Lab 1 - SQL Injection

Overview

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before being sent to the back-end database servers. Many web applications take inputs from users, and then use these inputs to construct SQL queries, so they can get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. SQL injection is one of the most common attacks on web applications. In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics: - SQL statements: SELECT and UPDATE statements - SQL injection - Prepared statement

Lab Setup

We'll be using **SEED Ubuntu-20.04 VM** for this lab. Make sure you've downloaded and installed the VM on you local machine. Follow this page for detailed instructions on how to download and install this VM. Add the following entry in `/etc/hosts` file :

```
10.9.0.5    www.seed-server.com
```

If there's already an entry corresponding to `www.seed-server.com`, remove it and add the above.

Container Setup

The lab uses docker containers. Two containers will be used here, one for hosting a web application and one for the database for the web application. Inside the VM, download the *Labsetup.zip* file from here. Unzip it, enter the

Labsetup folder, and use the `docker-compose.yml` file to set up the lab environment. Run `docbuild` and `dockup` consecutively int the same folder as the `docker-compose.yml` to start the containers. Now, you need to enter into the web-application container. For this, run `dockps` and find out the container id of the web-application container. Then execute the command `docksh <container_id>` and you'll get a shell inside the container. Then inside that shell, run the following commands one after another:

```
cd /etc/apache2/sites-enabled/
rm 000-default.conf
service apache2 restart
```

Now if you hit `10.9.0.5` in firefox, you'll see the vulnerable web application. For more details on working with docker, check this.. As its your very first lab, its highly recommended that you get yourself familiar with the docker commands. You'll need those in upcoming labs.

About the Web Application

We have created a web application, which is a simple employee management application. Employees can view and update their personal information in the database through this web application. There are mainly two roles in this web application: Administrator is a privilege role and can manage each individual employees' profile information; Employee is a normal role and can view or update his/her own profile information.

Tasks

Task 1

We will use the login page from `10.9.0.5` for this task. The login page asks users to provide a user name and a password. The web application authenticate users based on these two pieces of data, so only employees who know their passwords are allowed to log in. Your job, as an attacker, is to log into the web application without knowing any employee's credential.

To help you started with this task, we explain how authentication is implemented in the web application. The PHP code `unsafe_home.php`, located in the`/var/www/SQL_Injection` directory, is used to conduct user authentication. The following code snippet shows how users are authenticated.

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
    FROM credential
    WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

```
$result = $conn -> query($sql);

// The following is Pseudo Code
if(id != NULL) {
    if(name=='admin') {
        return All employees information;
    }
    else if (name !=NULL){
        return employee information;
    }
}
else { Authentication Fails;
}
```

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the `credential` table. The SQL statement uses two variables `input_uname` and `hashed_pwd`, where `input_uname` holds the string typed by users in the username field of the login page, while `hashed_pwd` holds the sha1 hash of the password typed by the user. The program checks whether any record matches with the provided username and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

Your task is to log into the web application as the very first user in the table which should be **Alice**. But the constraint is you have to assume that *you do not know any of the usernames*. So, you can't use an username directly here. Write the input that you gave in Username and Password field in your report. Also attach the ss after logging in.

Task 2

Assume that you know the number of fields in the `credential` table and also the name of the fields/columns. But you don't know anything about the ordering of the columns. By exploiting the SQL injection vulnerability, how can you determine what's the column number of the `name` field? Write the input that you need to provide in the login page to achieve the mentioned task and also explain why that would work.

Task 3

How can you determine which version of mysql is being used by the web application? And what is the version?

Task 4

Assume the database contains DES-encrypted value for the *password* and *eid* columns. In mysql, `DES_ENCRYPT()` function can be used to encrypt a string with DES encryption technique. The format of the function is: `DES_ENCRYPT(plaintext_string_to_encrypt, [key_number | key_string])`. In the second argument, you can either provide a `key_number` which is a value between 0 to 9 to select a key from DES key file or a `key_string` with which the plaintext will be encrypted. Now, consider the following SQL statement which is sent to the database where the *password* and *eid* values are given by users. Is the given statement vulnerable to SQL Injection? If so, how you can exploit the vulnerability.

```
$sql = "SELECT * FROM employee
        WHERE eid=DES_ENCRYPT('$eid', 6) and password=DES_ENCRYPT('$passwd', 'key')";
```

Task 5

With reference to the previous problem, the encryption is now being done inside PHP code using `openssl_encrypt()` function instead of doing it inside the SQL statement. Does this altered program have SQLi vulnerability? Why or why not?

```
$method = "DES-EDE3";
$key = "1569";
$options = 0;

// transform the key from hex to string
$key = pack("H*", $key);

// encrypt
$encrypted_eid = openssl_encrypt($eid, $method, $key, $options);
$encrypted_pwd = openssl_encrypt($passwd, $method, $key, $options);

$sql = "SELECT * FROM employee
        WHERE eid='$encrypted_eid' and password='$encrypted_pwd'";
```

Task 6

What can be the values of `$eid` and `$passwd` to exploit the SQLi vulnerablity in the following multiline SQL statement?

```
SELECT * FROM employee
WHERE eid= '$eid' AND
password='$password'
```

Hint: *Look up on different commenting techniques in mysql.*

Task 7

A disgruntled employee wants to take revenge on his boss whose name is `Reddington`. He found a way to exact his revenge through the password reset menu of the company website. He wants to set his boss's salary to `1$` and also set `Reddington`'s password to a known value so that he can later log back in as `Reddington`. Given the below SQL statement, how can that employee achieve his goal? Note that, the values in `$eid`, `$name`, `old_pwd`, `new_pwd` are controlled by the user and `Salary` is the name of the field where salary info is stored.

```
$hashed_newpwd = hash('sha256', $new_pwd);
$hashed_oldpwd = hash('sha256', $old_pwd);

$sql = "UPDATE employee
        SET name='$name', password='$hashed_newpwd'
        WHERE eid = '$eid' and password='$hashed_oldpwd'";
```

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not result in full marks.