Securing a Basic PHP Login System **Against Common Vulnerabilities**

In this workshop, you will be given a simple, working PHP login system that is intentionally insecure. **Your task** is to identify and fix common security issues by applying essential PHP security practices.

PART 1: Setup

Workshop10

1. You are given five files:
    a. signup.php
    b. login.php
    c. dashboard.php
    d. db.php
    e. setup.sql

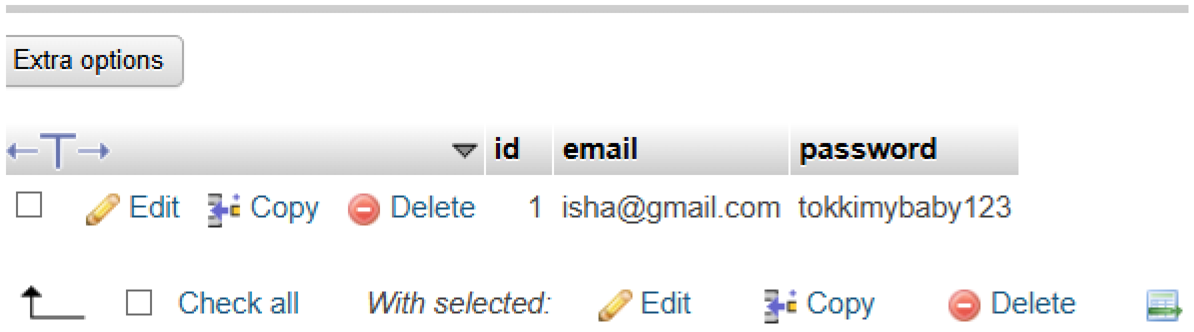
2. Create a new project **Workshop10** inside your htdocs folder and place all given files inside that folder.

   Execute setup.sql in the SQL tab of the phpmyadmin.

PART 2: Understanding the Insecure Login Flow

Identify the following problems:

1. User input is directly used in SQL queries
2. Passwords are stored without hashing and compared as plain text
3. No validation on email or password fields
4. Detailed login error messages are displayed
5. Session ID is not regenerated after login
6. No CSRF protection



	id	email	password
Edit Copy Delete	1	isha@gmail.com	tokkimybaby123

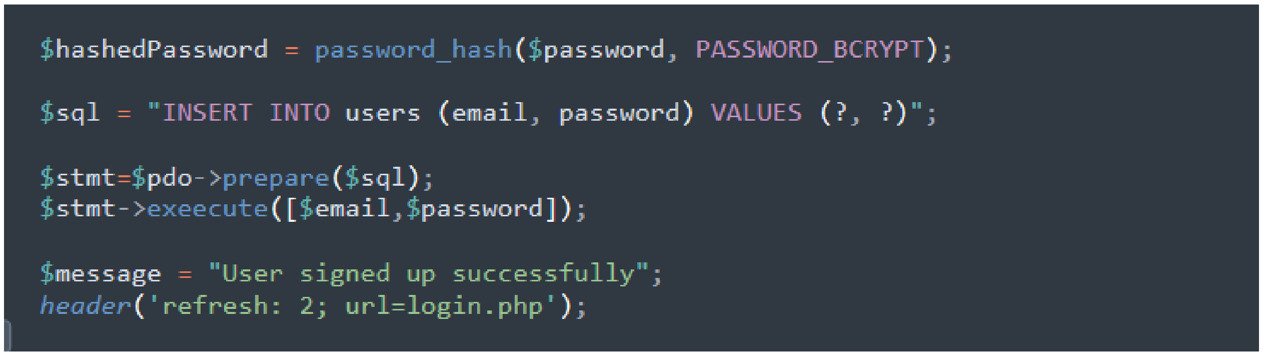
PART 3: Securing Password Storage

Update **signup.php** to:

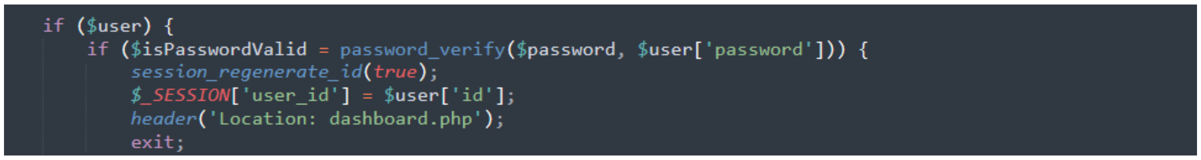
1. Hash password using **password_hash()**.

Update **login.php** to:

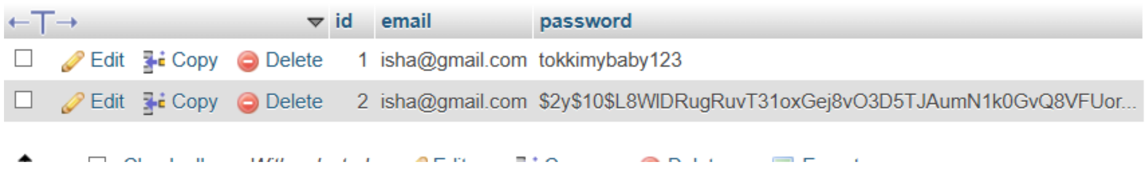
1. Remove plain-text password comparison.
2. Verify passwords using **password_verify()**.



```php
$hashedPassword = password_hash($password, PASSWORD_BCRYPT);

$sql = "INSERT INTO users (email, password) VALUES (?, ?)";

$stmt=$pdo->prepare($sql);
$stmt->exeecute([$email,$password]);

$message = "User signed up successfully";
header('refresh: 2; url=login.php');
```



```php
if ($user) {
    if ($isPasswordValid = password_verify($password, $user['password'])) {
        session_regenerate_id(true);
        $_SESSION['user_id'] = $user['id'];
        header('Location: dashboard.php');
        exit;
```



←T→	id	email	password
☐ Edit Copy Delete	1	isha@gmail.com	tokkimybaby123
☐ Edit Copy Delete	2	isha@gmail.com	$2y$10$L8WlDRugRuvT31oxGej8vO3D5TJAumN1k0GvQ8VFUor...

PART 4: Preventing SQL Injection

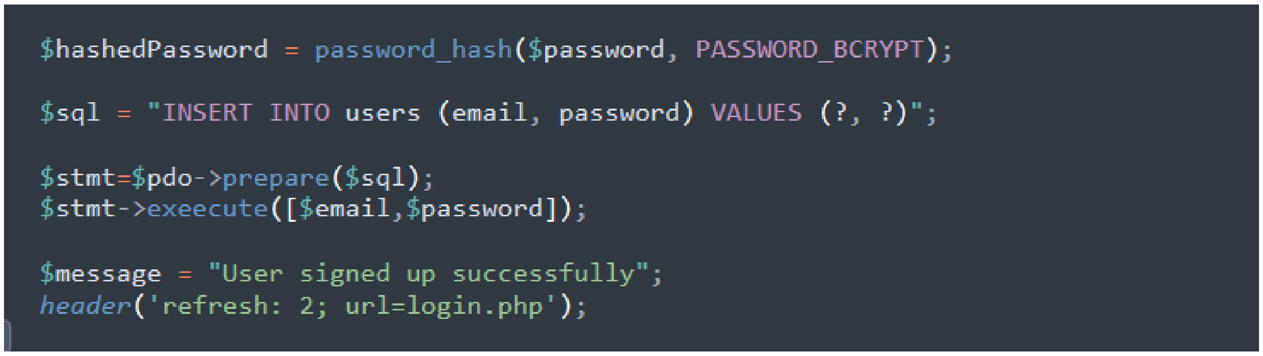
1. Locate the SQL query used to

a. Insert user using email and password in **signup.php**. b.

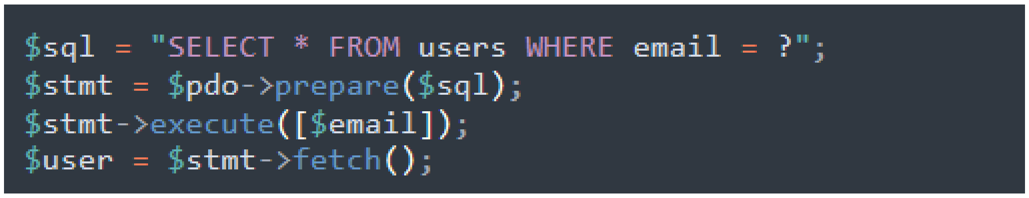
Fetch the user by email in **login.php** and **dashboard.php**. 2.

Replace the insecure query with a **prepared statement**. 3. **Bind**

user input safely when executing the query.



```
$hashedPassword = password_hash($password, PASSWORD_BCRYPT);

$sql = "INSERT INTO users (email, password) VALUES (?, ?)";

$stmt=$pdo->prepare($sql);
$stmt->exeecute([$email,$password]);

$message = "User signed up successfully";
header('refresh: 2; url=login.php');
```

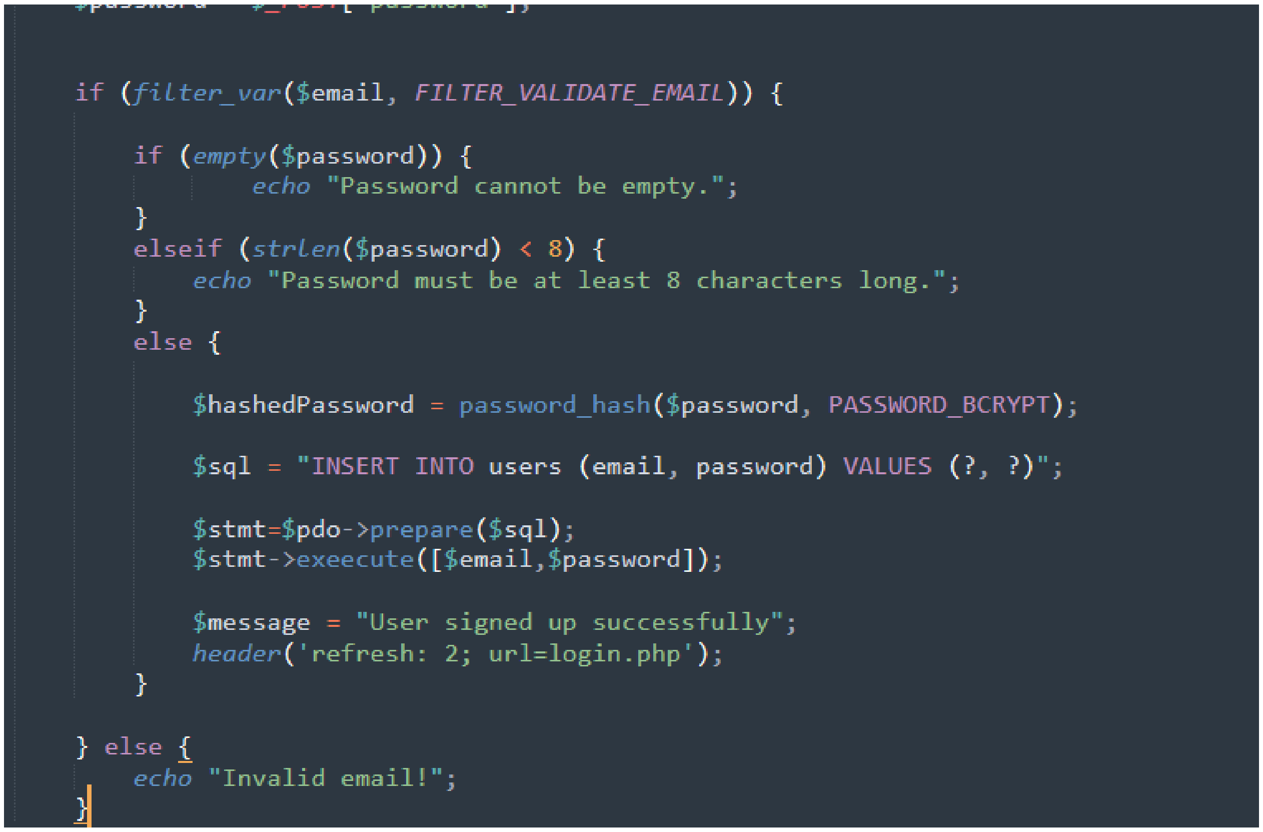


```
$sql = "SELECT * FROM users WHERE email = ?";
$stmt = $pdo->prepare($sql);
$stmt->execute([$email]);
$user = $stmt->fetch();
```

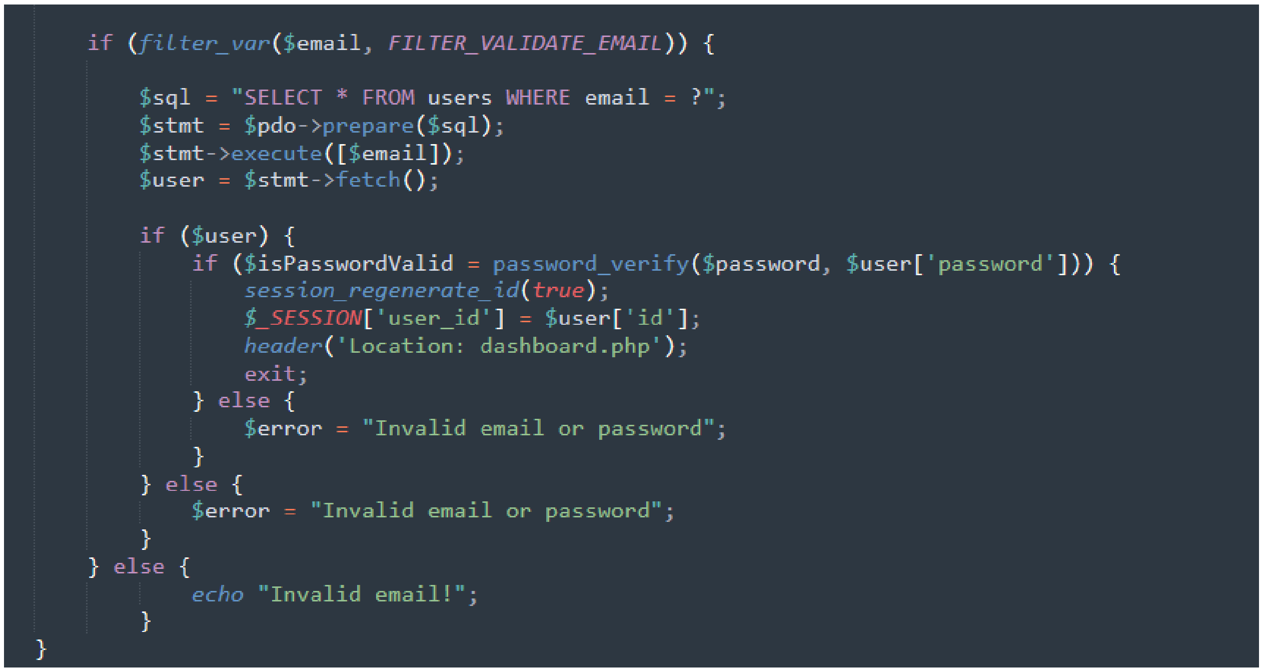
PART 5: Input Validation and Sanitization

1. Go to **signup.php**.
2. Validate the email field using **filter_input()** or **filter_var()**.
3. Ensure the password field:
    a. Is not empty
    b. Meets a minimum length requirement
4. Go to **login.php**. Ensure both fields are **not empty**.
5. In **dashboard.php**, sanitize all output before displaying it back to the user.

Use **htmlspecialchars()** while displaying user email to prevent XSS attacks. (This is the application of the **Validate Early, Escape Late** principle)



```
    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {

        if (empty($password)) {
                echo "Password cannot be empty.";
        }
        elseif (strlen($password) < 8) {
            echo "Password must be at least 8 characters long.";
        }
        else {

            $hashedPassword = password_hash($password, PASSWORD_BCRYPT);

            $sql = "INSERT INTO users (email, password) VALUES (?, ?)";

            $stmt=$pdo->prepare($sql);
            $stmt->exeecute([$email,$password]);

            $message = "User signed up successfully";
            header('refresh: 2; url=login.php');
        }

    } else {
        echo "Invalid email!";
    }
```

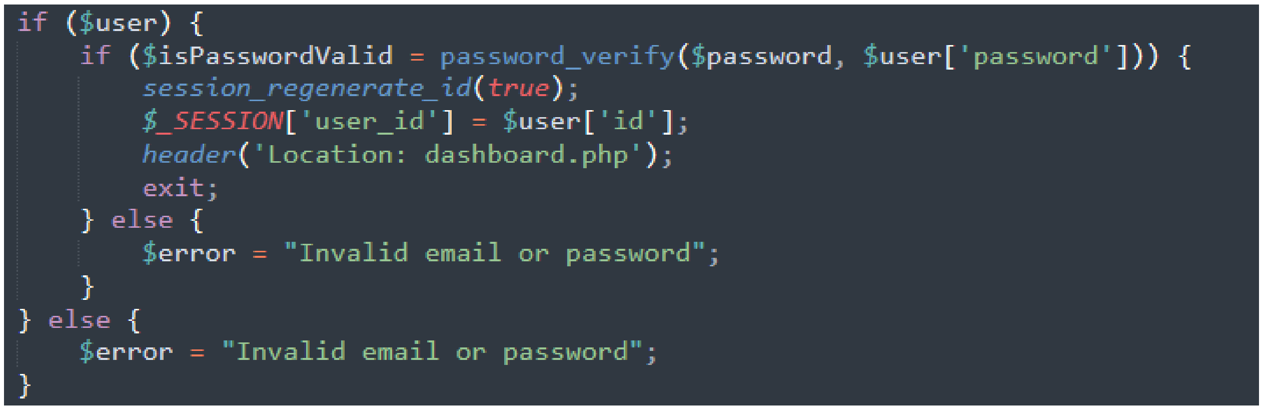


```
    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {

        $sql = "SELECT * FROM users WHERE email = ?";
        $stmt = $pdo->prepare($sql);
        $stmt->execute([$email]);
        $user = $stmt->fetch();

        if ($user) {
            if ($isPasswordValid = password_verify($password, $user['password'])) {
                session_regenerate_id(true);
                $_SESSION['user_id'] = $user['id'];
                header('Location: dashboard.php');
                exit;
            } else {
                $error = "Invalid email or password";
            }
        } else {
            $error = "Invalid email or password";
        }
    } else {
            echo "Invalid email!";
        }
}
```

PART 6: Secure Error Handling

1. Replace specific error messages such as: "Email does not exist", "Incorrect password"
2. **Use a single generic message**: "Invalid email or password"

(This helps prevent **account enumeration attacks** which happen when an attacker can discover whether a user account exists based on system responses.)

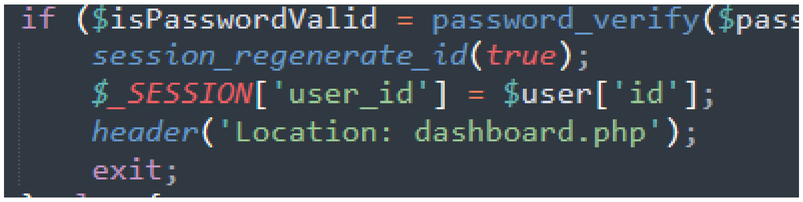


```php
if ($user) {
    if ($isPasswordValid = password_verify($password, $user['password'])) {
        session_regenerate_id(true);
        $_SESSION['user_id'] = $user['id'];
        header('Location: dashboard.php');
        exit;
    } else {
        $error = "Invalid email or password";
    }
} else {
    $error = "Invalid email or password";
}
```

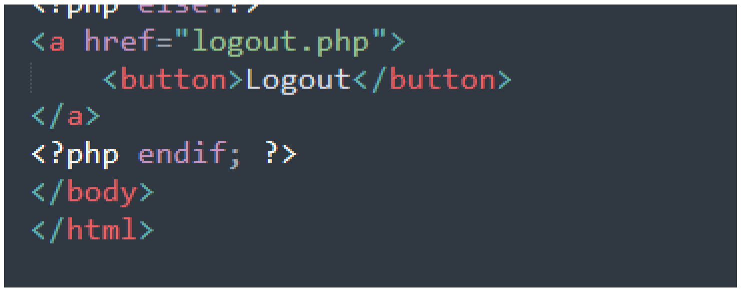
PART 7: Session Security

1. Start the session securely at the top of login.php. **(Already done)**
2. On successful login:
    a. Regenerate the session ID using **session_regenerate_id(true)**.
    b. Store only the **user ID** in the session. **(Already done)**
3. In **dashboard.php**, implement a logout mechanism that:
    a. **Destroys** the session
    b. **Unsets** all session variables
    c. Authenticated users should:
        i. See a logout button
        ii. **Be able to logout** when clicked on that button
    d. Unauthenticated users should:
        i. See a login button
        ii. Navigate to **login.php** when clicked on that button

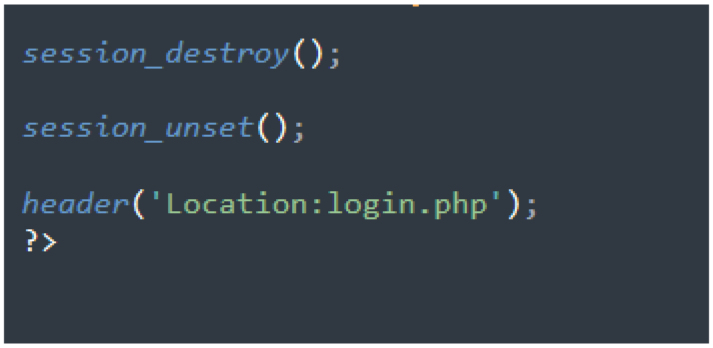
(This helps prevent **session fixation attacks** where an attacker forces a user to use a known session ID and then hijacks the session after login.)



```php
if ($isPasswordValid = password_verify($pass
    session_regenerate_id(true);
    $_SESSION['user_id'] = $user['id'];
    header('Location: dashboard.php');
    exit;
```



```html
<?php else:?>
<a href="logout.php">
    <button>Logout</button>
</a>
<?php endif; ?>
</body>
</html>
```



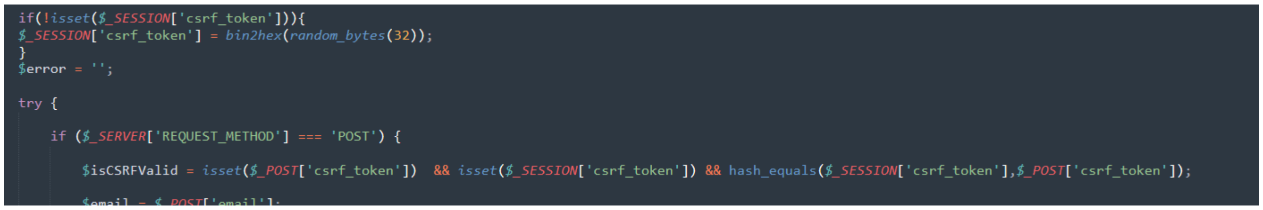
```php
session_destroy();

session_unset();

header('Location:login.php');
?>
```

PART 8: CSRF Protection

1. **Generate a CSRF token** when rendering the login form.
2. Store the token in the session.
3. Add the token as a hidden input field in the form.
4. On form submission:
    a. Verify the CSRF token
    b. **Reject** the request if the **token is missing or invalid**

(This ensures login requests **originate from your application** to prevent **CSRF attacks**.

CSRF attack means tricking a logged-in browser into sending a request it did not mean to send.)

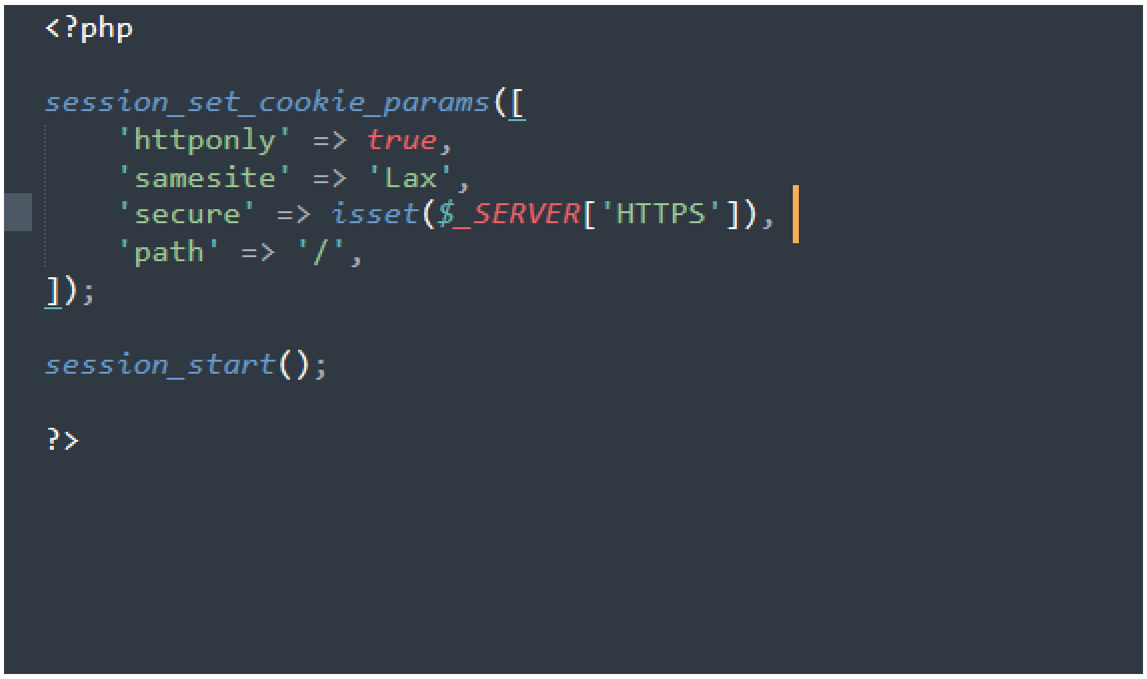


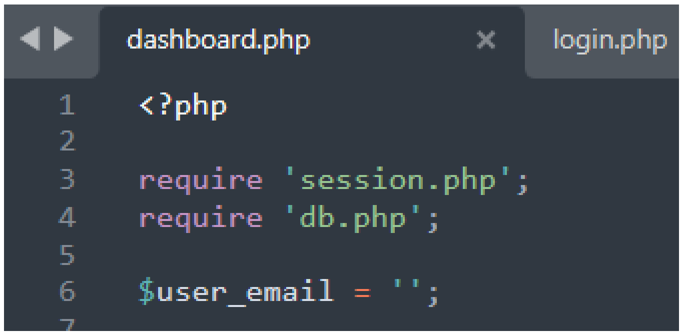
```php
if(!isset($_SESSION['csrf_token'])){
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
$error = '';

try {

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

        $isCSRFValid = isset($_POST['csrf_token'])  && isset($_SESSION['csrf_token']) && hash_equals($_SESSION['csrf_token'],$_POST['csrf_token']);

        $email = $_POST['email'];
```

PART 9: Secure Session Cookie Configuration

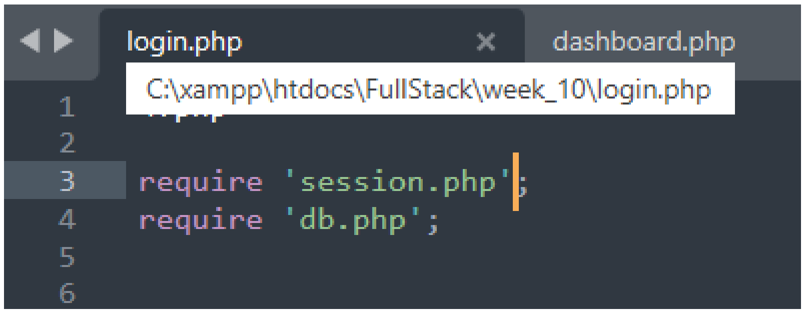
1. Before calling session_start(), **configure session cookie parameters**.
2. Set the following flags:
    a. HttpOnly: Prevent JavaScript access to session cookies
    b. SameSite (Lax): Reduce risk of CSRF attacks
3. Ensure these settings are applied consistently across the application.
    a. Create a file called **session.php**
    b. Migrate all the session related code in that file including the cookie parameters
    c. **require 'session.php';** in login and dashboard files

(This strengthens session protection at the browser level)


```php
<?php

session_set_cookie_params([
    'httponly' => true,
    'samesite' => 'Lax',
    'secure' => isset($_SERVER['HTTPS']),
    'path' => '/',
]);

session_start();

?>
```

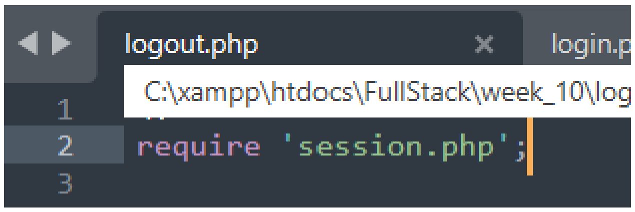


```php
<?php

require 'session.php';
require 'db.php';

$user_email = '';
```



C:\xampp\htdocs\FullStack\week_10\login.php

```php
<?php

require 'session.php';
require 'db.php';
```



```php
require 'session.php';
```

PART 10: Testing and Verification

1. Test login with:

   a. Valid credentials

   b. Incorrect password

   c. Invalid email format

2. Attempt the following attacks:

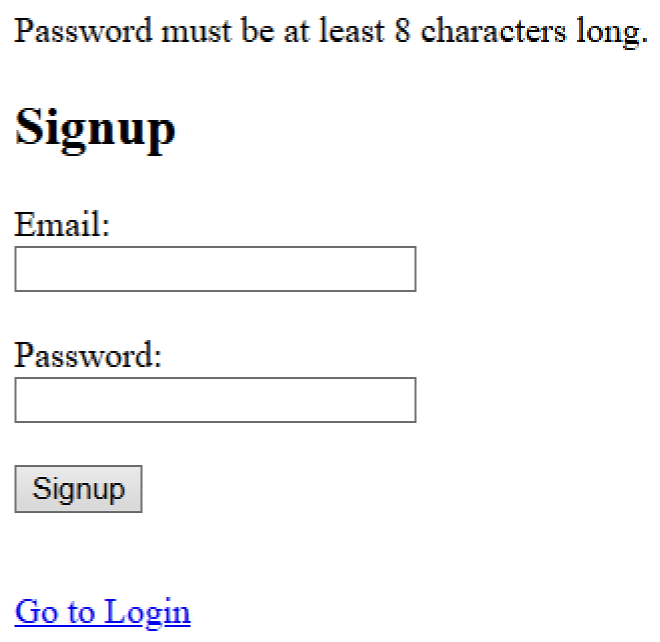
   a. SQL injection payloads in the email field b.

Submitting the form without a CSRF token 3.

Confirm that:

   a. The application fails securely

   b. No sensitive error information is exposed

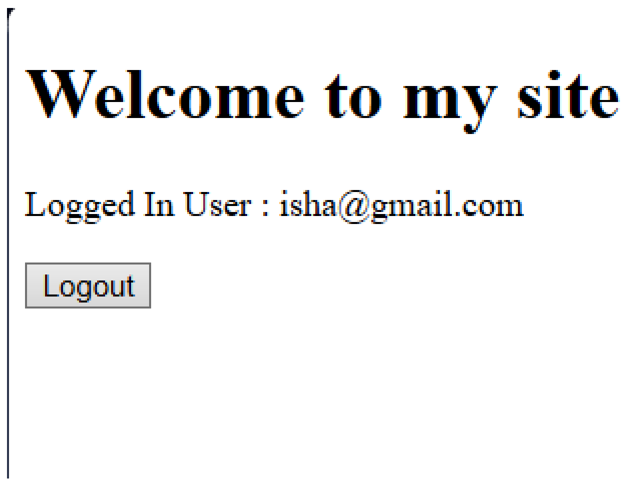


Password must be at least 8 characters long.

Signup

Email:

Password:



Welcome to my site

Logged In User : isha@gmail.com

Login

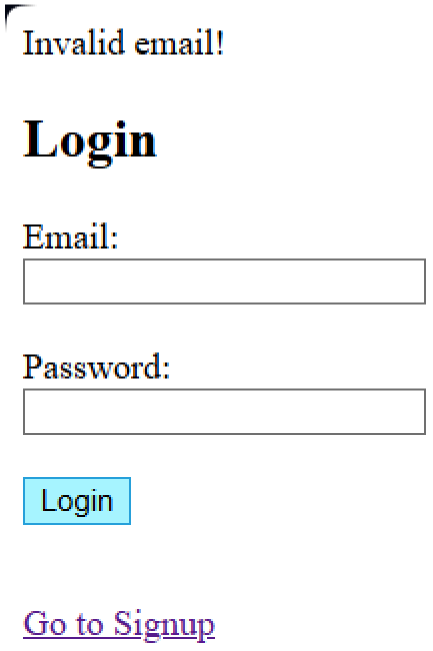
Invalid email or password

Email:

Password:

Login

Go to Signup



Invalid email!

Login

Email:

Password: