOGIN FORM CODE

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Lovejoy Antiques Login Form</title>
        <!-- CSS used for styling form to be usable and easy to traverse -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <link rel="stylesheet" href="style1.css">
        <!-- Source of Google ReCaptcha -->
        <script src="https://www.google.com/recaptcha/api.js" async defer></script>
    </head>
    <body>
        <div class="container">
            <div class="row">
                <div class = "col-md-12">
                    <h2><strong>LoveJoy Antique Evaluations: </strong><br>Login Screen</h2><br>
                    <!-- On submission, logincheck.php is called. This checks the inputs of the user &
                        if credential match a user in db, user is logged in -->
                    <form action="logincheck.php" method="post">
                        <div class="form-group">
                            <label for="username"></label>
                        </div>
                        <div class="form-group">
                            <!-- Text field for Username. Required tag ensures not empty -->
                            <input type="text" name="username" placeholder="Username" class="form-control" required>
                            <label for="password"></label>
                        </div>
                        <div class="form-group">
                            <!-- Password text field is also required. The type password 'hides' the input from the user-->
                            <input type="password" name="password" placeholder="Password" class="form-control" required>
                        </div>
                        <div class="form-group">
                            <!-- Link to forgot password page -->
                            <a href="forgotpassword.php">Forgot Password?</a>
                        </div>
                        <div class="form-group">
                            <!-- Google ReCaptcha on site. Used to stop botnet attacks required for Obfuscation -->
                        <div class="g-recaptcha" data-sitekey="6LcGjIYdAAAAANXPT-bad-hcaw464I12s4ty9fIY"></div>
                        </div>
                        <!-- Input Type hidden allows for data that cannot be seen or modified by users when a form is submitted.
                            Stores the unique token as the value to be passed to the logincheck.php Used in CSRF Protection -->
                        <input type="hidden" name="token" value="<?=$_SESSION['token']?>">
                        <div class="form-group">
                            <!-- Submit button -->
                            <input type="submit" name="submit" class="btn btn-primary" value="Login">
                        </div>
                        <!-- Link to registration form -->
                        <p>Don't have an account? <a href="index.html">Register here</a>.</p>
                    </form>
                </div>
            </div>
        </div>
```

1 2 3 4

Figure 5 – Login.php, form HTML code

ANNOTATION DESCRIPTIONS

1. JavaScript source for Google ReCAPTCHA.
2. Link to forgot password page (Figure 10).
3. Site Key for Google ReCAPTCHA, along with HTML code to present ReCAPTCHA.
4. Hidden session token used for CSRF Protection

```
include 'main.php';
// No need for the user to see the login form if they're logged-in so redirect them to the home page
if (isset($_SESSION['loggedin'])) {
    // If the user is logged in redirect to the Request Evaluation page
    header('Location: requestEval.php');
    exit;
}
// CSRF Protection
// When the user logs in, each & every login will require a 'token' that will be checked using Sessions in PHP
$_SESSION['token'] = md5(uniqid(rand(), true));
?>
```

1 2

Figure 6 – Login.php, php code checks if user is logged in, and redirects to evaluation page if they are

ANNOTATION DESCRIPTIONS

1. Checks if user is already logged in, redirects to evaluations page if so.
2. Creation of a random code used as a token to reference session is still the same.

CODE WHEN LOGIN FORM SUBMITTED

```php
<?php
include 'main.php';
//Brute force protection using Number of Attempts - Needed for obfuscation
/*Function is defined in main.php, and by giving $login_attempts as false, does not count as a login attempt.
Used to check if the user has any login atempts left, or if they have used all 5 attempts for the 10 mins*/
$login_attempts = loginAttempts($con, FALSE);
//If User has no attempts left, let user know to try again in 10 mins
if ($login_attempts == 0) {
    exit('Unable to login! Please try again in 10 mins');
}

// Token generated in login.php is checked whether it hasnt been sent OR whether its different to the token in the session
if (!isset($_POST['token']) || $_POST['token'] != $_SESSION['token']) {
    //Token invalid, user must try again
    exit('Incorrect token provided!');
}

// Code for checking Captcha has been completed!
if (isset($_POST['submit']) && $_POST['g-recaptcha-response'] != "") {
    // Secret Key for Captcha
    $secret = '6LcGjIYdAAAAAO4hhuI4VBPYBs3VHts6G3wEs_1z';
    //Verification of Response i.e. has response been completed correctly
    $verifyResponse = file_get_contents('https://www.google.com/recaptcha/api/siteverify?secret=' . $secret . '&response=' . $_POST['g-recaptcha-response']);
    $responseData = json_decode($verifyResponse);
    // If successful completion, continue
    if ($responseData->success) {
        ;
    } else {
        //Otherwise, exit PHP code and notify user
        exit("Error with Captcha! Please try again!<br><a href='login.php'> Click to return to the login form</a>");
    }
} else {
    //If not completed, notify user
    exit("Please complete Captcha to Login!<a href='login.php'> Click to return to the login form</a>");
}
```

Figure 7 – Logincheck.php, Brute force protection, CSRF Protection & Botnet Attack Prevention

ANNOTATION DESCRIPTIONS

1. Using login attempts function defined in main.php (Figure 25), if the user has failed 5 login attempts within the last 10 minutes, the user must wait 10 minutes from the last attempt to try again.
2. Checks if token created before logging is the same as now on submission.
3. Checks if the Google ReCAPTCHA has been completed correctly.

```php
// Protection against SQL injections & Cross-site scripting
$usernameentered = $usernameentered = mysqli_real_escape_string($con, $_POST['username']);
$usernameentered = htmlspecialchars($usernameentered);
$passwordentered = $passwordentered = mysqli_real_escape_string($con, $_POST['password']);
$passwordentered = htmlspecialchars($passwordentered);
// No empty checks needed due to 'required' property in Input Tags

// Code to prepare our SQL statement, preparing the SQL statement will further help with the prevention of SQL injection.
//Gets the user ID, password, activation code, role, ip & email of the user from the inputted username
$stmt = $con->prepare('SELECT id, password, activation_code, role, ip, email FROM accounts WHERE username = ?');
// Bind username entered to parameter username
$stmt->bind_param('s', $usernameentered);
$stmt->execute();
// Store the result so we can check if the account exists in the database.
$stmt->store_result();
// Check if the account exists:
if ($stmt->num_rows > 0) {
    //Bind the data retrieeved to variables.
    $stmt->bind_result($id, $password, $activation_code, $role, $ip, $email);
    $stmt->fetch();
    $stmt->close();
    // Account exists, now we verify the password. To do this, use function password_verify()
    //Does not required $password from data to be unhashed
    if (password_verify($passwordentered, $password)) {
        // Check if the account is activated, i.e. does not have an activation code stored in database
        if ($activation_code != 'activated') {
            // If user hasn't activated their account, output this msg which gives a link to resend account activation
            echo 'Please activate your account to login, <a href="resendactivation.php">Click here</a> to resend the activation email!';
            /* 2FA required if IP stored in database is not the same as current IP.
            The IP stored in the database is the IP of the session when user registered*/
        } else if ($_SERVER['REMOTE_ADDR'] != $ip) {
            //uniqid creates a unique code for 2FA
            $_SESSION['2FA'] = uniqid();
            //User id, email & 2FA code stored in URL. Checked on 2FA page so only verified users can access the page
            $link = 'twofactor.php?id=' . $id . '&email=' . $email . '&code=' . $_SESSION['2FA'];
            //Redirect to this page
            header("location: $link");
        } else {
            //User login was successful
            //Created a sessions to know the user that is logged in.
            //This function Updates the current session id with a newly generated one
            session_regenerate_id();
            //Loggedin set as TRUE, username, id & role all stored in the session
            $_SESSION['loggedin'] = TRUE;
            $_SESSION['name'] = $usernameentered;
            $_SESSION['id'] = $id;
            $_SESSION['role'] = $role;
            //if user is a member, redirected to Request Evaluation Page
            if ($role == 'Member') {
                header("Refresh:1; url=requestEval.php");
            }
            else {
                //Otherwise, if user is an Admin, redirect to view Evaluations Admin Page
                header("Refresh:1; url=viewRequests.php");
            }
        }
    } else {
        // Incorrect Password. Users number of attempts decreased by 1
        $login_attempts = loginAttempts($con, TRUE);
        echo 'Incorrect Username/Password, you have ' . $login_attempts . ' attempts remaining!';
        //Redirected to Login page
        header("Refresh:2; url=login.php");
    }
} else {
    // Incorrect Username, Users number of attempts decreased by 1
    $login_attempts = loginAttempts($con, TRUE);
    echo 'Incorrect Username/Password, you have ' . $login_attempts . ' attempts remaining!';
    header("Refresh:2; url=login.php");
}
?>
```

Figure 8 – Logincheck.php, SQL Injection & XSS prevention, along with checking login details.

ANNOTATION DESCRIPTIONS

1. Using predefined PHP functions to sanitize user inputs, to prevent SQL Injection and Cross-Site Scripting Software Attacks.
2. Prepared SQL statement (used to further prevent SQL injection), Selects user ID, password, activation code, role, IP & email address stored in database with matching entered username.
3. If username exists in database, binding of parameters to variables.
4. Verify the password entered is the exact same as the one stored in the database.
5. If user is yet to activate their account, give the user link to resend activation email.
6. If the IP address of the login is different to the last logon (or when the user registered), ask the user to complete two factor authentication (Figure 30). The email & and a code are stored in the URL of the page.

7. If successful logon, store username, id & role in session. If the user is an admin, redirect to view Evaluations page, otherwise, redirect user to request evaluations page.
8. If the incorrect username and/or password is entered, call login attempts, with TRUE (i.e. unsuccessful login attempt), and let the user know how many attempts they have left. Then redirect back to login page.

LOGIN ATTEMPTS DATABASE TABLE

```
CREATE TABLE IF NOT EXISTS `login_attempts` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `ip_address` varchar(255) NOT NULL,
    `attempts_left` tinyint(1) NOT NULL DEFAULT '5',
    `date` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Figure 9 – Table login_attempts store the IP address & dates of failed logins, and locks users accounts for 10 minutes after 5 failed attempts.

WHY THIS IS SECURE

PASSWORD POLICY

- Password in database remains encrypted and is not stored decrypted. Using function password_verify(), the password entered can be compared to stored password (Figure 8 Annotation 4).

VULNERABILTIES

- Using HTML validation, i.e. the required tag, the user submits the form with uncompleted fields (Figure 5).
- XSS and SQL Injection protection for username & password entered by user removing special characters used for XSS and escaping characters from the string used for SQL injection attacks (Figure 8 Annotation 1).
- Prepared statement also used to check username exists in database before any comparison is made (Figure 8 Annotation 2).
- Protection against CSRF (Cross-Site Request Forgery) by checking the token created on the login form is the same after submitting the form (Figure 7 Annotation 2).
- Therefore, the site can distinguish between legitimate authorized requests and forged authenticated requests, thwarting a potential attack by verifying the identity and authority of the requester.
- This is required as If the victim is an admin, CSRF can compromise the entire web application.
- I believe all countermeasures implemented here are of a good quality, with CSRF Implementation being somewhat simple but effective.

AUTHENTICATION

- Moreover, I have implemented features which require authentication before a user can login.
- If the IP address stored in the database for the account is different to the IP of the current login request, the user will be required to complete Two-Factor Authentication (2FA) (Figure 8 Annotation 6).
- The IP address for each login is checked and is initially stored on registration (Figure 3 Annotation 3).

- After a successful two factor authentication, the IP address is updated in the accounts table of the database, for example if the user has moved from their phone to their laptop (Figure 30 Annotation 5).
- By implementing this, even if a user's username & password is compromised, another user will not be able to login to the account unless they can present the 2FA code, sent only via email.
- Any user will not be able to access the Two-Factor authentication page as it requires a valid email address and code, stored in the session (Figure 30 Annotation 1).
- In addition, if a user has not activated their account through the link sent by email, they will be given a link to resend the activation email (Figure 8 Annotation 5).
- They are redirected to the resend activation email page (Figure 28). This page uses a prepared SQL statement to get the activation link from the entered email address (which is sanitized), and resends the activation link to the email, only if the email already exists and is not activated (Figure 28 Annotation 4 & 5).

OBFUSCATION

- Finally, this page includes excellent implementations of counter measures to botnet & brute force attacks.
- On form submissions, the login attempts, linked to the IP of the submission, not the data entered, is checked (Figure 7 Annotation 1).
- Using the function defined in (Figure 24), if the user has 5 non successful login attempts within 10 minutes, the user is temporarily blocked from logging in for 10 minutes (Figure 7 Annotation 1). By doing this, brute force attacks are prevented as the machine can only fail 5 times before being timed out for 10 minutes.
- In addition, by doing this by IP and not by username, the user attempting to brute force an attack, won't be able to attempt another user account after timing out, as their IP address is what is blocked, not the account their attempting to access (Figure 25 Annotation 3).
- Botnet attacks are also prevented using Google ReCAPTCHA (Figure 7 Annotation 3).
- If the ReCAPTCHA is not completed correctly, or at all, the user is given an error.
- This should prevent botnet attacks & simple brute force attacks as it requires human input after a significant amount of login attempts failed, along with relying on google cookies allowing for bot identification.

TASK 3 - IMPLEMENT PASSWORD STRENGTH AND PASSWORD RECOVERY

FORGOT PASSWORD FORM

```php
<?php
include 'main.php';
$msg = '';
// If email field not empty when submitted
if (isset($_POST['email'])) {
    //Protection against XSS
    //To check its an email, this is done via HTML attribute type=email & empty is checked using required attribute
    $email = mysqli_real_escape_string($con, $_POST['email']);
    $email = htmlspecialchars($email);
    // SQL statements that are prepared, allow for protection against SQL Injection
    $stmt = $con->prepare('SELECT * FROM accounts WHERE email = ?');
    $stmt->bind_param('s', $email);
    $stmt->execute();
    $stmt->store_result();
    // Check if the email exists in database
    if ($stmt->num_rows > 0) {
        $stmt->close();
        // If Email does exist
        // Creates a new & updates the reset code into the db.
        $uniqid = uniqid();
        $stmt = $con->prepare('UPDATE accounts SET reset_code = ? WHERE email = ?');
        $stmt->bind_param('ss', $uniqid, $email);
        $stmt->execute();
        $stmt->close();
        //Subject & Body of the email
        $subject = 'Password Reset';
        $reset_link = 'http://users.sussex.ac.uk/~bcc28/G6077/LovejoyAntiques/resetpassword.php?email=' . $email . '&code=' . $uniqid;
        $message = '<p>Please click the following link to reset your password: <a href="' . $reset_link . '">' . $reset_link . '</a></p>';
        //Adding subject & body to email & Using Email Template for nice formatting
        $email_template = str_replace('%link%', $message, file_get_contents('forgotpassword.html'));
        if (sendEmail($email, $email_template, $subject)) {
            $msg = 'Message has been sent. Please check your email (and Junk Mail)';
        } else {
            $msg = 'Message cannot be sent. Mail Error Occured';
            header("Refresh:5; url=login.php");
        }
    } else {
        $msg = 'No account found with that email!';
    }
}
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Lovejoy Antiques: Forgot Password </title>
        <link href="style1.css" rel="stylesheet" type="text/css">
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    </head>
    <body>
        <div class="container">
            <div class="row">
                <div class = "col-md-12">
                    <h2><strong>LoveJoy Antique Evaluations: </strong><br>Forgot Password</h2><br>
                    <form action="forgotpassword.php" method="post">
                    <div class="form-group">
                        <input type="email" name="email" placeholder="Your Email" id="email" class="form-control" required>
                    </div>
                    <div class="form-group">
                        <input type="submit" value="Submit" class="btn btn-primary">
                    </div>
                    <div class="msg"><?=$msg?></div><br>
                    <p>Remember your password? <a href="login.php">Login here</a>.</p>
                    </form>
                </div>
            </div>
        </div>
    </body>
</html>
```


Figure 10 – forgotpassword.php, if a user has forgotten their password, can be recovered through this page.

ANNOTATION DESCRIPTIONS

1. Sanitizing the email inputted to protect against SQL Injection/XSS Attacks.
2. Finds the account where the email entered is the same, otherwise error message presented that user does not exist with that email.
3. If a user does exist with that email, create a unique new code, called a reset code.
4. Store the code in the database linked to the account that wishes to reset their password.
5. Create a link, which includes the user's email & the reset code generated.
6. Send an email to the user with this link.
7. HTML code for the form.

RESET PASSWORD FORM

```
<!DOCTYPE html>
<html lang='en'>
    <!-- HTML for Reset Password Form -->
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <title>Lovejoy Antiques Reset Password</title>
        <link rel="stylesheet" href="style1.css">
    </head>
    <body>
            <div class="container">
                <div class="row">
                    <div class = "col-md-12">
                        <!-- Form elements -->
                        <h2><strong>LoveJoy Antique Evaluations: </strong><br>Reset Password</h2><br>
                        <p><strong>Please complete your security question & then enter your new password</strong></p>
                        <!-- On Submit, redirects to PHP with email & code in link, used to idenify user & stop anyone accessing this page -->
                        <form action="resetpassword.php?email=<?=$_GET['email']?>&code=<?=$_GET['code']?>" method="post" autocomplete=off>
                            <div class="form-group">
                                <br>
                                <!-- Displays Security Question which was extracted in SQL Select Statement
                                This is requried to be correct to allow users to update their password for added Protection-->
                                <p style= "text-align:left;"><b>Security Question:</b> <?=$securityQuestion?></p>
                            </div>
                            <div class="form-group">
                                <!-- Textfield for Security Answer. Only allows for the letters, spaces & hypens-->
                                <input type="text" name="securityAnswer" placeholder="Answer to Security Question" pattern="[A-Za-z0-9 -]+" class="form-control" required>
                                <label for="password"></label>
                            </div>
                            <br>
                            <div class="form-group">
                                <!-- Password for user. Min length 10 characters. Type 'Password' hides inputs as *s. Further validation is done in PHP -->
                                <input type="password" name="new_password" placeholder="Password" class="form-control" minlength="10" required>
                                <label for="confirm_password"></label>
                                <small>Must be atleast 10 characters long & contain atleast one uppercase, lowercase, number & special character</small>
                            </div>
                            <div class="form-group">
                                <!-- Confirmation Password for user. Same validation as password.-->
                                <input type="password" name="confirm_password" placeholder="Confirm Password" minlength="10" class="form-control" required>
                            </div>
                                <!-- Error messages displayed here in red -->
                                <p class="msg"><?=$msg?></p><br>
                            <div class="form-group">
                                <!-- Submit Button. Onclick, executes PHP in document -->
                                <input type="submit" class="btn btn-primary" value="Submit">
                            </div>
                        </form>
                    </div>
                </div>
            </div>>
    </body>
</html>
```

Figure 11 – resetpassword.php, HTML code for the form

ANNOTATION DESCRIPTIONS

1. On submit if something was entered incorrectly, page is reloaded, with email & code preserved in URL.
2. Prints the security question chosen by the user on registration and provides a textbox for the user to enter their security question's answer.
3. Field to enter answer.
4. New Password, and confirmation of new password fields.
5. Any errors occurred in PHP, for example if passwords do not match, new password is not meeting strength requirements, or security answer is incorrect

```php
<?php
include 'main.php';
// Now we check if the data from the login form was submitted, isset() will check if the data exists.
//This is the message that will be displayed to the user if there are any errors
$msg = '';
//Checking email & reset code from URL
if (isset($_GET['email'], $_GET['code']) && !empty($_GET['code'])) {
    //Get Security Question & Answer, along with old password from Database.
    //Prepared Statement Used for good Practice
    $stmt = $con->prepare('SELECT securityQuestion, securityAnswer, password, username FROM accounts WHERE email = ? AND reset_code = ?');
    $stmt->bind_param('ss', $_GET['email'], $_GET['code']);
    $stmt->execute();
    $stmt->store_result();
    // If user exists.
    if ($stmt->num_rows > 0) {
        $stmt->bind_result($securityQuestion, $securityAnswer, $passwordOld, $username);
        $stmt->fetch();
        $stmt->close();
        //If not empty
        if (isset($_POST['new_password'], $_POST['confirm_password'])) {
            //Setting User's inputs to variables
            //Protection against XSS & SQL Injection
            $securityAnswerEnt = mysqli_real_escape_string($con, $_POST['securityAnswer']);
            $securityAnswerEnt = htmlspecialchars($securityAnswerEnt);
            $password = mysqli_real_escape_string($con, $_POST['new_password']);
            $password = htmlspecialchars($password);
            $confirm_password = mysqli_real_escape_string($con, $_POST['confirm_password']);
            $confirm_password = htmlspecialchars($confirm_password);

            //To avoid casing issues
            $securityAnswer=strtolower($securityAnswer);
            $securityAnswerEnt=strtolower($securityAnswerEnt);
```

Figure 12 – resetpassword.php, checking link is valid, and sanitizing inputs

ANNOTATION DESCRIPTIONS

1. Checks if link is valid, by seeing if email & reset code are present.
2. Prepared SQL statement gets security question, answer, password, and username of account with matching email & reset code.
3. Bind result of query to variables.
4. If both new password & confirm password are entered.
5. Sanitize inputs of security answer, new password & confirm password.
6. Convert security answer from database and answer entered to lowercase (needed for comparison to ensure no casing issues.

```php
        $uppercase = preg_match('@[A-Z]@', $password);
        $lowercase = preg_match('@[a-z]@', $password);
        $number    = preg_match('@[0-9]@', $password);
        $specialChars = preg_match('@[^\w]@', $password);
        // Used to Check to see if Password is based on username. This checks for palindrome & if user replaces letters like s,e,o & l with number/other charcters alike.
        //Converted to lowercase in case of Casing issues
        $lc_password = strtolower($password);
        $lc_username = strtolower($username);
        $denum_pass = strtr($lc_password,'5301!','seoll');

        //Password Stength
        if(!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 10 || strlen($password) > 100) {
            $msg = 'Password should be at least 10 characters (no more than 100) in length and
                should include at least one upper case letter, one number, and one special character.';
        } else if ($confirm_password != $password) {
            //Whether new passwords entered match
            $msg = 'Passwords do not match!';
        } else if (strcmp($securityAnswer , $securityAnswerEnt) !== 0) {
            //Whether security Answer matches one in database. This is required to update the password for added protection.
            $msg = 'Security Answer do not match';
        } else if (password_verify($password, $passwordOld)) {
            //Ensure new password is not the same as old password
            $msg = 'New Password cannot be the same as Old Password';
            //Further Password Entropy
        } else if (($lc_password == $lc_username) || ($lc_password == strrev($lc_username)) || ($denum_pass == $lc_username) || ($denum_pass == strrev($lc_username))) {
            $msg = "Password cannot be based on the username!";
        } else {
            //Prepared SQL Statement updates the password stored in database for user & resets the reset code to empty string
            $stmt = $con->prepare('UPDATE accounts SET password = ?, reset_code = "" WHERE email = ?');
            //Hashes passworded for protection in database
            $passwordenc = password_hash($password, PASSWORD_DEFAULT);
            $stmt->bind_param('ss', $passwordenc, $_GET['email']);
            $stmt->execute();
            $stmt->close();
            //Sucess message & redirect
            $msg = 'Password has been reset! Redirecting Now!';
            header("Refresh:3; url=login.php");
        }
    }
} else {
    //No rows from select statement. Possible reset code is not valid/expired.
    exit("Link Expired!<a href='login.php'> Click here to return to the login screen</a>");
}
} else {
    //Code/Email in URL not found. Incase user just types /resetpassword.php into URL.
    exit("Link Expired!<a href='login.php'> Click here to return to the login screen</a>");
    header("Refresh:3; url=login.php");
}
?>
```

Figure 13 – resetpassword.php, check password strength & ensure security question is correct

ANNOTATION DESCRIPTIONS

1. Set Password policy standards, searching for at-least 1 instance of each uppercase, lowercase, number, and special character.
2. Convert username & password to lower.
3. Check to ensure password entered follows policy standards, and between 10-100 characters.
4. Then check passwords match.
5. Then check if security question is correct.
6. Check new password is not the same as current/old password.
7. Finally check password is not based on username, i.e. a palindrome, the same, or using common number replacements.
8. Prepare SQL statement to update password in accounts table, with new password, that has met all previous standards. Hash the new password, and redirect user to login form.
9. If link has already been used, if user has requested a reset again after this link was sent, or user is attempting to access form with incorrect URL (missing email and/or reset code)

EACH PASSWORD POLICY ELEMENT IMPLEMENTED

- Must be at least 10 characters long (Figure 15 Annotation 2).
- Cannot be longer than 100 characters long (Figure 15 Annotation 2).
- Must contain at least one uppercase letter (Figure 15 Annotation 1 & 2).
- Must contain at least one lowercase letter (Figure 15 Annotation 1 & 2).
- Must contain at least one number/digit (Figure 15 Annotation 1 & 2).
- Must contain at least one special character (Figure 15 Annotation 1 & 2).
- Password must be identical to confirmation password (Figure 15 Annotation 3).
- Password can be the same as previous password (Figure 15 Annotation 4).

- Password cannot be the same as the username (Figure 15 Annotation 5).
- Password cannot be a palindrome of the username (or vice versa) (Figure 15 Annotation 5).
- Password cannot be the username with common character replacement (even when reversed) (Figure 15 Annotation 5).


```php
//Password Strength Check
$uppercase = preg_match('@[A-Z]@', $password);
$lowercase = preg_match('@[a-z]@', $password);
$number    = preg_match('@[0-9]@', $password);
$specialChars = preg_match('@[^\w]@', $password);
// Used to Check to see if Password is based on username. This checks for palindrome & if user replaces letters like s,e,o & l with number/other charcters alike.
//Converted to lowercase in case of Casing issues
$lc_password = strtolower($password);
$lc_username = strtolower($username);
$denum_pass = strtr($lc_password,'5301!','seoll');

//Password Stength
if(!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 10 || strlen($password) > 100) {
    $msg = 'Password should be at least 10 characters (no more than 100) in length and
        should include at least one upper case letter, one number, and one special character.';
} else if ($confirm_password != $password) {
    //Whether new passwords entered match
    $msg = 'Passwords do not match!';
} else if (strcmp($securityAnswer , $securityAnswerEnt) !== 0) {
    //Whether security Answer matches one in database. This is required to update the password for added protection.
    $msg = 'Security Answer do not match';
} else if (password_verify($password, $passwordOld)) {
    //Ensure new password is not the same as old password
    $msg = 'New Password cannot be the same as Old Password';
    //Further Password Entropy
} else if (($lc_password == $lc_username) || ($lc_password == strrev($lc_username)) || ($denum_pass == $lc_username) || ($denum_pass == strrev($lc_username))) {
    $msg = "Password cannot be based on the username!";
```

Figure 14 – Password Policy Check

ADDITIONAL SECURITY IMPLEMENTED

- All SQL statements are prepared to prevent SQL Injection (Figure 12,Annotation 2 & Figure 13,Annotation 8).
- All inputted data is sanitized to prevent XSS & to prevent SQL Injection further (Figure 12, Annotation 5).
- Authentication is used to ensure only users with valid links, i.e. email & reset code present in the URL are given access to this form. This link will only be accessible from emails sent by the web application (Figure 12, Annotation 1).
- Security Question must be answered correctly before password can be updated (Figure 13, Annotation 5).

TASK 4 & 5 - IMPLEMENT A "EVALUATION REQUEST" WEB PAGE & DEVELOP A FEATURE THAT WILL ALLOW CUSTOMERS TO SUBMIT PHOTOGRAPHS

REQUEST EVALUATION FORM CODE

```php
<?php
//Main.php required to perform check_loggedin function.
//Check the user is logged in, before they can access this page. Otherwise redirected to login page.
//Stops users with the URL from accessing a page without logging in
include 'main.php';
check_loggedin($con);
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <!-- WebPage where users can request an evaluation of their antique -->
        <title>Lovejoy Antiques Request Evaluation Form</title>
        <!-- Stylesheets used for consitent format -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <link rel="stylesheet" href="style1.css">
    </head>
    <body>
        <nav>
            <div>
                <ul>
                    <li><a href="requestEval.php" class="active">Request an Evaluation</a></li>
                    <!-- This link on nav bar is only shown to Admins. If user is a member, this will not appear on the navigation bar -->
                    <?php if ($_SESSION['role'] == 'Admin'): ?>
                        <li><a href="viewRequests.php"></i>View Evaluation Requests</a></li>
                    <?php endif; ?>
                    <!-- Link to log users out once done -->
                    <li style="float:right"><a href="logout.php"></i>Logout</a></li>
                </ul>
            </div>
        </nav>
        <div class="container">
            <div class="row">
                <div class = "col-md-12">
                    <h2><strong>LoveJoy Antique Evaluations: </strong><br>Request Evaluation Form</h2><br>
                    <!-- On submission, check is done on details submitted -->
                    <form action="requestCheck.php" method="post" autocomplete="off" enctype="multipart/form-data">
                        <div class="form-group">
                        <u><p style="text-align:left">Please select the image of the Antique to upload</p></u>
                        <!-- Input type of file, allows for a file to be uploaded. This is required along with a description & a choice of contact -->
                            <input type="file" name="antique_image" required>
                        </div>
                        <br>
                        <div class="form-group">
                            <u><p style="text-align:left">Please enter a description of the Antique</p></u>
                            <!-- Textarea allows user to leave a large description of the antique if wanted by the user. This is also required. -->
                            <!-- In Addition, CSS is used to give the textarea a red, dashed appearance if not completed, and a green border if completed. -->
                            <textarea autofocus name="desc" spellcheck minlength="10" rows="5" cols="50"
                                placeholder="Enter a brief description of the item..." required></textarea>
                        </div>
                        <br>
                        <!-- Instead of dropdown, as only two options, radio buttons used for contact choice. User can choose only between Email & Telephone contact -->
                        <div class="form-group">
                            <u><p style="text-align:left">How would you like to be contacted?</p></u>
                            <input type="radio" id="telephone" value=0 name="contactChoice"  required>
                            <label style="margin-right: 30px" for="email">Email</label>
                            <input type="radio" id="email" value=1 name="contactChoice" required>
                            <label for="telephone">Telephone</label><br>
                        </div>
                        <!-- Submission button -->
                        <div class="form-group">
                            <input type="submit" name="submit" class="btn btn-primary" value="Request">
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

Figure 15 – requestEval.php – Form to request an evaluation of an antique

ANNOTATION DESCRIPTIONS

1. Check the user is logged in. Using function defined in main.php (Figure 24, Annotation 3), if user is not logged into an account, redirect to login.php
2. Stylesheets used for formatting page
3. Link to View Evaluation Requests page (Figure 16). Only visible to users logged in with a role of 'Admin'. Role is set by default to 'Member' (Figure 4)
4. Input type that allows users to upload an image from their device.
5. Text area allows user to enter large description about the antique.
6. Radio buttons used to select preferred contact choice, by email or by telephone.

CODE WHEN FORM SUBMITTED

```php
<?php
//User must be logged in to access this page
include 'main.php';
check_loggedin($con);
//If form submitted & image has been selected.
if (isset($_POST['submit']) && isset($_FILES['antique_image'])) {
    //Gets the name of the image, the size and a temp name of the image along with any occuring errors.
    $img_name = $_FILES['antique_image']['name'];
    $img_size = $_FILES['antique_image']['size'];
    $tmp_name = $_FILES['antique_image']['tmp_name'];
    $error = $_FILES['antique_image']['error'];
    //Choice of contact detail stored from selection of Radio Buttons
    $choice = $_POST['contactChoice'];
    //Description of Antique is sanitized against SQL injection & XSS
    $desc = mysqli_real_escape_string($con, $_POST['desc']);
    $desc = htmlspecialchars($desc);
    //User ID is taken from the ID stored in the session, which is stored on login.
    $userid = $_SESSION['id'];
    //If there is no errors
    if ($error === 0) {
        //And the image does not exceed 1.25MB
        if ($img_size > 1250000) {
            echo 'Sorry, your file is too large. Please use a file below 1.25MB';
            header("Refresh:3; url=requestEval.php");
        }else {
            //Image extension extracted from image name
            $img_ex = pathinfo($img_name, PATHINFO_EXTENSION);
            //Converts extension to lowercase for easier comparison checks
            $img_ex_lc = strtolower($img_ex);
            //Array of allowed extensions, i.e. image extensions
            $allowed_exs = array("jpg", "jpeg", "png");
            //If extension of file uploaded is an image
            if (in_array($img_ex_lc, $allowed_exs)) {
                //New name of file is created.
                //Renaming file protects the database from attacks, when users rename photos to SQL Injection statements
                $new_img_name = uniqid("IMG-", true).'.'.$img_ex_lc;
                //Path which is stored in database, is created by adding uploads/ before image name as all images stored in a folder called uploads.
                $img_upload_path = 'uploads/'.$new_img_name;
                //Moves image file to uploads folder.
                move_uploaded_file($tmp_name, $img_upload_path);
                // Insert into Database. Prepared Statement used for further injection protection.
                $stmt = $con->prepare("INSERT INTO evaluations (description, contactDetail,  image_url, userid) VALUES (?, ?, ?, ?)");
                //Binding Parameters with user's inputs, plus user id which is a foreign key in evaluations table
                //Foreign key required to link account details like email & telephone to each evaluation
                //Email/Telephone not stored in the evaluation table, incase these updated, & it being redundant duplicate data.
                $stmt->bind_param('sisi', $desc, $choice ,$new_img_name, $userid);
                $stmt->execute();
                $stmt->close();
                //Successful Evaluation Request
                echo 'Successful Request Submitted! Redirecting back to Evaluation page!';
                header("Refresh:3; url=requestEval.php");
            }else {
                //Wrong file type
                echo 'You cannot upload files of this type. Please upload a .jpg, .jpeg or a .png';
                header("Refresh:2; url=requestEval.php");
            }
        }
    }else {
        //$_FILES returned an error
        echo 'Unknown Error Occured! '. $error;
        header("Refresh:3; url=requestEval.php");
    }
}else {
    //If submit has not occured or no image has been uploaded (this would be caught by HTML)
    header("Refresh:2; url=requestEval.php");
}
?>
```

Figure 16 – requestCheck.php – Checks the data entered the evaluation form, and only uploads complete evaluations

ANNOTATION DESCRIPTIONS

1. Gets the image name, size, temp name, and any errors occurred from the file upload. Contact choice set to what was decided on form.
2. Sanitize the description of the antique, entered by the user.
3. If image is larger than 1.25mb, image too large error.
4. Array created of allowed file extensions (images only).
5. If file uploaded is an image, create a new image name, set upload path to uploads/image_name, then upload the image to the folder ‘uploads’.
6. Prepared SQL Statement used to insert evaluation details, such as description, contact choice, image URL and the ID of the user (used as a foreign key) to get the name & email/telephone for the view evaluations page.
7. If the file uploaded returned an error, or users submission is missing details, post error message and redirect back to request evaluations page.

EVALUATIONS DATABASE TABLE

```
CREATE TABLE IF NOT EXISTS `evaluations` (
    `evalid` int(11) NOT NULL AUTO_INCREMENT,
    `description` varchar(255) NOT NULL,
    `contactDetail` varchar(100) NOT NULL,
    `image_url` varchar(255) NOT NULL,
    `userid` int(11) NOT NULL,
    PRIMARY KEY (`evalid`),
    FOREIGN KEY (`userid`) REFERENCES `accounts`(`id`) ON UPDATE CASCADE ON DELETE SET NULL
)ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Figure 17 – Evaluations database table

WHY THIS IS SECURE

VULNERABILTIES

- SQL Injection & Cross-Site Scripting prevented using sanitation methods (Figure 16 ,Annotation 2).
- Renaming the image file also helps prevent these vulnerabilities as the image URL is directly injected into an INSERT SQL statement (Figure 16, Annotation 5).
- Prepared SQL statements continued to be used, to ensure the SQL command is executed safely, further preventing SQL Injection vulnerabilities (Figure 16, Annotation 6).
- By checking the file uploaded is an image, and not any kind of file (Figure 16, Annotation 4), vulnerabilities such as malicious files being uploaded is avoided.
- In addition, ensuring the image is smaller than a certain size, prevents the user from maliciously uploading an image that fills the server's storage, which could cause errors, and affect performance.

AUTHENTICATION

- User identity is enforced from the moment the user accesses this page. If the user is not logged in, defined in (Figure 24, Annotation 3), they are immediately redirected to the login page. This is done by checking whether the session variable 'loggedin' has been updated to TRUE, only done after successful logins.
- This means users even with the URL will still not be able to access this site, without logging in (Figure 15, Annotation 1).
- Finally, the navigation bar has a link to the View Evaluation Requests page (Figure 16). However, this is only visible to users logged in with a role of 'Admin', which can be found in the session after a successful login.

TASK 6 – REQUEST LISTING PAGE

CODE OF THE PAGE

```
<!DOCTYPE html>
<!-- Webpage that displays all evaluation requests to an admin -->
<html>
    <head>
        <meta charset="UTF-8">
        <title>Lovejoy Antiques View Evaluation Requests</title>
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <link rel="stylesheet" href="style1.css">
    </head>
    <body class="loggedin">
        <nav class="navtop">
            <div>
                <ul>
                    <!-- Navigation Bar -->
                    <li><a href="requestEval.php">Request an Evaluation</a></li>
                    <li><a href="viewRequests.php" class="active"></i>View Evaluation Requests</a></li>
                    <li style="float:right"><a href="logout.php"></i>Logout</a></li>
                </ul>
            </div>
        </nav>
        <div class="container">
            <div class="row">
                <div class = "col-md-12">
                    <h2><strong>LoveJoy Antique Evaluations: </strong><br>View Evaluation Requests</h2><br>
```

Figure 19 - – viewEval.php, HTML code generating navigation bar

```
<?php
include 'main.php';
//Checks if user is logged in (as this page requires users to be logged in to view
//If not logged in, re-directed to login page
check_loggedin($con);
// Ensures only admins can view this page. Otherwise, instantly re-drects Member users to the request page
if ($_SESSION['role'] != 'Admin') {
    header("Location: requestEval.php");
    exit;
}
```

1

2

Figure 18 – viewEval.php, PHP check to ensure user is logged in to access, and is an admin

CODE FOR LISTING GENERATION

```
<?php
//Prepare statements are not needed here, since user is not able to input any data!
//Using an INNER JOIN allows for the name, email and telephone number of the requestor to be stored in a temp table with the evaluation details
//This means, the requestors name & contact details do not need to be stored in the evaluation table. Ensuring no duplicate redundant data is stored.
$sql = 'SELECT accounts.fullname, evaluations.description, evaluations.contactDetail, accounts.email,
    accounts.telephone, evaluations.image_url FROM evaluations INNER JOIN accounts ON evaluations.userid=accounts.id';
$result = mysqli_query($con, $sql);
$results = mysqli_query($con, $sql);
//If no evaluations requests have been submitted by users, this is displayed
if ($results->num_rows === 0) {
    echo "<p>No evaluation requests submitted</p>";
    exit();
}
//Otherwise, table generated
echo "<table>";
echo "<tr>";
//Displays the name of the requestor
echo "<th>Name of Requestor</th>";
//Description of the antique, which was entered by the requestor
echo "<th>Description of Antique</th>";
//The email or telephone number of the requestor
echo "<th>Choice of Contact</th>";
//An image of the antique
echo "<th>Image of Antique</th>";
echo "</tr>";
//Whiel loop used to print a row for each evaluation in the table
while ($rows = mysqli_fetch_array($result)) {
    echo "<tr>";
    echo "<td>".$rows['fullname']."</td>";
    echo "<td>".$rows['description']."</td>";
    //If contact choice chosen was 'email', display the email of the requestor in the table
    if ($rows['contactDetail'] == 0) {
        echo "<td>".$rows['email']."</td>";
    }
    else {
    //Otherwise, if the user wished ot be contacted by telephone, their telephone number is displayed.
        echo "<td>".$rows['telephone']."</td>";
    }
    //Images are printed to be the same size, for a clean & consistent look.
    //Through CSS, admins can hover over the image for an enlargered look at the image.
    echo "<td><img width='200' height='200' src='uploads/".$rows['image_url']."' ></img></td>";
    echo "</tr>";
}
echo "</table>";
```

3

4

5

6

7

8

Figure 20 – viewEval.php – Displays the evaluations in a structured table, if there are any, otherwise message stating no evaluations yet.

ANNOTATION DESCRIPTIONS

1. Check if user is logged in. If not, redirect immediately to login form.
2. If user is not an admin, but logged in, redirect immediately to request evaluation page.
3. Prepared SQL statement uses INNER JOIN to get user's full name and contact details along with request details using foreign key, userID found in evaluations.
4. If there has been evaluation requested, continue otherwise, let Admin know, there is no current requests.
5. Displays table headings.
6. For each request in evaluations, display User's name & contact detail preference, along with the antiques description and image.
7. If the user chose for email as preferred contact method, display their email. Otherwise, display their telephone number.
8. Exports image into a square. User may hover over any image to increase the scale factor and see a larger version of the image.

WHY THIS IS SECURE

- This page only requires authentication as it is a page that does not require inputs from a user & should only be accessed by admins.
- If a malicious user has the URL to this page, they will be redirected to the login page, if not logged in (Figure 19, Annotation 1).
- If the user is logged in, but not an administrator, by searching the URL, this user would be immediately redirected to the request evaluation page (Figure 19, Annotation 2).
- Only the preferred contact method chosen by the user when submitting the request (Figure 15, Annotation 6), is displayed to Admin (Figure 20, Annotation 7).
- **To view this page,** use the username 'Admin' & the password 'castleDonkeyleft!2'.

OTHER FILES USED

CREDENTIALS

```php
<?php
    define('EMAIL', 'EMAIL@example.com');
    define('PASS', 'Password');
?>
```

Figure 21 – credentials.php

DESCRIPTION

This file stores the credentials of the mail account being used to send the account activation and password reset links, along with the two-factor authentication codes from. For security reasons I have replaced the actual email & password for this screenshot. Storing in a separate file allows for easy updating of these attributes, along with encrypting this file for security.

CONFIGURATION OF DATABASE

```php
<?php
// Hostname
define('db_host', 'krier.uscs.susx.ac.uk');
// Username & password for database connection
define('db_username', 'bcc28');
define('db_password', 'Mysql_433883');
// Name of the database we are connecting to
define('db_name', 'G6077_bcc28');
```

Figure 22 – config.php

DESCRIPTION

This file defines the database host, username, password, database name & charset. This is required within ‘main.php’ to enable a connection to the database. By storing this in a separate file allows for the values to be updated easily if needed.

LOGOUT OF WEBSITE

```php
<?php
//session initialized & destroyed
session_start();
session_destroy();
// Redirect to the login page:
header('Location: login.php');
?>
```

Figure 23 – logout.php

DESCRIPTION

When a logged in user has finished requesting their evaluations, or viewing evaluation requests, they need to be able to logout of the ongoing session. Code destroys the session in progress and redirects the user to the login page.

MAIN

```php
<?php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;
// This file contains the database connection, initializing of sessions
include_once 'config.php';
// Start the session
session_start();
// Establish a connection to the database
$con = mysqli_connect(db_host, db_username, db_password, db_name);
if (mysqli_connect_errno()) {
    exit('Connection to Database Failed: ' . mysqli_connect_error());
}

/* The below function will check if the user is logged-in. If they are not, redircted back to
login page. Stops users typing in URL of a page without logging in. For example used on Request
Evaluation page & View Evaluations Page*/
function check_loggedin($con) {
    if (!isset($_SESSION['loggedin'])) {
        // If the user is not logged in redirect to the login page.
        header('Location: login.php');
        exit;
    }
}
```


Figure 24 – Main.php, initializing database connection & declaration of function which checks if user has logged in.

ANNOTATION DESCRIPTIONS

1. Including ‘config. php’ required for definitions of db_host, db_name, db_password & db_username.
2. Connection to MySQL database, using the constant variables defined in ‘config .php’. If there is an error, error message displayed.
3. Function definition, which is used over multiple pages, so to ensure code is efficient. This function uses sessions to check if the variable ‘loggedin’ has been set to FALSE, i.e. the user is not logged in. If the user has not signed in, and attempts to access a page, only available to logged in users, they are redirected to the login page. After signing into the website, this variable ‘loggedin’ would be set as TRUE.

```php
//Brute force protection using login attempts. Users are given 5 attempts to login
function loginAttempts($con, $update = TRUE) {
    //Gets IP and date, which if failed attenpt, is saved in DataBase
    $ip = $_SERVER['REMOTE_ADDR'];
    $now = date('Y-m-d H:i:s');
    if ($update) {
        //SQL Injection protected by using Prepared Statements
        //INSERTS in db login_attempts the ip-address and date/time of the failed login
        /*If IP of the failed attempt is a duplicate in the table,
        i.e. user has failed now more than once on same device, that row is updated instead with the attempts-1*/
        $stmt = $con->prepare('INSERT INTO login_attempts (ip_address, `date`) VALUES (?,?)
            ON DUPLICATE KEY UPDATE attempts_left = attempts_left - 1, `date` = VALUES(`date`)');
        $stmt->bind_param('ss', $ip, $now);
        $stmt->execute();
        $stmt->close();
    }
    //Finds all failed attempts for a specific IP
    $stmt = $con->prepare('SELECT * FROM login_attempts WHERE ip_address = ?');
    $stmt->bind_param('s', $ip);
    $stmt->execute();
    $stmt->store_result();
    // Check there is a row:
    if ($stmt->num_rows > 0) {
        //Bind the data retrieeved to variables.
        $stmt->bind_result($id, $ips, $attempts, $date);
        $stmt->fetch();
        $stmt->close();
        //If the user has run out of attempts
        // The user can try to again in 10 mins
        $expire = date('Y-m-d H:i:s', strtotime('+10 minutes', strtotime($date)));
        if ($now > $expire) {
            //If user has expiray time.
            //Delete the record from the table.
            $stmt = $con->prepare('DELETE FROM login_attempts WHERE ip_address = ?');
            $stmt->bind_param('s', $ip);
            $stmt->execute();
            $stmt->close();
            $attempts = 5;
            return $attempts;
        }
        else {
            return $attempts;
        }
    }
    //First login, so 5 attempts left
    return 5;
}
```

1
2
3
4
5
6

Figure 25 – main.php, declaration of function which checks how many login attempts user has left.

ANNOTATION DESCRIPTIONS

1. Stores the current date & time, along with the IP address of the user
2. If an incorrect attempt occurs, insert IP address and date/time of attempt into login_attempts table. If there is already a row with the IP address, subtract 1 from number of attempts, and update the date.
3. Selects row from table where IP addresses are the same.
4. If there is a row. Save the id, IP address, number of attempts and date.
5. Create an expiry time 10 minutes from date extracted from database
6. If the user has waited 10 minutes after account was locked, remove row from table and reset attempts. Otherwise, return the number of attempts left.

```php
/*This function takes the recipient email, the html template & subject
and sends an email with an activation/reset password link or 2FA code*/
function sendEmail($email, $email_template, $subject) {
    require 'PHPMailer.php';
    require 'Exception.php';
    require 'SMTP.php';
    //Contains SMTP Email(Username) & Password
    require 'credential.php';

    //PHPMailer setup - initialize new object
    $mail = new PHPMailer(true);
    // Set mailer to use SMTP
    $mail->isSMTP();
    // Specify SMTP server
    $mail->Host = 'smtp.gmail.com';
    // Enable SMTP authentication
    $mail->SMTPAuth = true;
     // SMTP username & password
    $mail->Username = EMAIL;
    $mail->Password = PASS;
    // Enable TLS encryption
    $mail->SMTPSecure = 'tls';
    // TCP port to connect to
    $mail->Port = 587;

    //From email account created for this coursework
    $mail->setFrom(EMAIL, 'Lovejoy Antique Evaluations');
    // Add a recipient
    $mail->addAddress($email);
    $mail->addReplyTo(EMAIL);
    // Set email format to HTML as using HTML template
    $mail->isHTML(true);
    //Adding subject & body to email
    $mail->Subject = $subject;
    $mail->Body    = $email_template;
    //Sending the email
    if(!$mail->send()) {
        //Email sending failed
        return 0;
    } else {
        //Email sending success
        return 1;
    }
}
?>
```

Figure 26 – main.php, declaration of function that is used to send emails to users.

ANNOTATION DESCRIPTIONS

1. For PHPMailer to work, we require to have these three following files in the webspace to define classes for PHPMailer, Exception & SMTP objects. We also require the credentials of the email account as we are setting the SMTP username & password, along with the sent from address.
2. This section of code creates a new PHPMailer object. The SMTP Server host, server, username, password, port & encryption method need to be set here.
3. Using the variables given to the function, the recipient, subject & body of the email are set
4. If there is an error sending the email, return 0, otherwise return 1.

ACCOUNT ACTIVATION

```php
<?php
include 'main.php';
$msg = '';
// First we check if the email and code exists, these variables will appear as parameters in the URL sent to the user!
//GET method is safe to use, as getting code & email from the URL, not being entered by users into a field. Still sanitizing incase user knows this.
$email = mysqli_real_escape_string($con, $_GET['email']);
$code = mysqli_real_escape_string($con, $_GET['code']);

if (isset($_GET['email'], $_GET['code']) && !empty($_GET['code'])) {
    //Prepare Statement 1. Protects Against SQL Injection & 2. Is used to all stored data from the user where the code & email in URL match.
    $stmt = $con->prepare('SELECT * FROM accounts WHERE email = ? AND activation_code = ?');
    // Binding the ? marks to the email & code retrieved from URL.
    $stmt->bind_param('ss', $email, $code);
    $stmt->execute();
    // Store the result so we can check if the account exists in the database.
    $stmt->store_result();
    if ($stmt->num_rows > 0) {
        $stmt->close();
        // Account exists with the requested email and code.
        $stmt = $con->prepare('UPDATE accounts SET activation_code = ? WHERE email = ? AND activation_code = ?');
        // Set the new activation code to 'activated', this is how we can check if the user has activated their account already.
        $activated = 'activated';
        //Updates the activation code to the text 'activated'.
        $stmt->bind_param('sss', $activated, $email, $code);
        $stmt->execute();
        $stmt->close();
        //Displayed on page activate.php
        $msg = 'Your account is now activated, you can now <br><a href="login.php">Login here</a> or be redirected in a moment!';
        //Redirects user to login page if activation successful
        header("Refresh:4; url=login.php");
    } else {
        //User is most likely already activated, i.e. this link is an old link sent to the user. Gives option to login or resend activation email!
        $msg = 'The account is already activated or this link has expired! To login click <a href="login.php">here</a><br>
            Or to activate your account, <a href="resendactivation.php">click here</a> to resend the activation email!';
    }
} else {
    //Code & Email not found in URL. This may mean user has just typed url into browser. Redircted to login page.
    $msg = 'Activation code & email not found! <a href="login.php">Login here</a> or wait to be redirected';
    header("Refresh:3; url=login.php");
}
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Lovejoy Antiques - Account Activation Page</title>
        <link href="style1.css" rel="stylesheet" type="text/css">
    </head>
    <body>
        <div>
        <!-- displays error/success message! -->
            <p><?=$msg?></p>
        </div>
    </body>
</html>
```


Figure 27 – activate.php, activation page directed to after clicking link in email

ANNOTATION DESCRIPTIONS

1. SQL Injection & XSS Prevention to escape special characters and/or HTML elements within the URL.
2. Finds the account which has the same email & activation code.
3. If the user can be found with matching email & activation codes.
4. Update the user's activation code, to 'activated', changing the activation status to complete. Prepared SQL statement used to prevent SQL Injection
5. If not code, code likely expired, or user is already activated so link has expired.
6. Code & URL missing from email, user may be trying to access this page manually, rather than through link.

```php
<?php
//Include Main.php for database connection
//MailSetup also needed, as activation code sent via Email
include 'main.php';
// Output error/success message
$msg = '';
//isset() will check if the email has been entered by the user.
if (isset($_POST['email'])) {
    //SANITIZE INPUT against XSS & SQL Injection
    $email = mysqli_real_escape_string($con, $_POST['email']);
    $email = htmlspecialchars($email);
    //Prepare our SQL, preparing the SQL statement will prevent SQL injection.
    //Activation Code is selected from database, where email = email entered by user on form & activation code is not empty or already activated
    $stmt = $con->prepare('SELECT activation_code FROM accounts WHERE email = ? AND activation_code != "activated"');
    // In this case we can use the account ID to get the activation code
    $stmt->bind_param('s', $_POST['email']);
    $stmt->execute();
    $stmt->store_result();
    //Check the account, with this email, exists in database
    if ($stmt->num_rows > 0) {
        //if it does indeed exist and needs to be activated
        $stmt->bind_result($activation_code);
        $stmt->fetch();
        $stmt->close();
        //Account exist, the $msg variable will be used to show the output message (on the HTML form)
        $subject = 'Account Activation Required';
        $activate_link = 'http://users.sussex.ac.uk/~bcc28/G6077/LovejoyAntiques/activate.php?email=' . $email . '&code=' . $activation_code;
        $message = '<p>Please click the following link to activate your account!: <a href="' . $activate_link . '">' . $activate_link . '</a></p>';
        $email_template = str_replace('%link%', $message, file_get_contents('activation.html'));
        //Sending the email
        if (sendEmail($email, $email_template, $subject)) {
            $msg = 'Message has been sent. Please check your email (and Junk Mail)';
        } else {
            $msg = 'Message cannot be sent. Mail Error Occured';
            header("Refresh:5; url=login.php");
        }
    } else {
        //Account already Activated or Email entered invalid
        $msg = "No activation is required or email doesn't exist in database";
    }
}
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>LoveJoy Antiques: Resend Activation Email</title>
        <!-- Stylesheets used for CSS Formatting -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <link rel="stylesheet" href="style1.css">
    </head>
    <body>
        <div class="container">
            <div class="row">
                <div class = "col-md-12">
                    <h2><strong>LoveJoy Antique Evaluations: </strong><br>Resend Activation Email</h2><br>
                    <form action="resendactivation.php" method="post">
                        <div class="form-group">
                            <label for="email"></label>
                        </div>
                        <div class="form-group">
                            <!-- Form text field to allow user to enter email. Required Field, with pattern to ensure a valid email format is entered -->
                            <input type="email" name="email" placeholder="Your Email" class="form-control" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$" required>
                        </div>
                        <div class="form-group">
                            <!-- Submit Button -->
                            <input type="submit" class="btn btn-primary" value="Submit">
                        </div>
                        <!-- Outputs Error/Success Message  -->
                        <p><?=$msg?></p>
                        <p>Account already activated? <a href="login.php">Login here</a>.</p>
                    </form>
                </div>
            </div>
        </div>
    </body>
</html>
```

1
2
3
4
5

Figure 28 – resendactivation.php, user enters their email, and new activation email is sent.

ANNOTATION DESRCIPTIONS

1. Sanitation of user's input, their email. This is to protect against vulnerabilities like SQL Injection & Cross-Site Scripting.
2. Find activation code from accounts, where email is the same as entered email, and account is not activated.
3. If account with email does exist, bind activation code to a variable.
4. Creates link adding email & activation code to the end. Needed so only authenticated users with this link can activate their accounts.
5. Sends activation link to user as email has been confirmed (Figure 26).

TWO FACTOR AUTHENTICATION

```html
<!DOCTYPE html>
<html>
    <head>
        <!-- HTML code for Two Factor Page -->
        <meta charset="UTF-8">
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <title>Lovejoy Antiques: Two-Factor Authentication</title>
        <link rel="stylesheet" href="style1.css">
    </head>
    <body>
        <div class="container">
            <div class="row">
                <div class = "col-md-12">
                    <h2><strong>LoveJoy Antique Evaluations: </strong><br>Two-Factor Authentication</h2><br>
                    <p style="padding:10px;margin:0;">Please enter the 6-digit code, that was sent to your email address, below.</p>
                    <!-- Onclick, executes POST method in PHP above -->
                    <form action="" method="post">
                        <div class="form-group">
                            <!-- Input Field for 2FA code. Pattern used to ensure only 6 digits entered -->
                            <input type="text" name="code" placeholder="2FA Code sent via Email" class="form-control" pattern="[0-9A-Za-z]{6}" required>
                        </div>
                        <div class="form-group">
                            <!-- Error/Success Message Displayed Here -->
                            <div class="msg"><?=$msg?></div><BR>
                            <!-- Submit Button -->
                            <input type="submit" value="Submit" class="btn btn-primary">
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </body>
</html>
```

Figure 30 – twofactor.php: html section which produces textfield where user enters code sent via email.

```php
<?php
//As mail is sent out
include 'main.php';
// Output message
$msg = '';
// Verify the ID and email and 2FA code provided in URL are present
//And 2FA session code, is equal to code in URL
if (isset($_GET['id'], $_GET['email'], $_GET['code'], $_SESSION['2FA']) && $_SESSION['2FA'] == $_GET['code']) {
    // Prepare our SQL, preparing the SQL statement will prevent SQL injection.
    $stmt = $con->prepare('SELECT email, 2FA_code, role FROM accounts WHERE id = ? AND email = ?');
    $stmt->bind_param('ii', $_GET['id'], $_GET['email']);
    $stmt->execute();
    // Store the result so we can check if the account exists in the database.
    $stmt->store_result();
    // If the account exists with the email & ID provided...
    if ($stmt->num_rows > 0) {
        //Bind the email, 2FA code and role selected from accounts to the following variables
        $stmt->bind_result($email, $acc_code, $role);
        $stmt->fetch();
        $stmt->close();
        // 2FA Code submitted in the form
        if (isset($_POST['code'])) {
            //Code submitted = the code in the database
            //Santize Input
            $codeEnt = mysqli_real_escape_string($con, $_POST['code']);
            $codeEnt = htmlspecialchars($codeEnt);
            if ($codeEnt == $acc_code) {
                // Code accepted, update the IP address to this login address
                $ip = $_SERVER['REMOTE_ADDR'];
                $stmt = $con->prepare('UPDATE accounts SET ip = ? WHERE id = ?');
                $stmt->bind_param('si', $ip, $_GET['id']);
                $stmt->execute();
                $stmt->close();
                //Regenerate session, updating logged in as TRUE
                session_regenerate_id();
                $_SESSION['loggedin'] = TRUE;
                $_SESSION['id'] = $_GET['id'];
                $_SESSION['role'] = $role;
                $msg = '2FA Code has been accepted! You can now access the website <a href="requestEval.php">here</a>!';
            } else {
                //Code is not accepted, therefore code incorrect/expired
                $msg = 'Incorrect code provided!';
            }
        } else {
            //Send the access code email using the twofactor.html template
            //Creates 6 digit random unique code for the 2FA code
            $code = strtoupper(substr(md5(uniqid(mt_rand(), true)), 0, 6));
            //Updates the 2FA code in the database, to the newly created code
            $stmt = $con->prepare('UPDATE accounts SET 2FA_code = ? WHERE id = ?');
            $stmt->bind_param('si', $code, $_GET['id']);
            $stmt->execute();
            $stmt->close();

            //Setting Subject
            $subject = 'Your Two-Factor Access Code';
            //Replacing the template string with the actual code
            $email_template = str_replace('%code%', $code, file_get_contents('twofactor.html'));
            //Adding subject & body to email
            if (sendEmail($email, $email_template, $subject)) {
                $msg = 'Message has been sent. Please check your email (and Junk Mail)';
            } else {
                $msg = 'Message cannot be sent. Mail Error Occured';
                header("Refresh:5; url=login.php");
            }
        }
    } else {
        //No user found in select statement
        exit('Incorrect email and/or code provided!');
    }
} else {
    //URL does not contain an email &/or code
    exit('No email and/or code provided!');
}
?>
```

Figure 29 – twofactor.php: php check to ensure code matches one sent to the email

ANNOTATION DESCRIPTIONS

1. Prepared SQL statement which retrieves user's email, 2FA code and role from accounts table where ID & email stored in URL match
2. If there is a row with matching details, and the user has entered a 2FA code into the textfield. SQL Injection & XSS prevention used to sanitize code entered.
3. Store the IP of the current session, and update user's row in accounts with new login IP address.
4. User has completed two-factor check, so is able to click link and access request evaluations page.
5. If no code has been entered on the page (i.e. user has just loaded the page), create a new 2FA code & store in database
6. Send user email with newly generated two factor code.

FINAL SELF EVALUATION

CRITERIA	GRADING
PASSWORD POLICY	VERY GOOD (Encryption method simple)
VULNERABILITIES	VERY GOOD (CSRF was simple however)
AUTHENTICATION	EXCELLENT
OBFUSICATION	EXCELLENT
OTHER SECURITY FEATURES	POOR/NONE

Site also hosted here: https://184514.000webhostapp.com/ which does include some other security features (HTTP secure connection through DigiCert) but is linked to another database. Performance on this site was slow in general, when compared to Sussex server.