Database Systems Lab

Lab # 05

SQL Injection

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2005

Semester Spring 2022

Department of Computer Science,
National University of Computer and Emerging Sciences FAST
Peshawar Campus

Contents

SQL Injection:

SQL injection refers to the act of someone inserting a MySQL statement to be run on your database without your knowledge. Injection usually occurs when you ask a user for input, like their name, and instead of a name they give you a MySQL statement that you will unknowingly run on your database.

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

Look at the following example which creates a SELECT statement by adding a variable ($txtUserId) to a select string. The variable is fetched from user input through POST method.

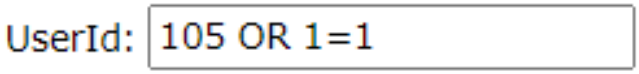
```
$txtUserId = $_POST["UserId"];
$txtSQL = "SELECT * FROM Users WHERE UserId =
$txtUserId";
```

The rest of this section describes the potential dangers of using user input in SQL statements.

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:



Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

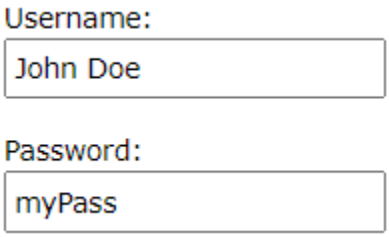
The SQL statement above is much the same as this:

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

SQL Injection Based on ""="" is Always True

Here is an example of a user login on a web site:



Example

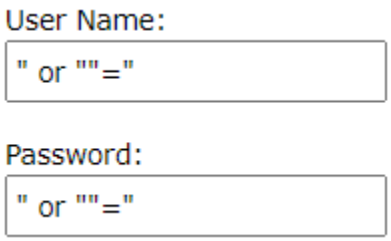
```
$uName = $_POST["username"];
$uPass = $_POST["userpassword"];

sql = “SELECT * FROM Users WHERE Name ='$uName’ AND Pass =' $uPass”
```

Result

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:



The code at the server will create a valid SQL statement like this:

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since **OR** ""="" is always TRUE.

SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

Example

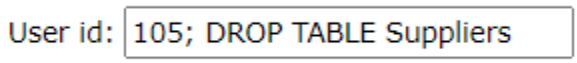
```
SELECT * FROM Users; DROP TABLE Suppliers
```

Look at the following example:

Example

```
$txtUserId = $_POST["UserId"];
$txtSQL = "SELECT * FROM Users WHERE UserId = $txtUserId";
```

And the following input:



The valid SQL statement would look like this:

Result

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

The following examples shows how to build parameterized queries in PHP.

```
$user_id = $_POST[“User_id”];

$query_1 = $conn->prepare("select * from profile where user_id = ?");
$query_1->execute([$user_id]);
```

```
$user_id = $_POST[“User_id”];
$first_name = $_POST[“first_name”];
$last_name = $_POST[“last_name”];

$query_1 = $conn->prepare("insert into profile values(?,?,?)");
$query_1->execute([$user_id, $first_name, $last_name]);
```

```
$user_id = $_POST[“User_id”];
$first_name = $_POST[“first_name”];
$last_name = $_POST[“last_name”];

$query_1 = $conn->prepare("update profile set first_name = ?, last_name = ?
where user_id = ?");
$query_1->execute([$user_id, $first_name, $last_name]);
```

Parameterized queries can be used for any situation where untrusted input appears as data within the query, including the `WHERE` clause and values in an `INSERT` or `UPDATE` statement.