SQL INJECTION REPORT

December 9th, 2022

Yathusan Thulasinathan - 6735955
Adam Shariff - 6768600
Maisam Anjum - 6804298
Steve John Abraham Jayaseelan - 6856694

Executive Summary

Our group was tasked with exploring a cyber security topic. The topic we decided to tackle was SQL Injection. To understand the topic we replicated an SQL attack, then replicated preventative measures to test how to stop said attack. We had the following goals to accomplish in this report.:

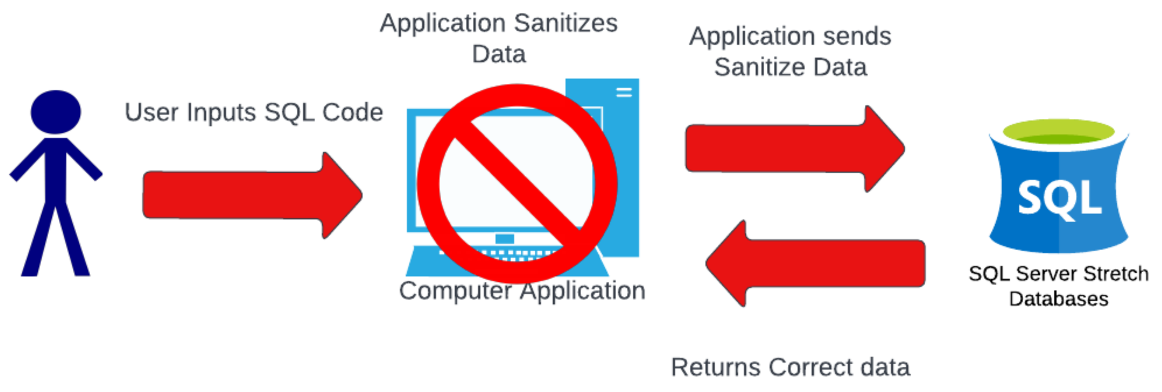
- To understand how a SQL attack is performed and create a replicable procedure to perform a sample attack.
- To understand how to defend from an SQL injection by performing a preventative measure/defense.

Summary of Results

An SQL injection involves using an application's queries to a database to inject code into the database. An application will take in user input and send it to a database. An example of the input is a login page with the user inputting login credentials. A malicious user can use this as a point of entry into a database. Instead of sending credentials, the user could input database code. When the application sends a query to the database, the database will run the code. This can lead to results such as returning the entire database or even destroying all the data within the database.

Using this knowledge we were able to recreate an SQL injection. We created our own login database as well as a login page to interface with it. Using this we were able to demonstrate the injection and retrieve user information from the database. We were able to send SQL instructions to return data from the database.

We were also able to modify the vulnerable code to prevent the injection. We employed data sanitization to prevent the user input from the application being read as code. This being the main way to prevent SQL injections.


User Inputs SQL Code
Application Sanitizes Data
Computer Application
Application sends Sanitize Data
SQL
SQL Server Stretch Databases
Returns Correct data

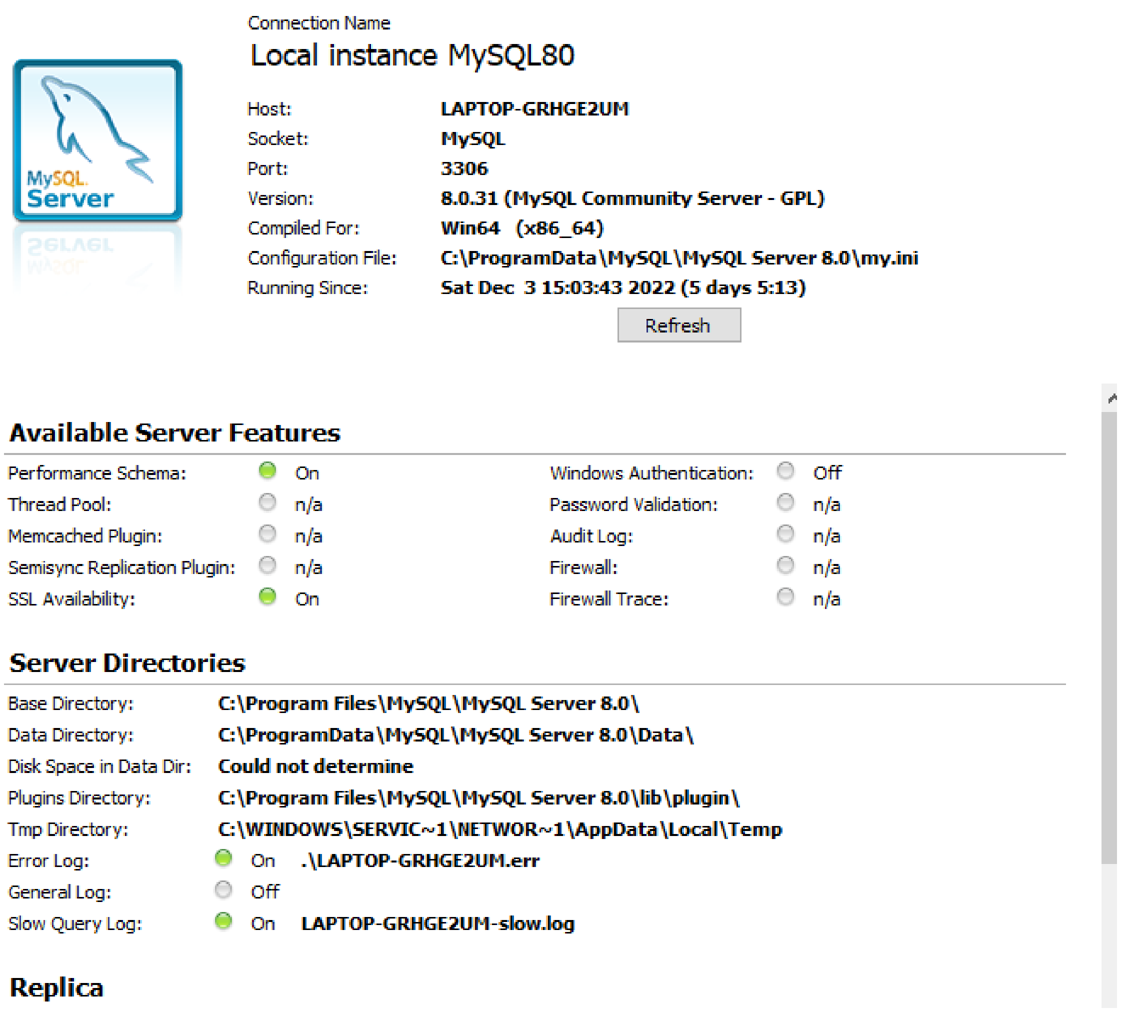
ATTACK NARRATIVE

Tools Used:
The stack utilized : HTML, MySQL, Apache, PHP

Setting Up Database:

Server version : MySQL Community Server version 8.0.31
App used to host the server : MySQL Workbench

Hosting the server and website:



Connection Name

Local instance MySQL80

Host:	LAPTOP-GRHGE2UM
Socket:	MySQL
Port:	3306
Version:	8.0.31 (MySQL Community Server - GPL)
Compiled For:	Win64 (x86_64)
Configuration File:	C:\ProgramData\MySQL\MySQL Server 8.0\my.ini
Running Since:	Sat Dec 3 15:03:43 2022 (5 days 5:13)

Refresh

Available Server Features

Performance Schema:	● On	Windows Authentication:	○ Off
Thread Pool:	○ n/a	Password Validation:	○ n/a
Memcached Plugin:	○ n/a	Audit Log:	○ n/a
Semisync Replication Plugin:	○ n/a	Firewall:	○ n/a
SSL Availability:	● On	Firewall Trace:	○ n/a

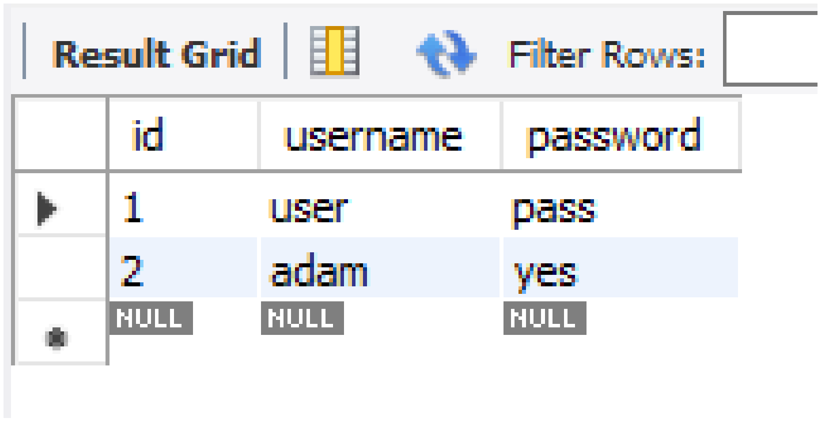
Server Directories

Base Directory:	C:\Program Files\MySQL\MySQL Server 8.0\
Data Directory:	C:\ProgramData\MySQL\MySQL Server 8.0\Data\
Disk Space in Data Dir:	Could not determine
Plugins Directory:	C:\Program Files\MySQL\MySQL Server 8.0\lib\plugin\
Tmp Directory:	C:\WINDOWS\SERVIC~1\NETWOR~1\AppData\Local\Temp
Error Log:	● On .\LAPTOP-GRHGE2UM.err
General Log:	○ Off
Slow Query Log:	● On LAPTOP-GRHGE2UM-slow.log

Replica

Figure 1 - MySQL dashboard settings

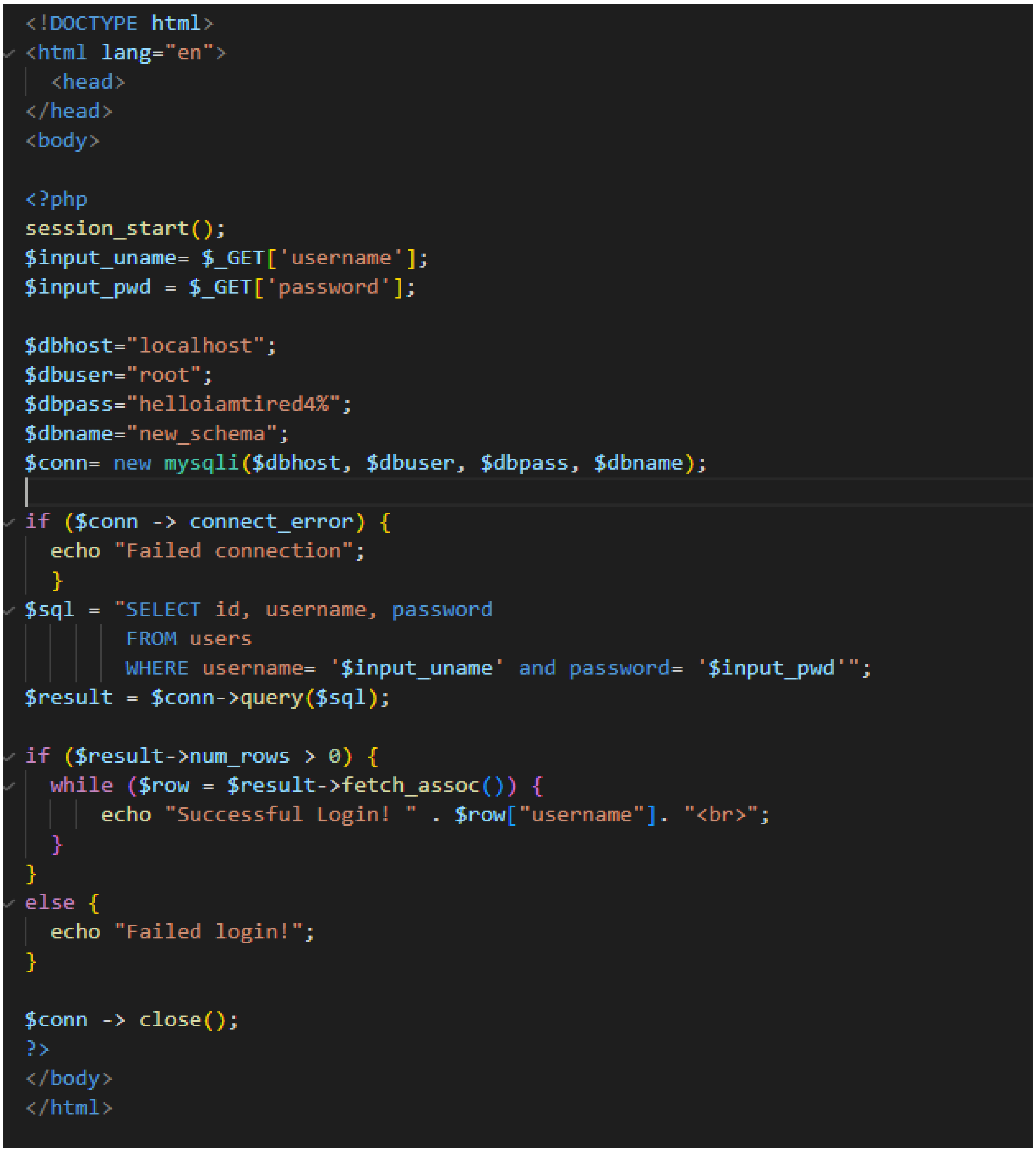
- Download MySQL server
- Run the server on localhost using MySQL Workbench, set a username and password to connect to the database
- Create a new database on the server
- Create a table called users, it has three columns: id, username and password
- Add any user you want with the id, username and password and each user will be 1 row



id	username	password
1	user	pass
2	adam	yes
NULL	NULL	NULL

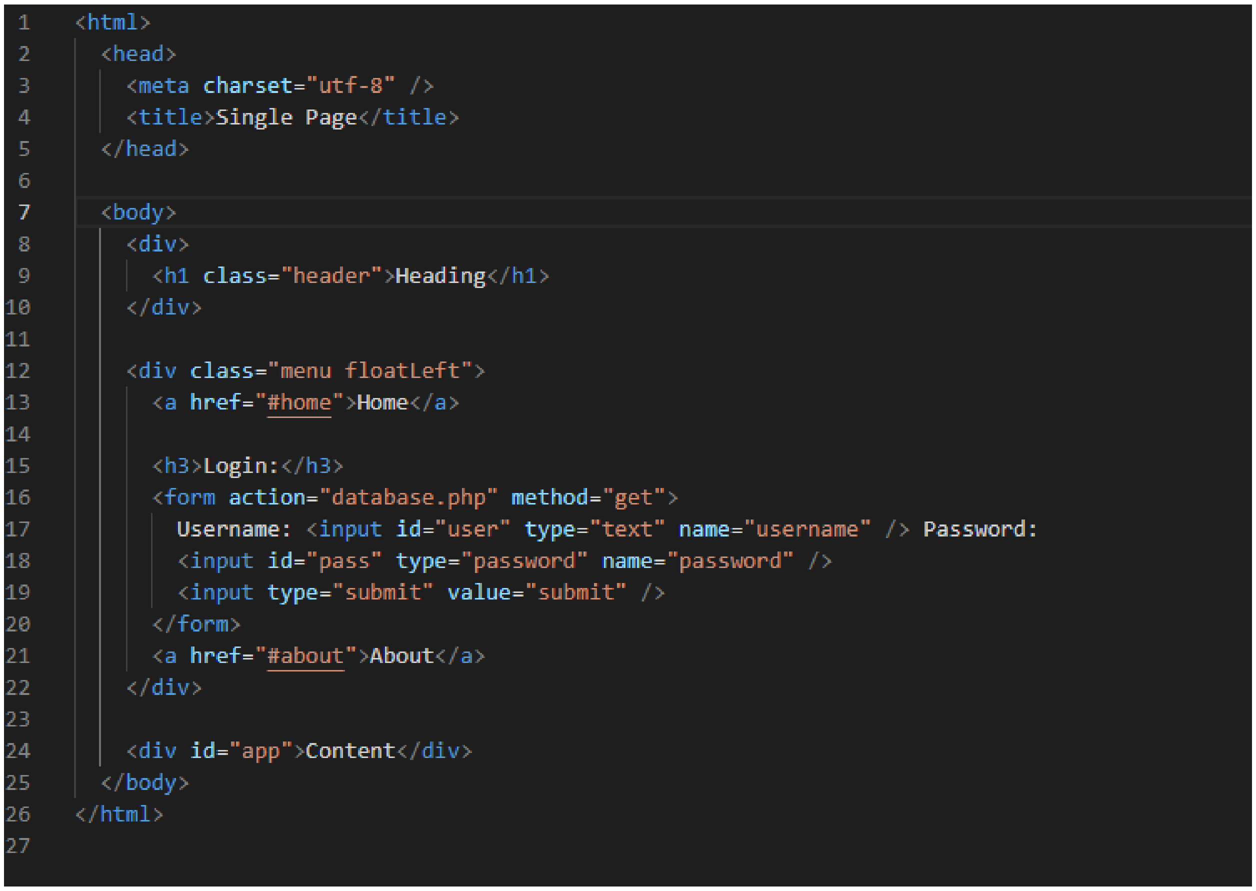
Figure 2 - the database

HTML code was written for a simple login page linked to the database created.



```
<!DOCTYPE html>
<html lang="en">
  <head>
</head>
<body>

<?php
session_start();
$input_uname= $_GET['username'];
$input_pwd = $_GET['password'];

$dbhost="localhost";
$dbuser="root";
$dbpass="helloiamtired4%";
$dbname="new_schema";
$conn= new mysqli($dbhost, $dbuser, $dbpass, $dbname);

if ($conn -> connect_error) {
  echo "Failed connection";
  }
$sql = "SELECT id, username, password
        FROM users
        WHERE username= '$input_uname' and password= '$input_pwd'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  while ($row = $result->fetch_assoc()) {
      echo "Successful Login! " . $row["username"]. "<br>";
  }
}
else {
  echo "Failed login!";
}

$conn -> close();
?>
</body>
</html>
```

Figure 3 - the unchanged and vulnerable initial login script

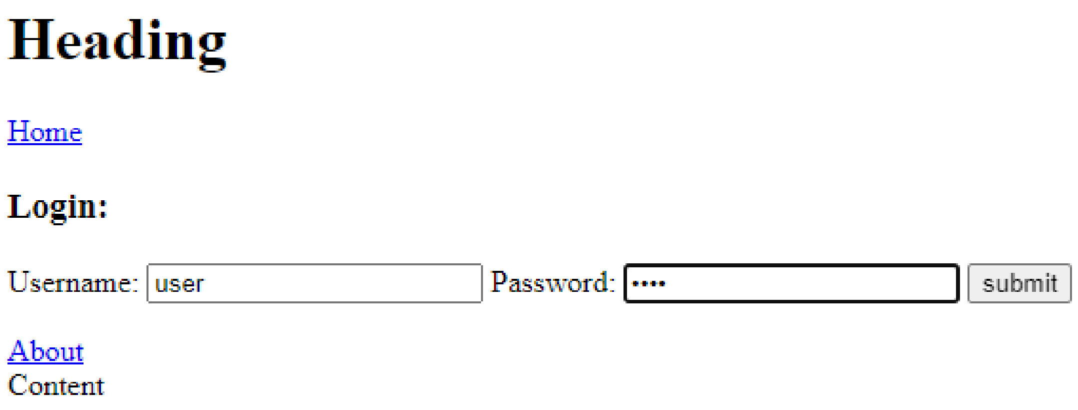


```html
<html>
  <head>
    <meta charset="utf-8" />
    <title>Single Page</title>
  </head>

  <body>
    <div>
      <h1 class="header">Heading</h1>
    </div>

    <div class="menu floatLeft">
      <a href="#home">Home</a>

      <h3>Login:</h3>
      <form action="database.php" method="get">
        Username: <input id="user" type="text" name="username" /> Password:
        <input id="pass" type="password" name="password" />
        <input type="submit" value="submit" />
      </form>
      <a href="#about">About</a>
    </div>

    <div id="app">Content</div>
  </body>
</html>
```

Figure 4 - HTML code utilized within the login page

Host for the website : Local server (on PC)

This is the query that is used for the MySQL database to ensure that the login credentials that are typed into the login page are correct. (The normal login way)
SELECT id, username, password
FROM users
WHERE username= '$input_uname' and password= '$input_pwd'
Upon a successful login , the webpage shows “login successful + ‘username’”.


Heading

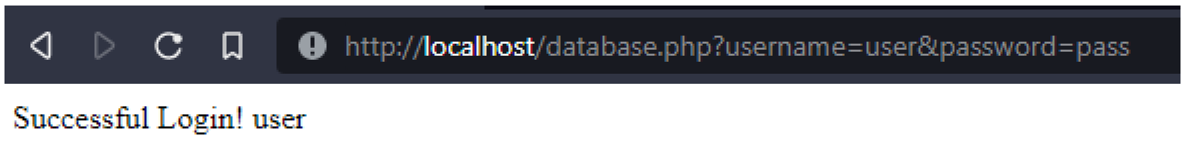
Home

Login:

Username: user Password: •••• submit

About
Content

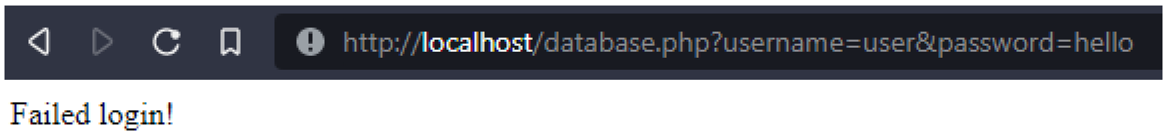
Figure 5 - user inputs shown with correct inputs for login


http://localhost/database.php?username=user&password=pass

Successful Login! user

Figure 6 - Standard successful login

While attempting a proper login , upon entering wrong credentials the login fails.

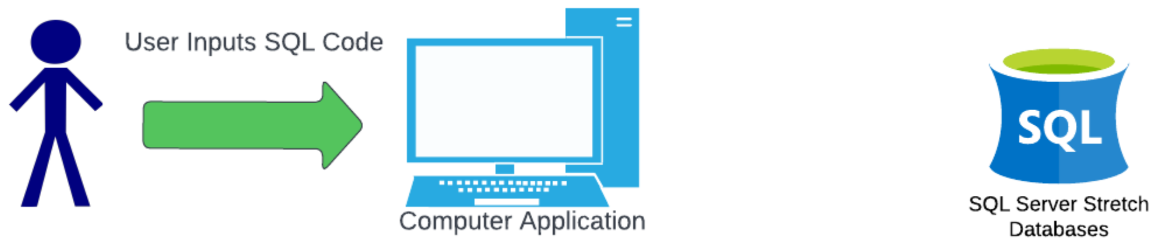

http://localhost/database.php?username=user&password=hello

Failed login!

Figure 7 - Demonstrates failed login

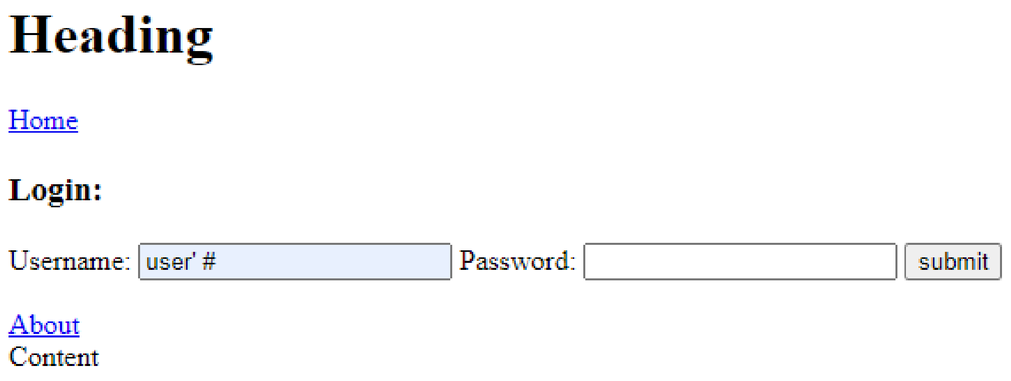
Attack Procedure

While comparing the entered username and password with the database, an interruption can be made such that the password check condition fails.



The normal user name login way can be altered by the following two ways

1) user'# -> logs in as actual user



Heading

Home

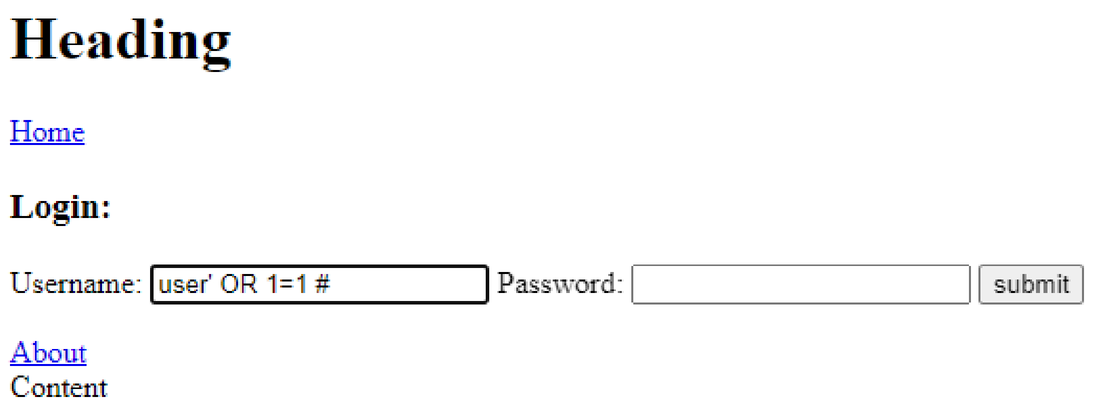
Login:

Username: user' # Password: [ ] submit

About
Content

Figure 8 - Sample of possible injection

2) ‘1=1# ->which lets us login in as all users



Heading

Home

Login:

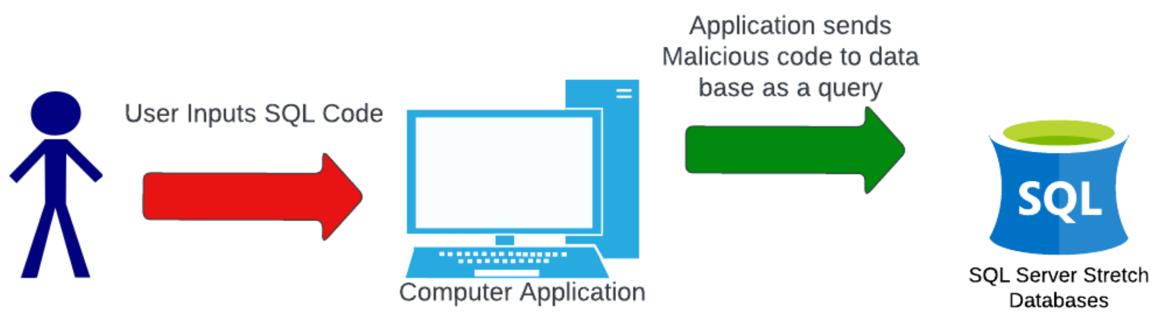
Username: user' OR 1=1 # Password: submit

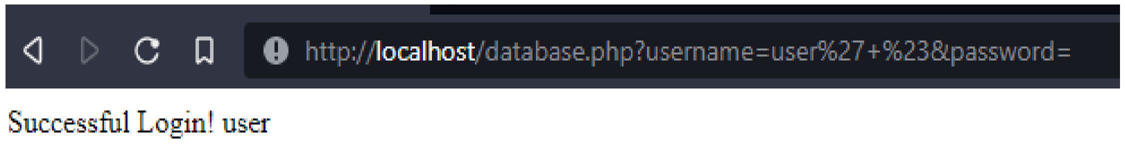
About
Content

Figure 9 - Sample of possible injection

This is the possible execution based on the given injection

- Anything after # is commented so "and password=password " is never actually done when the SQL runs the query and the login for the user is done without any password or a random password entered.
- 1=1 is always true. So it just logs into any of the accounts that the database holds.


User Inputs SQL Code
Computer Application
Application sends Malicious code to data base as a query
SQL
SQL Server Stretch Databases


```
http://localhost/database.php?username=user%27+%23&password=
```

```
Successful Login! user
```

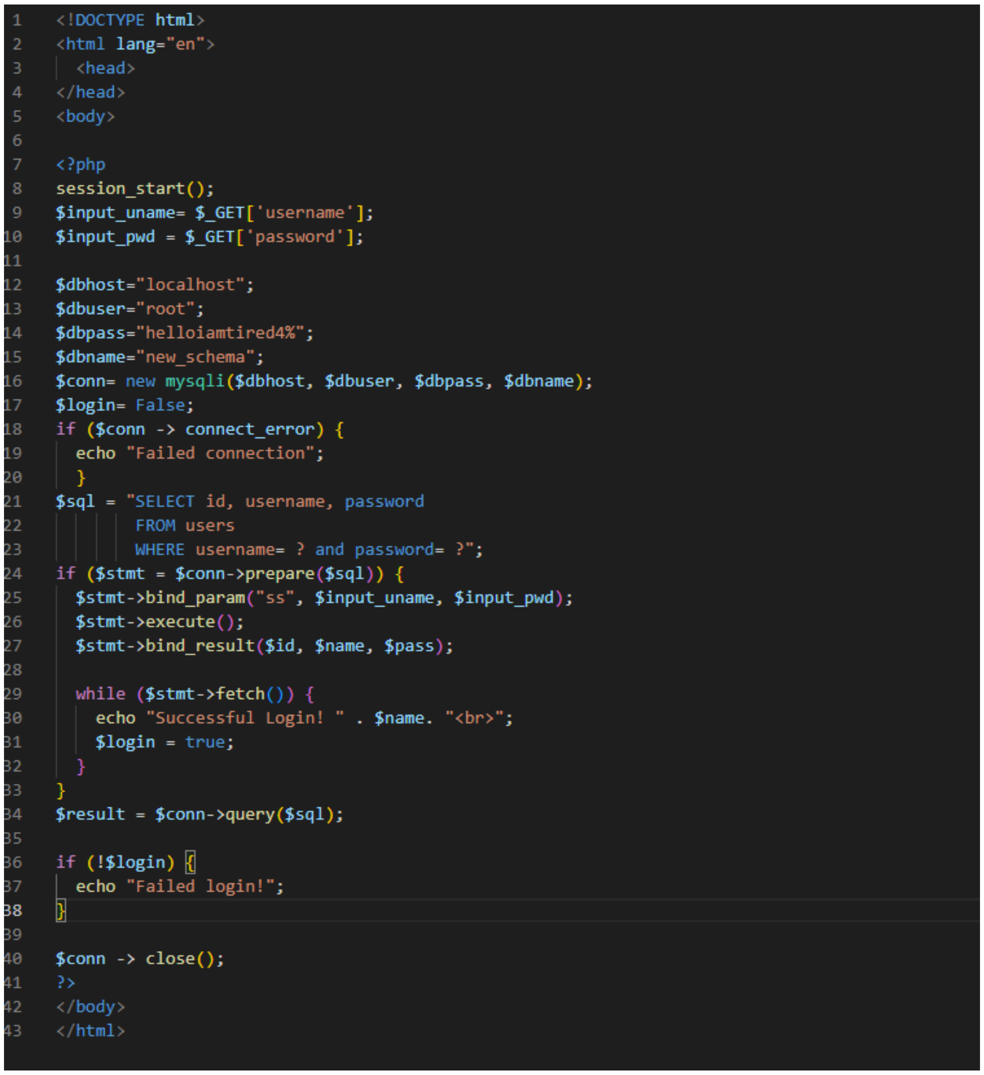
Figure 10 - Demonstrates successful login

Defense Against the Attack

The primary mechanism that is used in the SQL injection attack is to exploit a user input system that allows for unintended inputs. The vulnerability is present within the interface when user inputs can allow for access into the database by running scripts that the designer may not have known to protect themselves from, the system is tricked into running input as code.

In order to defend against this, we must control how the users are allowed to interface with the program such that the exploits can be blocked. It would be possible to limit all user inputs to preset values in a dropdown, however this would not work in all applications because discrete and known values would not be a reasonable design choice in situations such as a login system.

In this case, we must utilize a prepared statement which specifically separates the code and data into its own channels. How a prepared statement works is that it sends a SQL statement template to the database, the database parses, compiles and the query and it stores the result without executing the SQL query. Then at a later time we bind values to the parameters (here we do that using bind_param()) and then we ask the database to execute the statement. Thus the code part of the database is always the same but the data can be different every time the SQL database is executed.

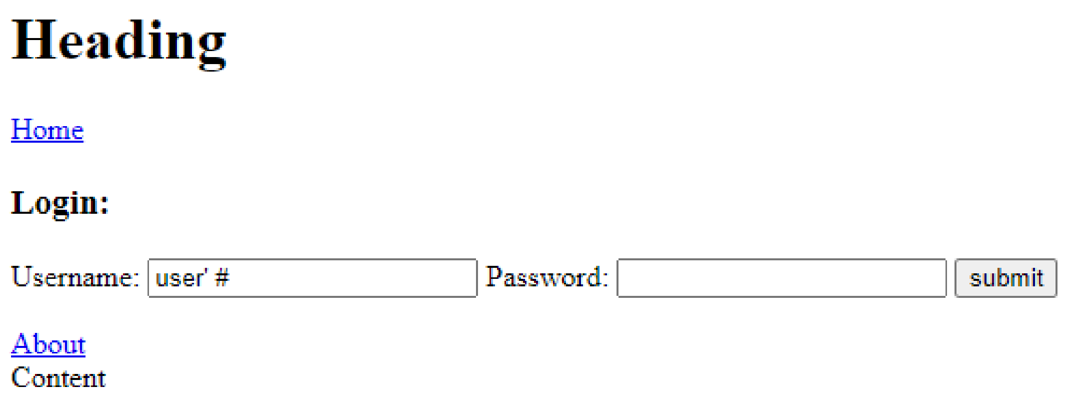


```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4   </head>
5   <body>
6
7   <?php
8   session_start();
9   $input_uname= $_GET['username'];
10  $input_pwd = $_GET['password'];
11
12  $dbhost="localhost";
13  $dbuser="root";
14  $dbpass="helloiamtired4%";
15  $dbname="new_schema";
16  $conn= new mysqli($dbhost, $dbuser, $dbpass, $dbname);
17  $login= False;
18  if ($conn -> connect_error) {
19    echo "Failed connection";
20    }
21  $sql = "SELECT id, username, password
22          FROM users
23          WHERE username= ? and password= ?";
24  if ($stmt = $conn->prepare($sql)) {
25    $stmt->bind_param("ss", $input_uname, $input_pwd);
26    $stmt->execute();
27    $stmt->bind_result($id, $name, $pass);
28
29    while ($stmt->fetch()) {
30      echo "Successful Login! " . $name. "<br>";
31      $login = true;
32    }
33  }
34  $result = $conn->query($sql);
35
36  if (!$login) {
37    echo "Failed login!";
38  }
39
40  $conn -> close();
41  ?>
42  </body>
43  </html>
```

Figure 11 - the PHP code with implements defense

In our defense, we will apply the prepared SQL statement to the input before we allow it to run so that we can confirm that no unintended consequences are to occur.

- At line 25 in figure x, we can see that the PHP script will take in the credentials as a string data type, which will prevent the user input from being run as a script to access the data. The prepared statement method sets the credentials as ss meaning that they are identified as a string,
- execute the query, which is attempting to log in to the page.
- If the results are valid, it will allow a login and bind the result into Username, Password, and ID, else if a script is attempted here, it will go no further and return a failed login output.



Heading

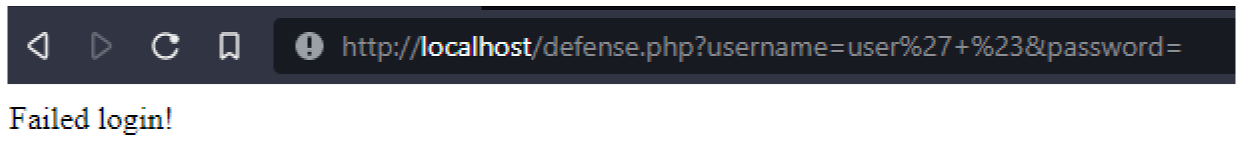
Home

Login:

Username: user' # Password: submit

About
Content

Figure 12 - Attempting to use suspicious inputs



http://localhost/defense.php?username=user%27+%23&password=

Failed login!

Figure 13 - The defense successfully prevents the attack

When the same attack procedure is attempted as seen in page x, we can see that the system will simply take in the username as a string type value, and attempt to use the user input as an attempted username. Because this likely would not match the database's accepted login credentials, it will return a failed login output, and the script would not run

<u>**Effectiveness of Defense**</u>

The method of sanitization and input filtering are very effective methods to protect against this particular flavor of attack, because it will dismantle the systems that will allow it to function.. Although we may still be vulnerable to alternative attacks, The effectiveness of this defense against SQL injections is nearly perfect, as it will eliminate any possibility of code and data being mixed together, in fact, prepared statements separate the code and data into its own channels.

Conclusion - Recommendations, Risk Rating

In conclusion, an SQL injection attack is a very effective method to exploit inputs that are set up without proper security precautions, however with some applied concepts in cybersecurity to vet the user inputs before attempting to execute code, we can successfully protect ourselves from this type of attack. Some recommendations to ensure security are as follows:

1. **Ensure that inputs are sanitized before they are executed in the program**. This means that one or several methods should be applied to user inputs before they are used. Some examples include utilizing prepared statements such as we have done in our report, filter inputs so that all unnecessary characters are illegal and will not run, maintaining reasonable bounds and parameters on what is a valid input.
2. **Grant privileges on as-needed basis.** In order to utilize commands on the database such as insert, delete, search, the program requires permission to make these changes. By minimizing permissions granted, the security concerns are minimized as well.
3. **Utilize standard cybersecurity patterns in system design.** Within the architecture, we can apply more complex design principles such as using multiple databases, delegating authentication to external sources, conducting regular testing, 2-factor authentication, and data encryption within the database. These industry standard techniques will add security layers to make exploitation more difficult.

Risk rating:

The overall risk identified as a result of the SQL Injection attack is Moderate. A direct path from an external attacker to the database was discovered. It is reasonable to believe that a malicious entity would be able to successfully execute the attack and access private information from the database through the attack, however, using some design techniques and cybersecurity principles, we can eliminate the risk of this attack entirely.