SQL INJECTION

CONTENTS

### A Brief Tutorial of SQL

### Interacting With Database in Web Application

### Launching SQL Injection Attacks

### The fundamental Cause

### Countermeasures

### Summary

# A Brief Tutorial of SQL

SQL stands for Structured Query Language. It is the language that is used to communicate with a relational database management system such as MySQL. A Relational Database contains multiple tables usually named for an entity type, with columns to store attributes for that entity.

Here is an example of an employee table-Table

Description automatically generated

The power of a relational database is that it has the ability to link multiple tables together. Every relational database needs a column that can serve as a unique ID for that table. Sometimes databases can be configured in a manner to do that automatically for you using the auto increment. In the example above the unique ID could be the Employee ID because employees could have the same name, but each employee would have a different ID. A key uniquely identifies a row within a table. Keys of other tables can be references to create links between data. The Relational Database Management System we will work in is MySQL. Large web applications use RDMS on the backend to store large amounts of data. When we go to a web application with a search term that information is retrieved from a database.

SQL is a standard language for issuing queries to databases and not only querying but inserting data, deleting data, adding tables, adding columns, creating users, and modifying any aspect of a database. The server side-code of a web app such as PHP will send SQL commands to a database in response to user input.

Let's start our lab environment which will be in the form of docker containers that was provided to me by my teacher. Go ahead and login to your Seed Lab environment open up a web browser and install the zip file Lab7setup.zip using the following link <https://www.cs.lewisu.edu/~perryjn/42100sp22/>

Unzip the file on our local machine and change your directory to where the .yml file is

Text

Description automatically generated

Text

Description automatically generated

Open a terminal and run the following to open up a sql command line interface

Graphical user interface, text, application

Description automatically generated

We can login to mysql using the password dees-

Text

Description automatically generated

Once in we can see what databases are on the system –

Text

Description automatically generated

If you would like to use a database you could do use <databasename>;-

Graphical user interface, text, application

Description automatically generated

You can see tables by running the command show tables;

Table

Description automatically generated

The SQL query you use to search for data will always start with select. Lets log back into mySQL and use the sqllab\_users database

Graphical user interface, text, application

Description automatically generated

Graphical user interface, table

Description automatically generated

You can create a new database using the following command-

Text

Description automatically generated

Run the command use dbtest;

Run show tables;

Notice how it returns with an empty set.

Let us now create a table-

Table

Description automatically generated

We can enter in data into the tables using the INSERT command-

Table

Description automatically generated

You can add in more logic to the query by adding a WHERE clause after the table name. What goes after where is Boolean. It will match and return all the rows where that condition specified in the query is true-

Diagram

Description automatically generated

You can do non-exact matches using like-

Diagram

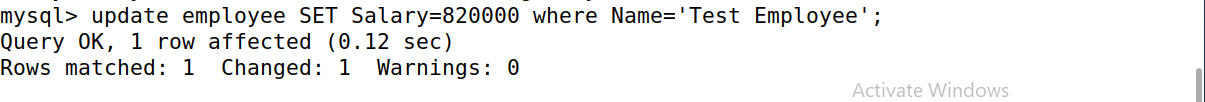
Description automatically generated with low confidence

You can use OR and and NOT in your queries like this-

A picture containing chart

Description automatically generated

Here is a query an attacker could craft in an attack that will return every row in the databases. 1=1 is a Boolean condition that is always true. Without any knowledge of what's in the rows in the database it will match every row in the database. -

You can update columns like this-

Select a single column-

Text

Description automatically generated

# Interacting With Database in Web Application

In web applications the end user never directly interacts with a database. The web application code will execute SQL statements into the database on the users behalf. Take the example where the user is creating profile information in that scenario the web app code could be using an insert sql statement. The Web app code is the intermediary between the end user and the database-

Chart

Description automatically generated

If the application is coded insecurely the end user could input data into the web application that could give them access to the backend database. This is the basis of a sql injection.

Let us move on to PHP which is a server-side language. The clients never really see the PHP code. The server gets the PHP page and it runs the PHP code on the server and replaces that code with its output.

Let's write a PHP app from scratch that uses an HTML form to get an employee's ID and password, then queries the database to get that employee’s salary.

A SQL query can be constructed as a string in PHP code (this is not wise from a security point of view)

Important lines of code to understand

$conn = new mysqli($dbhost, $dbuser,$dbpass, $dbname); //returns new connection object

We could then call methods on the $conn object by using ->

$result = $conn->query($querystring); //send the query and store the result object

We must remember that a SQL query may return multiple results as rows. To process it, we loop through the rows:

While($row= $result->fetch\_assoc()) { //as long as there's more data $row won't be null … }

There is a variable built into php that is an array $\_GET. It understands php will have all variables of a get request that we can pull from here. Get request sends variable data in the URL to the server PHP has programmatic access to this.

Take this example-

Graphical user interface, text, application

Description automatically generated

How could one pass variables to this get request? You could do it by making a form, or it could be a part of the URL, like this-

Graphical user interface, text, email

Description automatically generated

This is how we can setup a database connection and construct a query as a string in php-

<! -- Demonstrating a PHP app to query a SQL database -->

<html>

<body>

Hello, this is a webpage.

<?php

// first get the user vars from the GET

$eid = $\_GET['EID'];

$pwd = $\_GET['Password'];

//printf("EID given was: $eid, Password was: $pwd");

//setup the db connection

$dbhost = "10.0.9.6";

$dbuser = "root";

$dbpass = "dees";

$dbname = "dbtest";

$conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);

//construct the query string based on the user input

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='$eid' and Password='$pwd'";

// issue the query to the database.

$result = $conn->query($querystring);

// loop through the result rows and print them.

if ($result) {

while ($row = $result->fetch\_assoc()) {

printf("Name: %s -- Salary: %s -- SSN: %s\n",

$row["Name"], $row["Salary"], $row["SSN"];)

}

$result->free();

}

else{

{printf("No result; bug or table doesn't exist."); }

}

$conn->close();

?>

</body>

</html>

That code returns this-

Graphical user interface, text

Description automatically generated

Let us add a form now to this php app-

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

It returns the following when filling out the form-

Graphical user interface, text, application, email

Description automatically generated

# Launching SQL Injection Attacks

Input that is fed to a command processing system such as SQL can contain meta characters that prematurely end the input and allow direct commands to be issues to the system

EXAMPLE- Bob’; drop table suppliers

The code we wrote has no input sanitization.

Lets do a SQL injection on the web app we wrote.

We need to enter something into the EID and password fields that will

1. Give us an employee’s salary and ssn without the password, with just the eid

I entered in the following- Graphical user interface, text, application

Description automatically generated

In the EID box we enter in the employee ID and a hash mark. That comments out the rest of the query after that. Which part of the code does this effect?-

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='$eid' and Password='$pwd'";

What we entered in the form changes the query to

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='EID5001'; # ‘ and Password='$pwd'”;

1. Can we get every employee’s salary and SSN?

YES WE CAN!!

Graphical user interface, text, application, email

Description automatically generated

Notice it the query we crafted returned every record in the database and for both the first and second injections we do not have to enter anything for the password since the # allows us to bypass the rest of the query by treating it as a comment.

We do not even have to enter an ID-

Graphical user interface, text, application, email

Description automatically generated

We now change the query to-

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='whatever'; OR 1=1 # and Password='$pwd'”;

We have failed to separate data and code with the way we have wrote this program. We take input directly and concatenate it with a query string, stitch it together and feed it to a database. Our data turns into code. This is bad!! We put metacharacters in the input to escape from the environment and turn data into code. We take in two strings from the user in the following code(the bold is where out input goes)-

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='**$eid'** and Password='**$pwd**'";

Meta characters are characters that are part of the command language rather than the input data. The important meta character in the following code is the apostrophe like the one after ‘$eid’-

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='**$eid'** and Password='**$pwd**'";

This apostrophe is what separates user input from the command part of the query. Inside the two single quotes is a string from the user outside of the double quotes is the command language itself. This is why we use the apostrophe in our attack when we enter in this data.

Let's take a look back at this example-

If this was entered in the username it would delete a table from the database.

The query would become-

$querystring = "SELECT Name, Salary, SSN

FROM employee

Where EID='**Bob'; drop table suppliers** and

Password='**$pwd**'";

SQL doesn’t know that the apostrophe put in by the user is not the apostrophe ending this string. The quote metacharacter in the input closes the string and causes what is after to be interpreted as a command.

# The Fundamental Cause

Mixing data and code together is the cause of several types of vulnerabilities and attacks. The issue with this type of vulnerability is that we mix two pieces of information together. One comes from the user which is untrusted and the other piece is provided by the program which is usually trusted. Once mixed boundaries between the two disappear. As an example, in SQL injection cases when constructing a SQL query the developer knows where the user data should be placed inside the SQL statement. When the user data is merged into the SQL statement, if the data contains keywords or characters reserved for code, they will alter the original boundaries between code and data.

Secondly after 2 pieces of information are mixed the result is passed to a parser. SQL uses a SQL parser for XSS it uses the HTML parser. The parser needs to separate code and data so that it can execute the code. If the data contains keywords or special characters it will be treated as code.

# Countermeasures

There are three approaches one can implement to protect against SQL injection attacks

1. Getting rid of code (Filtering)
2. Turning Code into data(encoding)
3. Clearly separating code and data

Filtering and Encoding Data-

Before mixing user data with code we can inspect the data and filter out any characters that could potentially be interpreted as code. For example, the apostrophe is used in SQL injection attacks so if we can get rid of it or encode it then we can effectively prevent the parser from treating it as code. Encoding a special character tells the parser to treat the encoded character as data not as code.

Example- 

PHP’s mysqli extension has a method called mysqli::realescape.string which can be used to encode the characters that have special meanings in SQL statements including NULL, carriage return (\r) newline(\n), apostrophe(‘) double quote(“), underscore(\_), percentage(%), backslash(\)

IMPORTANT- The filtering or escaping approach does not address the fundamental cause of the problem. Data and code are still being jumbled up together. This approach does make the code more secure, but escaping special characters can be bypassed by carefully constructing the injection. This would NOT be an approach that should be promoted.

The best way to prevent SQL injection attacks is to separate code from data. This way we can ensure data can never become code. For SQL statements we can send code and data in separate channels to the DB server so the DB parser knows not to retrieve any code from the data channel. We can accomplish this by using SQL prepared statements.

Example -

Text

Description automatically generated

Why prepared statements can prevent SQL injection attacks-

Using Prepared statements, trusted code is sent via a code channel. The untrusted user provided data is sent through a data channel. In this situation the database clearly knows the boundary between code and data. When it gets data from the data channel it will not parse the data. An attacker can attempt to hide code in data, but the code will not be treated as code therefore it won't execute.

# Summary

Web applications store data in databases. When data needs to be accessed the code from the application constructs SQL statements on the user's behalf. Web applications need to ensure that user input does not contain any SQL code. The best approach is to separate code from data. This way we can construct a SQL statement and send the data and code separately to the back-end database. This way we can ensure that even if the user-provided data contains code the code will only be treated as data resulting in no negative impacts on the database. We see that prepared statements can help us achieve this goal.