Binghamton University, Watson School of Engineering

Project 3:
SQL Injection Prevention

Name: Rahul Verma

Science of Cyber Security – CS 559-01

Prof. Guanhua Yan

November 14, 2022

Contents

SQL INJECTION PREVENTION

1. Previous work and intro to SQLI prevention:

In the first project, the approach used was vulnerable where the attacker can change the statement by entering a SQL injection query directly in the input username and password.

```
14    $query=mysqli_query($connection,"SELECT * FROM userpass WHERE user='$user'AND pass='$pass'");
```

Here, we can clearly see that the source code in the first project was vulnerable to manipulation. So, if the attacker inserted a vulnerable SQL query, the database information was easily accessible to him. So, when the attacker enters username and password as (' or 1=1 --), then the SQL query became "SELECT * FROM userpass WHERE user = '$user' AND pass ='' or 1=1--". The attacker gets access to the database since the logic statement "1=1" is always true and the rest of the statement is commented out.

In the second project, we detected SQL injection using Naïve Bayes Classifier, where the method of detection was based on machine learning. The Naïve Bayes classifier was used to filter out the numerous dangerous spam keywords using Bayesian decision theory, where we also teach the algorithm to filter out the terms that we consider to be spam and might harm the web application and so it keeps learning and making accurate decisions as time progresses. The web application's source code wasn't required to be changed in that method.

However, when the attacker tries brute forcing all the SQL injection queries, we need to prevent those by implementing prevention mechanism. Nowadays, we have advanced mechanisms to combat SQL injection attacks, using Prepared Statements in which before any parameters which are given by the user as an input are entered, the SQL statements are submitted to the SQL database server as well as compiled or converted by the SQL database server, MySQLI real escape string which is used to circumvent whatever special characters that an attacker may input in the username or password fields of login form on a website and Input Validation. Implementing these three mechanisms makes sure that the website will be completely safe against SQL injection attacks.

2. How to prevent against SQLI:

- To know the security threats that exist:
  The system's security is only at risk from user input. When performing a SQL Injection attack, the attacker can insert the queries as special characters, turning them into Boolean statements and gaining access to the database. The attacker can alter the data

by deleting, updating, or inserting fraudulent information after gaining access to the database. The HTML or the application's client side is where the input is provided.

- Vulnerabilities connected to the system development for the website:
  The client enters input username and password that are then cross-checked against the database to see if they match. If they do and pass, the client or user is allowed permission to access the database. Because of this, if the username and password entries are not properly checked, the attacker may exploit this and enter a customized query, giving them access to the database.

3. Prepared Statements:

Before any parameters which are given by the user as an input are entered, prepared statements, which are SQL statements are submitted to the SQL database server as well as compiled or converted by the SQL database server. We must build the query as well as the input separately for a prepared statement, sometimes referred to as a parameterized query. In this case, we specify all the SQL logic upfront and compile it, then enter parameters into the query immediately before execution. This is one of the methods which will help in completely safeguarding the website against any kind of SQL injection attacks, especially the one where hackers use brute force to keep injecting different SQL queries to gain access to the database and other private information of the users stored in the database.

Approach used in the first project which is vulnerable:

```
// Establish connection
$connection=mysqli_connect("localhost","root","");
$db=mysqli_select_db($connection,"test");

// Vulnerable approach
$query=mysqli_query($connection,"SELECT * FROM userpass WHERE user='$user'AND
pass='$pass'");
$row=mysqli_fetch_array($query);
```

Here, if we give the input as (1’ or ‘1’ = ‘1) the statement will become “ SELECT * FROM userpass WHERE user = ‘$user’ AND pass =‘1’ or ‘1’ = ‘1’ ” and the attacker gets access to the database since the logic statement “1=1” is always true. Due to the mixing of SQL code and user data, an attacker is now able to inject code, alter the SQL statement's structure, steal data, edit data, and even insert system instructions.

The following is an example of a safe approach using prepared statement:

```php
// Establish connection
$connection=mysqli_connect("localhost","root","");
$db=mysqli_select_db($connection,"test");

// Using "?" as placeholders
$stmt = $connection->prepare('SELECT * FROM table WHERE text = ? AND number = ?');

// Bind parameters to statement, here, "s" for string and "i" for integer
$stmt->bind_param("si", $text, $number);

// Final Step is to execute
$stmt->execute();
```

Here, we first define the structure of the query. We write the query without parameters and use "?" as placeholders. This string will be sent to the SQL server to be converted into SQL code. We can then send over the query's parameters separately. Here, "si" denotes that we are providing two parameters, the first of which is a string and the second of which is an integer. Finally, we execute the query.

4. Implementation and working of prepared statement:

```php
20  //SQLI prevention using prepared statement
21  $stmt = $connection->prepare('SELECT * FROM userpass WHERE user = ? AND pass = ?');
22  $stmt->bind_param('ss', $user, $pass);
23  $stmt->execute();
24  $result = $stmt->get_result();
```

As we can see in the "login.php" file we have successfully implemented prepared statement with the help of which we can make our website completely secure against SQL injection attacks consisting of brute force approach.

On line 21, almost all the SQL statement is the same as before, we create an object called $stmt and then we call the prepare method, here as we cannot use variable names directly and instead, we substitute "?" in the place of $user and $pass.

On line 22, we are going to execute bind parameters which has 2 different parameters which defines what the "?" means, the first parameter will be "ss" as both the variables are string, also we are binding the "?" to a variable, which here is $user and $pass.

On line 23, we execute the query and on line 24, we store the result in the object called $result.

5. MySQLI real escape string:

mysqli_real_escape_string() is used to circumvent whatever special characters that an attacker may input in the username or password fields of login form on a website as well as to insert input in a table in a database that contains special characters, circumventing the special characters totally shields the website from SQLI attacks, which often include manipulating the SQL query in the source code with special characters.

Syntax and example for the usage of MySQLI real escape string is shown below:

```
// mysqli_real_escape_string() syntax:
mysqli_real_escape_string(connection_var, string_var)

// mysqli_real_escape_string() example:
$example = mysqli_real_escape_string($connection, $example);
```

6. Implementation and working of MySQLI real escape string:

```
14   //SQLI prevention using mysqli_real_escape_string
15   $user=stripcslashes($user);
16   $pass=stripcslashes($pass);
17   $user=mysqli_real_escape_string($connection, $user);
18   $pass=mysqli_real_escape_string($connection, $pass);
```

As we can see in the “login.php” file we have successfully implemented mysqli_real_escape_string() with the help of which we can make the entered SQL query to be considered as a string. Thus, protecting against SQL injection attack.

On line 15 and 16 we use stripcslashes() to remove the back slashes in front of the provided words.

On line 17 and 18 we can see that we have used mysqli_real_escape_string() for $user and $pass which will take care of the username and password input if it contains any special characters used by hackers for SQL injection and make it completely considered like a string.

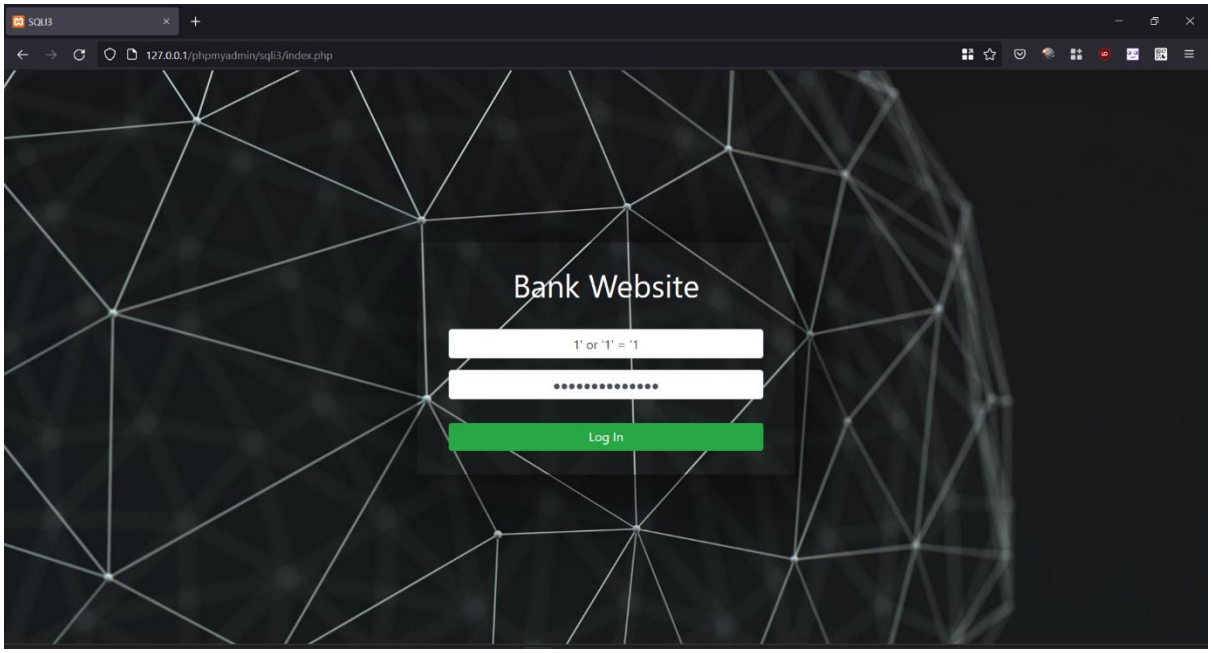
7. Input Validation implementation and working:

```
<!-- SQLI Prevention using input validation -->
<script>
    function validate() {
        var bool = /^[-\w\.\$@\*\!]{1,10}$/.test(document.getElementById("user").value);
        var next = /^[-\w\.\$@\*\!]{1,15}$/.test(document.getElementById("pass").value);
        if (bool == false || next == false) {
            document.getElementById("form").action = "invalid.php";
        }
    }
</script>
```

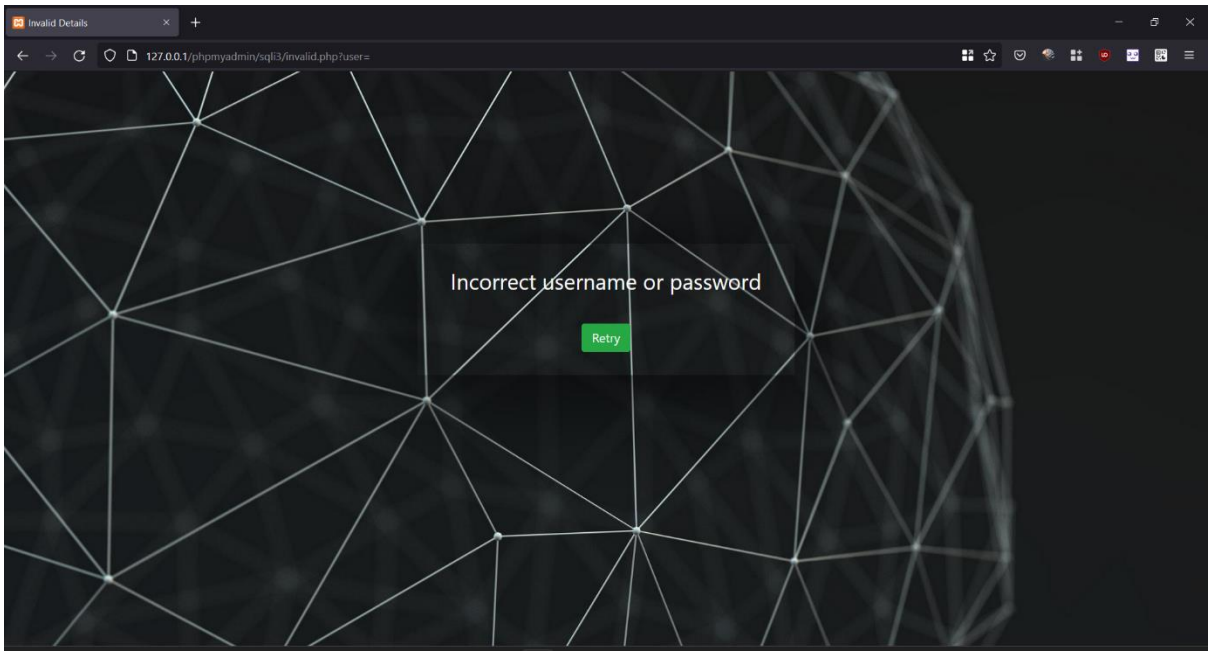
The validate function verifies the username and password. To match the inputs with the regular expression, we use this. The validation will fail even if one of both, user or pass, are false, preventing the attacker from accessing the database. Its username and password inputs are handled by bool and next, respectively. Only the special characters listed as being permitted are allowed.

An error will be raised and validation will fail if the attacker inputs something different from what is allowed, sending the user to "invalid.php" which is the error page. The password field allows for a maximum of 15 characters.

8. Screenshots:



The attacker should get access once the string for bypassing the security check is submitted. However, due to prepared statement in which before any parameters which are given by the user as an input are entered, the SQL statements are submitted to the SQL database server as well as compiled or converted by the SQL database server, mysqli_real_escape_string() where we circumvent whatever special characters that an attacker may input in the username or password fields of login form on a website and input validation, it doesn't happen.



Due to the special characters in the string that are not permitted, it displays an error and directs the attacker to the error page. This is how the prevention mechanism against SQL injection works using prepared statement, mysqli_real_escape_string() and input validation.

9. Bibliography:

- https://en.wikipedia.org/wiki/Prepared_statement
- https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php
- https://www.w3schools.com/php/php_mysql_prepared_statements.asp
- https://www.youtube.com/watch?v=_jKylhJtPmI
- https://www.php.net/manual/en/mysqli.real-escape-string.php
- https://www.w3schools.com/php/func_mysqli_real_escape_string.asp
- https://www.geeksforgeeks.org/php-mysqli_real_escape_string-function/
- https://logz.io/blog/defend-against-sql-injections/
- https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/
- https://www.red-gate.com/simple-talk/databases/sql-server/learn/sql-injection-defense-in-depth/