Concordia Institute for Information System Engineering (CIISE)

Concordia University

INSE 6961 – Graduate Seminar in Information and Systems Engineering

Graduate Seminar Report – 2

Udemy Course: Learning Ethical Hacking from Scratch

Section Covered: Website Hacking – SQL Injection Vulnerabilities

Total Time covered for this Segment: 60 Mins

Submitted to:

Professor Ayda Basyouni

Submitted By:

Nithish Reddy Yalaka – 40164619

SQL Injection:

What is SQL Injection?

SQL Injection (or SQLi) is a web vulnerability which allows an attacker to query SQL commands to the application which retrieves sensitive data of the database such as username and passwords. This attack can also be used to bypass login pages and gain unauthorized access to a website. We target an input field in the web application and pass SQL queries. If the application is not sanitizing the data, the application could be prone to SQL Injection. We can either automate this attack using a tool like SQL Map or try inserting SQL Queries manually.

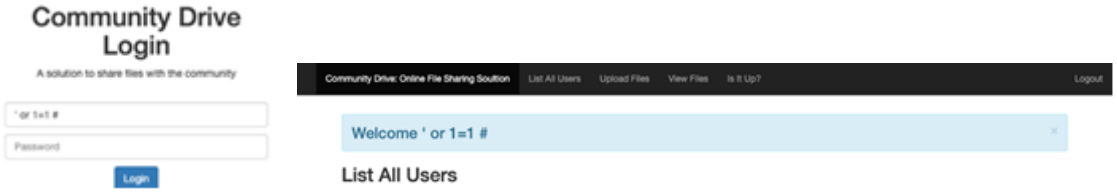
Why SQL Injection so dangerous?

SQL attacks are everywhere, even the top browser like yahoo, google gets exploited with this attack. They are very hard to protect against and it's very easy to make a mistake to make these exploits available for exploitation. The other reason that they're very dangerous is that they give you access to the database, and the attacker has everything he needs, and there is no need to do reverse php shell also. SQL injection can be used to read files outside the ww route. They are also used to upload a PHP shell. Overall, SQL injections are very dangerous and very useful for the attackers if they fine one.

Discovering SQL Injection in POST

We must break each and every page in order to find SQL injections. o Try to inject some stuff here anytime you see a text box or a parameter on this webform, for example, home.php. So, to break up the page and make it look distinct, try using a single quotation, and, or an order by statement. When you add a comment, everything that comes after it will be ignored. We can use the hash as the comment, which means that everything entered after the hash will be ignored by the system. So, we effectively slashed the password, and after inserting the one we did, the system will believe we provided the correct username and password. As a result, we should be able to login to the page after pasting the code into the form. Because we'll enter a password, close the quote, and then paste the code we want to run on the machine right here, which will be run on the target system.

Payload: Trying blind SQL injection (' or 1=1 #) on the login page. Below screenshot shows the successful query firing and logs us into the application.



Bypassing Logins using SQL Injection:

We got an understanding from the preceding work that the system will accept any code that is fed into it. Select * from accounts, where username is equal to the username entered here, and password is equal to the password entered here, according to the statement.

Let's test whether we can use it to log in without a password, even if we don't know it, and we can do it with the admin. So, the username for admin will be admin, and the password for admin will be unknown. So, we'll type in any random password and see whether it works. So that's what we're doing, and 1 = 1 is what our code will be. After I execute this, once I inject this, this will go in here, same as before. As a result, our code will look like this: Select * from accounts, where admin is the username and AAA is the password. This is now incorrect. Alternatively, 1 Equals 1, which is correct. So, whenever you have an or, everything is fine if or requirement is met. That is how our statement functions. As a result, select * from the accounts and here, user is the same as admin, hence username is the same as admin. And the password is equal to AAA, which is incorrect. Alternatively, 1 = 1, which I'll accept. Then it'll actually execute it, allowing us to log into the admin without even knowing the admin password. So, let's try injecting this right now. Now, based on the code we've written on the webform and how you want to implement it, and how you're visualizing the code, bypassing, and logging may be done in a variety of ways. When you place a single quote here, you won't see this notice in most circumstances. When you do it, it can sometimes make your job a lot easier. If you don't, you'll have to make some assumptions as to what it looks like.

Reading Information from Database using SQL Injection:

Let's have a look at how many columns are picked by default on this page, as well as how much information is selected and shown on this page. We're going to use the order by command to do this. Let's try ordering some numbers by one and see what happens. When we ordered by one, we got something that was fine, but when we ordered by 100,000, we got an error. Let's try ordering by ten and see if we get an error, then order by five and see if it works, and order by six and we get an error. So, we know there are five columns that are being taken from a specific database, which is the accounts table and displayed on this page, by doing this. Now, let's see if we can create our own choose statement and have it run on the target computer. Right now, the statement is pick * from accounts where the username is zaid, and we're going one by one. So, let's see if we can correct this and get it to choose something we like. we'll use this format to choose things normally, but because we're trying to do many selects from the URL, we'll have to use a union first and then say select. Then we need to visualize what's going on with this app. Now we know there are five records being picked in this web application. As a result, five columns have been chosen.

Now, we're performing one, two, three, four, and five, as directed by the command. So, let's see what happens if we execute this and it combined that selection with another selection and showed us something else, and as you can see here, we're only seeing two, three, and four, which means that whatever value you put in a number two, three, or four, whatever you want to select, if you put it in there, it'll be displayed in this page in this specific location, and you can also see that you have results for two. So, whatever you type in box two will also appear in box three. Let's see what we can do with the database. Instead of number two, type database, and instead of number three, type username or user, and finally, type database version, and this will choose the current database, the current user privileges, and the database version. The login appears to be owasp10, which was the second option, so that's the database we're looking for. owasp10 is the database's name. We're currently logged in as root@localhost, which means we're the root user. We're only

concerned with this and what we've injected, and we've injected the version, which we can see is MySQL 5.0.51. We now know that the database to which we're linked is owasp10. Each database is usually assigned to a certain user in most real-world scenarios.

Normally, you can only choose items, tables, columns, and data from the current database. Because we logged in as root, the web application linked with it is connected to the data base via the root connection, allowing us to access other databases, but this will not be the case in real-life settings.

Discovering Database Tables using SQLI:

owasp10 is our target database, based on the preceding conclusion. Let's try to figure out what tables are in that database. So, this is our select statement, which is a union select 1 with these items. We can either leave this or change these to null because we're only going to select one thing now and set it to null, and number two, we're going to select table name from Information Schema, which is MySQL's default database and contains information about all other databases. As a result, we're going to choose the table name from information schema.table. In this case, we're picking a table called tables from a database called information schema, and the column we're picking is called table name. So, from the table tables, we're selecting the table name information schema from the database.

How can we prevent SQLI Vulnerabilities?

SQL injections are extremely harmful, as they are both common and easy to detect. We may locate them on some of the most well-known websites, such as Yahoo and Google. People try to prevent these vulnerabilities by applying filters, which can make it appear as if there are no exploits, but if we try harder by using encodings, multiple types of encoding, or a proxy, we can defeat most of these filters. Some programmers use a blacklist to block the use of union, which in turn prevents the usage of insert and other related functions. However, the blacklist isn't totally safe and can be circumvented. Even with the whitelist, the output will be very similar to the blacklist. The best method to do this is to program your web application in such a way that code cannot be injected and subsequently executed.

We can fix the SQL Injection vulnerability by sanitizing the input data in the username and password field. We can pass the data through strip_tags() {A PHP inbuilt function for XSS Prevention} and real_escape_strings() {A PHP inbuilt function for SQL Injection prevention with MySQL}

The ideal method to do this is to use a parameterized statement, which separates the data from the code.

References:

https://concordia.udemy.com/course/learn-ethical-hacking-from-scratch/learn/lecture/5369454#overview

Instructor : Zaid Sabih