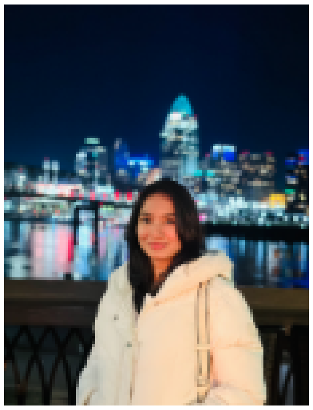
WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Srujana Reddy

Email: vadagasy@mail.uc.edu



Lab 3 - Secure Web Application Development in PHP/MySQL

Overview :

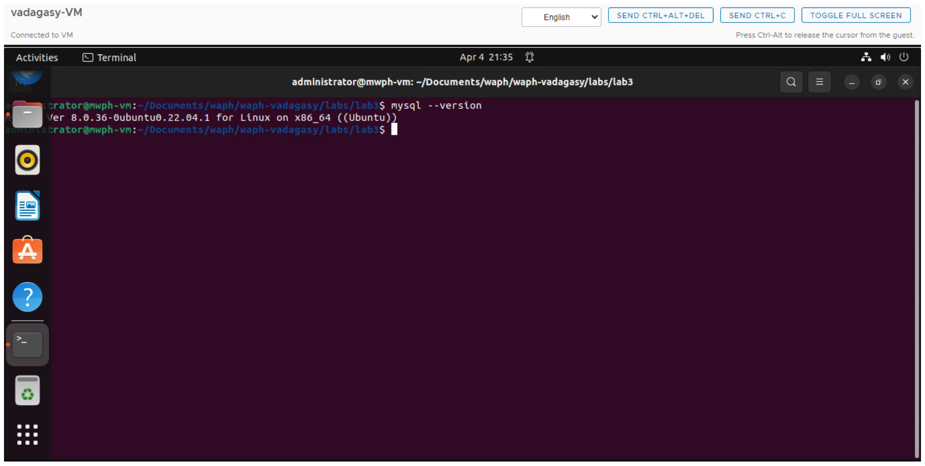
In this lab, I explored in creating web applications with PHP and MySQL, extending the knowledge from Lab 3. My task wasto address and find security vulnerabilities in web development. I developed a web application and tested its security by performing SQL injection and XSS attacks to bypass the login system. To counter these vulnerabilities, I implemented input validation using prepared statements and sanitized the output. I documented each step, including the attack demonstrations, in a README file. For the final documentation, I used Pandoc to convert the markdown content and screenshots into a comprehensive PDF file.

https://github.com/vadagasy/waph-vadagasy/blob/main/lab/lab3/README.md

a. DataBase Setup and Management

MySQL installation

I checked for the Mysql Installation and Installed it and verfied it bu checking it's version.


vadagasy-VM
Connected to VM
English
SEND CTRL+ALT+DEL
SEND CTRL+C
TOGGLE FULL SCREEN
Press Ctrl-Alt to release the cursor from the guest.
Activities
Terminal
Apr 4 21:35
administrator@mwph-vm: ~/Documents/waph/waph-vadagasy/labs/lab3

```
rator@mwph-vm:~/Documents/waph/waph-vadagasy/labs/lab3$ mysql --version
Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))
rator@mwph-vm:~/Documents/waph/waph-vadagasy/labs/lab3$
```

Create a New database, Database User and permission

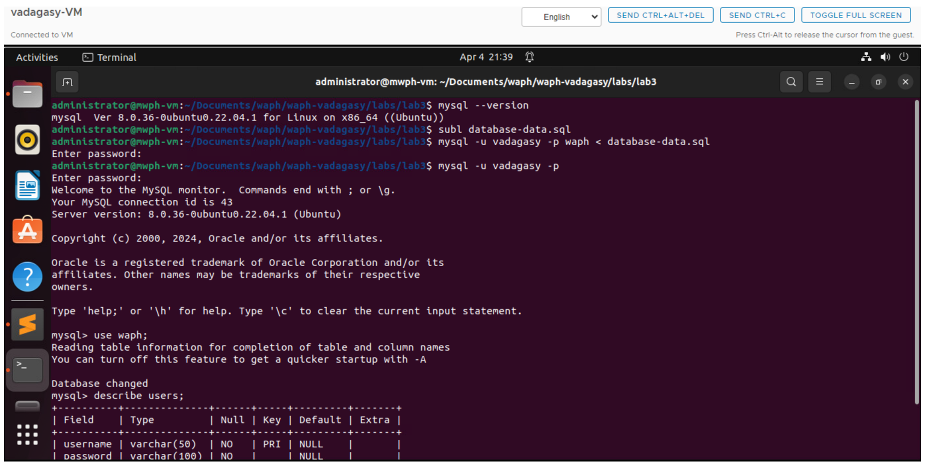
For this task, I created a file named 'database-account.sql' it contains the SQL commands to create a new user and give permissions to this user. I successfully created a new user in the database and was able to log in with that user's name and password I then committed and pushed these changes to GitHub, ensuring that the code and updates were properly versioned and stored in the repository.

```
create database waph;
    CREATE USER 'vadagasy'@'localhost' IDENTIFIED BY 'Pa$$w0rd';
    GRANT ALL ON waph.* TO 'vadagasy'@'localhost';
```

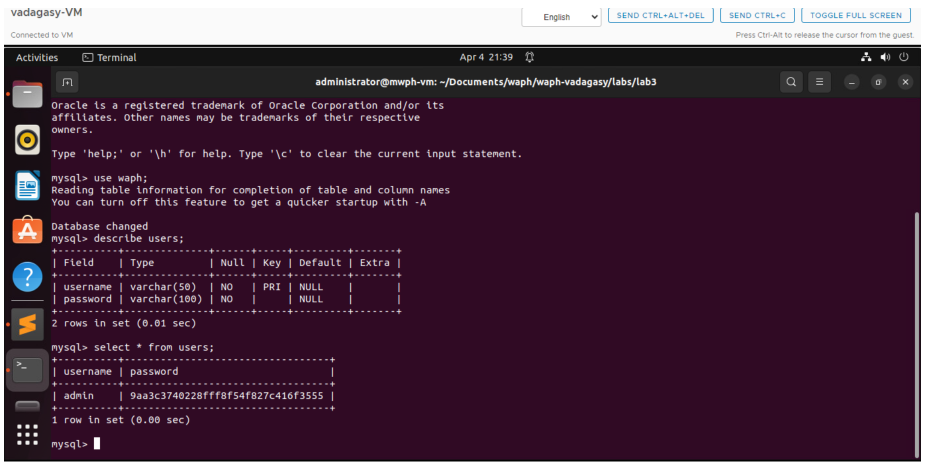
Create a new table Users and insert data into the table

I created a table 'users' in the database, which includes fields for the user's name and password. next, I inserted a record with a randomly chosen username and password. I used an SQL SELECT query inorder to check it. The retrieved data, as shown in the screenshot, indicates that the password is stored in a hashed format for security.

```
drop table if exists users;
create table users(
    username varchar(50) PRIMARY KEY,
    password varchar(100) NOT NULL);
INSERT INTO users(username,password) VALUES ('admin',md5('MyPa$$w0rd'));
```


vadagasy-VM
Connected to VM
English
SEND CTRL+ALT+DEL
SEND CTRL+C
TOGGLE FULL SCREEN
Press Ctrl-Alt to release the cursor from the guest.
Activities
Terminal
Apr 4 21:39
administrator@mwph-vm: ~/Documents/waph/waph-vadagasy/labs/lab3

```
administrator@mwph-vm:~/Documents/waph/waph-vadagasy/labs/lab3$ mysql --version
mysql  Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))
administrator@mwph-vm:~/Documents/waph/waph-vadagasy/labs/lab3$ subl database-data.sql
administrator@mwph-vm:~/Documents/waph/waph-vadagasy/labs/lab3$ mysql -u vadagasy -p waph < database-data.sql
Enter password:
administrator@mwph-vm:~/Documents/waph/waph-vadagasy/labs/lab3$ mysql -u vadagasy -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
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> describe users;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| username | varchar(50)  | NO   | PRI | NULL    |       |
| password | varchar(100) | NO   |     | NULL    |       |
```


vadagasy-VM
Connected to VM
English
SEND CTRL+ALT+DEL
SEND CTRL+C
TOGGLE FULL SCREEN
Press Ctrl-Alt to release the cursor from the guest.
Activities
Terminal
Apr 4 21:39
administrator@mwph-vm: ~/Documents/waph/waph-vadagasy/labs/lab3

```
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> 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                         |
+----------+----------------------------------+
| admin    | 9aa3c3740228fff8f54f827c416f3555 |
+----------+----------------------------------+
1 row in set (0.00 sec)

mysql> 
```

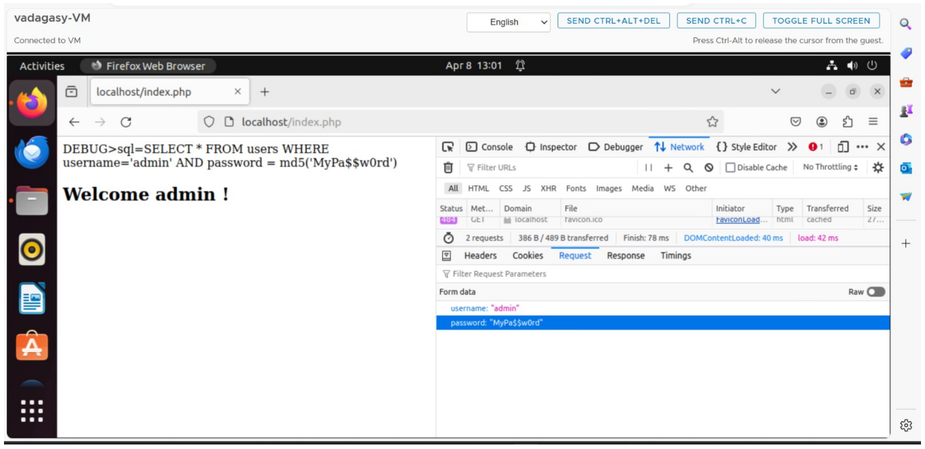
b. A Simple (Insecure) Login System with PHP/MySQL

To work with database programming, I installed the PHP-MySQLi extension, which enables PHP to interact with MySQL databases. Then, I created a form that serves as a login page, which collects username and password. Next, I created an `index.php` file that contains a `checklogin` function. This function is tasked with verifying the login details submitted through the form, checking them against the database to authenticate users.

```
index.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();
    }
    unction checklogin_mysql($username, $password) {
        $mysqli = new mysqli('localhost','vadagasy' /*DB username*/,'Pa$$w0rd'
        /*DB password*/,'waph' /*DB name*/);
                if ($mysqli->connect_errno){
                    printf("Datebase connection failed: %s\n",
                    $mysql->connect_error);
                    exit();
        }
?>
```

```
form.php

<form action="index.php" method="POST" class="form login">
    Username:<input type="text" class="text_field" name="username" /> <br>
    Password: <input type="password" class="text_field" name="password" /> <br>
    <button class="button" type="submit">Login</button>
  </form>
```



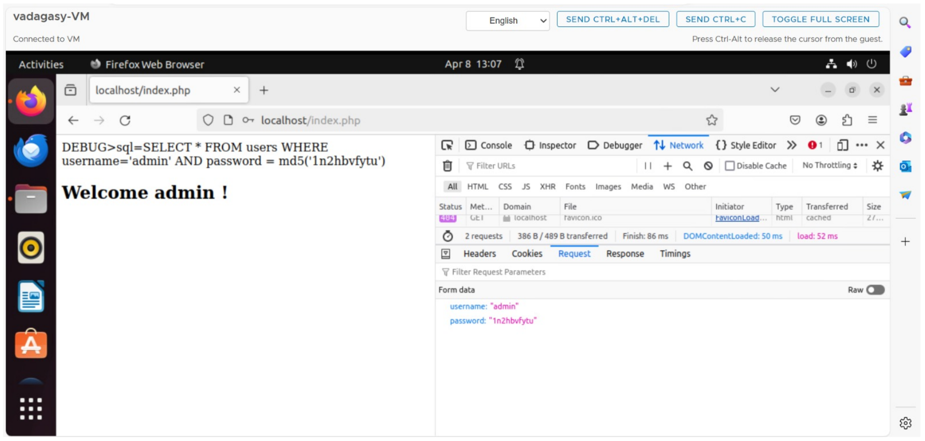
```
DEBUG>sql=SELECT * FROM users WHERE
username='admin' AND password = md5('MyPa$$w0rd')

Welcome admin !
```

```
Status  Met...  Domain     File         Initiator      Type  Transferred  Size
404     GET     localhost  favicon.ico  FaviconLoad... html  cached       27...
2 requests   386 B / 489 B transferred   Finish: 78 ms   DOMContentLoaded: 40 ms   load: 42 ms

Form data
username: "admin"
password: "MyPa$$w0rd"
```

c. Performing XSS and SQL Injection Attacks

Sql injection attacks

These attacks, such as SQL injection, occur primarily because of unverified or correct input validation. Without thorough checking, attackers can take advantage of this by inserting harmful SQL code into the query inputs. The risk is especially high when applications use plain SQL queries that directly concatenate with user inputs, creating a pathway for such attacks. Ensuring security is a critical aspect of application development, and the absence of proactive security measures can significantly increase the vulnerability to SQL injection attacks.



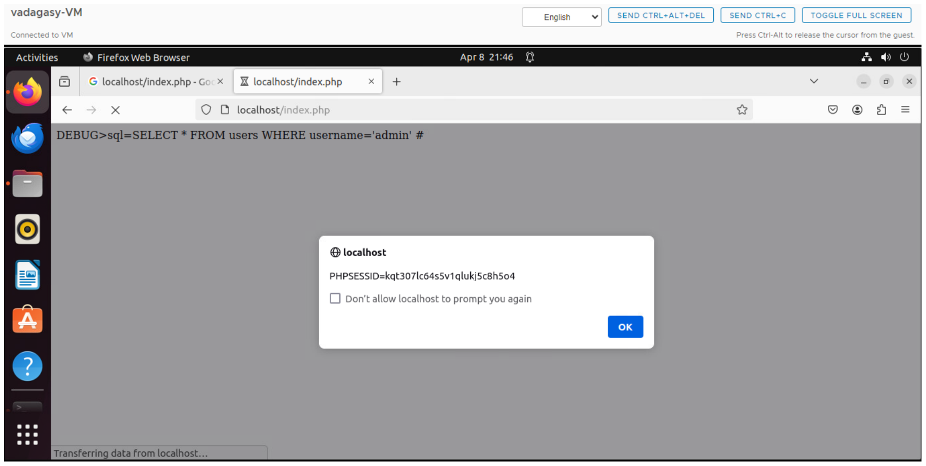
```
DEBUG>sql=SELECT * FROM users WHERE
username='admin' AND password = md5('1n2hbvfytu')
```

Welcome admin !

Form data

```
username: "admin"
password: "1n2hbvfytu"
```

Cross-site Scripting(XSS)



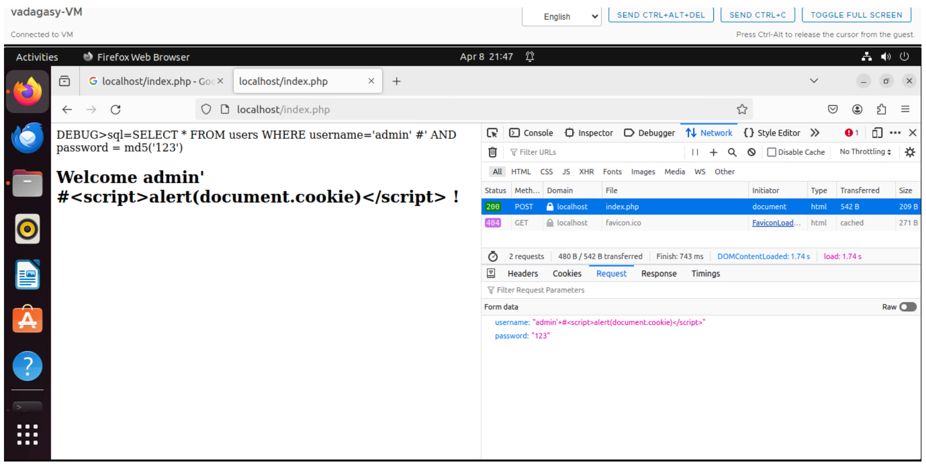
```
DEBUG>sql=SELECT * FROM users WHERE username='admin' #
```

localhost

PHPSESSID=kqt307lc64s5v1qlukj5c8h5o4

☐ Don't allow localhost to prompt you again

OK



```
DEBUG>sql=SELECT * FROM users WHERE username='admin' #' AND
password = md5('123')
```

Welcome admin' #<script>alert(document.cookie)</script> !

```
Status  Meth...  Domain     File         Initiator      Type  Transferred  Size
200     POST     localhost  index.php    document       html  542 B        209 B
404     GET      localhost  favicon.ico  FaviconLoad... html  cached       271 B

2 requests | 480 B / 542 B transferred | Finish: 743 ms | DOMContentLoaded: 1.74 s | load: 1.74 s

Form data
  username: "admin'+#<script>alert(document.cookie)</script>"
  password: "123"
```

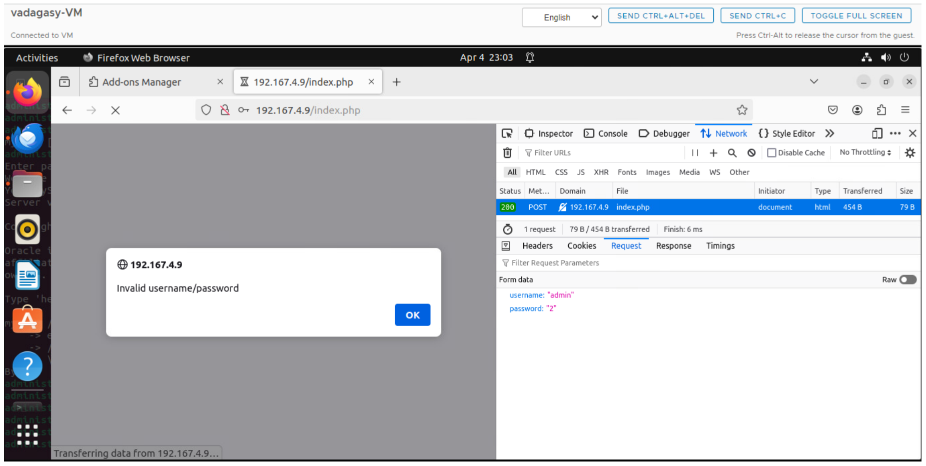
d.Prepared Statement Implementation

Prepared Statement for SQL Injection Prevention

I utilized prepared statements to sanitize the output and avoid vulnerabilities with SQL injection attacks. The code below has been used to prevent SQL injection attacks.

code:

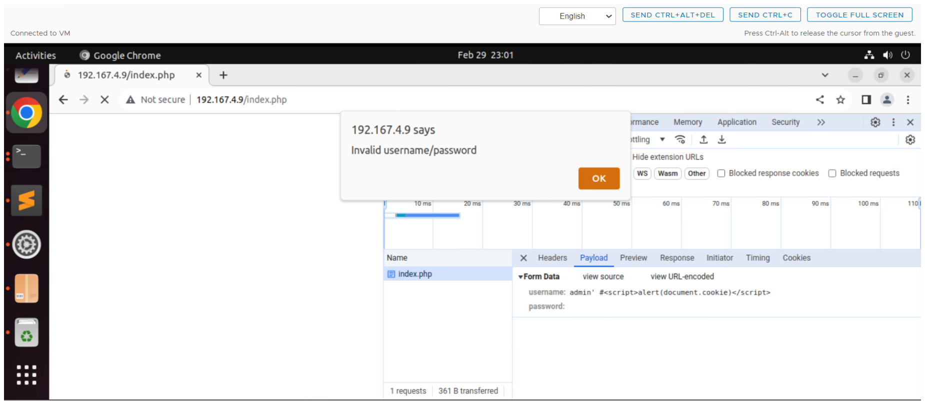
```
$prepared_sql = "SELECT * FROM users WHERE username= ? " . " AND password=md5(?);";
          $stmt = $mysqli->prepare($prepared_sql);
          $stmt->bind_param("ss", $username,$password);
          $stmt->execute();
          $result=$stmt->get_result();
```



```
192.167.4.9
Invalid username/password
OK
```

```
Status  Met...  Domain       File       Initiator  Type  Transferred  Size
200     POST    192.167.4.9  index.php  document   html  454 B        79 B

1 request | 79 B / 454 B transferred | Finish: 6 ms

Form data
  username: "admin"
  password: "2"
```

Invalid login with implementation of Prepared statement



192.167.4.9 says
Invalid username/password

```
Form Data
username: admin' #<script>alert(document.cookie)</script>
password:
```

Logged in succesfully with the implementation of prepared statements after providing correct credentials.

Security Analysis

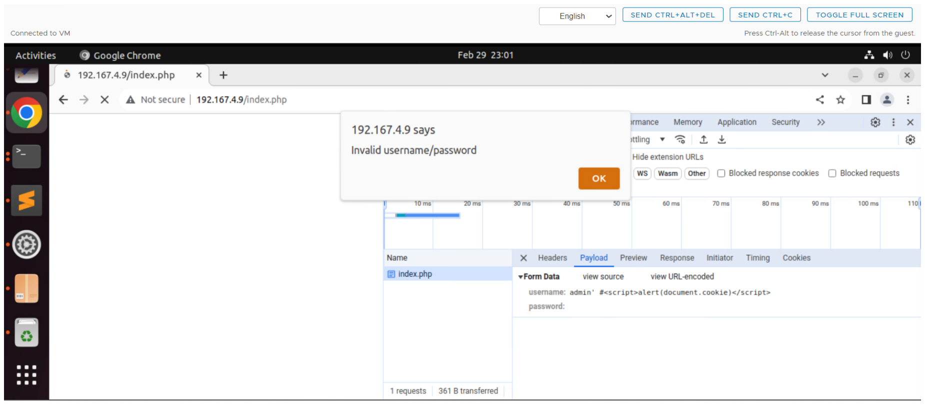
Explaination of Prepared Statements: The prepared statements employ parametrized queries, which keep user input and SQL code apart. The escaping notion will also be used to user input in prepared sentences.Given that various escape concepts need changing the special characters that users enter.

Implement Sanitization : This line

Welcome ?php echo $_POST[“username”];?> besides the SQLi vulnerability

since the input variable is shown immediately in index.php, making it susceptible to XSS attacks. In order to get around this, I've utilized htmlspecialchars, which translate the special characters into the appropriate HTML entities so that they're seen as plain text rather than HTML tags.

```
<h2> Welcome <?php echo $_POST["username"]; ?> !</h2>
```



192.167.4.9 says
Invalid username/password

```
Form Data
username: admin' #<script>alert(document.cookie)</script>
password:
```

Discussions

what if the username/password are empty? The code does not check to see if the username and password fields in the checklogin_mysql() method are empty before attempting to utilize them. In the event that the user submits the form without entering any information, this might lead to unanticipated consequences or errors.

what if there are any database errors? Even while the code includes error handling for unsuccessful database connections ($mysqli->connect_error), it does not handle other possible database failures that may occur during query execution. For example, when there is a syntax mistake in the SQL query or when the database server is having issues, the code fails quietly and doesn't alert the user.

what if the provided username is not exactly the same as the username from the database.? Because MD5 hashes quickly and doesn't need salting, it is seen as risky when used to store passwords. It is also susceptible to brute-force and rainbow table attacks. It is advised to use more secure hashing algorithms, such as Argon2 or bcrypt, to increase security. These techniques each need a different salt for each password. Case sensitivity in username authentication is highlighted by the existing system, which obtains user records from the database only if the supplied and stored usernames and passwords exactly match. If there is a difference in case between the username that the user entered and the one that is kept in the database, this might cause problems.