**LAB07 SQL injection and avoiding**

Name: Truong Dang Truc Lam Students ID: B2111933 Class: M04

**System environment for developing**

| Resources | Sender(attacker) | Receiver(victim) | Homepage |
| --- | --- | --- | --- |
| OS | Windows |  |  |
| IP address |  |  |  |
| URL |  | http://localhost/Injection |  |
| Web browser | Chrome |  |  |
| CSS language |  |  |  |
| Web server |  | Apache |  |
| Web application |  | PHP |  |
| DB server script |  | MySQL |  |
| Others |  |  |  |

**[Model A]**

**1. Search web application database query model for SQL injection test**

SQL injection occurs when malicious code is inserted into an SQL statement, potentially altering its intended behavior. This can lead to unauthorized access, data modification, or even system compromise.

Common SQL Injection Techniques:

* **Single-quote injection:** Adding a single quote to the input can break the SQL statement: SELECT \* FROM users WHERE username = 'admin' OR '1'='1';
* **Numeric injection:** Injecting numbers that can be interpreted as SQL code: SELECT \* FROM users WHERE user\_id = 1 OR 1=1;
* **Union-based injection:** Using the UNION operator to combine multiple queries: SELECT \* FROM users WHERE user\_id = 1 UNION SELECT \* FROM users;
* **Error-based injection:** Exploiting error messages to extract information: SELECT \* FROM users WHERE user\_id = 1 AND 1/0=1;

**2. Code full process of web application database query module for SQL injection test**

This is a vulnerable code from a web application:

import mysql.connector

def execute\_query(query, params):

try:

db = mysql.connector.connect(

host='your\_host',

user='your\_user',

password='your\_password',

database='your\_database'

)

cursor = db.cursor()

cursor.execute(query, params)

results = cursor.fetchall()

cursor.close()

db.close()

return results

except mysql.connector.Error as e:

print(f"Error: {e}")

return []

user\_input = "admin' OR 1=1 --"

query = "SELECT \* FROM users WHERE username = '%s'" % user\_input

results = execute\_query(query, [])

Expected Output: All users from the database.

**3. How to prevent and protect SQLI**

### 1. Parameterized Queries and Prepared Statements:

* Use prepared statements: This separates the SQL query from the data, preventing direct injection of malicious code.
* Bind parameters: Bind user-provided values to placeholders in the query, ensuring they are treated as literals.

### 2. Input Validation:

* Validate user input: Check for invalid characters, data types, and length restrictions.
* Sanitize input: Remove or escape special characters that are used for injection.

### 3. Error Handling:

* Avoid revealing sensitive information: Do not return detailed error messages that could be exploited by attackers.
* Log errors: Log errors to track potential attacks and identify vulnerabilities.

### 4. Least Privilege Principle:

* Grant minimal permissions: Ensure that database users have only the necessary privileges to perform their tasks.

### 5. Security Audits and Testing:

* Regular audits: Conduct regular security audits to identify vulnerabilities and weaknesses.
* Penetration testing: Simulate attacks to assess the effectiveness of your security measures.

**4. Code full process of web application database query module for avoiding SQL injection**

Web Application Database Query Module (Safe):

import mysql.connector

def execute\_query(query, params):

try:

db = mysql.connector.connect(

host='your\_host',

user='your\_user',

password='your\_password',

database='your\_database'

)

cursor = db.cursor()

cursor.execute(query, params)

results = cursor.fetchall()

cursor.close()

db.close()

return results

except mysql.connector.Error as e:

print(f"Error: {e}")

return []

# Example usage with prepared statements

user\_input = "admin' OR 1=1 --"

query = "SELECT \* FROM users WHERE username = %s"

results = execute\_query(query, (user\_input,))

print(results)

Expected result: [('admin', 'password', 'other\_data')]

**5. Explain your test environment of software for test**

I use XAMPP, including Apache, PHP and MySQL. This is my code that prevent the login function from SQLI:

<?php

require '../vendor/autoload.php';

require\_once "config.php";

require\_once "session.php";

$error = '';

if(isset($\_POST['login'])){

$username = trim($\_POST['username']);

$password = trim($\_POST['password']);

$remember\_me = isset($\_POST['remember-me-box']) ? 1 : 0;

$stmt = $db->prepare("select \* from acc where idacc = ?");

$stmt->bind\_param("s", $username);

$stmt->execute();

$result = $stmt->get\_result();

$row = $result->fetch\_assoc();

if (mysqli\_num\_rows($result)>0){

$password = hash("sha256", $password);

$validpass=$row['pass'];

if(strcmp($password, $validpass)==0) {

$\_SESSION['idacc']= $row['idacc'];

$\_SESSION['name']= $row['name'];

$\_SESSION['admin']= $row['admin'];

$\_SESSION['darkmode']= $row['darkmode'];

if ($remember\_me)

setcookie("idacc", $\_SESSION['idacc'], time() + (86400 \* 30), "/", "", true, true);

else

setcookie("idacc", $\_SESSION['idacc'], 0, "/"); // Expires with session

if($\_SESSION['admin']){

header("location: ../admin/index.php");

exit;

}

else{

header("location: ../user/index.php");

exit;

}

}

else{

$error = "Not a valid password.";

}

}

else{

$error = "The username does not exist.";

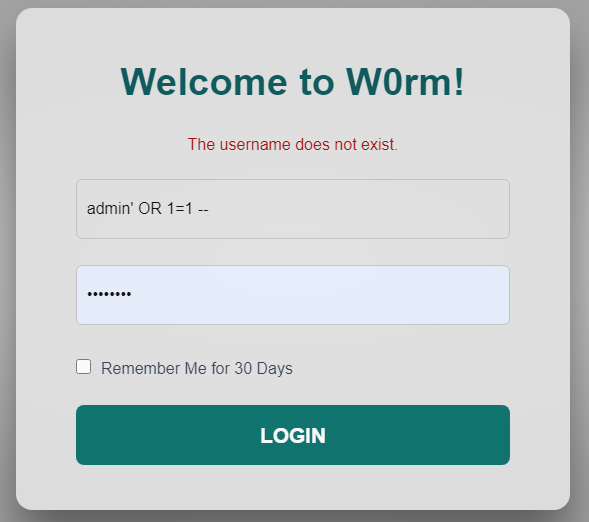
}

mysqli\_close($db);

}

?>

Let's check the result:



This is the result

==================================================================

Reference site

<https://www.w3schools.com/sql/sql_injection.asp>

<https://developer.okta.com/blog/2020/06/15/sql-injection-in-php>

<https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/>

<https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection>