Name: Omar Ahmad Ayesh

Secure web application project documentation

Table of contents:

Environment

In this project I use the following environment:

1)Ampps : is software stack that provides tools and configurations to implement a web application.

2)Frontend language : Html , CSS , JavaScript .

3)backend language : PHP / Database: MySQL .

4)server : Apache server .

Security controls implementation and code :

1) Input Validation and Sanitization : the purpose of validation and sanitization is to Ensure all user inputs are validated and sanitized and to

prevent malicious input from users and to avoid XSS and SQL injection attacks.

- XSS (Cross-Site Scripting): is a web security vulnerability that allows an attacker to compromise the interactions that users have with a malicious scripts .
- SQL injection : is a web security vulnerability that allows an attacker to change the queries that an application makes to its database and get sensitive information.
- How to prevent XSS :

First -Sanitize Code :

- filter_input(INPUT_POST, 'field',FILTER_SANITIZE_SPECIAL_CHARS)

  this code Sanitizes fields by converting special characters like

  (<, >, ‘ , “ and &) into their HTML entity equivalents, preventing XSS.

- filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL): this code Removes unwanted characters from the email input, ensuring it only contains valid email characters.

Second-validation code:

- $email = filter_var($email, FILTER_VALIDATE_EMAIL) : Checks if the sanitized email input is in a valid email format. If the validation fails, $email will be false.

```php
// Sanitize and validate inputs
$firstName = filter_input(INPUT_POST, 'fName', FILTER_SANITIZE_SPECIAL_CHARS);
$lastName = filter_input(INPUT_POST, 'lName', FILTER_SANITIZE_SPECIAL_CHARS);
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
$email = filter_var($email, FILTER_VALIDATE_EMAIL); // Validate email format
$password = $_POST['password'];
```

- How to prevent SQL injection : use prepared statements with bound parameters instead of directly including user input in SQL queries. This ensures that user input is treated as data, not executable code.

Code: bind_param("s", $email); takes the email as ? and then bind it to the sql query not direct .

```
$stmt = $conn->prepare("SELECT * FROM users WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
```

---

2) CSRF Protection : (Cross-Site Request Forgery) : generally CSRF tokens are implemented in any form submission in the website to protect against cross site request forgery this attack happen when a malicious website tricks a user's browser into making an unwanted request to a different site that the user is already authenticated on by forms submitted like (sign in and sign up).

A CSRF token is a unique, random value generated by the server and included in every form that needs protection. When the user submits the form, the token is sent along with the request. The server then checks if the token matches the one stored in the user's session. If it does, the request is valid; if not, it's rejected.

Code: generation function generate the random token and but it in the users session , then the CSRF token is included in the HTML form as a hidden input field to send it with the form . then before the submission process the token is validated to check if it is valid if not error will occur.

```
// CSRF Token Function
function generateCSRFToken() {
    if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32)); // Generate a random to
    }
    return $_SESSION['csrf_token'];
}

// Generate the CSRF token for the form
$csrfToken = generateCSRFToken();
```

```
<!-- CSRF Token Hidden Field -->
<input type="hidden" name="csrf_token" value="<?php echo htmlspecialchars($csrfToken, ENT_QUOTES,
<input type="submit" class="btn" value="Sign In" name="signIn">
```

```php
if (isset($_POST['signUp'])) {
    if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        // CSRF token validation failed, show an error message
        $_SESSION['error_message'] = "Invalid CSRF token.";
        header("Location: index22.php");
        exit();
    }
```

---

3) Secure Session Management: Using secure cookies to store JWT tokens with HttpOnly and Secure flags.

ensures that sensitive information like authentication tokens (e.g., JWT tokens) is safely stored and transmitted between the client and the server using cookies.

When a user logs in successfully A JWT(Jason web token) is created during the login and signed using secret key. This token contains information like the user's email, securely **encoded**. We should not put sensitive info.

The JWT is sent to the user's browser as a secure cookie. When the user visits a page like homepage.php or otp_verify : The server checks the cookie containing the JWT.

The JWT is **decoded** and verified to : Ensure it wasn't tampered and Ensure it hasn't expired. If the JWT is valid, the server retrieves the user's data (like email) from the token.

HTTPOnly flag: Prevents JavaScript access to cookies, protecting against XSS attacks.

Secure flag : Ensures the cookie is sent over HTTPS only.

Library needed: php-jwt-main and Firebase\JWT\JWT .

Then it produce (auth_token) cookies then in the homepage the server decodes the cookies and checks the JWT if it has the right data the user will be able to move to the homepage.

Code : generate a jwt token and set into cookies after the user successful sign in contain the gmail and expiration time.

```php
    // Generate JWT Token for authenticated user
    $payload = [
        "email" => $email,
        "iat" => time(), // Issued at time
        "exp" => time() + (60 * 60) // Token expiration time (1 hour)
    ];
    $jwt = JWT::encode($payload, $secretKey, 'HS256');

    // Set JWT in secure cookie
    setcookie("auth_token", $jwt, [
        "expires" => time() + (60 * 60), // 1 hour expiration
        "path" => "/",
        "secure" => true,
        "httponly" => true,
        "samesite" => "Strict"
    ]);
```

Code: in the homepage decoding and checking then retrieve the data from the database using the gmail in the JWT :

```php
// Check if the auth_token cookie exists
if (isset($_COOKIE['auth_token'])) {
    try {
        // Decode JWT from the cookie
        $jwt = $_COOKIE['auth_token'];
        $decoded = JWT::decode($jwt, new Key($secretKey, 'HS256'));
        // Get the email from the decoded JWT payload
        $email = $decoded->email;

        // Fetch user details from the database using the email
        $query = mysqli_query($conn, "SELECT firstName, lastName FROM users WHERE email='$email'");
        $user = mysqli_fetch_assoc($query);

        if ($user) {
            $firstName = $user['firstName'];
            $lastName = $user['lastName'];
        } else {
            // Redirect to login if the user is not found in the database
            header("Location: index22.php");
            exit();
        }
    } catch (Exception $e) {
        // Redirect to login if the JWT is invalid or expired
        header("Location: index22.php");
        exit();
```

4) Use HTTPS: to Ensure the application is served over HTTPS so encrypt data in transit. **(HyperText Transfer Protocol Secure)** is an extension of HTTP that uses **encryption** to secure data during transmission, it ensures that all data exchanged between the server and the user is encrypted.

How to implement : Generate a self signed trusted https certification

Require installation of openssl-64bit : this is used to generate ssl trusted certification by running commands in the admin command prompt.

First generate a private key using : openssl genrsa -out localhost.key 2048

to create localhost.key file

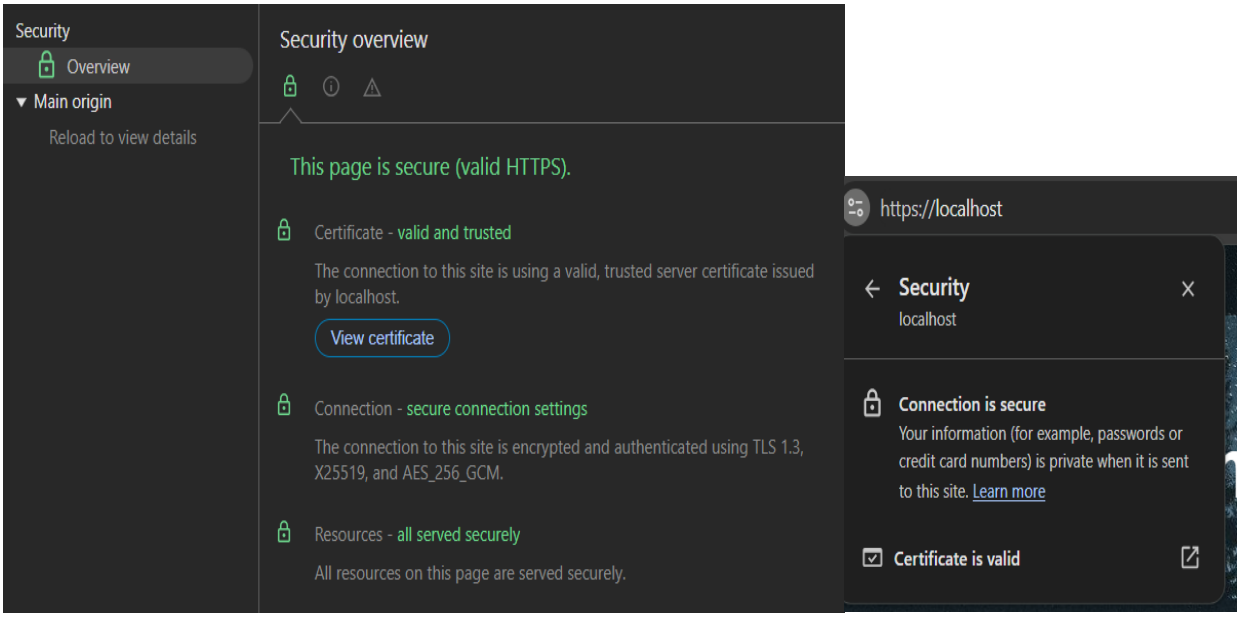
Second create a certificate: openssl req -new -key localhost.key -out localhost.csr . to create localhost.crt file

Then go to Apache server configurations and include the files .

Then go to the control panel-manage computer certificates and add it there.

Then access the web page and press trust certificate then navigate to https.

Then when you go to inspect then security you will notice that :



5) User Registration and Authentication using 2F

User registration if user have an account (sign in) the user will put the Gmail and password .

If the user don’t have an account (sign up) the user will put the first,last name and Gmail and password .

Two-Factor Authentication (2FA) adds an additional layer of security by requiring a second form of verification after entering a Gmail and password.an OTP(one time password)(6-digit) will be sent to the user Gmail. using SMTP(simple mail transfer protocol)

Code Requirements: install PHPMailer library .

Include the PHP-Mailer.php in the code and include the SMTP.php .

In the code a random number is generated using (rand(000000,999999) ; and sent by the Gmail and stored in the PHP session of the user so the server can validate the code later. The mail is sent by Gmail defined during the coding process .

```php
$otp = rand(100000, 999999);
$_SESSION['otp'] = $otp;
$_SESSION['email'] = $email;

$mail = new PHPMailer(true);
try {
    $mail->isSMTP();
    $mail->Host = 'smtp.gmail.com';
    $mail->SMTPAuth = true;
    $mail->Username = 'omarayesh2000@gmail.com';
    $mail->Password = 'jbxl ohsq yujq cvic';
    $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
    $mail->Port = 587;

    $mail->setFrom('omarayesh2000@gmail.com', '6-digit otp');
    $mail->addAddress($email);

    $mail->isHTML(true);
    $mail->Subject = 'Your OTP Code';
    $mail->Body = "<p>Your OTP code is: <b>$otp</b></p>";

    $mail->send();
    header("Location: otp_verify.php");
    exit();
} catch (Exception $e) {
    $_SESSION['error_message'] = "OTP email could not be sent. Mailer Error:
```

Then the validation process in the otp_verify code file it first check if the otp is submitted then check if the user session otp match the entered otp if yes the user will have access to homepage if not the user can't access the homepage . and check if the session expired or no otp is sent.

```php
// Check if OTP is submitted
if (isset($_POST['verifyOtp'])) {
    $enteredOtp = $_POST['otp'];

    // Validate the OTP
    if (isset($_SESSION['otp']) && $_SESSION['otp'] == $enteredOtp) {
        // OTP is correct, log the user in
        $_SESSION['email'] = $email; // Ensure session email is set
        unset($_SESSION['otp']); // Remove OTP from the session
        header("Location: homepage.php"); // Redirect to the homepage
        exit();
    } else {
        // OTP is incorrect
        $error_message = "Invalid OTP. Please try again.";
    }
}

// Check if the session has expired or no OTP is set
if (!isset($_SESSION['otp']) || !isset($_SESSION['email'])) {
    header("Location: index22.php");
    exit();
}
?>
```

---

6)hashing function : ensuring that in the Database the user password is stored hashed. By (bcrypt)

Passwords are hashed before storage using the password_hash function.

$hashedPassword = password_hash($password, PASSWORD_DEFAULT);

Hashes are verified during login with password_verify(inserted, hashed)

password_verify($password, $hashedPassword);

in the Database it should look like this:

	id	firstName	lastName	email	password
Edit Copy Delete	1	omar	ayesh	omar778899001133@gmail.com	$2y$10$zjDiXH.FIWxB2F9TMY8VluaqJx7HgdPmRDTylGMlSIB...
Edit Copy Delete	2	ali	moh	al@gmail.com	$2y$10$phjYbZ1O0P5xi.Fz8Dg3wO2.EsyRlf9u5/UA0eJyN2J...