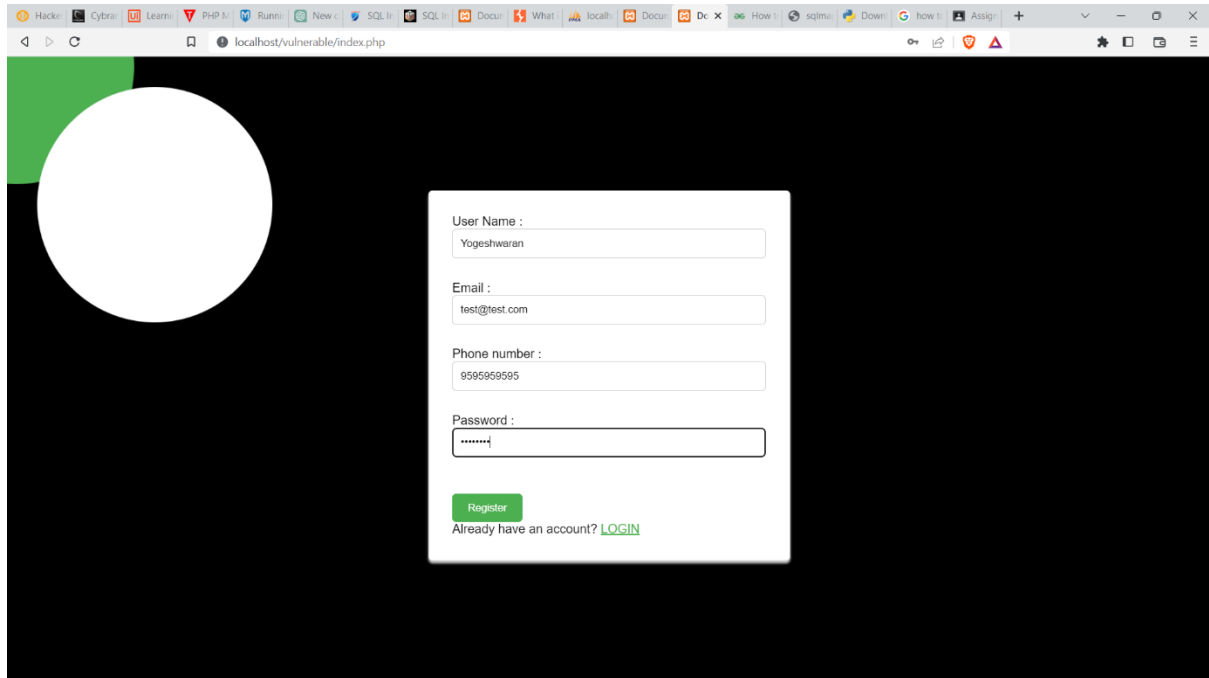
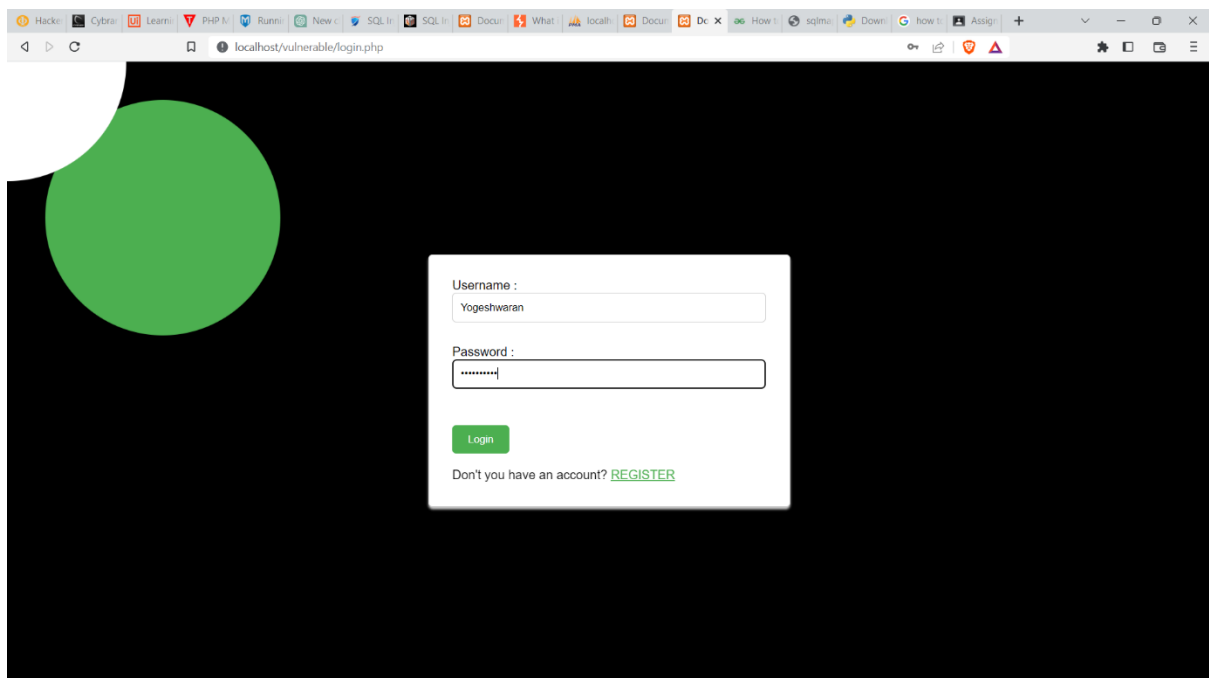


I have created a small webpage using PHP with three pages (a Login page, a page to display user informations-Dashboard page after successful login and a page to register the account.)



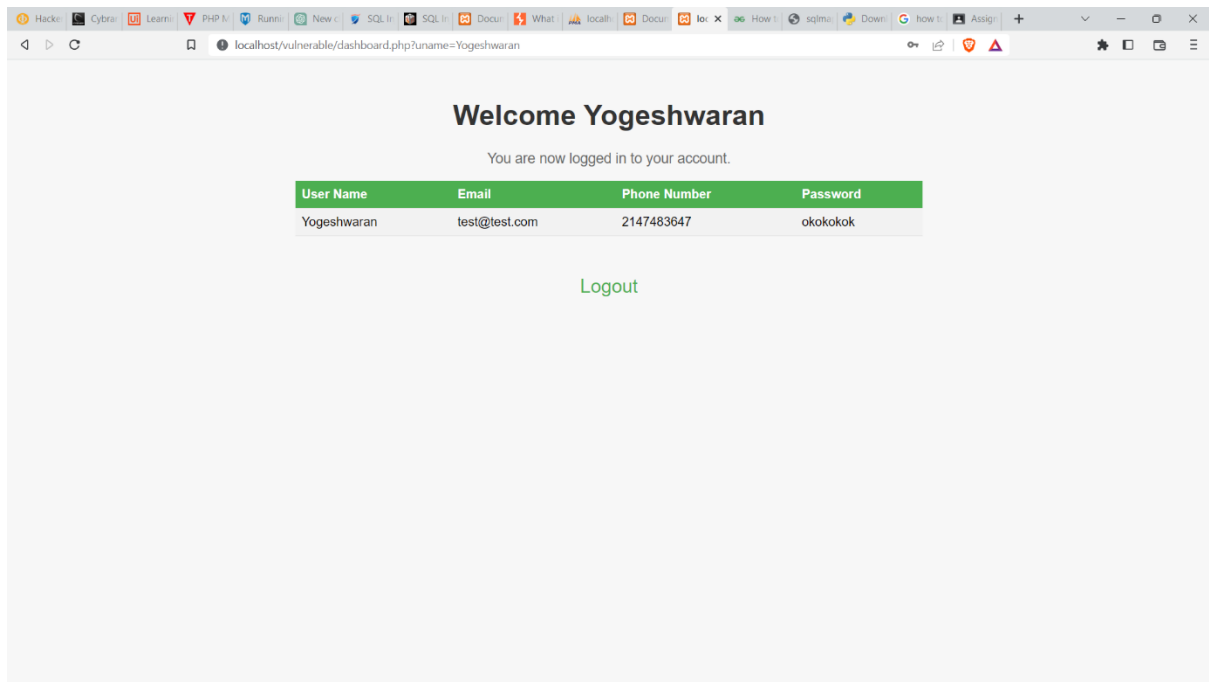
A screenshot of a web browser displaying the 'Register' page. The browser's address bar shows 'localhost/vulnerable/index.php'. The page has a black background with a large green circle on the left. A white registration form is centered, containing fields for 'User Name', 'Email', 'Phone number', and 'Password'. The 'User Name' field contains 'Yogeshwaran', 'Email' contains 'test@test.com', and 'Phone number' contains '958959595'. The 'Password' field is masked with asterisks. Below the fields is a green 'Register' button and a link that says 'Already have an account? [LOGIN](#)'.

Register page



A screenshot of a web browser displaying the 'Login' page. The browser's address bar shows 'localhost/vulnerable/login.php'. The page has a black background with a large green circle on the left. A white login form is centered, containing fields for 'Username' and 'Password'. The 'Username' field contains 'Yogeshwaran' and the 'Password' field is masked with asterisks. Below the fields is a green 'Login' button and a link that says 'Don't you have an account? [REGISTER](#)'.

Login page



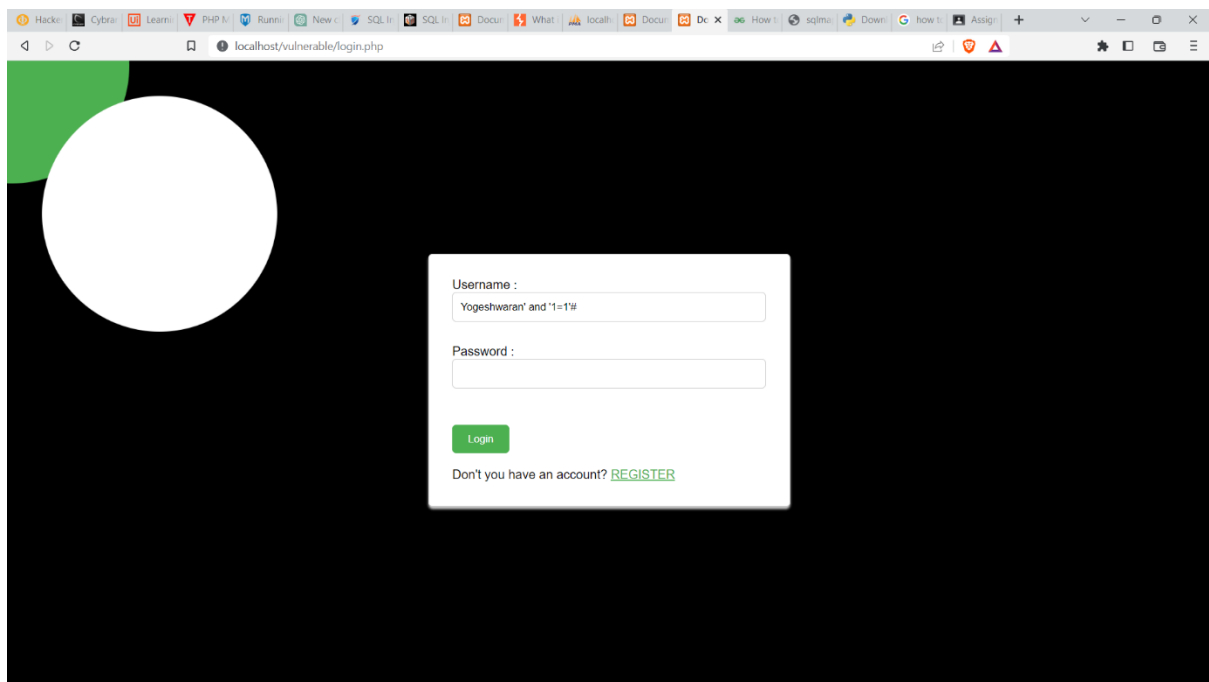
Dashboard Page

SQL Injection:

SQL injection is a type of cyber attack in which an attacker injects malicious SQL (Structured Query Language) code into a web application's input fields, with the aim of gaining unauthorized access to a database or executing unintended actions.

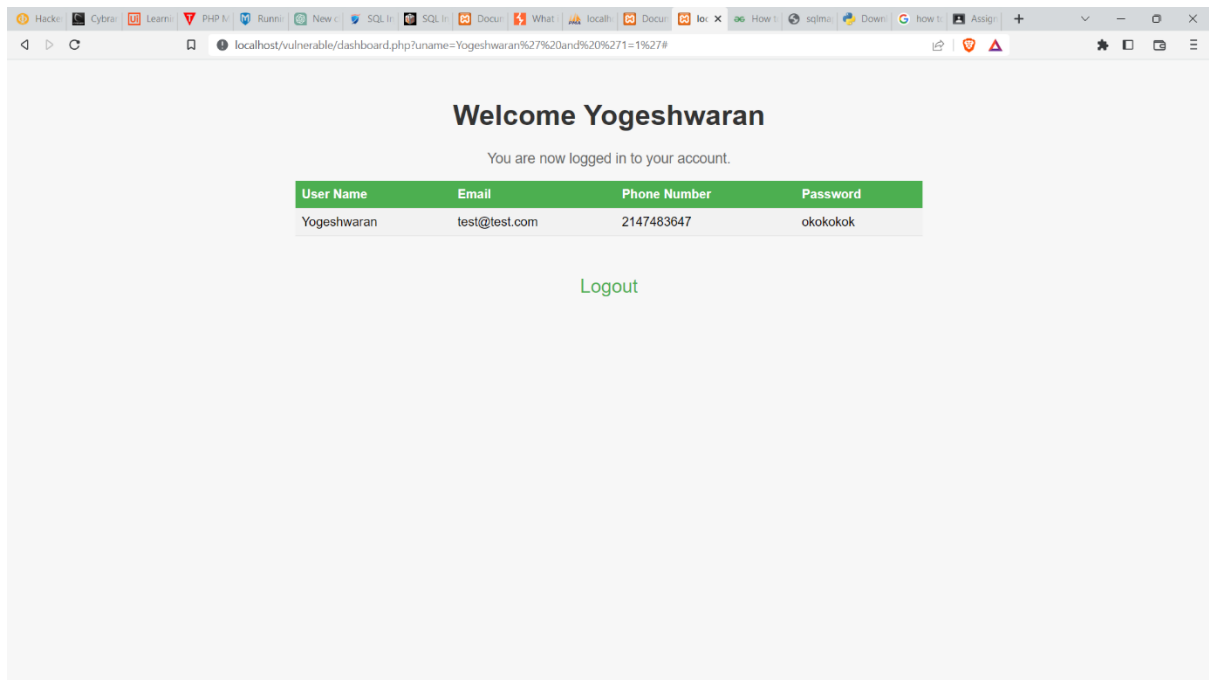
The attack works by taking advantage of poorly sanitized user inputs that are directly incorporated into SQL queries without proper validation, enabling the attacker to manipulate the database queries to achieve unintended outcomes. SQL injection can be used to extract sensitive information, such as usernames and passwords, modify or delete data in the database, and execute arbitrary commands on the server.

Trying to perform SQL Injection through the login page:



Username: Yogeshwaran' and '1=1'##

The SQL injection through the login page works and it redirect the attacker to user detail page “dashboard.php”.



Redirected page

Preventing SQL Injection:

Preventing SQL injection attacks involves using secure coding practices, such as input validation and parameterized queries, which ensure that user inputs are sanitized before being used in SQL queries. Additionally, regularly updating and patching web applications can also help prevent SQL injection vulnerabilities from being exploited.

Vulnerable code:

```
register.php
1  <?php
2  include("config/config.php");
3  if(isset($_GET['submit']))
4  {
5      $uname=$_GET['uname'];
6      $email=$_GET['email'];
7      $phnumber=$_GET['phnumber'];
8      $password=$_GET['password'];
```

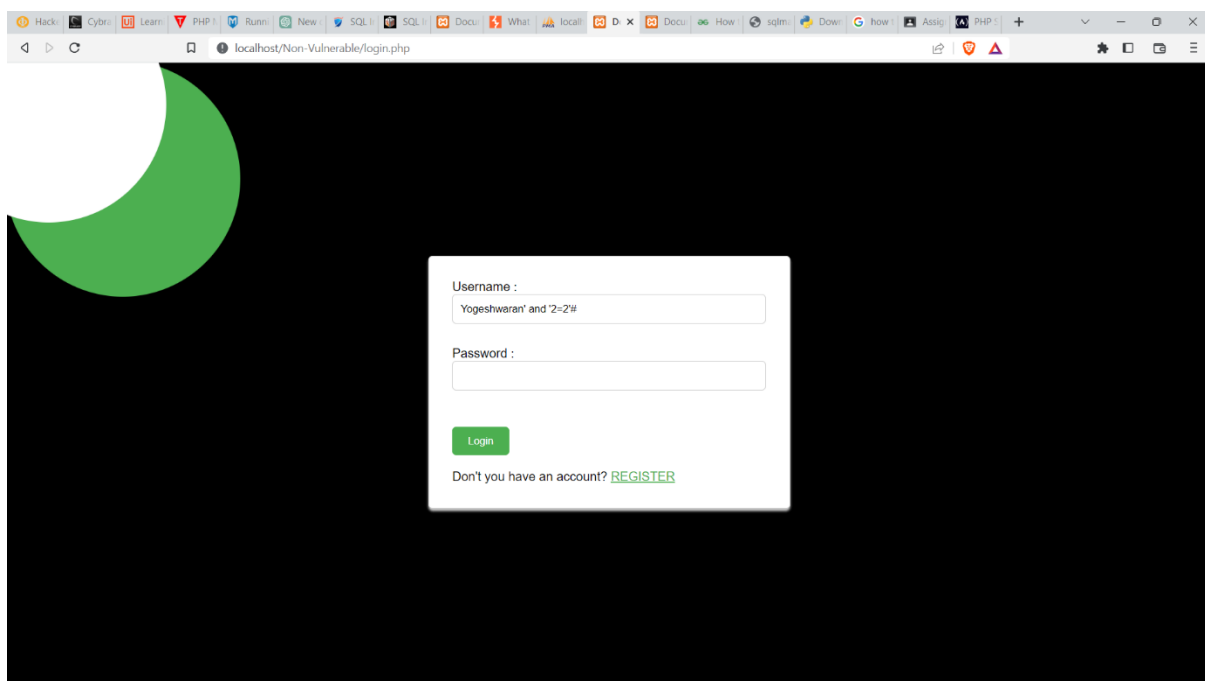
Secure code:

```
register2.php
1  <?php
2  include("config/config.php");
3  if(isset($_POST['submit']))
4  {
5      $uname = mysqli_real_escape_string($mysqli, $_POST['uname']);
6      $email = mysqli_real_escape_string($mysqli, $_POST['email']);
7      $phnumber = mysqli_real_escape_string($mysqli, $_POST['phnumber']);
8      $password = mysqli_real_escape_string($mysqli, $_POST['password']);
9  }
```

Sanitizing the input to prevent from SQL Injection attack

After secure coding:

Trying to perform SQL Injection through the login page:



Attempt to bypass the login



Username or Password is incorrect

It refuses the connection

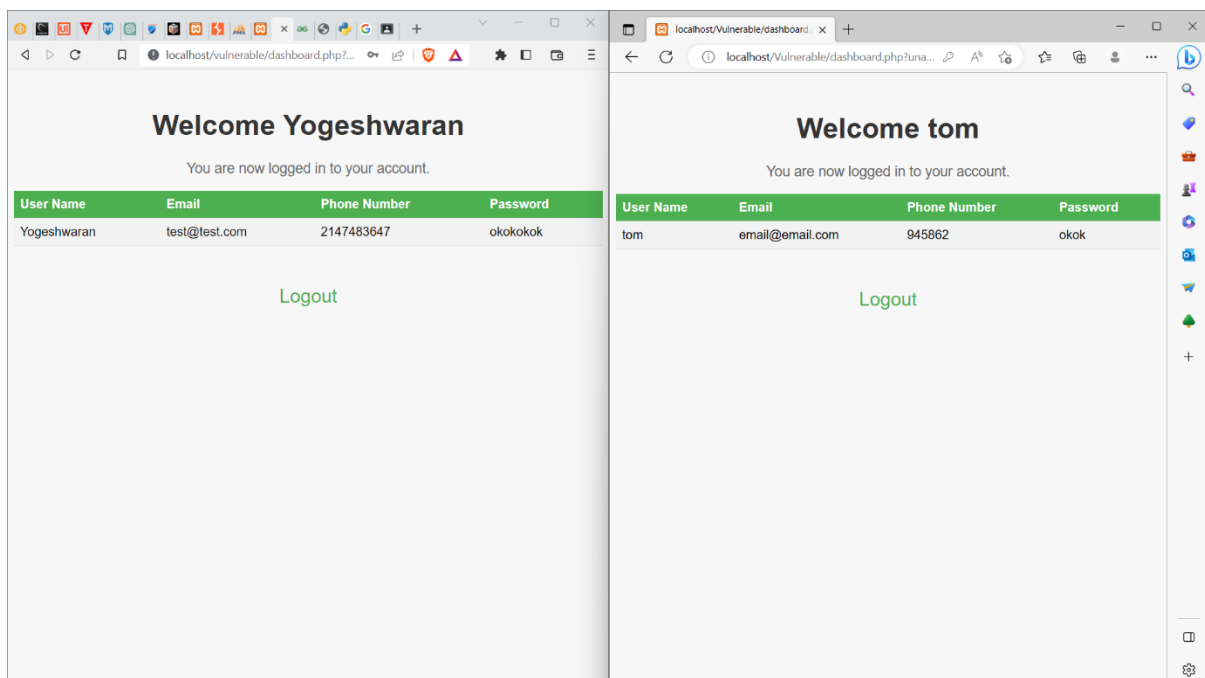
SESSION Hijacking:

Session hijacking is a type of cyber attack where an attacker gains unauthorized access to a user's session on a web application or website by intercepting and stealing their session cookies or tokens.

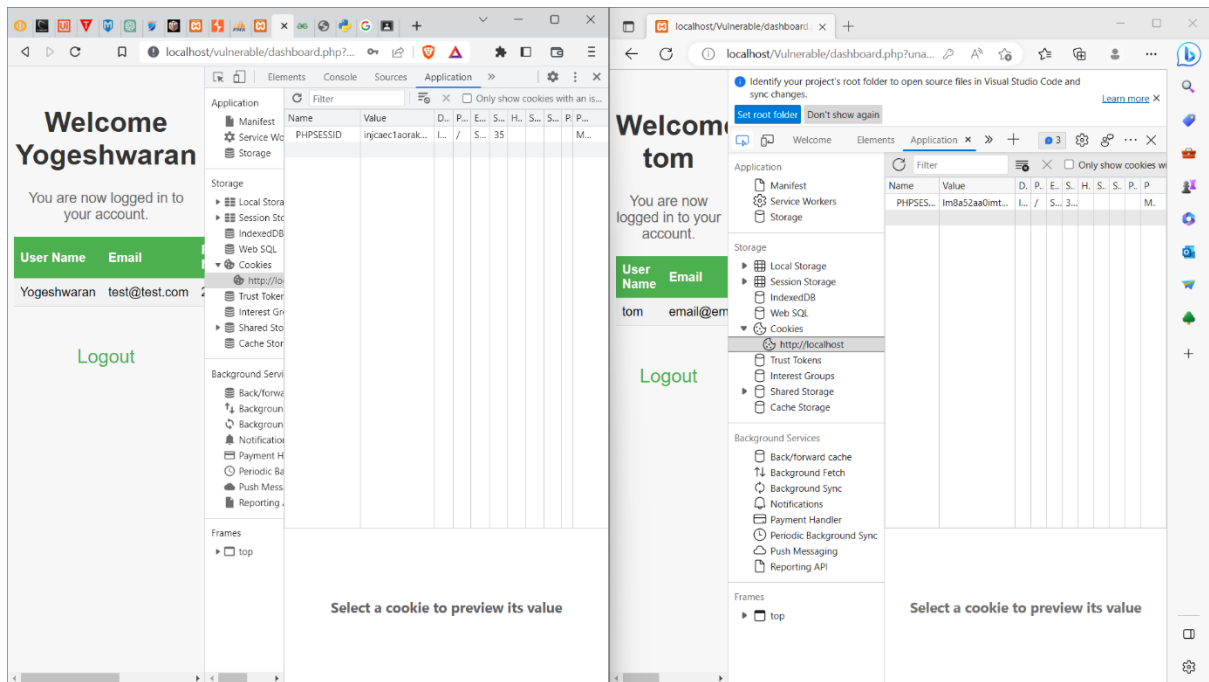
Session cookies are used by web applications to maintain the state of a user's session between page requests. When a user logs into a web application, the server creates a unique session identifier and sends it to the user's browser in the form of a cookie. The browser stores the cookie and sends it back to the server with each subsequent request, allowing the server to identify and maintain the user's session.

In a session hijacking attack, the attacker intercepts the user's session cookie, either by sniffing it from network traffic or stealing it through a malicious script or phishing attack. The attacker can then use the stolen cookie to impersonate the user and gain access to the user's account, with all the privileges and permissions associated with that account.

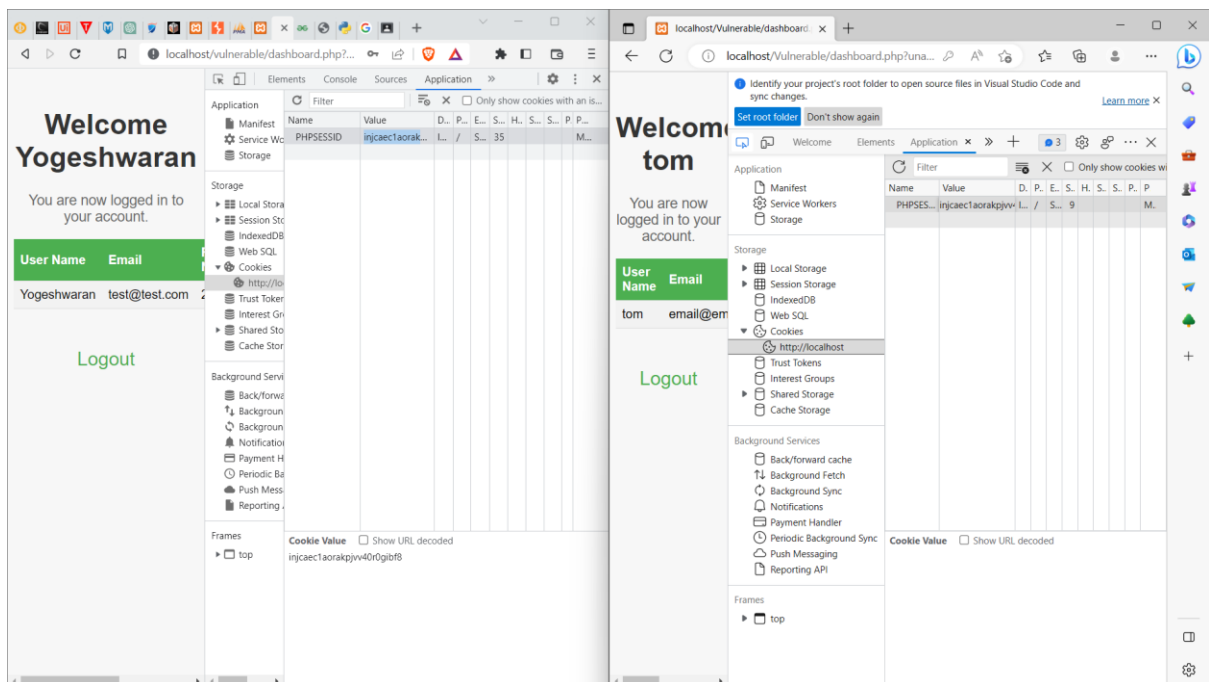
Trying to perform Session hijacking:



Two different users



Different session id's



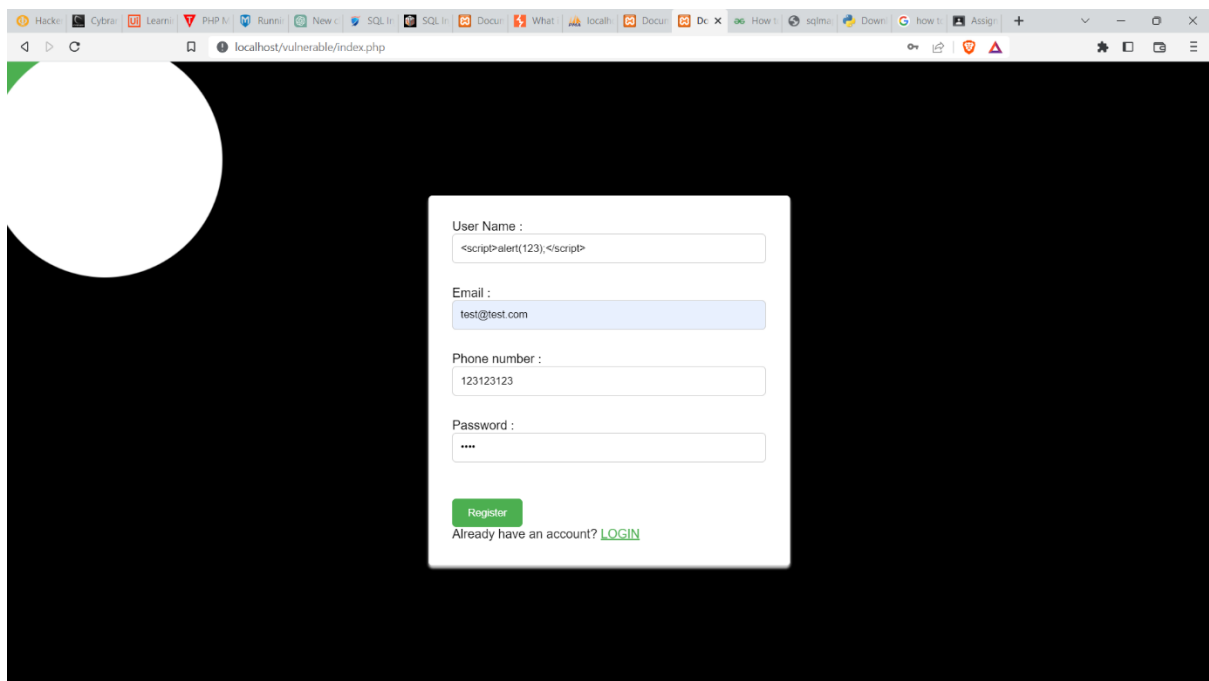
Replacing the session id of one user to another user and reload it

XSS:

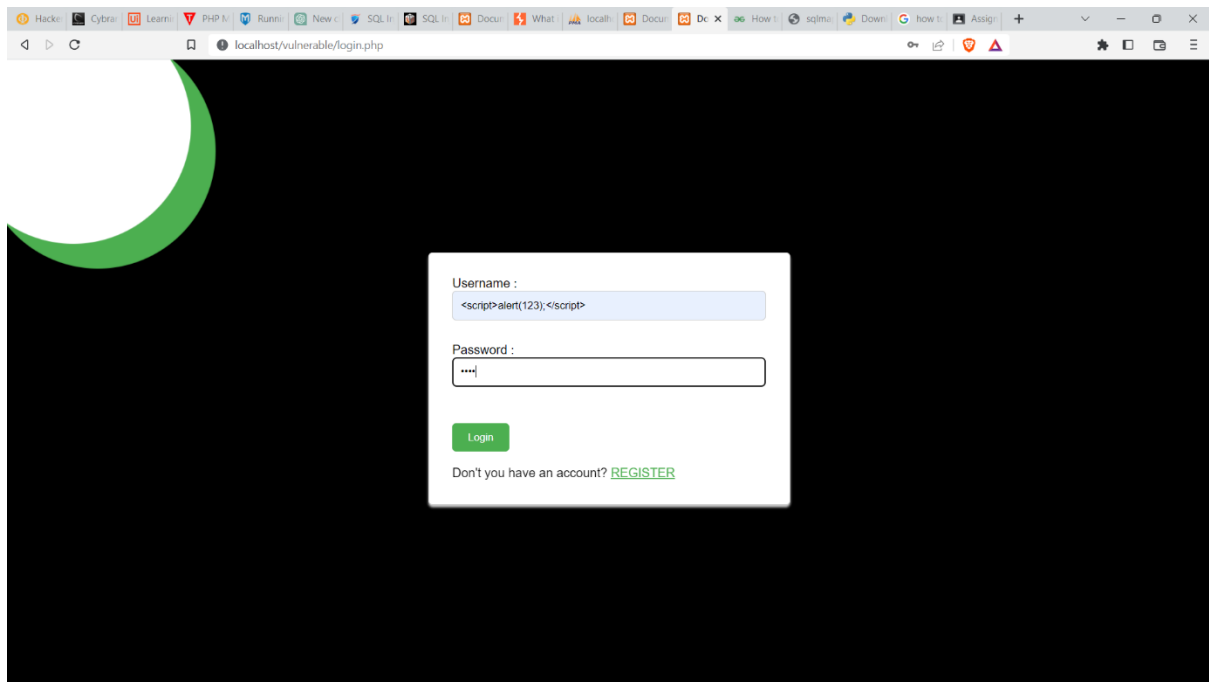
XSS (Cross-Site Scripting) is a type of cyber attack in which an attacker injects malicious scripts into a website or web application, allowing them to steal sensitive information from users, modify website content, or redirect users to malicious sites.

XSS attacks exploit vulnerabilities in a website's input validation process, such as failing to sanitize user input or validate user input before displaying it on the website. The attacker can inject malicious code in various forms, such as in comments, search fields, or input forms. When a user interacts with the affected web page, the malicious script executes, enabling the attacker to steal the user's session cookies or other sensitive information, or manipulate the website content.

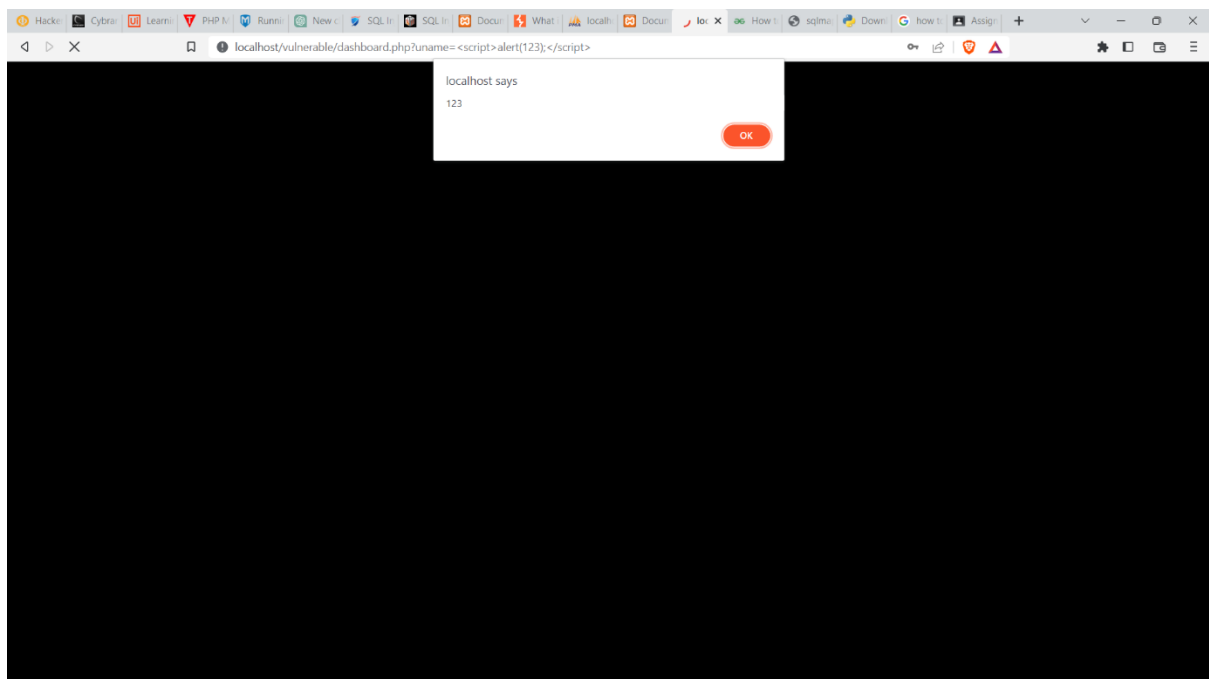
Trying to perform XSS:



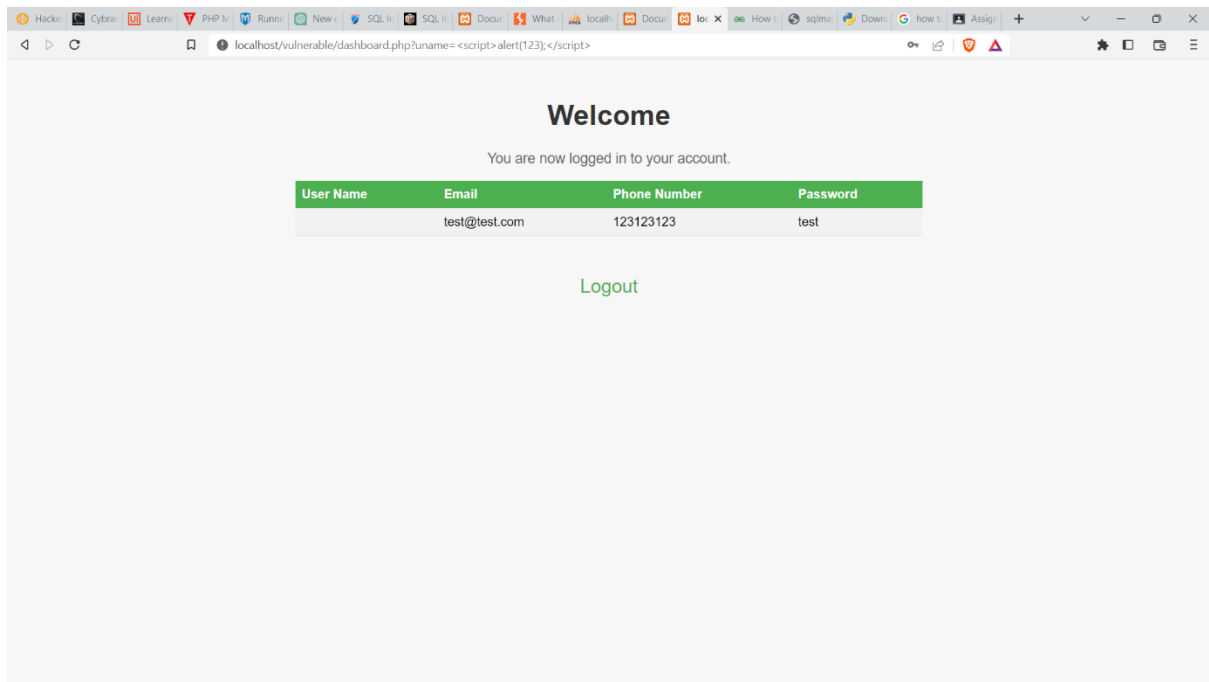
Register the User with XSS code “<script>alert(123);</script>”



Trying to login with XSS script as Username



XSS works



Redirected to Dashboard page

Preventing XSS:

Preventing XSS attacks involves implementing secure coding practices, such as properly sanitizing user input, using output encoding to prevent the injection of malicious scripts, and implementing Content Security Policy (CSP) to restrict the execution of untrusted scripts. Web developers should also regularly update and patch their web applications and educate users on how to identify and avoid suspicious links or websites. Additionally, using browser extensions or plugins that block malicious scripts can also help prevent XSS attacks.

Vulnerable code:

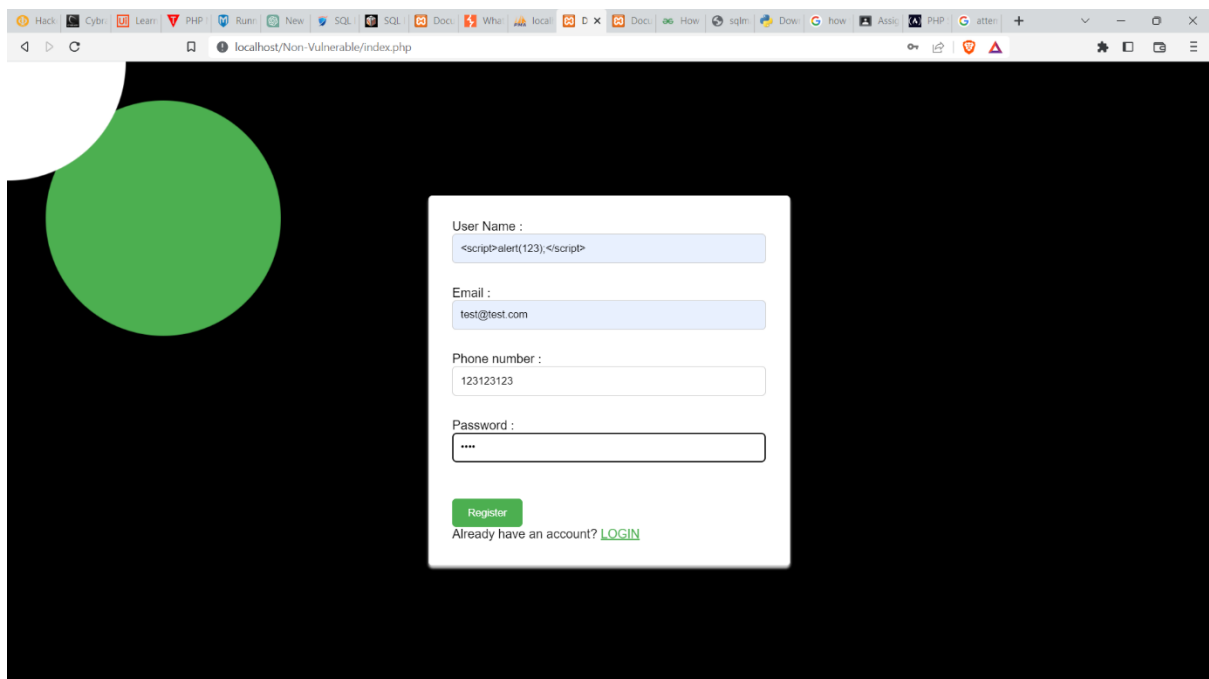
```
register.php
1  <?php
2  include("config/config.php");
3  if(isset($_GET['submit']))
4  {
5      $uname=$_GET['uname'];
6      $email=$_GET['email'];
7      $phnumber=$_GET['phnumber'];
8      $password=$_GET['password'];
```

Secure code:

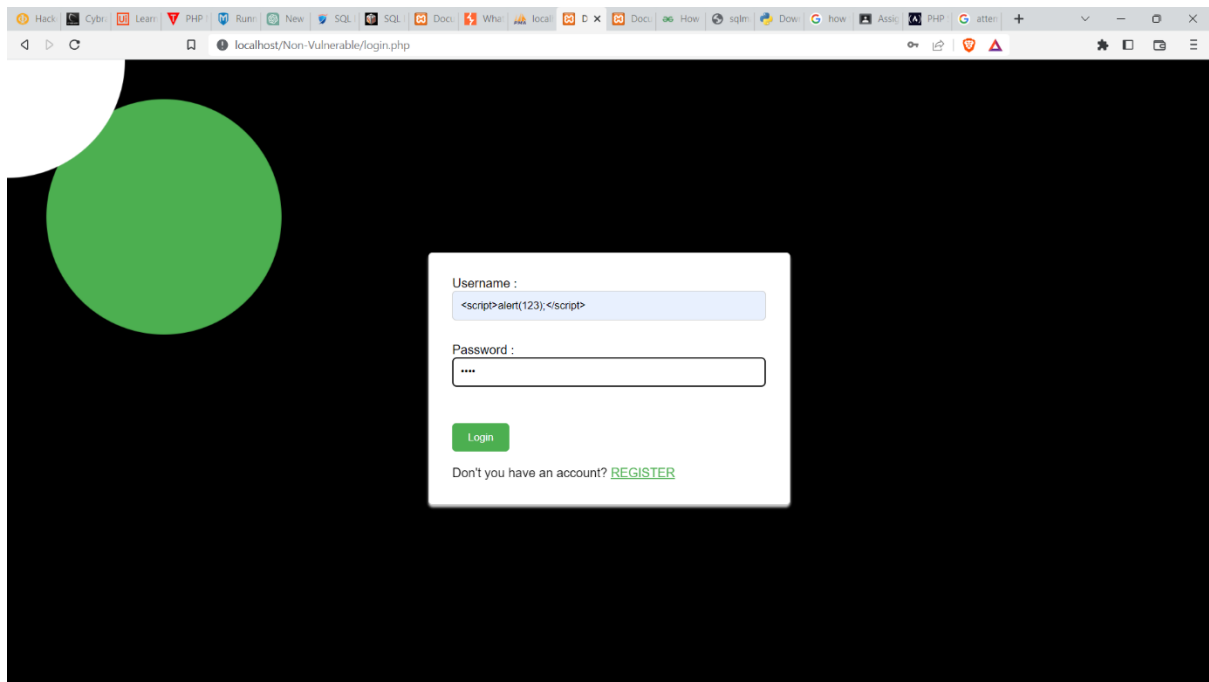
```
register2.php
1  <?php
2  include("config/config.php");
3  if(isset($_POST['submit']))
4  {
5      $uname = mysqli_real_escape_string($mysqli, $_POST['uname']);
6      $email = mysqli_real_escape_string($mysqli, $_POST['email']);
7      $phnumber = mysqli_real_escape_string($mysqli, $_POST['phnumber']);
8      $password = mysqli_real_escape_string($mysqli, $_POST['password']);
9
10     // sanitize the input
11     $uname = htmlspecialchars($uname, ENT_QUOTES, 'UTF-8');
12     $email = htmlspecialchars($email, ENT_QUOTES, 'UTF-8');
13     $phnumber = htmlspecialchars($phnumber, ENT_QUOTES, 'UTF-8');
14     $password = htmlspecialchars($password, ENT_QUOTES, 'UTF-8');
15
16     // prepare the statement
17     $stmt = $mysqli->prepare("INSERT INTO user (uname, email, phnumber, password) VALUES (?, ?, ?, ?)");
18
19     // bind parameters
20     $stmt->bind_param("ssss", $uname, $email, $phnumber, $password);
21
22     // execute the statement
23     $result = $stmt->execute();
24 }
```

After secure coding:

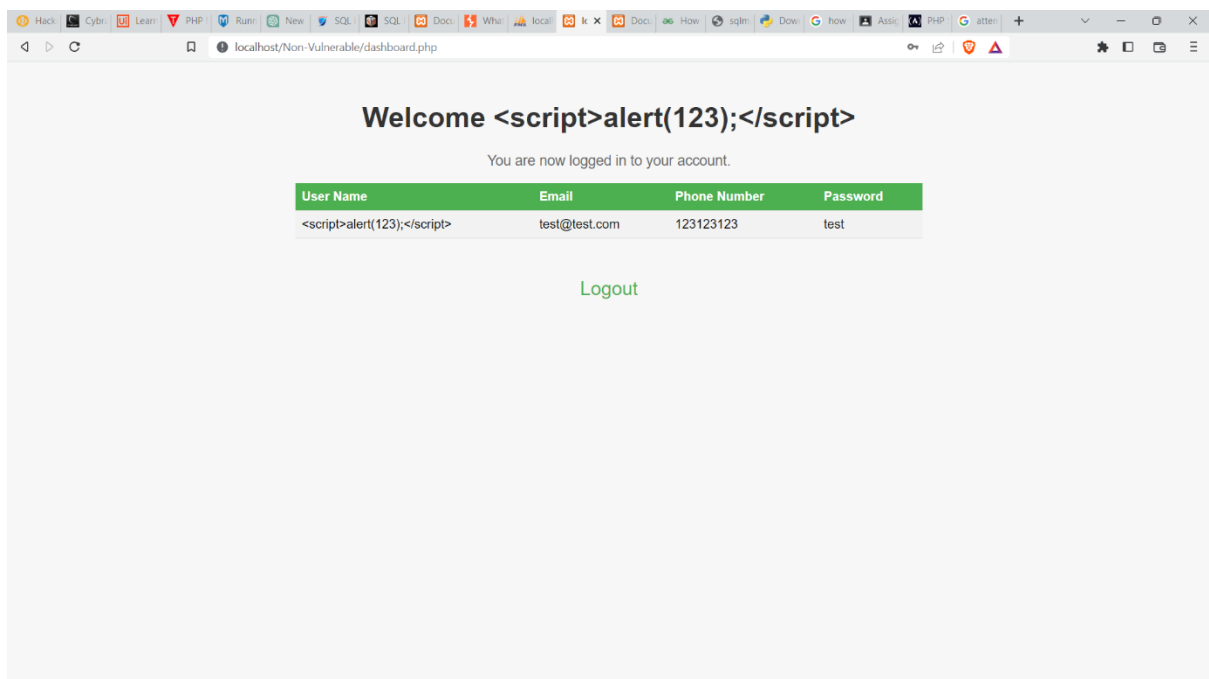
Trying to perform XSS:



Registering the user name with XSS script



Trying to login with XSS script as Username



Secure code sanitizes the input from XSS