21 Mar 2021

CS253 ASSIGNMENT 4 REPORT - SQL INJECTION - ATUR GUPTA (190203)

1. DETAILS OF THE WEAKNESS AS YOU UNDERSTAND IT?

SQL injection is a web security vulnerability that allows an attacker to get access to some information in the database by making specific queries exploiting the weak security of the website. This information is generally that data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access.

This of course can prove disastrous since the attacker will be able to see the confidential information of other users on the website, or might even access their accounts and use it for illegal things.

2. THE ATTACK MODEL UNDER WHICH YOU HAVE BUILT YOUR EXPLOIT. IS THERE ANY SPECIFIC REQUIREMENT FOR THE BUG TO BE EXPLOITED (SPECIFIC OPERATING SYSTEM, HARDWARE, LIBRARIES, ETC.)?

I created a website for this assignment that registers the new users and then show a secret message to the registered users whenever they log in.

And then exploiting the vulnerability of the website, I use SQLi attack to read the secret message by sending a special query, which has nothing to do with me being a registered user or not.

So, the requirements for this bug to be exploited are just that first, we are able to host our local website on a server and second, we are able to send queries to the server on the web page opened in a browser.

3. HOW TO RUN YOUR VULNERABLE SYSTEM TO EXPOSE THE EXPLOIT?

Below are the steps to host the website and then run the query on my system -

1. Open the nautilus as sudo to be able to paste the php files in the root directory.

2. Paste the files login.php, registration.php and server.php in the file /opt/lampp/htdocs/FirstProject (Since I'm using xampp on my system, this is where I have created a folder named FirstProject)

3. Now go to /opt/lampp and run manager-linux-x64.run to start hosting the website using the apache server. Open the app and click on start all.

4. Go to /localhost/dashboard on chromium to see if the homepage of XAMPP is visible. If yes, our server is working fine.

5. Now go to /localhost/FirstProject/registration.php. You'll see the webpage where you can register as a new user.

6. After registering with your details, go to **Log In.**

7. Fill the user details that you used just before to register yourself and click on log in. You will see the secret message now.

8. Give any random string of alphabets as username and password and convince yourself that secret message doesn't show up for just any random string.

9. Now input the special string - "' or 2=2-- " (without the double quotes) as the username and any string whatsoever as the password and you'll be able to see the secret message.

Vulnerability exists.

4. EXPLAIN HOW YOUR EXPLOIT WORKS (IN THE ACCOMPANYING PROGRAM/SYSTEM).

Below is a part of the vulnerable server.php file -

$username = $_POST['username'];

$password = $_POST['password']

// PHP $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post"

$password = md5($password); // encrypting the password

$data = "SELECT * FROM user WHERE username = '$username' AND password = '$password' ";

// Query that searches a user with the given username and password in the database.

The problem in this code is that we’re taking the string provided by the user and putting it **AS IT IS** in our sql query, which can wreak havoc on our system. Simpy because the attacker now essentially can alter the file server.php to anything.

He/she just has to close the left quote by putting one quote in the starting of the 'username' query, write whatever they want and then comment out the whole line afterwards.

<u>Our Special string($username) - </u>' or 2=2--

After replacing our special string in the query with $username, we get -

$data = "SELECT * FROM user WHERE username = '' or 2=2-- AND password = '$password' ";

Now, notice what it has done.

First quote of special string is working as the closed quote of username and, **or** keyword results in true since the statement at right of it is true i.e **2=2.**

-- comments out the rest of the line.

So new query becomes -

$data = "SELECT * FROM user WHERE username = '' or 2=2

So, finally $data will contain true and believe that there does exist a user with those credentials in our database. This way we have bypassed the security of our website.

5. EXPLAIN HOW TO MITIGATE THIS WEAKNESS, I.E. EXACTLY PIN-POINT WHAT WAS WRONG IN THE PROGRAM/SYSTEM AND WHAT IS THE WAY THE PROGRAM SHOULD HAVE BEEN WRITTEN.

The problem is with how we store the query given by the user in our variables.

$username = $_POST['username'];

$password = $_POST['password'];

Instead of this, we should use a function provided to us by mysql itself – **mysqli_real_escape_string**()

So, we should store our user-given strings like -

$username = mysqli_real_escape_string($db, $_POST['username']);

$password = mysqli_real_escape_string($db, $_POST['password']);

// The mysqli_real_escape_string() function escapes special characters in a string for use in an SQL query, taking into account the current character set of the connection. Special Characters - NUL (ASCII 0), \n, \r, \, ', ", etc.

6. REFERENCES

A. PHP and MYSQL Tutorial

https://www.w3schools.com

B. SQL Injection

https://owasp.org/www-community/attacks/SQL_Injection

https://www.youtube.com/watch?v=ciNHn38EyRc