WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student Name: Charan Sai Venaganti

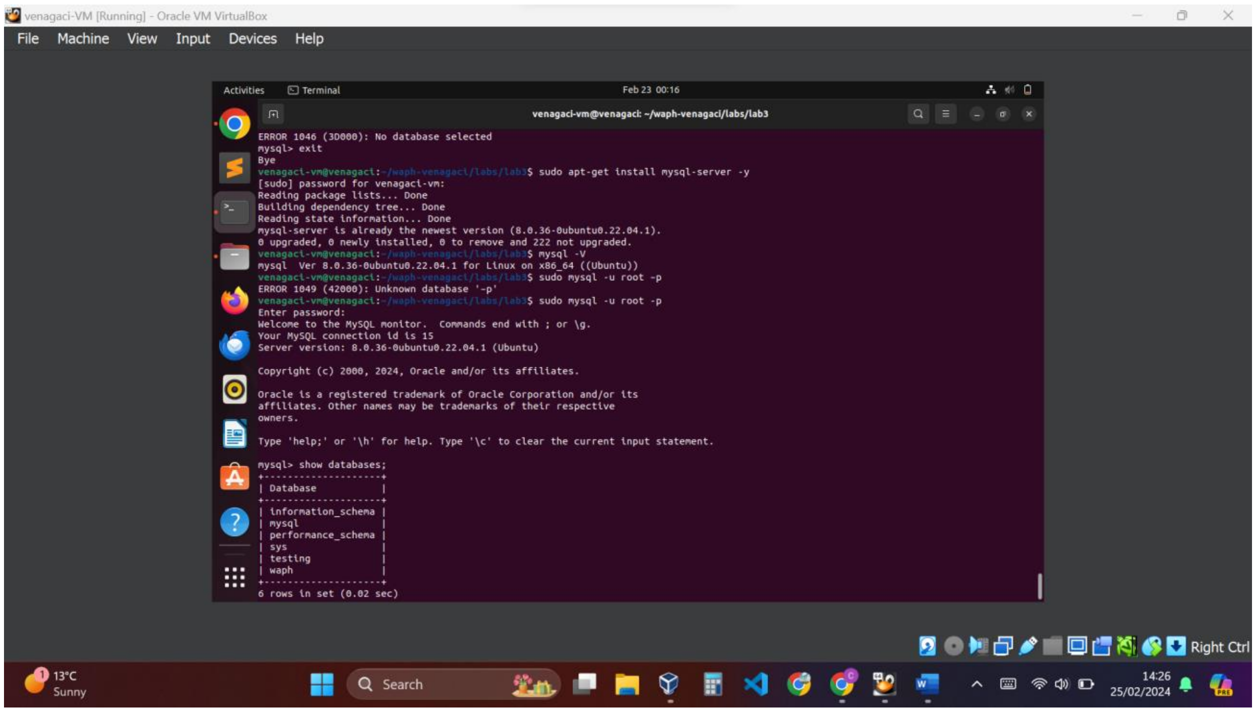
Email: venagaci@mail.uc.edu

Repository URL: (https://github.com/venagaci/waph-venagaci.git)

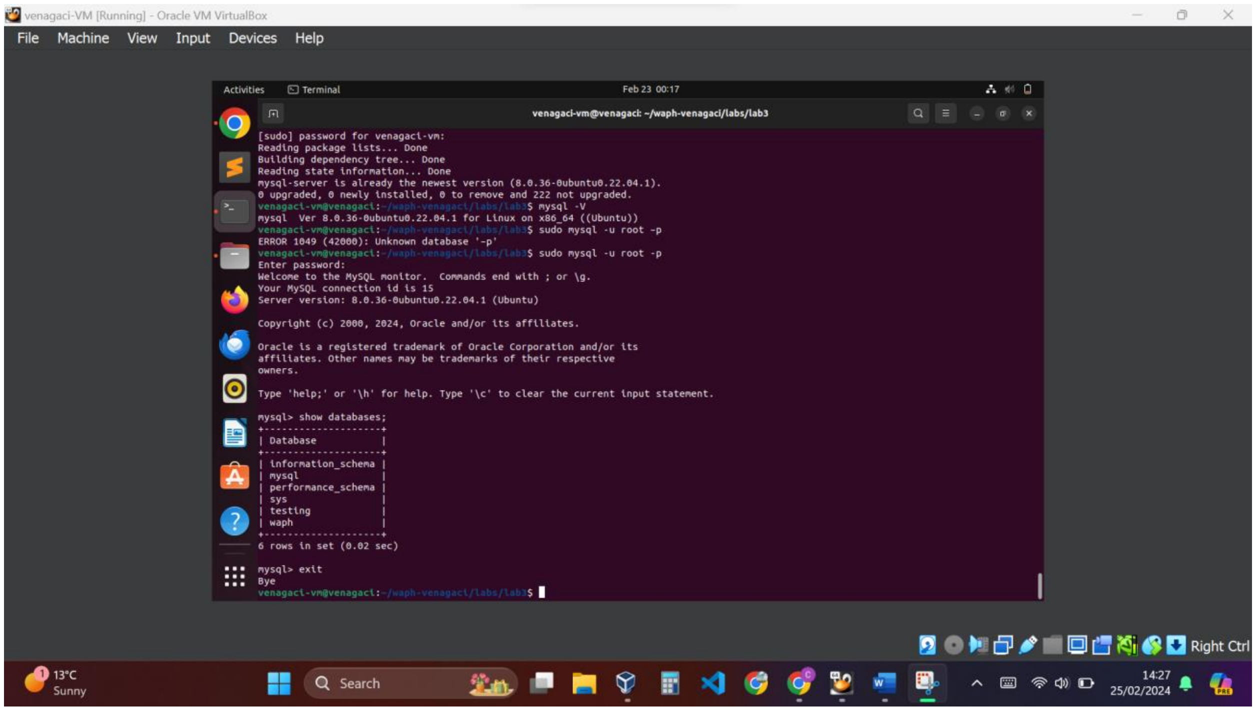
Overview: *Through the completion of various tasks outlined in the assignment, I have gained a comprehensive understanding of creating, managing, and securing web applications using PHP and MySQL. Beginning with database setup and management, including the creation of tables and user permissions, I then progressed to developing an insecure login system and subsequently executing XSS and SQL Injection attacks to understand their vulnerabilities. By implementing prepared statements and output sanitization techniques, I fortified the system against these attacks while also analyzing the significance of these security measures. Through this assignment, I have not only learned practical skills in web development and security but also gained insights into the importance of proactive security measures and the potential vulnerabilities present in web applications***.**

a. Database Setup and Management

MySQL Installation



```
ERROR 1046 (3D000): No database selected
mysql> exit
Bye
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo apt-get install mysql-server -y
[sudo] password for venagaci-vm:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mysql-server is already the newest version (8.0.36-0ubuntu0.22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 222 not upgraded.
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ mysql -V
mysql  Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo mysql -u root -p
ERROR 1049 (42000): Unknown database '-p'
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
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> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testing            |
| waph               |
+--------------------+
6 rows in set (0.02 sec)
```



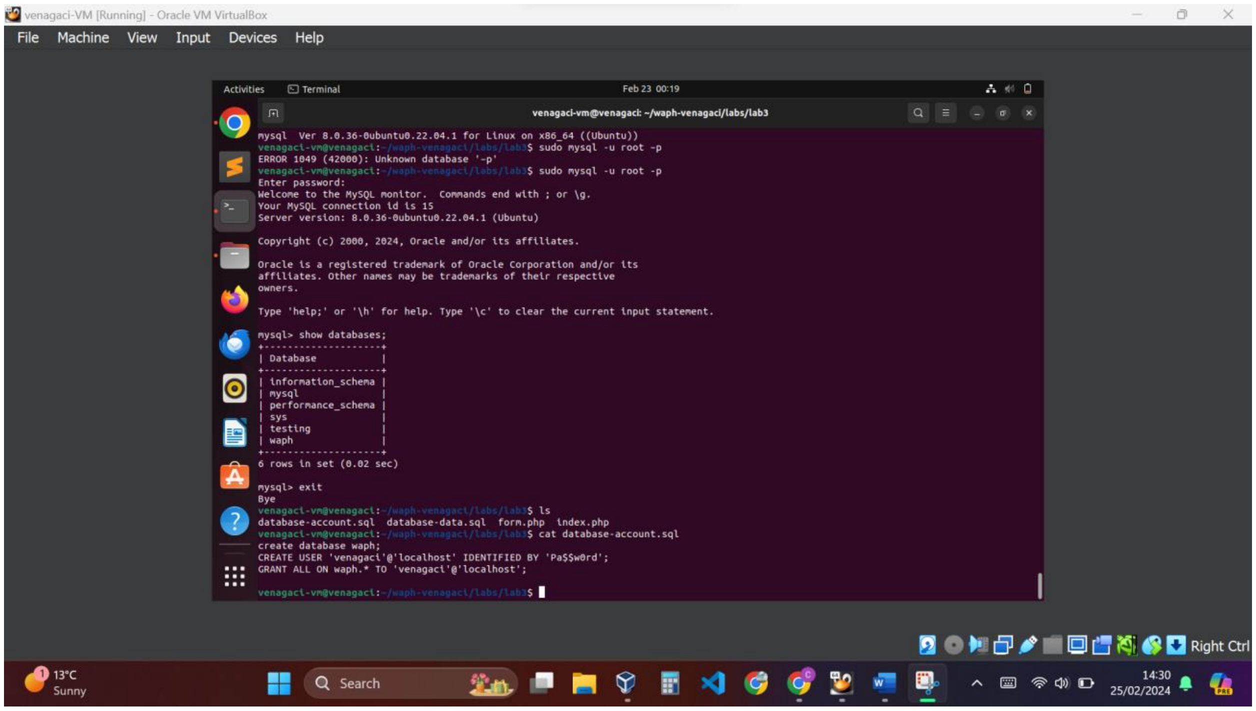
```
[sudo] password for venagaci-vm:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mysql-server is already the newest version (8.0.36-0ubuntu0.22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 222 not upgraded.
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ mysql -V
mysql  Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo mysql -u root -p
ERROR 1049 (42000): Unknown database '-p'
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
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> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testing            |
| waph               |
+--------------------+
6 rows in set (0.02 sec)

mysql> exit
Bye
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$
```

Summary: *The process begins with the installation of MySQL server using the command `sudo apt-get install mysql-server -y`, ensuring a smooth and automated installation process. Verification of the successful installation follows by executing `mysql -V`, confirming the version information of the installed*

*MySQL server. Finally, connection to the MySQL server is established using `sudo mysql -u root -p`, prompting for the root user's password and granting access to the server. This comprehensive process covers the installation, verification, and connection steps necessary for setting up and accessing MySQL server, ensuring a functional environment for database management and operations***.**

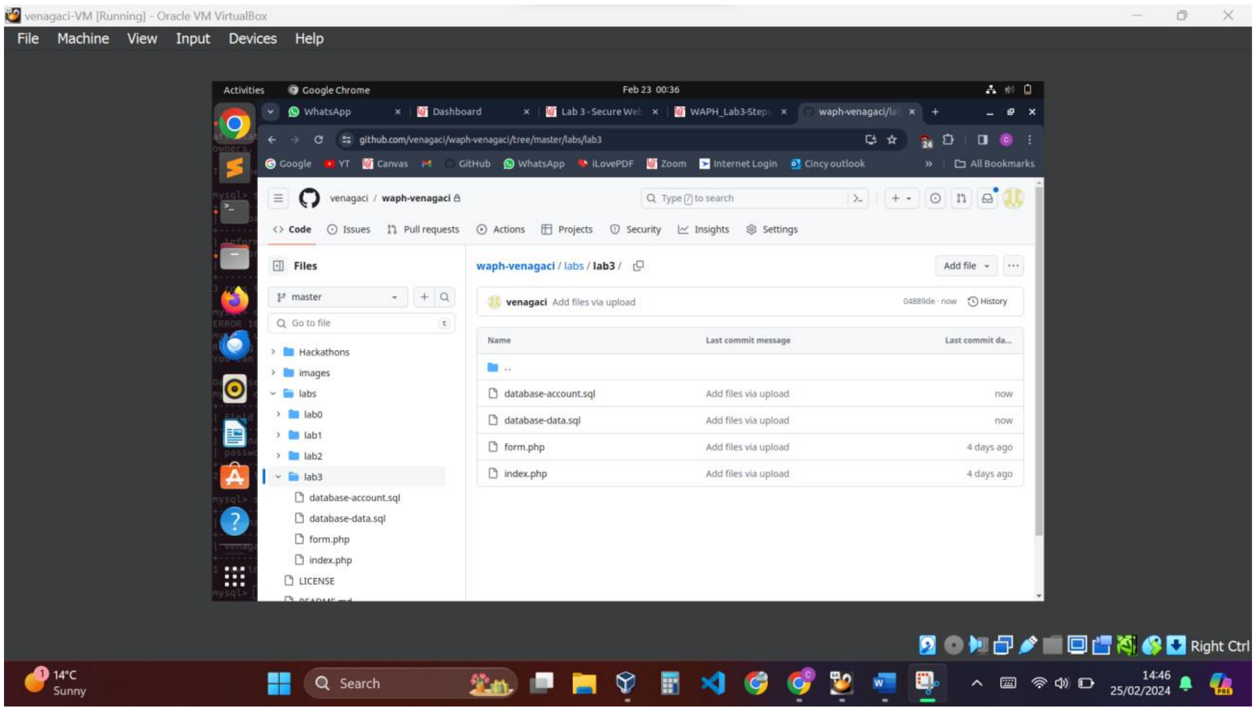
Create a New Database, Database User and Permission

The following screenshot has the content in the file database-account.sql


venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Feb 23 00:19
venagaci-vm@venagaci: ~/waph-venagaci/labs/lab3
Right Ctrl
13°C Sunny
Search
14:30 25/02/2024

```
mysql  Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo mysql -u root -p
ERROR 1049 (42000): Unknown database '-p'
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
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> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testing            |
| waph               |
+--------------------+
6 rows in set (0.02 sec)

mysql> exit
Bye
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ ls
database-account.sql  database-data.sql  form.php  index.php
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ cat database-account.sql
create database waph;
CREATE USER 'venagaci'@'localhost' IDENTIFIED BY 'Pa$$w0rd';
GRANT ALL ON waph.* TO 'venagaci'@'localhost';

venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ 
```

Summary of that code: *The `form.php` script executes essential MySQL commands for initializing a database environment. It creates a database named `waph` for storing application data, generates a user named `venagaci` with the password `'Pa$$w0rd'` to facilitate authentication, and grants full privileges within the `waph` database to this user. This sequence establishes the groundwork for database management, user authentication, and access control, providing a foundation for subsequent web application development tasks.*



github.com/venagaci/waph-venagaci/tree/master/labs/lab3

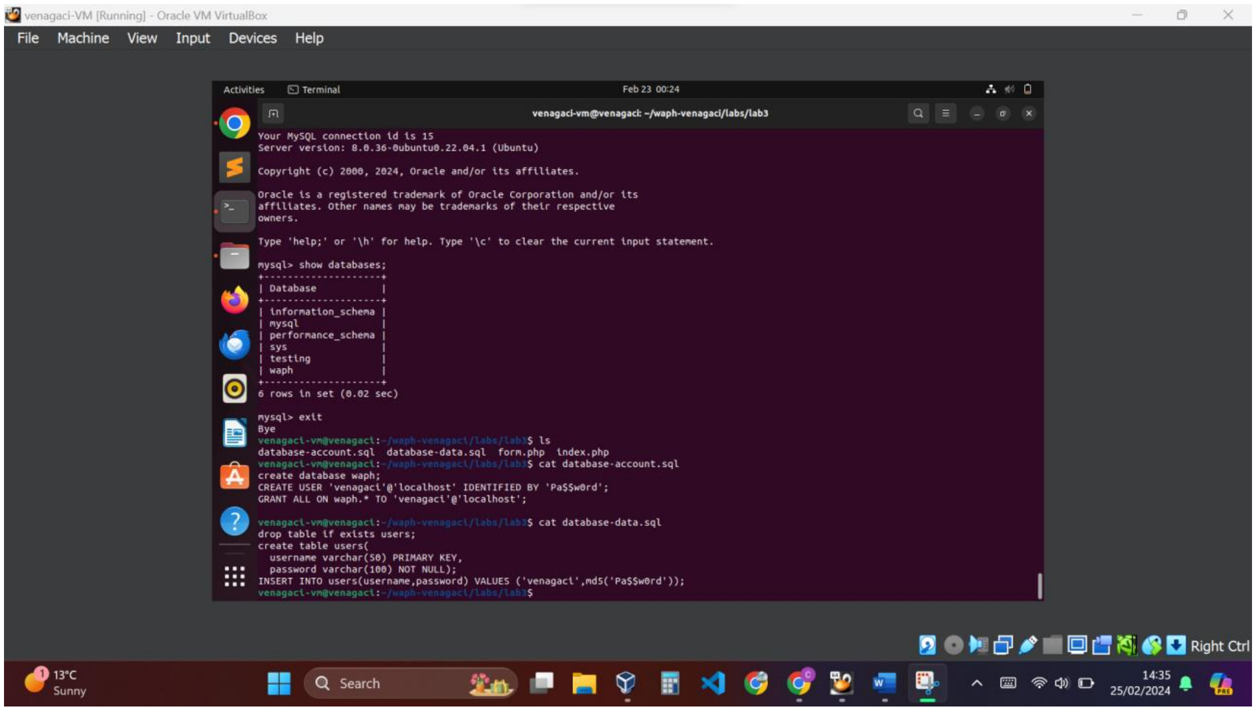
waph-venagaci / labs / lab3 /

venagaci Add files via upload 04889de · now History

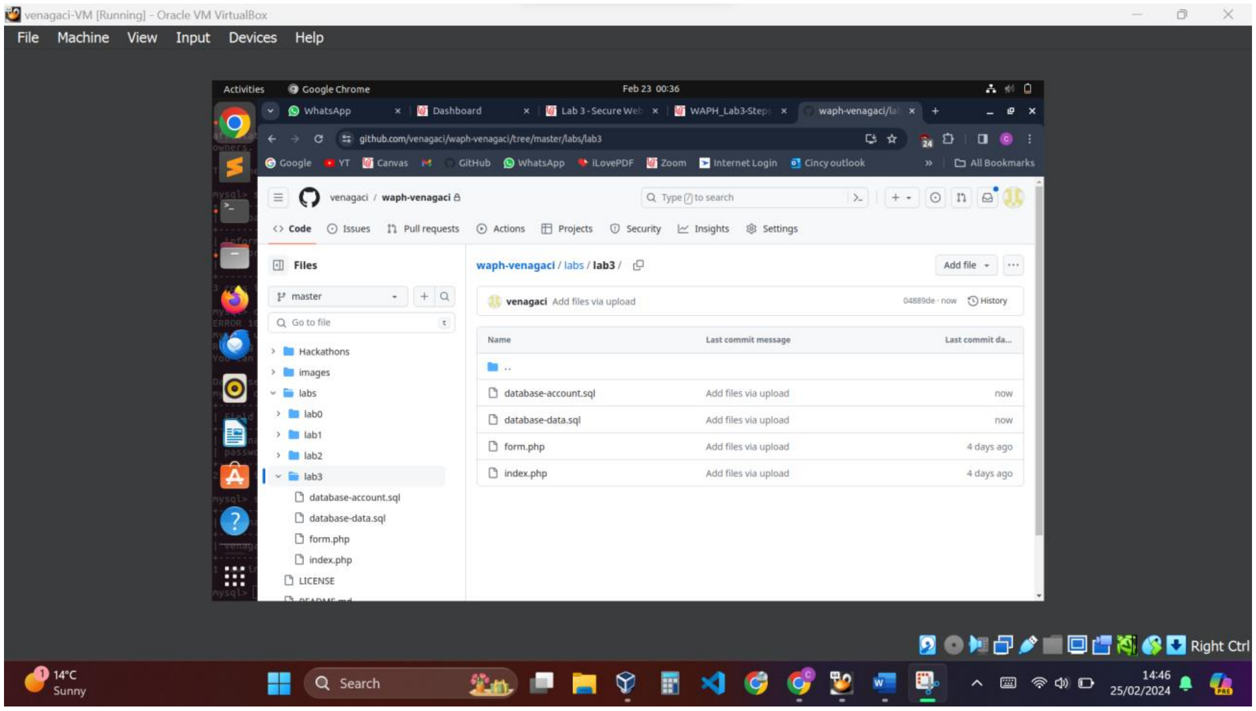
Name	Last commit message	Last commit da...
..		
database-account.sql	Add files via upload	now
database-data.sql	Add files via upload	now
form.php	Add files via upload	4 days ago
index.php	Add files via upload	4 days ago

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

The below screenshot has the content of the file database-data.sql.



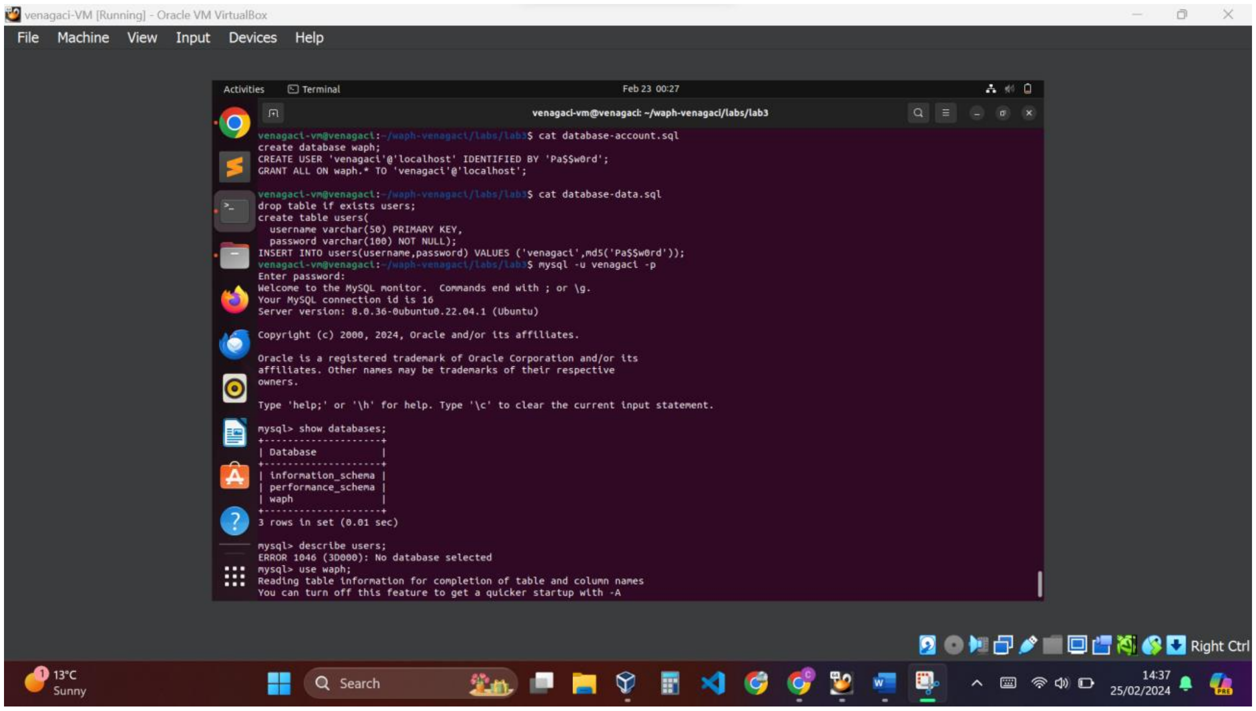
```
Your MySQL connection id is 15
Server version: 8.0.36-0ubuntu0.22.04.1 (Ubuntu)

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> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testing            |
| waph               |
+--------------------+
6 rows in set (0.02 sec)

mysql> exit
Bye
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ ls
database-account.sql  database-data.sql  form.php  index.php
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ cat database-account.sql
create database waph;
CREATE USER 'venagaci'@'localhost' IDENTIFIED BY 'Pa$$w0rd';
GRANT ALL ON waph.* TO 'venagaci'@'localhost';

venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ cat database-data.sql
drop table if exists users;
create table users(
  username varchar(50) PRIMARY KEY,
  password varchar(100) NOT NULL);
INSERT INTO users(username,password) VALUES ('venagaci',md5('Pa$$w0rd'));
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$
```

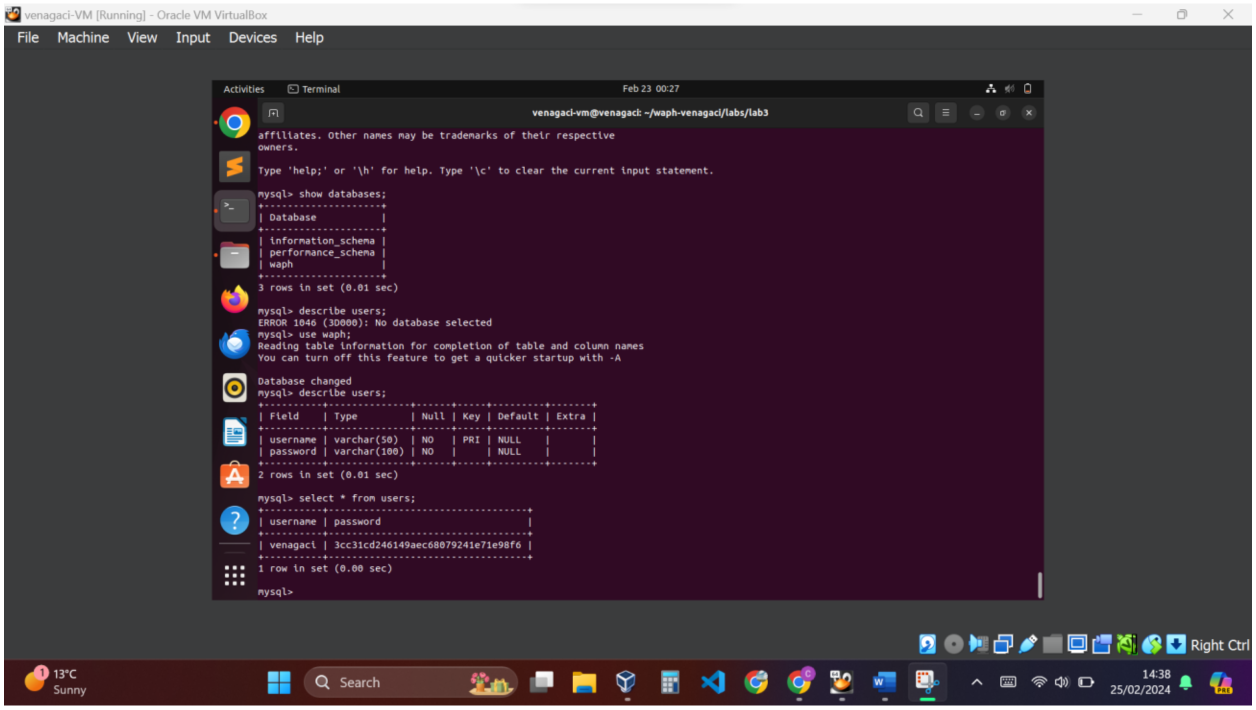

github.com/venagaci/waph-venagaci/tree/master/labs/lab3

waph-venagaci / labs / lab3 /

venagaci Add files via upload 04889de · now History

Name	Last commit message	Last commit da...
..		
database-account.sql	Add files via upload	now
database-data.sql	Add files via upload	now
form.php	Add files via upload	4 days ago
index.php	Add files via upload	4 days ago


```
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ cat database-account.sql
create database waph;
CREATE USER 'venagaci'@'localhost' IDENTIFIED BY 'Pa$$w0rd';
GRANT ALL ON waph.* TO 'venagaci'@'localhost';

venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ cat database-data.sql
drop table if exists users;
create table users(
  username varchar(50) PRIMARY KEY,
  password varchar(100) NOT NULL);
INSERT INTO users(username,password) VALUES ('venagaci',md5('Pa$$w0rd'));
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ mysql -u venagaci -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
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> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| waph               |
+--------------------+
3 rows in set (0.01 sec)

mysql> describe users;
ERROR 1046 (3D000): No database selected
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
```

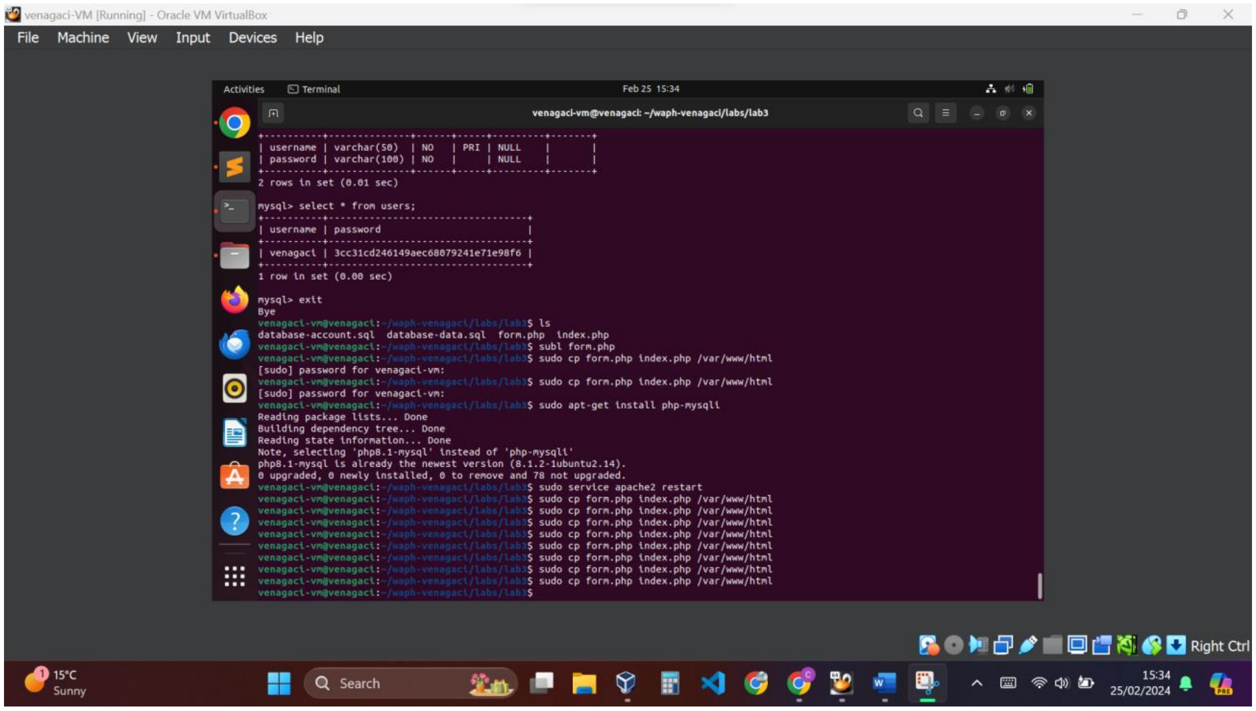

venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Feb 23 00:27
venagaci-vm@venagaci: ~/waph-venagaci/labs/lab3
Right Ctrl
13°C Sunny
Search
14:38 25/02/2024

```
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> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| waph               |
+--------------------+
3 rows in set (0.01 sec)

mysql> describe users;
ERROR 1046 (3D000): No database selected
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> describe users;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| username | varchar(50)  | NO   | PRI | NULL    |       |
| password | varchar(100) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> select * from users;
+----------+----------------------------------+
| username | password                         |
+----------+----------------------------------+
| venagaci | 3cc31cd246149aec68079241e71e98f6 |
+----------+----------------------------------+
1 row in set (0.00 sec)

mysql>
```

Summary of that code*: The process involves two key components: the `form.php` script and the `database-data.sql` file. Firstly, `form.php` orchestrates the initialization of the MySQL database environment by creating the `waph` database and generating a MySQL user named `venagaci` with the password `'Pa$$w0rd'`, along with granting full privileges within the `waph` database to this user. Secondly, `database-data.sql` complements this setup by defining the structure and initial data for the `users` table within the `waph` database. It ensures a clean slate by dropping the `users` table if it exists, creates the table with appropriate columns for username and password storage, and inserts an initial user record with the username 'venagaci' and the hashed password. Together, these components establish the foundational framework for MySQL database management, user authentication, and data storage, laying the groundwork for subsequent web application development tasks.*

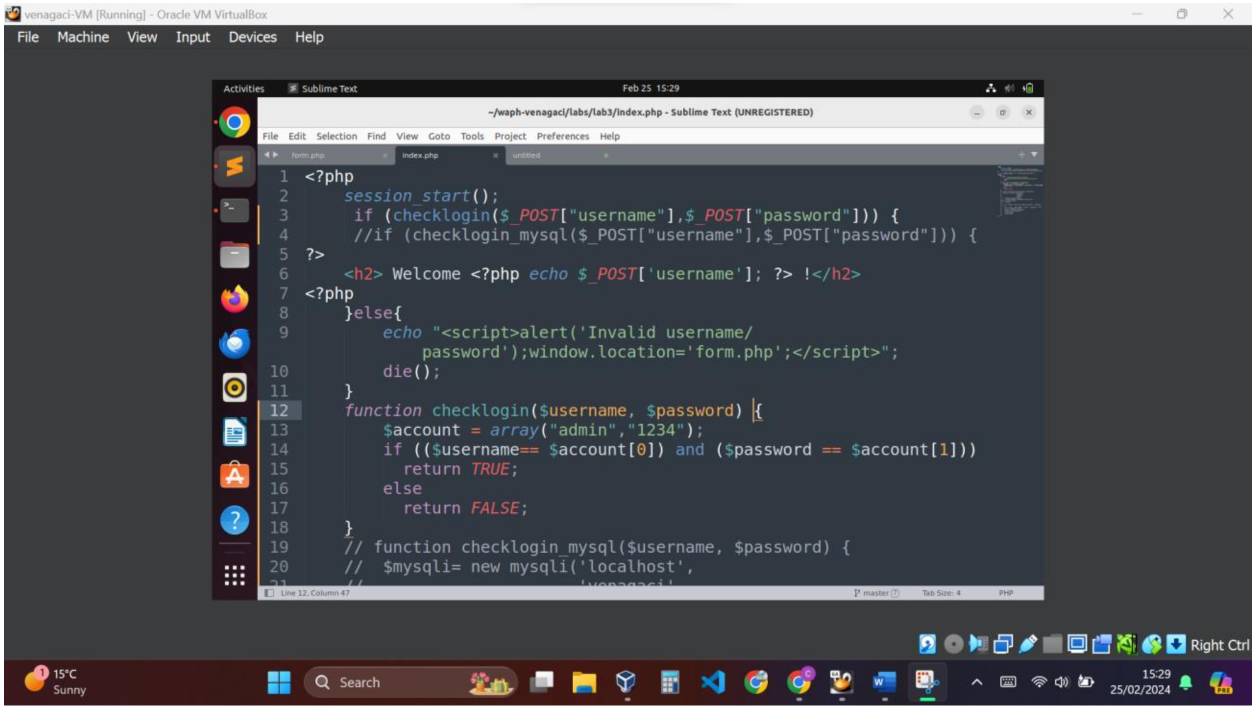
b. A Simple (Insecure) Login System with PHP/MySQL

Installation Screenshots:



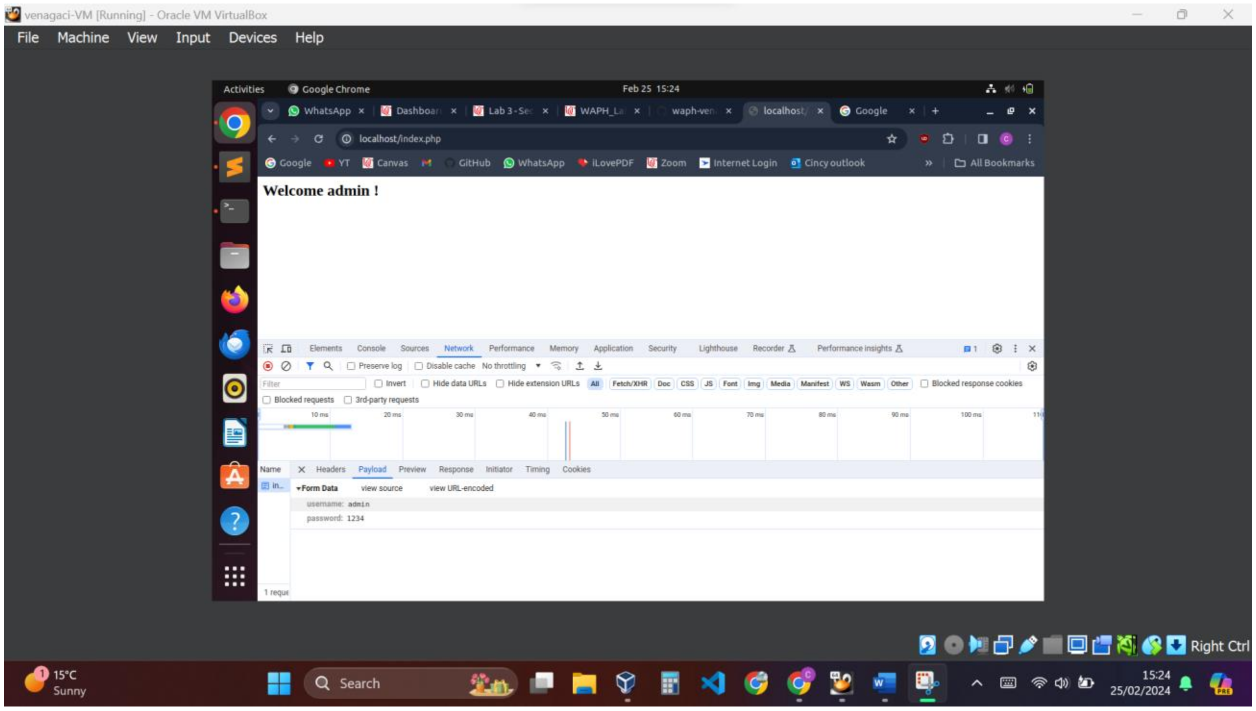
```
+----------+--------------+------+-----+---------+-------+
| username | varchar(50)  | NO   | PRI | NULL    |       |
| password | varchar(100) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> select * from users;
+----------+----------------------------------+
| username | password                         |
+----------+----------------------------------+
| venagaci | 3cc31cd246149aec68079241e71e98f6 |
+----------+----------------------------------+
1 row in set (0.00 sec)

mysql> exit
Bye
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ ls
database-account.sql  database-data.sql  form.php  index.php
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ subl form.php
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
[sudo] password for venagaci-vm:
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
[sudo] password for venagaci-vm:
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo apt-get install php-mysqli
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'php8.1-mysql' instead of 'php-mysqli'
php8.1-mysql is already the newest version (8.1.2-1ubuntu2.14).
0 upgraded, 0 newly installed, 0 to remove and 78 not upgraded.
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo service apache2 restart
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$ sudo cp form.php index.php /var/www/html
venagaci-vm@venagaci:~/waph-venagaci/labs/lab3$
```

Check login code in index.php file:



```
<?php
    session_start();
     if (checklogin($_POST["username"],$_POST["password"])) {
     //if (checklogin_mysql($_POST["username"],$_POST["password"])) {
?>
    <h2> Welcome <?php echo $_POST['username']; ?> !</h2>
<?php
    }else{
        echo "<script>alert('Invalid username/password');window.location='form.php';</script>";
        die();
    }
    function checklogin($username, $password) {
        $account = array("admin","1234");
        if (($username== $account[0]) and ($password == $account[1]))
          return TRUE;
        else
          return FALSE;
    }
    // function checklogin_mysql($username, $password) {
    //  $mysqli= new mysqli('localhost',
```

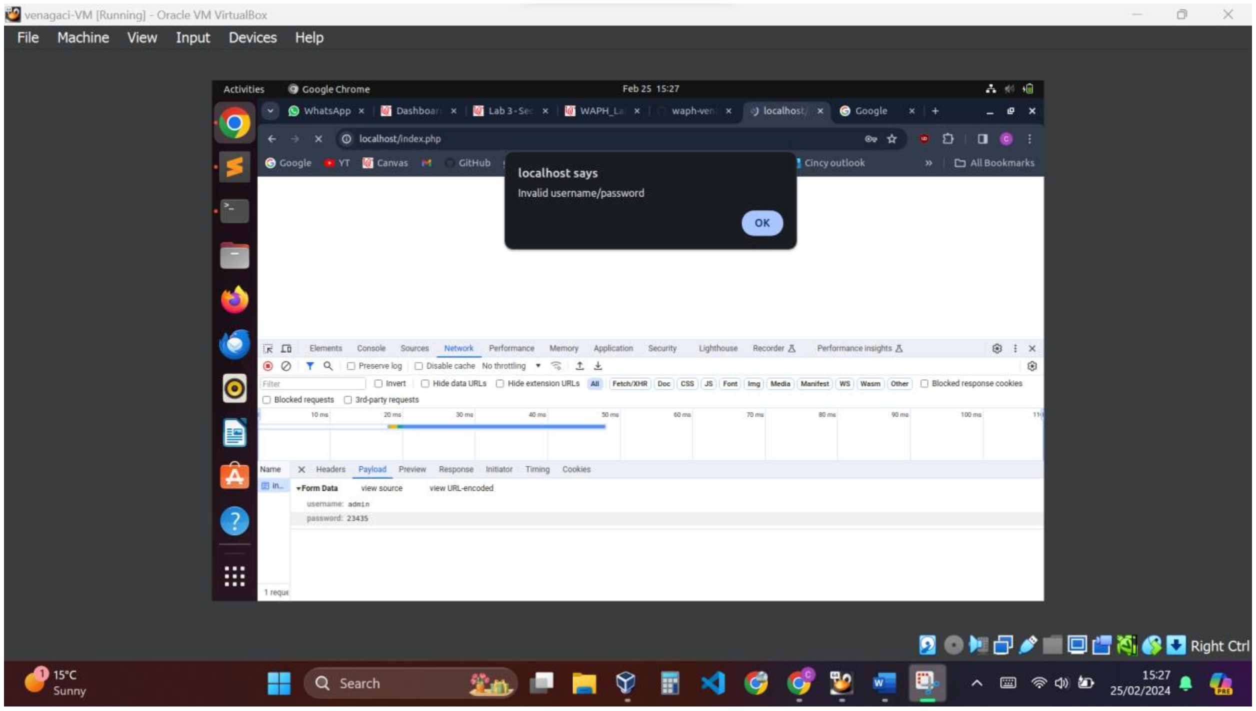
With Check login function – Correct password:


venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
localhost/index.php
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder
Name Headers Payload Preview Response Initiator Timing Cookies

Welcome admin !

```
Form Data    view source    view URL-encoded
username: admin
password: 1234
```

With Check login function – Wrong password:

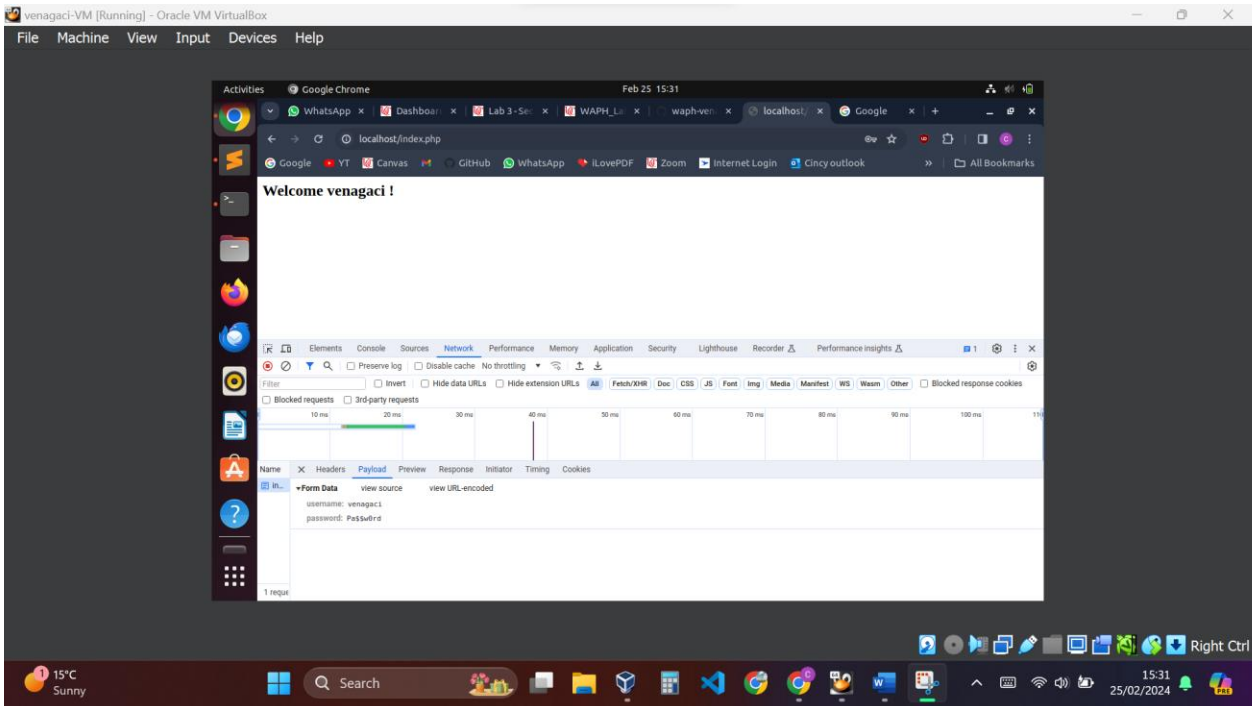

venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
localhost/index.php
OK
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder
Name Headers Payload Preview Response Initiator Timing Cookies

localhost says
Invalid username/password

```
Form Data    view source    view URL-encoded
username: admin
password: 23435
```

***Summary**: I have created an `index.php` file containing PHP code for a basic login system. This system utilizes a hardcoded array of credentials (`"admin"` and `"1234"`) for authentication. Upon login attempt, the script checks if the provided username and password match the values in the array. If the credentials are correct, it displays a welcome message; otherwise, it alerts the user about invalid credentials and redirects them back to the login page (`form.php`). I have deployed this `index.php` file alongside `form.php` onto a web server directory (`/var/www/html`) using the command `sudo cp form.php index.php /var/www/html`. Testing involved logging in with both correct and incorrect credentials to validate the login functionality. Implementing secure authentication methods and input validation would enhance the security and reliability of the login system.*

With checklogin_mysql – Correct Password:



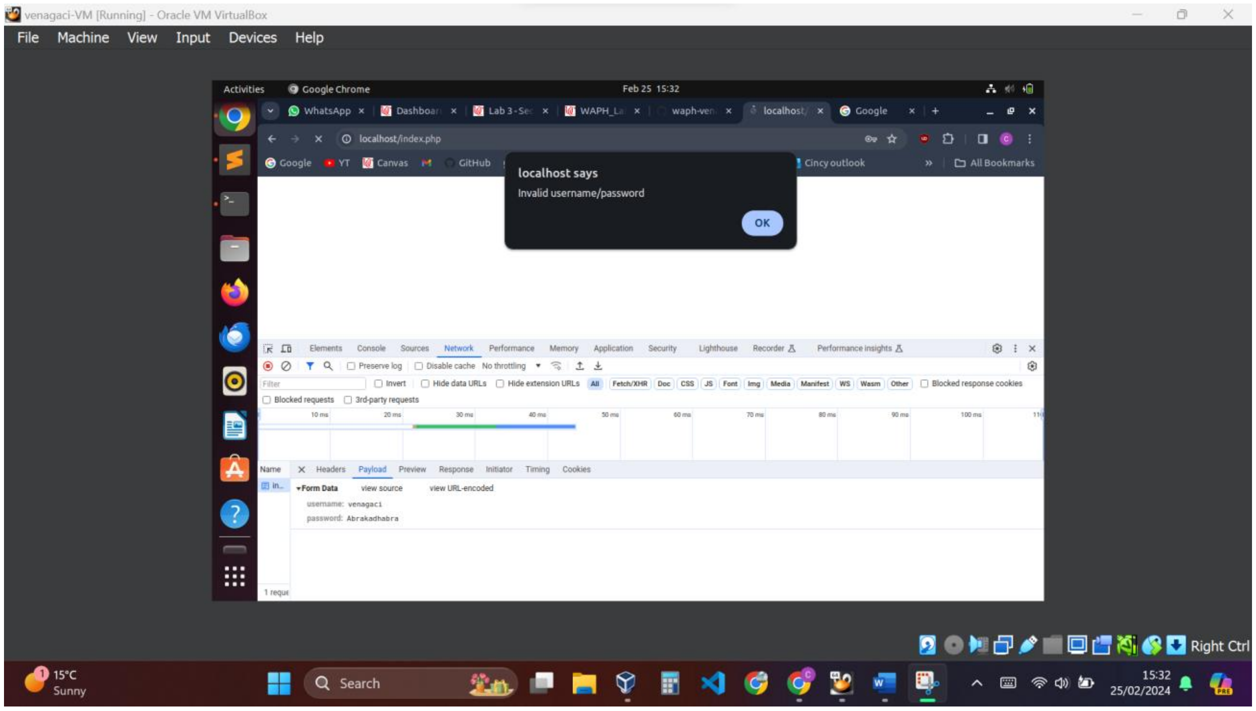
localhost/index.php

Welcome venagaci !

```
Form Data    view source    view URL-encoded
  username: venagaci
  password: Pa$$w0rd
```

venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Google Chrome
Feb 25 15:31
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights
Preserve log Disable cache No throttling
Name Headers Payload Preview Response Initiator Timing Cookies
15°C Sunny
Search
15:31 25/02/2024
Right Ctrl

With checklogin_mysql – Wrong Password:



localhost/index.php

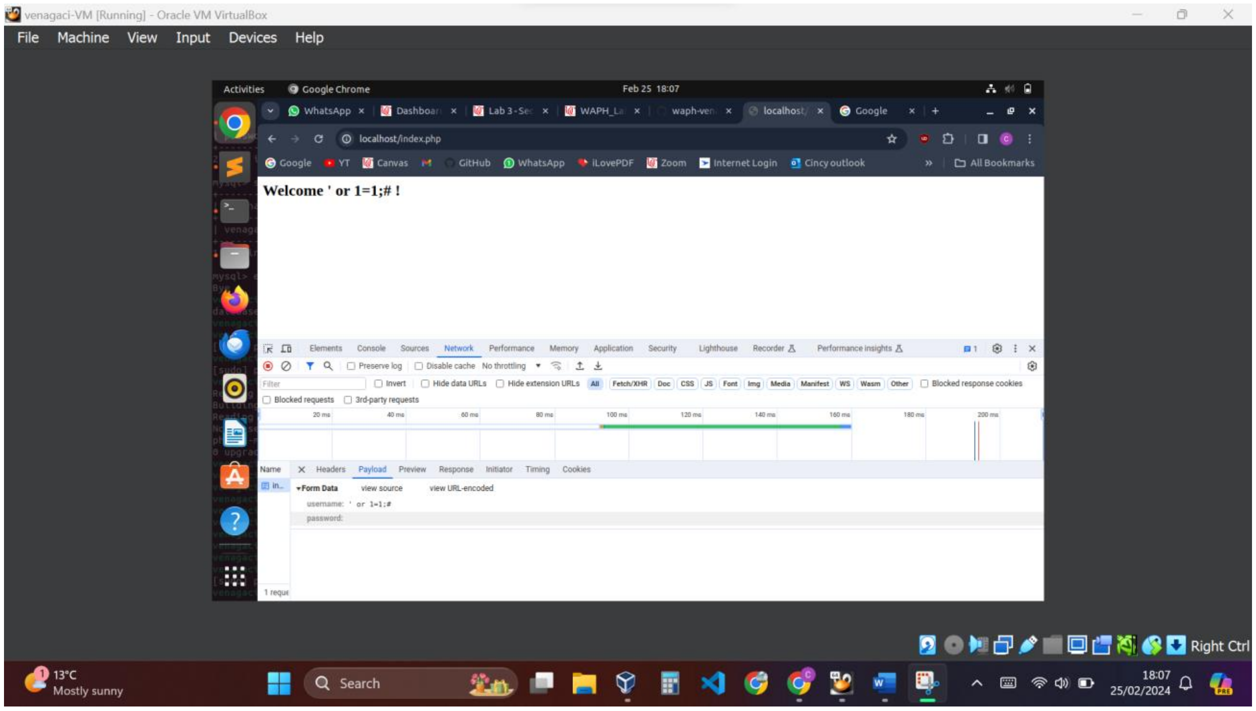
localhost says
Invalid username/password

```
Form Data
username: venagaci
password: Abrakadhabra
```

Summary: *The updated index.php script starts a session and checks user credentials using the checklogin_mysql() function. If the provided username and password match an entry in the users table of the MySQL database, it displays a welcome message with the username. Otherwise, it triggers an alert indicating invalid credentials using JavaScript and redirects the user back to the form.php page. The checklogin_mysql() function establishes a connection to the MySQL database using provided credentials, then constructs and executes a SQL query to select a user with the given username and password hash. If a matching user is found in the database, the function returns true, indicating successful authentication, otherwise, it returns false.The deployment process involves copying both form.php and index.php to the web server directory (/var/www/html) using the sudo cp command. Upon logging in with correct credentials, the user is welcomed with a personalized message, whereas incorrect credentials trigger an alert and redirect.*

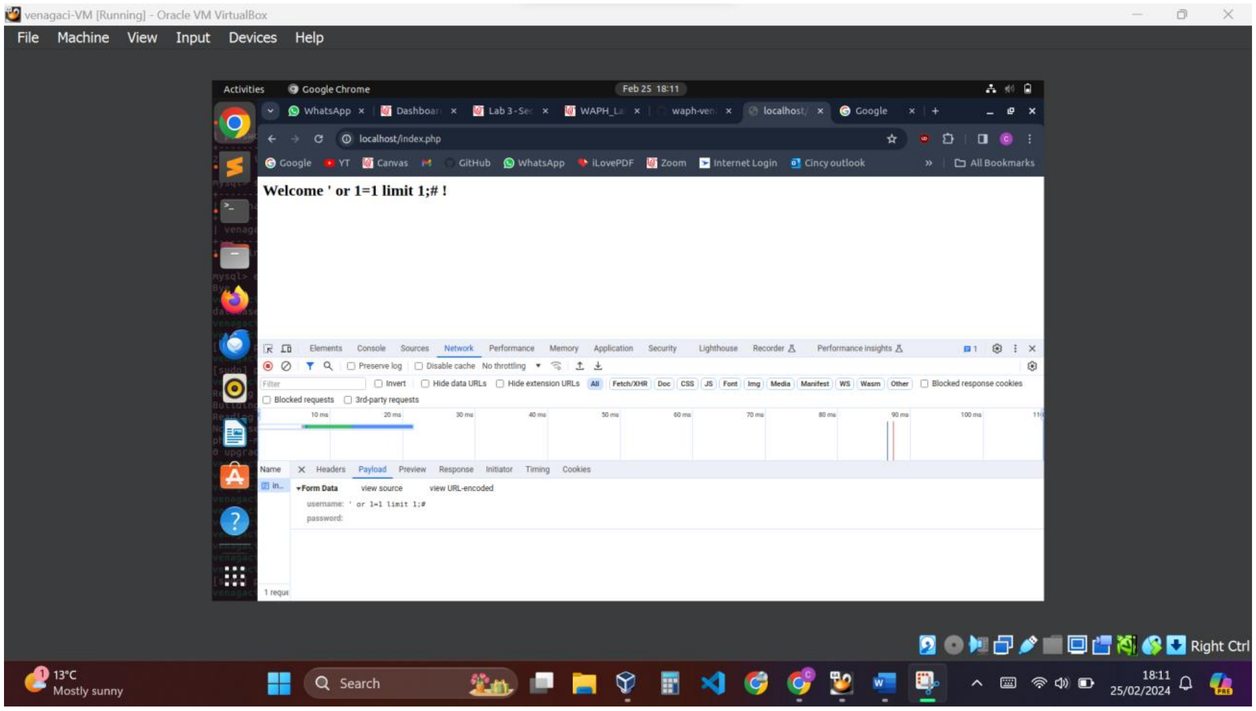
c. Performing XSS and SQL Injection Attacks

SQL Injection Attack:


venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Google Chrome Feb 25 18:07
localhost/index.php
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights
Name Headers Payload Preview Response Initiator Timing Cookies
13°C Mostly sunny
Search
18:07 25/02/2024

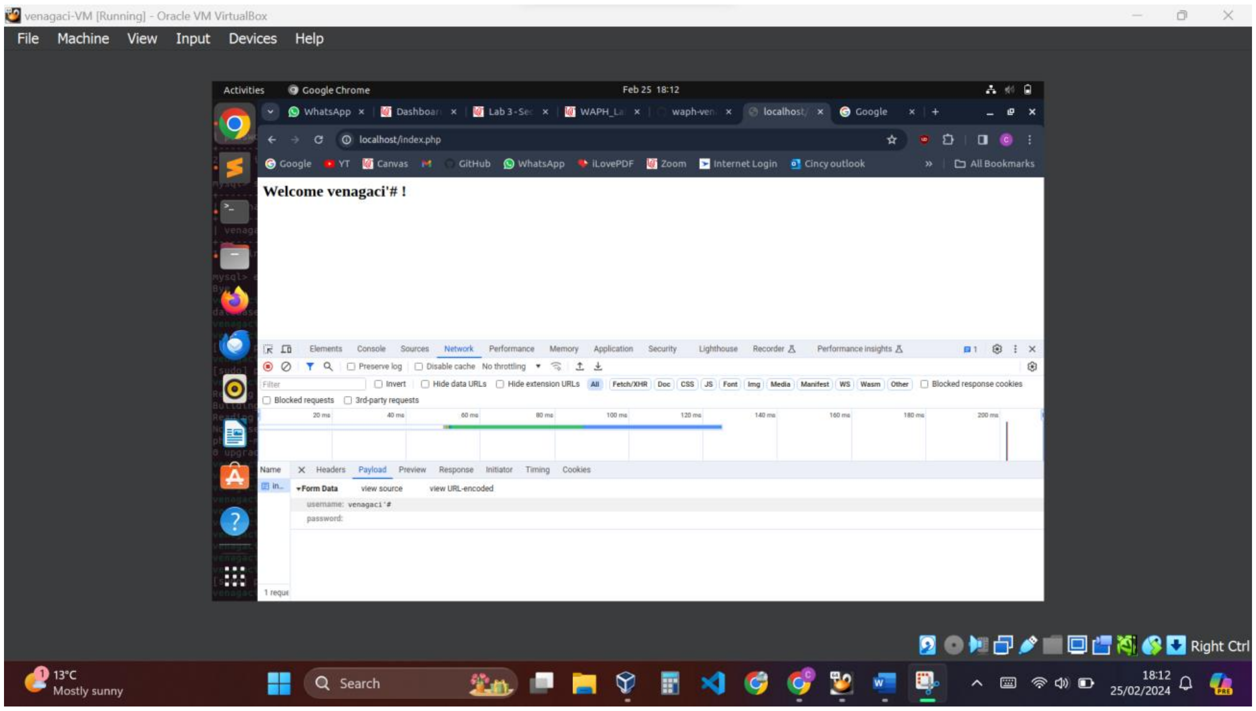
Welcome ' or 1=1;# !

```
▾Form Data    view source    view URL-encoded
  username: ' or 1=1;#
  password:
```


venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Google Chrome Feb 25 18:11
localhost/index.php
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights
Name Headers Payload Preview Response Initiator Timing Cookies
13°C Mostly sunny
Search
18:11 25/02/2024

Welcome ' or 1=1 limit 1;# !

```
▾Form Data    view source    view URL-encoded
  username: ' or 1=1 limit 1;#
  password:
```



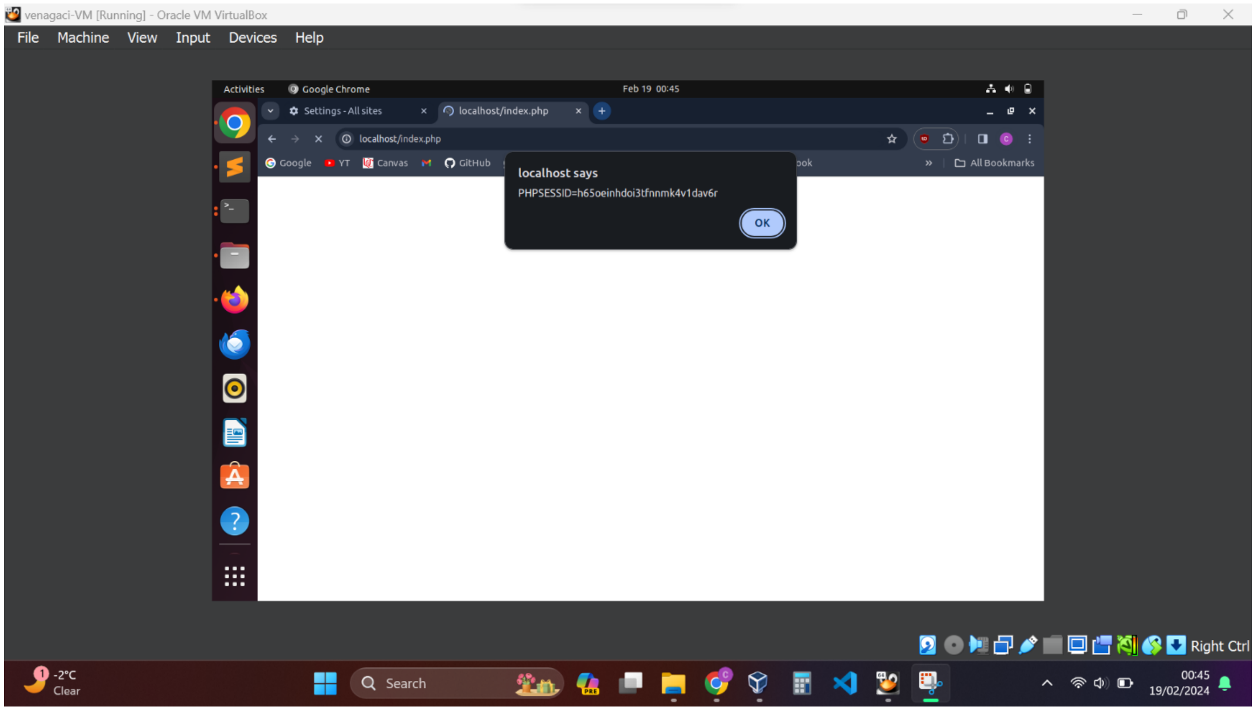
Welcome venagaci'# !

```
username: venagaci'#
password:
```

Explain why such attacks happen?

Solution: *Such attacks, including SQL Injection and Cross-Site Scripting (XSS), often occur due to vulnerabilities in web applications, which malicious actors exploit to gain unauthorized access or execute harmful scripts. SQL Injection attacks occur when user input is not properly sanitized before being included in SQL queries. In this code, the checklogin_mysql() function constructs a SQL query using user-provided input ($username and $password) without proper validation or parameterization. Attackers can manipulate input fields to inject SQL code into the query, potentially bypassing authentication checks or accessing sensitive data. For example, an attacker might input ' OR '1'='1' as the username and any password, effectively by passing the authentication check because the injected SQL always evaluates to true.*

Cross-site Scripting:

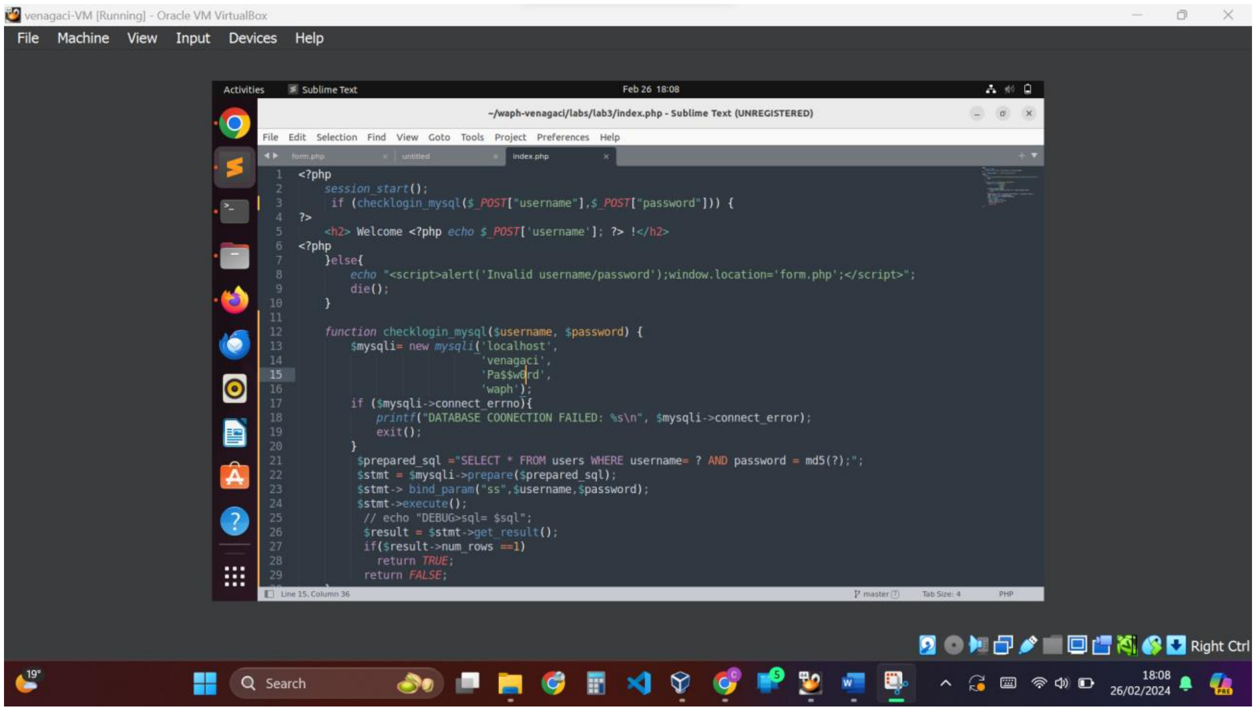

localhost says
PHPSESSID=h65oeinhdoi3tfnnmk4v1dav6r

Discuss the vulnerability in your report:

The vulnerability in the provided code stems from the lack of input validation and sanitization. Specifically, the `checklogin_mysql()` function concatenates user-provided input directly into SQL queries, making the code susceptible to SQL Injection attacks where attackers can manipulate input to execute arbitrary SQL commands. Additionally, if user input is echoed back to the browser without proper escaping, it could lead to Cross-Site Scripting (XSS) vulnerabilities, allowing attackers to inject and execute malicious JavaScript code in users' browsers. These vulnerabilities pose significant security risks, highlighting the importance of implementing robust input validation, parameterized queries, and output sanitization to mitigate potential threats and ensure the security of the web application.

d. Prepared Statement Implementation:

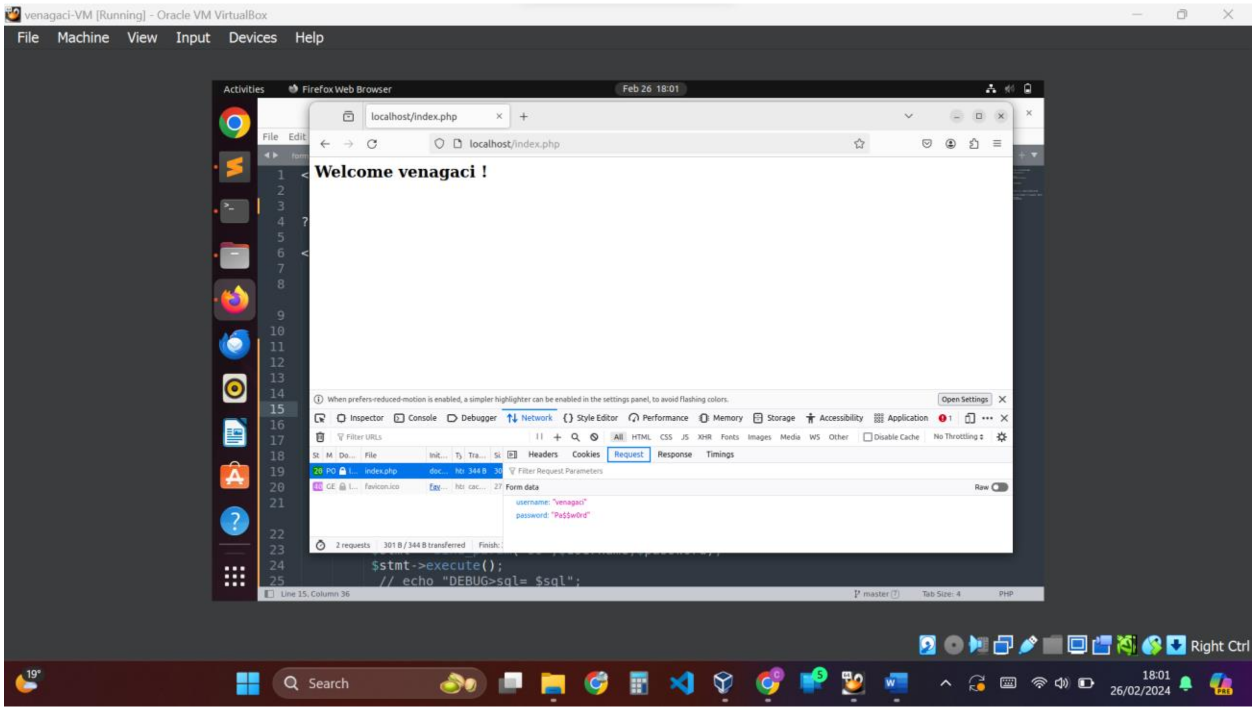
Screenshot of the new PHP code:



~/waph-venagaci/labs/lab3/index.php - Sublime Text (UNREGISTERED)

```php
<?php
    session_start();
     if (checklogin_mysql($_POST["username"],$_POST["password"])) {
?>
    <h2> Welcome <?php echo $_POST['username']; ?> !</h2>
<?php
    }else{
        echo "<script>alert('Invalid username/password');window.location='form.php';</script>";
        die();
    }

    function checklogin_mysql($username, $password) {
        $mysqli= new mysqli('localhost',
                            'venagaci',
                            'Pa$$w0rd',
                            'waph');
        if ($mysqli->connect_errno){
            printf("DATABASE COONECTION FAILED: %s\n", $mysqli->connect_error);
            exit();
        }
         $prepared_sql ="SELECT * FROM users WHERE username= ? AND password = md5(?);";
         $stmt = $mysqli->prepare($prepared_sql);
         $stmt-> bind_param("ss",$username,$password);
         $stmt->execute();
          // echo "DEBUG>sql= $sql";
          $result = $stmt->get_result();
          if($result->num_rows ==1)
            return TRUE;
          return FALSE;
```

Login with correct password:

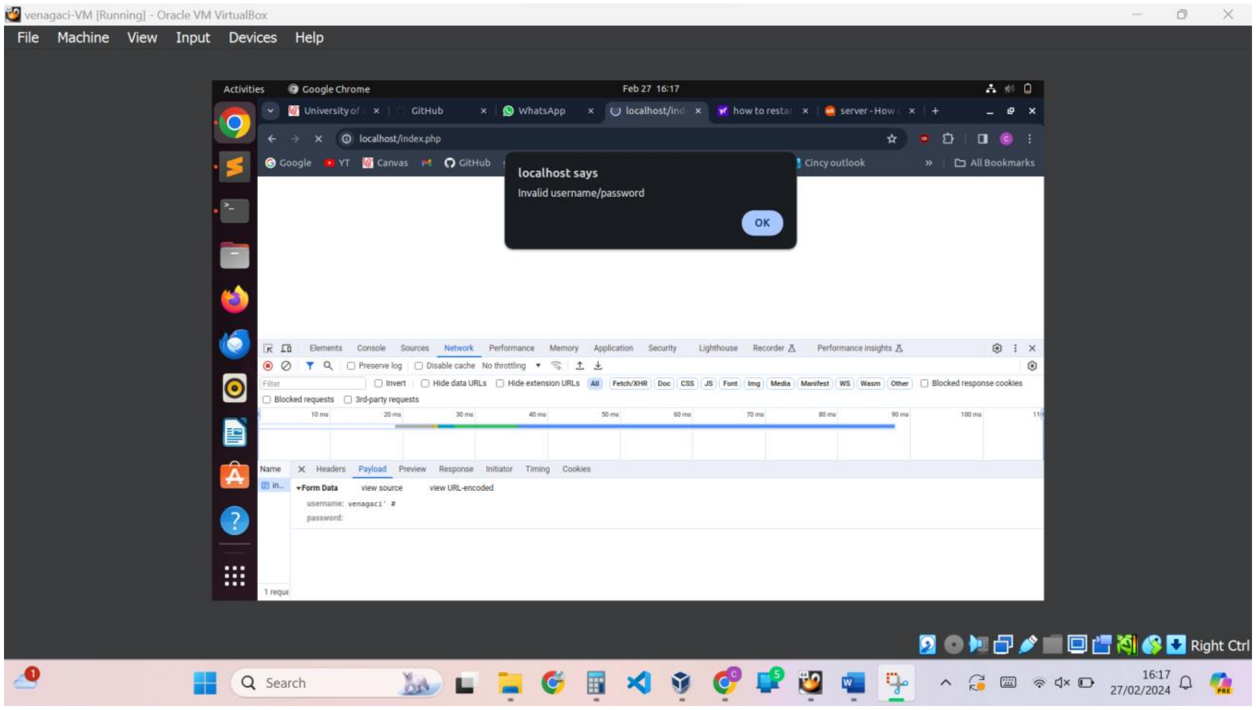


localhost/index.php

Welcome venagaci !

```
Form data
username: "venagaci"
password: "Pa$$w0rd"
```

No SQL Injection attacks can be performed:

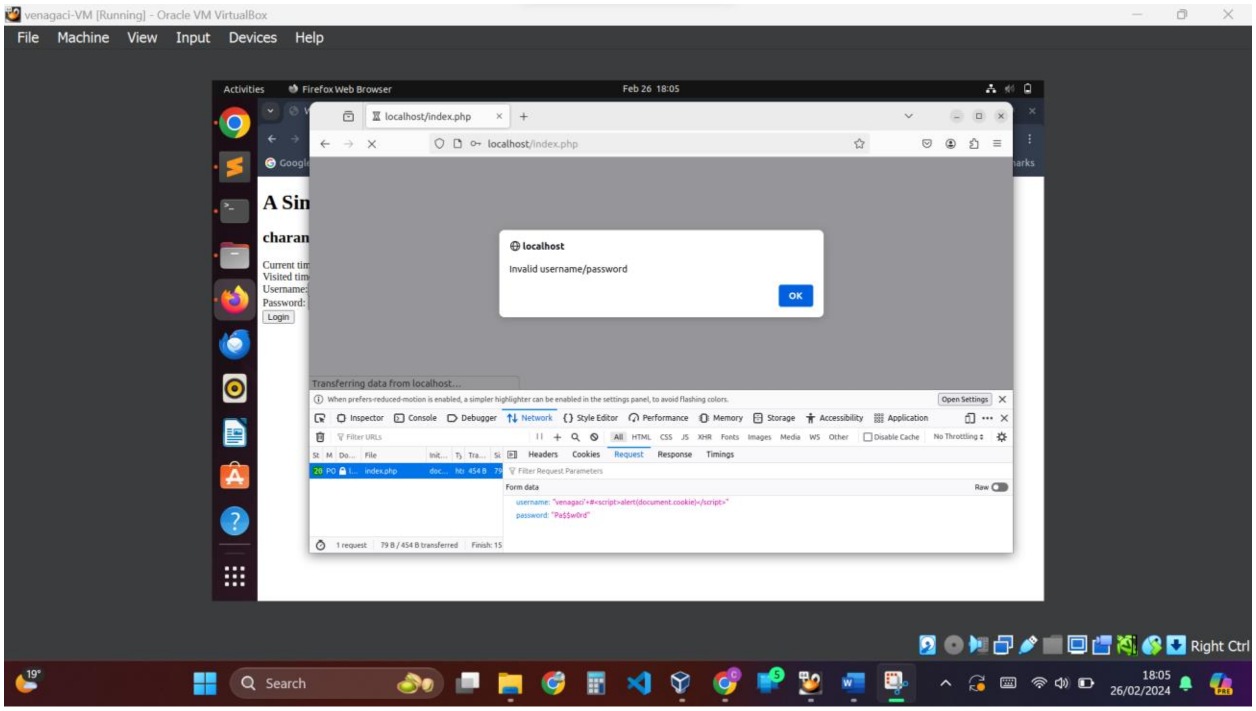

venagaci-VM [Running] - Oracle VM VirtualBox

localhost/index.php

```
localhost says
Invalid username/password
```

```
Form Data
  username: venagaci' #
  password:
```

No Cross-scripting attacks can be performed:

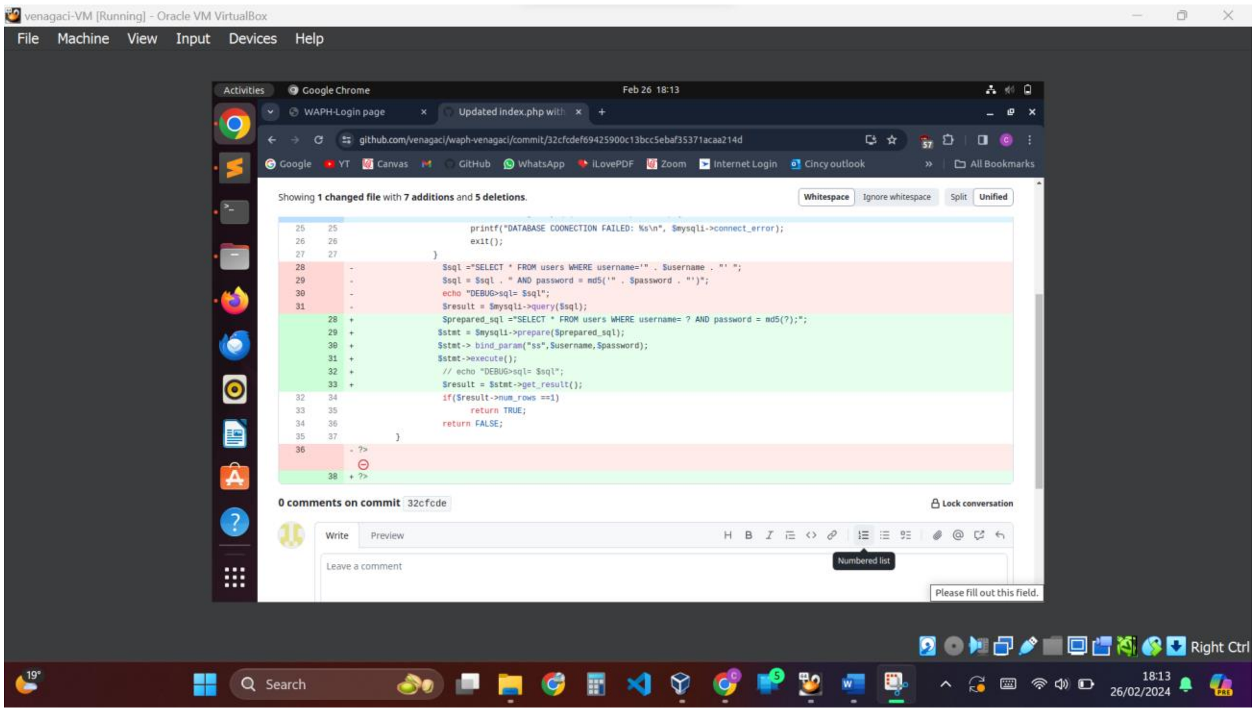

venagaci-VM [Running] - Oracle VM VirtualBox

localhost/index.php

```
localhost
Invalid username/password
```

```
Form data
  username: "venagaci'+#<script>alert(document.cookie)</script>"
  password: "Pa$$w0rd"
```

Updated index.php with Prepared statements in github repository:


venagaci-VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Google Chrome
Feb 26 18:13
WAPH-Login page
Updated index.php with
github.com/venagaci/waph-venagaci/commit/32cfcdef69425900c13bcc5ebaf35371acaa214d
Google YT Canvas GitHub WhatsApp iLovePDF Zoom Internet Login Cincy outlook All Bookmarks
Whitespace Ignore whitespace Split Unified
Write Preview
Leave a comment
Numbered list
Please fill out this field.
Lock conversation
Search
Right Ctrl
18:13
26/02/2024

Showing **1 changed file** with **7 additions** and **5 deletions**.

```
                         printf("DATABASE COONECTION FAILED: %s\n", $mysqli->connect_error);
                         exit();
                 }
-                   $sql ="SELECT * FROM users WHERE username='" . $username . "' ";
-                   $sql = $sql . " AND password = md5('" . $password . "')";
-                   echo "DEBUG>sql= $sql";
-                   $result = $mysqli->query($sql);
+                   $prepared_sql ="SELECT * FROM users WHERE username= ? AND password = md5(?);";
+                  $stmt = $mysqli->prepare($prepared_sql);
+                  $stmt-> bind_param("ss",$username,$password);
+                  $stmt->execute();
+                   // echo "DEBUG>sql= $sql";
+                   $result = $stmt->get_result();
                    if($result->num_rows ==1)
                         return TRUE;
                    return FALSE;
             }
- ?>
+ ?>
```

0 comments on commit 32cfcde

Summary: *The updated `index.php` script incorporates significant improvements to mitigate SQL Injection vulnerabilities. It begins by starting a session and then calls the `checklogin_mysql()` function to authenticate users. This function establishes a connection to the MySQL database using the `mysqli` object and utilizes prepared statements with parameterized queries to execute SQL commands. The use of prepared statements with bound parameters ensures that user input is treated as data rather than executable SQL code, effectively preventing SQL Injection attacks. When a user attempts to log in, the function queries the database to verify the provided username and password. If a matching user is found, the user is welcomed with a personalized message; otherwise, an alert is triggered, and the user is redirected to the login page. The deployment process involves copying both `form.php` and `index.php` to the web server directory (`/var/www/html`). Despite attempts to execute SQL Injection attacks, they fail due to the use of prepared statements, which treat user input as data rather than executable SQL code, effectively neutralizing the attack vector. This demonstrates the effectiveness of implementing secure coding practices, such as using prepared statements and parameterized queries, to bolster the security of web applications and mitigate common vulnerabilities like SQL Injection.*

Security Analysis:

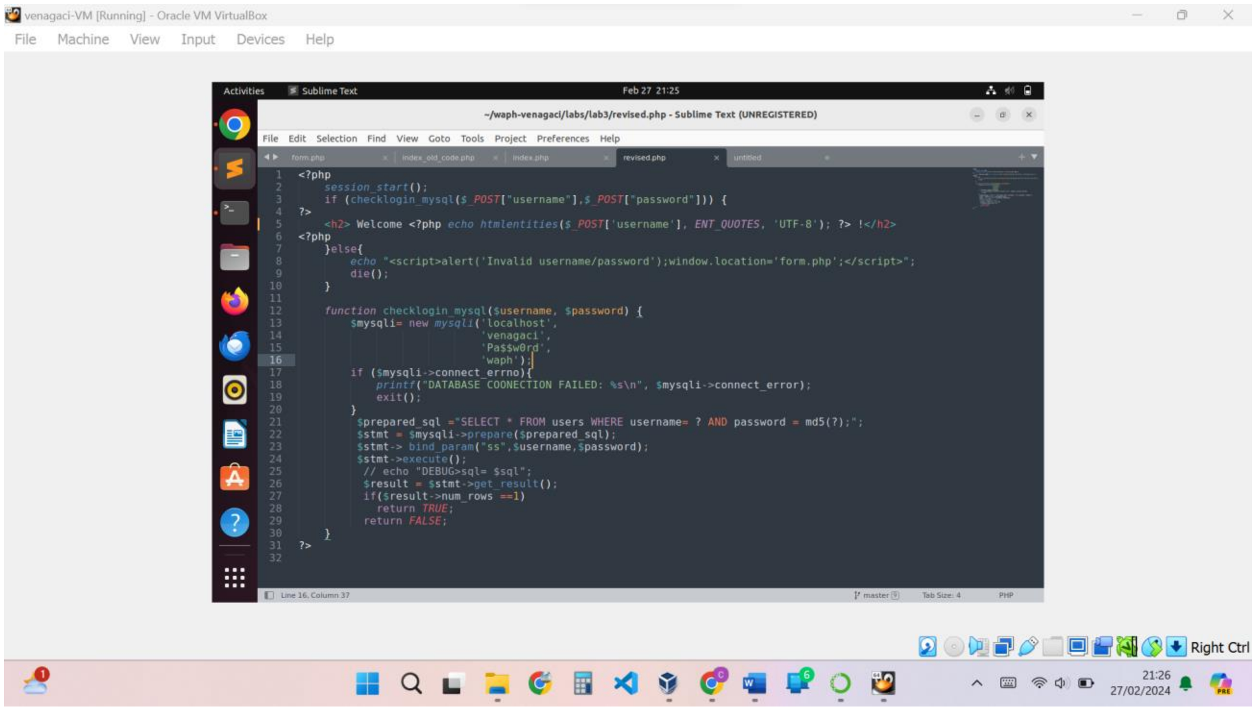
Prepared Statement Explanation: **Discuss why prepared statements can prevent SQL injection attacks (2.5 pts)**

Solution: Prepared statements have a parameterization feature that isolates SQL logic from user input, they provide a strong defense against SQL injection attacks. Pre-compiled SQL statements guarantee that user input is handled simply as data and not as a component of the SQL command structure by pre-compiling queries with placeholders for parameters. As a result, the arguments are tied to the placeholders during query execution, eliminating the chance of injected SQL code being executed. Furthermore, prepared statements neutralize attacker efforts to insert malicious SQL code into the query by automatically escaping special characters like quotes and semicolons in the arguments. The likelihood of SQL injection vulnerabilities is greatly decreased by this automatic escaping and sanitization process, improving the overall security posture of web applications.

Prepared statements are not limited to automatic escape and parameterization; they also have other advantages. By merely compiling and optimizing the query once, they encourage the reuse of execution plans by using the same plan for subsequent executions. In addition to lowering query compilation and execution overhead, this optimization stops attackers from using control characters to manipulate query execution. Furthermore, prepared statements guarantee that only legitimate data values are allowed by enforcing data types for parameters. Potential SQL injection attacks are effectively prevented by raising a type mismatch error in the event that data of an incompatible type is attempted to be passed. All things considered, using prepared statements strengthens web applications' security posture considerably by offering a thorough defense against SQL injection issues.

Implement Sanitization: Enhance the code to sanitize outputs, mitigating XSS risks. Provide the revised code in the report with an explanation (2 pts)

Here is the Screenshot of revised code:



```php
<?php
    session_start();
    if (checklogin_mysql($_POST["username"],$_POST["password"])) {
?>
    <h2> Welcome <?php echo htmlentities($_POST['username'], ENT_QUOTES, 'UTF-8'); ?> !</h2>
<?php
    }else{
        echo "<script>alert('Invalid username/password');window.location='form.php';</script>";
        die();
    }

    function checklogin_mysql($username, $password) {
        $mysqli= new mysqli('localhost',
                            'venagaci',
                            'Pa$$w0rd',
                            'waph');
        if ($mysqli->connect_errno){
            printf("DATABASE COONECTION FAILED: %s\n", $mysqli->connect_error);
            exit();
        }
         $prepared_sql ="SELECT * FROM users WHERE username= ? AND password = md5(?);";
         $stmt = $mysqli->prepare($prepared_sql);
         $stmt-> bind_param("ss",$username,$password);
         $stmt->execute();
          // echo "DEBUG>sql= $sql";
          $result = $stmt->get_result();
          if($result->num_rows ==1)
            return TRUE;
          return FALSE;
    }
?>
```

Explanation of this revised code:

To enhance the prepared statements code to sanitize outputs and mitigate XSS risks, I can use PHP's htmlentities() function to encode special characters before echoing any user-provided data back to the browser. This function converts special characters such as <, >, ", ', and & to their corresponding HTML entities, preventing them from being interpreted as HTML or JavaScript code.

This code uses session_start() to start a session when the user submits their login credentials. This allows for persistent data storage during the user's interaction with the application. Next, using the checklogin_mysql() function, which safely connects to a MySQL database using the mysqli object, it confirms the username and password entered. This code makes sure that user input is rigorously treated as data by using prepared statements with restricted parameters. This prevents SQL injection issues by keeping potentially dangerous input apart from SQL logic. In order to reduce the possibility of cross-site scripting (XSS), the username is cleaned up using htmlentities() and the user is presented with a welcome message if authentication is successful. On the other hand, in this case that authentication is unsuccessful, a JavaScript alert is set off, informing the user that their credentials are invalid and rerouting them to the form.php login page, thus improving security and user experience.

This method uses numerous layers of security, which considerably reduces susceptibility. By encoding special characters before sending them to the webpage, htmlentities() sanitizes user input and ensures that data provided by the user is handled as plain text rather than executable code, hence preventing potential XSS attacks. Additionally, by prohibiting attackers from inserting malicious SQL code into the query, the usage of prepared statements with bound parameters in database queries mitigates SQL injection vulnerabilities. Together, these security features and error handling, which gives users insightful feedback, strengthen the login mechanism's resilience and robustness, protecting the web application from typical security threats and enhancing user confidence in its security posture.

Discussions (3pts): **Are there any programming flaws/vulnerabilities in the current code? For example, what if the username/password are empty? , what if there are any database errors?; what if the provided username is not exactly the same as the username from the database.**

Yes, there are several programming flaws and potential vulnerabilities in the provided code:

Empty Username or Password: When the username or password fields are left empty, the code does not know what to do. The checklogin_mysql() method will still run if either field is left empty, which could result in unexpected behavior or mistakes in the database query.

Database Errors: Although the code handles faults for the database connection ($mysqli->connect_errno), it does not address issues that can arise when the prepared statement is executed or when the query result is retrieved. Because there is no error handling in place, the user may see unhandled exceptions or errors, which could reveal private information about the configuration or structure of the database.

Mismatched Username: The query will return zero rows and invalid usernames, signifying an unsuccessful attempt at login, if the username supplied does not precisely match a username that is maintained in the database. Nevertheless, the code does not distinguish between a login failure brought on by an invalid username and a login failure brought on by an incorrect password. This lack of difference may cause consumers to become confused and may provide attackers access to more data for enumeration or brute-force operations.

This code will differentiate between failed login attempts caused by incorrect usernames and passwords to give users more clear feedback and lower the risk of information disclosure. It will also implement proper input validation to ensure that the username and password fields are not empty. These changes will address flaws and vulnerabilities in the database and make it easier to handle errors by implementing error handling for the prepared statement execution and result retrieval. The code might also benefit from additional security features like rate limitation to lessen brute-force attacks and the use of more robust password hashing algorithms rather than just MD5.