

WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sai Keerthi Vadnala

Email: vadnalsi@ucmail.uc.edu

Short bio: Sai Keerthi Vadnala has great interest in learning web development. and wants to explore more about it by doing hands-on projects.



Figure 1: Sai Keerthi vадnala headshot

Repository Information

Repository's URL: <https://github.com/Saikeerthi72/waph-vadnalsi.git>

This is a private repository for Sai Keerthi Vadnala to store all code from the course.

Lab 3 Overview

- Lab 3 involves the development and setup of web applications using PHP and MySQL, which includes the creation of a new database.
- The lab is designed to highlight potential security vulnerabilities within web applications and proposes strategies to mitigate these risks.
- Lab 3 includes performing SQL Injection and Cross scripting attacks. It emphasizes the importance of enhancing security by using prepared statements for database and sanitization of outputs to protect against attacks.

a. Database Setup and Management:

- Mysql installation:

- MySQL is installed using the command 'sudo apt-get install mysql-server -y'
- The command 'mysql -V' is used to check the version.
- Command 'sudo mysql -u root -p' is used for connection to the database.

- Create a New Database, Database User and Permission

- Firstly, 'database-account.sql' file is created.
- Here in this file, I have created a new database and granted access to new user with a username and password.
- The code of the file is as follows:
 - create database waph;
 - Firstly, I have created a database named 'waph'
 - CREATE USER vadnalsi@'localhost' IDENTIFIED BY 'waph@UC!2024';
 - Next, I have created user with username 'vadnalsi' and password 'waph@UC!2024'
 - GRANT ALL ON waph.* TO vadnalsi@'localhost';
 - Later, I have given grant all permissions to the user
- The SQL file was imported and executed using 'sudo mysql -u root -p < database-account.sql'.
- Using the command: 'mysql -u vadnalsi -p' we can login into MySQL.

- Create a new table Users and insert data into the table

- I have created user account and inserted data (username and password) into the table.
- Here the file named 'database-data.sql' is used.
- The content of this file includes:


```
drop table if exists users;

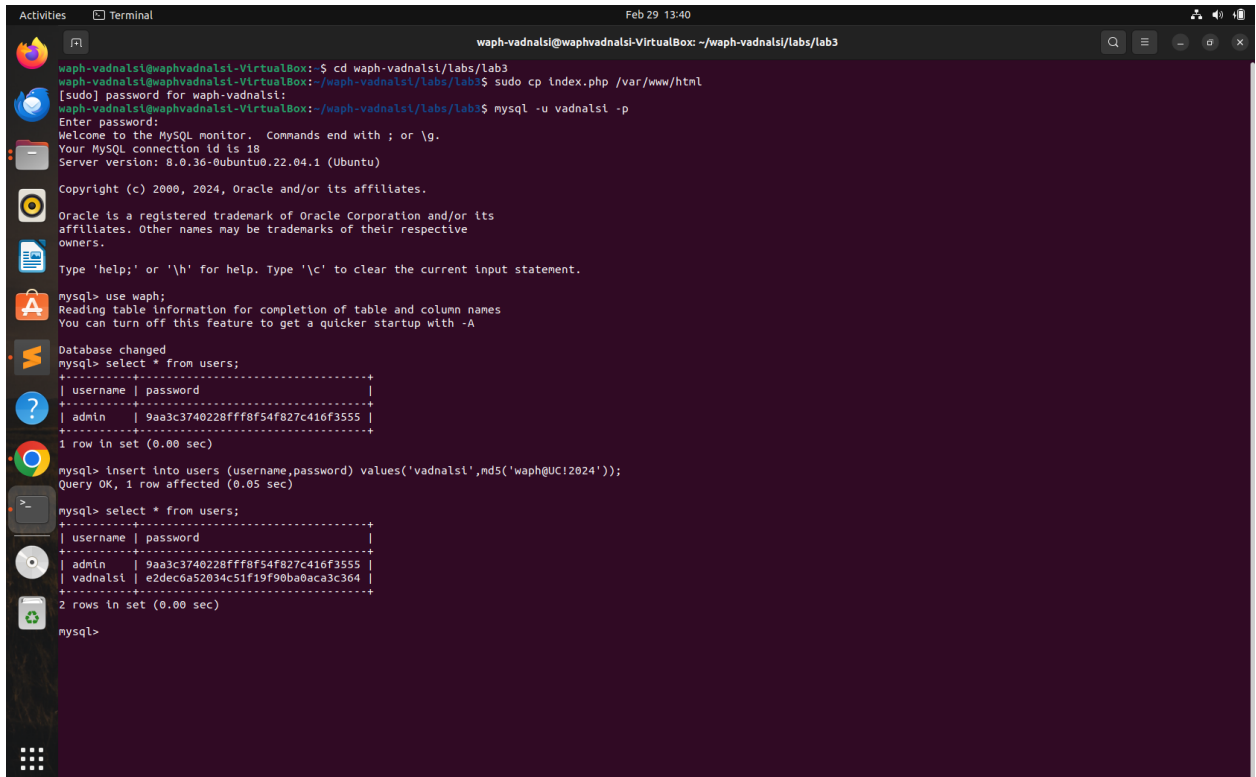
- Firstly, I have included a code to drop 'users' table if it already exists, else it creates a table named 'users'.

create table users(
username varchar(50) PRIMARY KEY,
password varchar(100) NOT NULL);

INSERT INTO users(username,password) VALUES ('admin',md5('Pa$$w0rd'));
```

-Next, the following code is about creating table named 'users' with columns username and password with type varchar. And inserted values into the table.

- Command 'mysql -u vadnalsi -p < database-data.sql' is used to create database-data.sql.
- Now I have entered command 'select * from users', which displays all user accounts and their hashed password.
- Below is the screenshot for the above task.



```
waph-vadnalsi@waphvadnalsi-VirtualBox: ~/waph-vadnalsi/labs/lab3
waph-vadnalsi@waphvadnalsi-VirtualBox: $ cd waph-vadnalsi/labs/lab3
waph-vadnalsi@waphvadnalsi-VirtualBox: ~/waph-vadnalsi/labs/lab3$ sudo cp index.php /var/www/html
[sudo] password for waph-vadnalsi:
waph-vadnalsi@waphvadnalsi-VirtualBox: ~/waph-vadnalsi/labs/lab3$ mysql -u vadnalsi -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.36-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use waph;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from users;
+-----+-----+
| username | password |
+-----+-----+
| admin    | 9aa3c3740228fff8f54f827c416f3555 |
+-----+-----+
1 row in set (0.00 sec)

mysql> insert into users (username,password) values('vadnalsi',md5('waph@UC12024'));
Query OK, 1 row affected (0.05 sec)

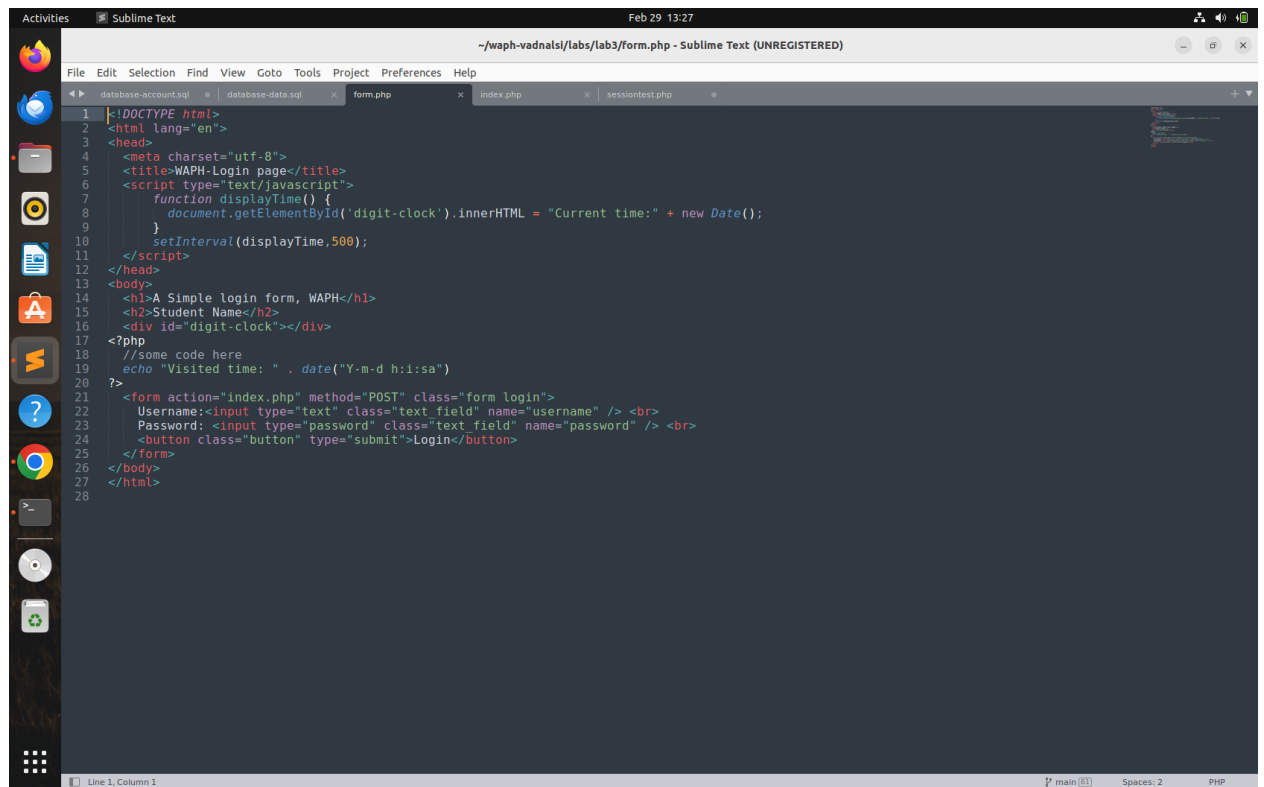
mysql> select * from users;
+-----+-----+
| username | password |
+-----+-----+
| admin    | 9aa3c3740228fff8f54f827c416f3555 |
| vadnalsi | e2dec6a52034c51f19f90ba0aca3c364 |
+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

b. A Simple (Insecure) Login System with PHP/MySQL

- For this task we need to install few pre-requisites, So I have installed 'php-mysqli', which is used for performing database PHP/MySQL operations.
- To install the php-mysqli command 'sudo apt-get install php-mysqli' is used.
- The Apache server was restarted using 'sudo service apache2 restart' command, this will restart the apache services.
- 'Form.php' is designed to include input fields for a username and password, along with a submission button.
- Upon clicking the submit button, the form data (username and password) are sent to 'index.php' for processing.

- Below is the code for form.php

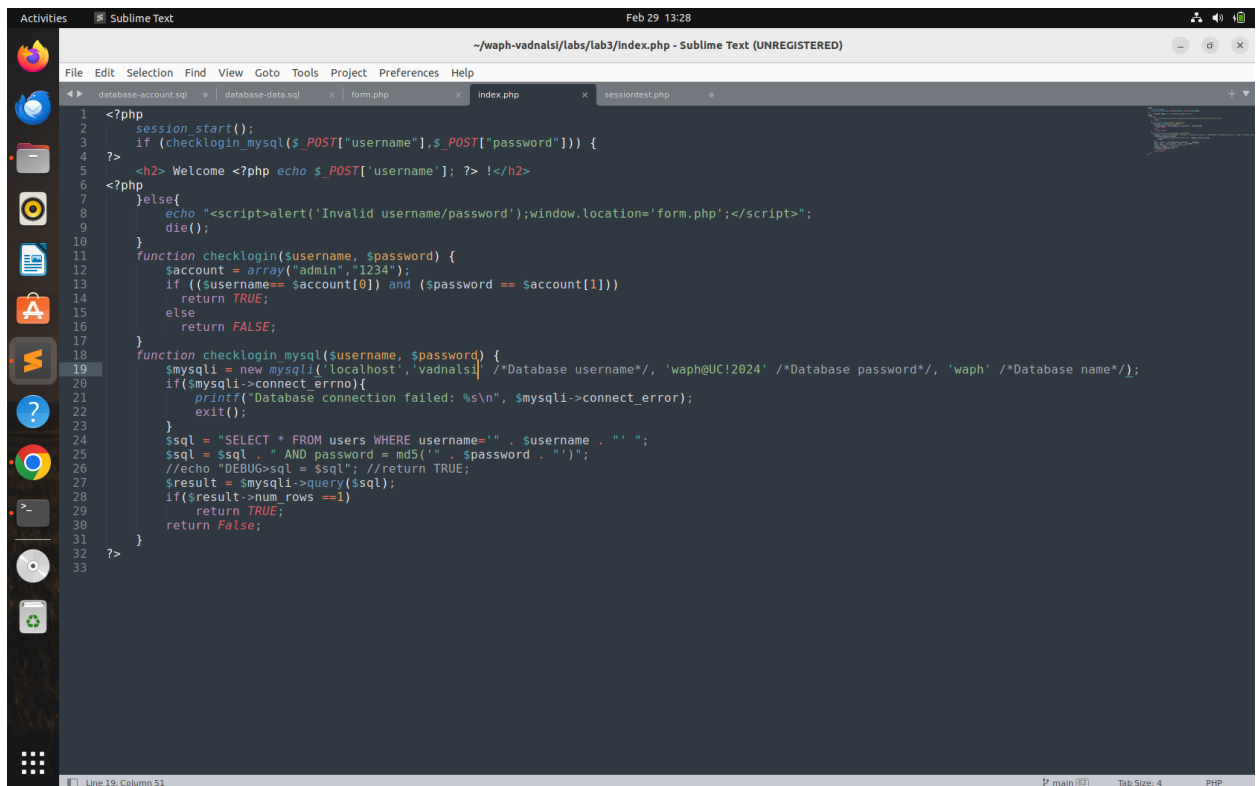


```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>WAPH-Login page</title>
6 <script type="text/javascript">
7     function displayTime() {
8         document.getElementById('digit-clock').innerHTML = "Current time:" + new Date();
9     }
10    setInterval(displayTime,500);
11 </script>
12 </head>
13 <body>
14 <h1>A Simple login form, WAPH</h1>
15 <h2>Student Name</h2>
16 <div id="digit-clock"></div>
17 <?php
18 //some code here
19 echo "Visited time: " . date("Y-m-d h:i:sa")
20 >?>
21 <form action="index.php" method="POST" class="form login">
22     Username:<input type="text" class="text field" name="username" /> <br>
23     Password:<input type="password" class="text field" name="password" /> <br>
24     <button class="button" type="submit">Login</button>
25 </form>
26 </body>
27 </html>
28

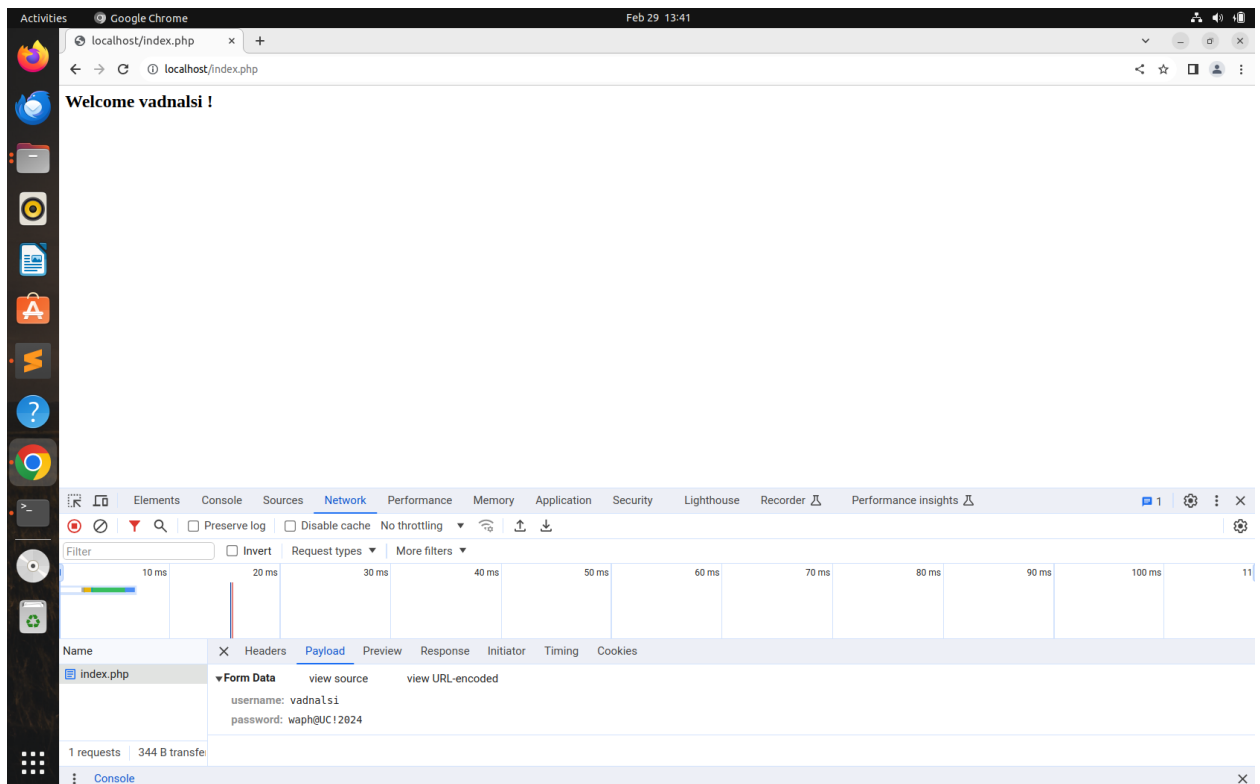
```

- Here we update 'index.php' file by incorporating code that establishes a connection to the database.
- Next step we create a `checklogin_mysql` function which is used to collect the user credentials with POST method and then authenticate.
- The authentication process follows: In case the credentials entered match with the ones in database it returns the web page, else the output would be as follows "invalid username and password".
- Below screenshot displays the code for index.php file.



```
1 <?php
2 session_start();
3 if (checklogin_mysql($_POST["username"],$_POST["password"])) {
4     ?> <h2> Welcome <?php echo $_POST['username']; ?> !</h2>
5 }
6
7 <?php
8 }else{
9     echo "<script>alert('Invalid username/password');window.location='form.php';</script>";
10    die();
11 }
12 function checklogin($username, $password) {
13     $account = array("admin","1234");
14     if (($username== $account[0]) and ($password == $account[1]))
15         return TRUE;
16     else
17         return FALSE;
18 }
19 function checklogin_mysql($username, $password) {
20     $mysqli = new mysqli('localhost','vadnalsi', /*Database username*/, 'waph@UC!2024' /*Database password*/, 'waph' /*Database name*/);
21     if($mysqli->connect_errno){
22         printf("Database connection failed: %s\n", $mysqli->connect_error);
23         exit();
24     }
25     $sql = "SELECT * FROM users WHERE username='".$username."' ";
26     $sql = $sql . " AND password = md5('".$password."')";
27     //echo "DEBUG>sql = $sql"; //return TRUE;
28     $result = $mysqli->query($sql);
29     if($result->num_rows ==1)
30         return TRUE;
31     return FALSE;
32 }
33 ?>
```

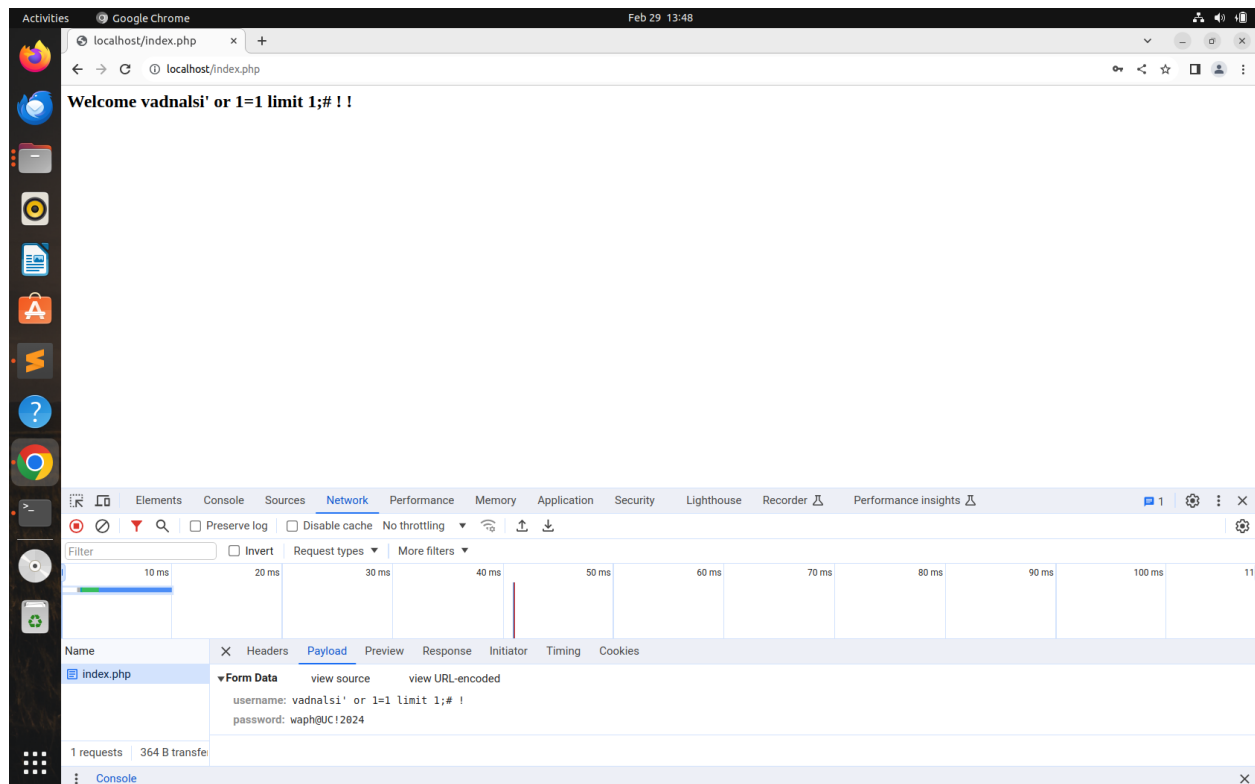
- When attempting to log in using the correct credentials, the system will successfully authenticate the user, and prints a “welcome admin” message.
- The output for this task is displayed below:



c. Performing XSS and SQL Injection Attacks

SQL Injection Attacks:

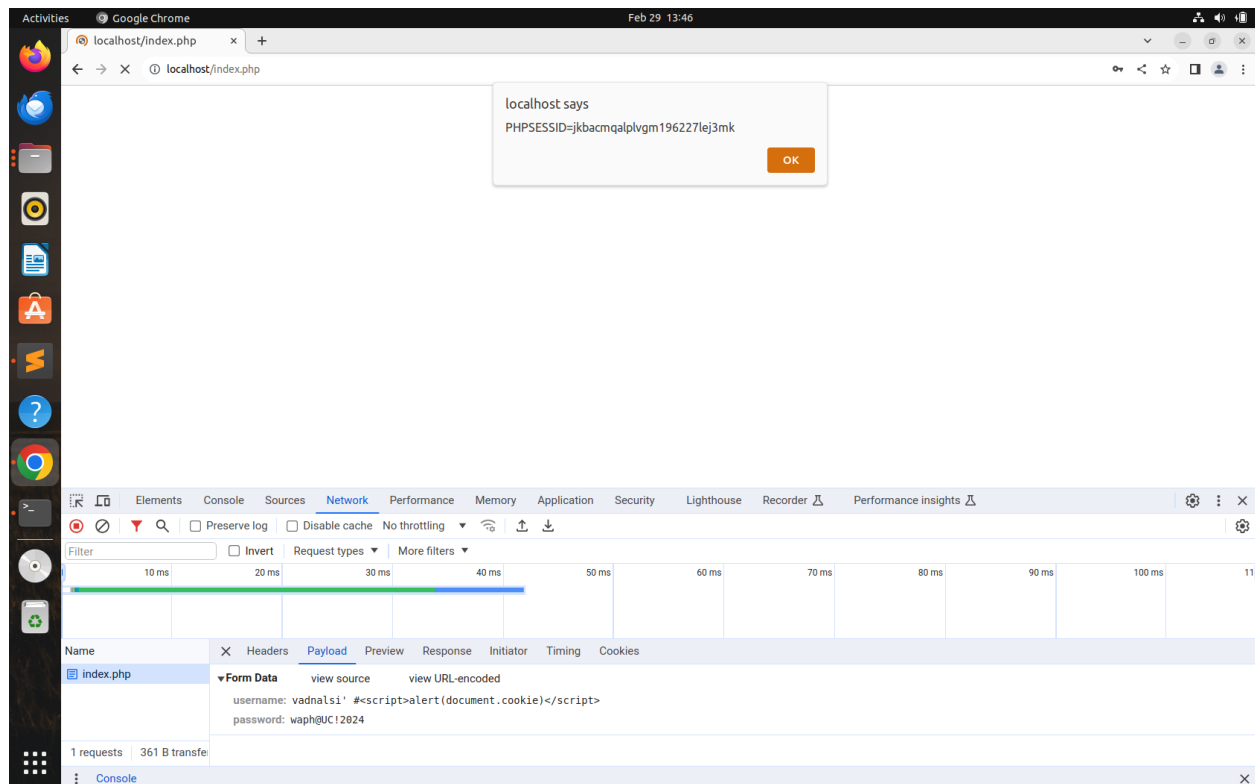
- To perform an SQL Injection attack, attackers inject a sql code in the input field. So, I have used a sql code and entered username field to bypass the login credentials. The code I entered in the input field is `vадnalsi' or 1=1 limit 1;#`.
- The login was successful and the demo of the task is shown in the below screenshot.



- SQL injection attacks occur due to security vulnerabilities in web applications. When input fields accept code without conducting any validation or sanitization, the application inadvertently constructs SQL queries based on this input.
- In this scenario, the password is in comments using `#` with which it is effectively bypassing the password check.

Cross scripting Attacks

- I have implemented an XSS attack.
- I have given the input for username as: `'#<script>alert(document.cookie)</script>'`.
- Upon submitting this code through the login form, it gives the below output:



- This security vulnerability in the web application allows an attacker to submit input containing malicious JavaScript code.
- This issue arises primarily from inadequate input validation.
- Furthermore, the vulnerability gets worse when the application outputs the same input without correctly encoding it.
-

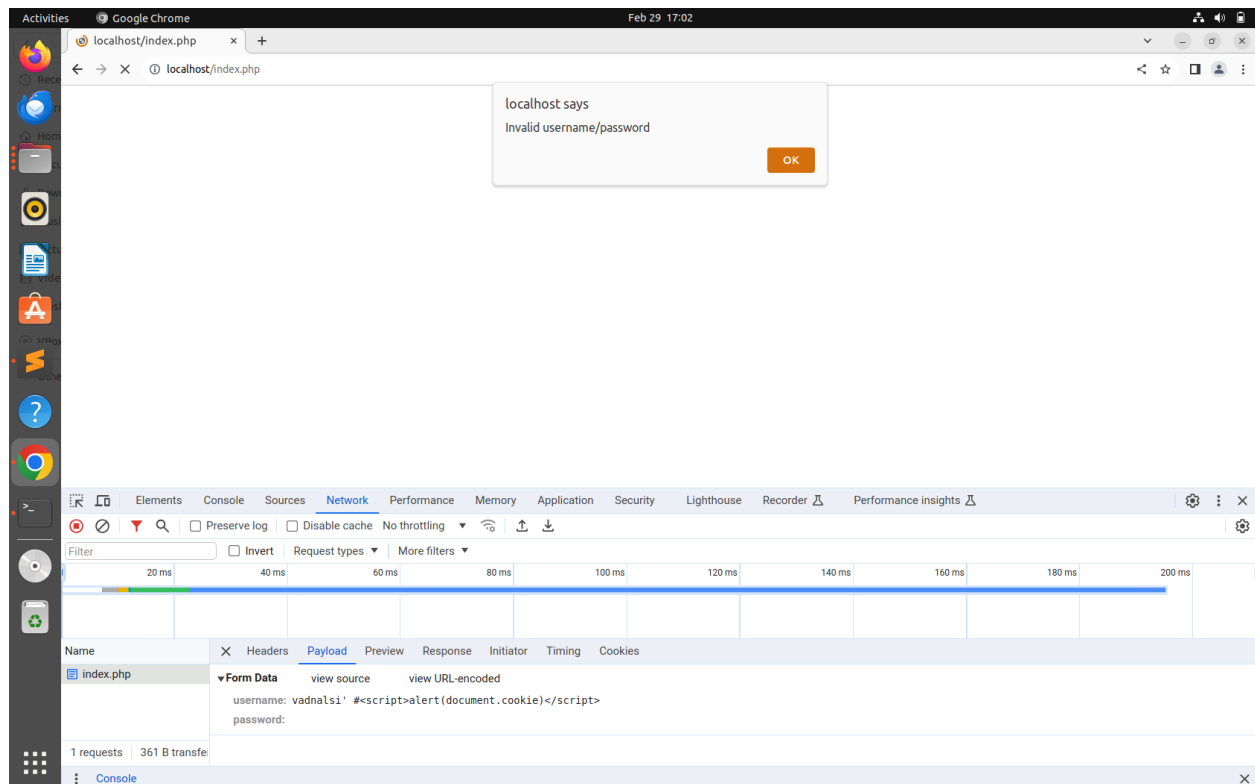
d. Prepared Statement Implementation

Prepared Statement for SQL Injection Prevention

- Prepared statements are used for enhancing security by validating input and securing web applications against hacking attempts. Here I have updated the `index.php` file to include SQL prepared statements.
- Later, we perform binding each input field to the prepared statement.
- Below is the screenshot,
- Modified code:


```
1 <?php
2 session_start();
3 if (checklogin_mysql($_POST['username'],$_POST['password'])) {
4
5
6 }
7 <h2> Welcome <?php echo $_POST['username']; ?> !</h2>
8
9 <?php
10 }else{
11     echo "<script>alert('Invalid username/password');window.location='form.php';</script>";
12     die();
13 }
14 function checklogin($username, $password) {
15     $account = array("admin","1234");
16     if (($username== $account[0]) and ($password == $account[1]))
17         return TRUE;
18     else
19         return FALSE;
20 }
21 function checklogin_mysql($username, $password) {
22     $mysqli = new mysqli('localhost','vadnalsi' /*Database username*/, 'waph@UC12024' /*Database password*/, 'waph' /*Database name*/);
23     if($mysqli->connect_errno){
24         printf("Database connection failed: %s\n", $mysqli->connect_error);
25         exit();
26     }
27     $sql = "SELECT * FROM users WHERE username=? AND password=MD5(?);";
28     $stmt = $mysqli->prepare($sql);
29     $stmt->bind_param("ss", $username, $password);
30     $stmt->execute();
31     //echo "DEBUG>sql = $sql"; //return TRUE;
32     $result = $stmt->get_result();
33     if($result->num_rows ==1)
34         return TRUE;
35     return FALSE;
36 }
37 ?>
```

- After implementing the new code with prepared statements in the login form, if the execution results in an "invalid username and password" message, this indicates that the login attempt was unsuccessful.
- Below is the screenshot,



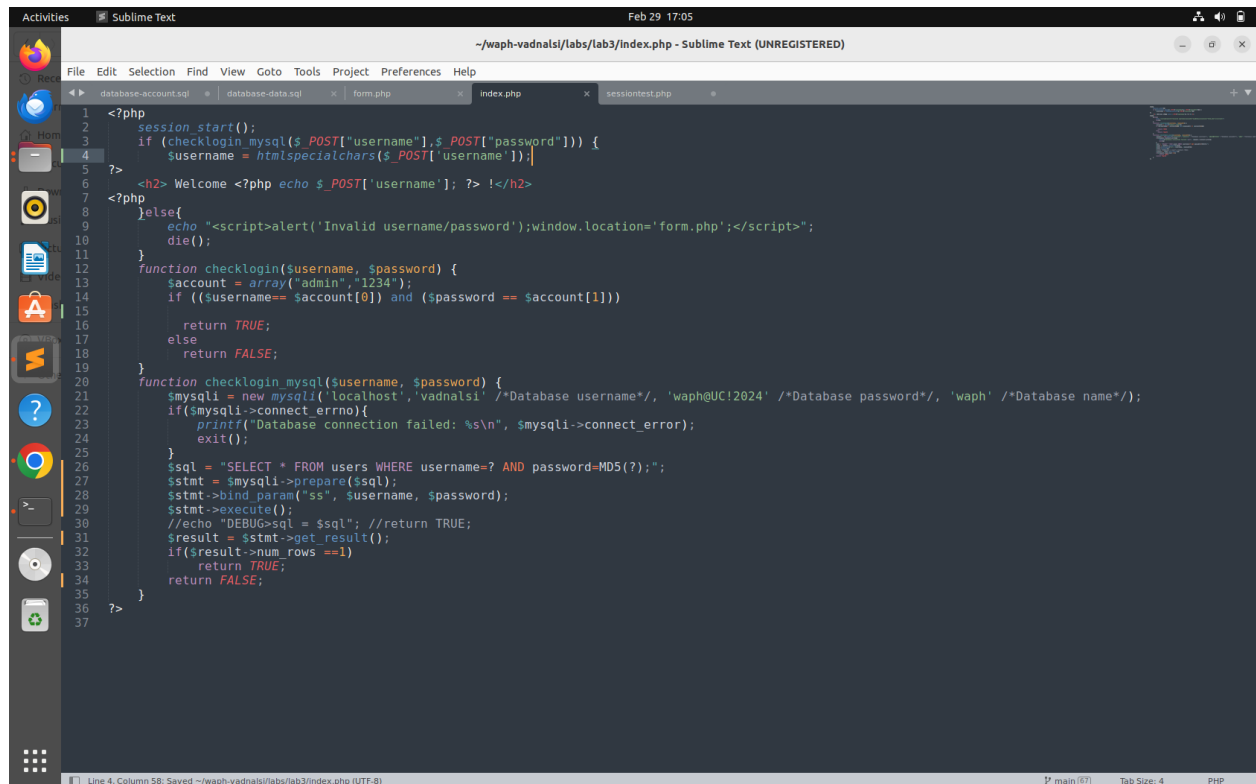
Security Analysis

Prepared Statement Explanation:

- Prepared statements help in easing SQL injection risks by differentiating the SQL command from user query.
- Prepared statements ensure that user inputs are never performed as part of the SQL command by clearly separating user data from SQL code. Attackers are prevented from injecting harmful SQL code by this separation.
- Any effort to modify the query's structure through injection is nullified since the SQL query structure in prepared statements is preset and compiled before obtaining user input.
- Since prepared statements handle all entered data as strings, they automatically sanitize user inputs. This essentially eliminates SQL injection vulnerabilities by automatically neutralizing special characters that might change SQL commands.

Implement Sanitization:

- To prevent from Cross-Site Scripting (XSS) attacks, I used the `htmlspecialchars()` function. This function is used in converting special characters into HTML entities.
- Implementing server-side input validations serves as another layer of security. This includes verifying email formats to ensure they meet standard criteria, assessing the length of inputs.
- Revised code:



```

1  <?php
2  session_start();
3  if (checklogin_mysql($_POST['username'],$_POST['password'])) {
4      $username = htmlspecialchars($_POST['username']);
5  }
6  <h2> Welcome <?php echo $_POST['username']; ?> !</h2>
7  <?php
8  }else{
9      echo "<script>alert('Invalid username/password');window.location='form.php';</script>";
10     die();
11 }
12 function checklogin($username, $password) {
13     $account = array("admin","1234");
14     if (($username == $account[0]) and ($password == $account[1]))
15         return TRUE;
16     else
17         return FALSE;
18 }
19 }
20 function checklogin_mysql($username, $password) {
21     $mysqli = new mysqli('localhost','vadnalsi','/*Database username*/', 'waph@UC12024' /*Database password*/, 'waph' /*Database name*/);
22     if($mysqli->connect_errno){
23         printf("Database connection failed: %s\n", $mysqli->connect_error);
24         exit();
25     }
26     $sql = "SELECT * FROM users WHERE username=? AND password=MD5(?);";
27     $stmt = $mysqli->prepare($sql);
28     $stmt->bind_param("ss", $username, $password);
29     $stmt->execute();
30     //echo "DEBUG>sql = $sql"; //return TRUE;
31     $result = $stmt->get_result();
32     if($result->num_rows ==1)
33         return TRUE;
34     return FALSE;
35 }
36 ?>
37

```

Discussions

- Absence of input validation, if username and password fields are not checked for empty values, the system may execute invalid queries, which could result in unauthorized or unexpected access.
- Inadequate Error Handling, when database problems occur, the lack of understandable error messages hinders users from identifying the issue and may even conceal serious system risks.
- If usernames aren't normalized for whitespace and case sensitivity, legitimate users may unintentionally deny access.

- Enhancing password security through stronger encryption methods is another area for improvement. While MD5 is used in the example, switching to more robust encryption techniques would significantly increase security.
- Despite employing prepared statements and input sanitization, there remain vulnerabilities, specifically when usernames and passwords are submitted as empty. This can cause the post variables to be null. So, it's crucial to validate inputs.