SQL Injection Lab

1. 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 a user's input is not correctly checked within the web application before being sent to the back-end database servers.

Many web applications accept inputs from users and use these inputs to construct SQL queries in order to access or update information in databases. When the SQL queries are not carefully constructed, SQL injection vulnerabilities can result. SQL injection attacks are one of the most frequent attacks on web applications.

For this lab, we modified a web application called Collabtive, disabling several countermeasures implemented by Collabtive. As a result, we created a version of Collabtive that is vulnerable to SQL injection attacks. Although our modifications are artificial, they are representative of mistakes made by many web developers. Your goals in this lab are to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attacks, and master techniques that can help defend against such attacks.

2. Lab Environment

The name of the VM image that supports this lab is called SEEDUbuntu12.04, built in September 2013. If you have a pre-built VM image prior to SEEDUbuntu12.04, you must download the new version from the SEED web site:
http://www.cis.syr.edu/~wedu/seed/lab_env.html

2.1 Environment Configuration

The lab requires three software packages, all of which are already installed in the provided VM image: 1. The Firefox web browser 2. The Apache web server 3. The Collabtive project management web application

In Firefox, you may want to use the LiveHTTPHeaders extension to inspect the HTTP requests and responses. The pre-built Ubuntu VM image provided to you has the Firefox web browser with the required extensions already installed.

Starting the Apache Server. The Apache web server is also included in the pre-built Ubuntu image. However, the web server is not started by default. You need to first start the web server

using the following command:

```
% sudo service apache2 start
```

The Collabtive Web Application. We use an open-source web application called Collabtive in this lab. Collabtive is a web-based project management system. It is already installed and configured in the pre-built Ubuntu VM image. We have also created several user accounts on the Collabtive server. To see all the users' account information, first log in as the admin (username admin, password admin). Once logged in as admin, other users' account information can be obtained from the User's Account Information Project on the front page.

DNS Configuration.

We have configured the Collabtive server to use the following URL:

The URL is only recognized from inside of the virtual machine, because we have modified the /etc/hosts file to map the domain name of the URL to the virtual machine's local IP address (127.0.0.1). You may map any domain name to a particular IP address by editing /etc/hosts. For example you can map http://www.example.com to the local IP address by appending the following entry to /etc/hosts:

```
127.0.0.1       www.example.com
```

If your web server and browser are running on two different machines, you must modify /etc/hosts on the browser's machine to map the domain name to the web server's IP address (*not* 127.0.0.1).

2.2 Turn Off the Countermeasures

PHP provides a mechanism called "magic quotes" to automatically defend against SQL injection attacks (this protection method is deprecated after PHP version 5.3.0). We must disable magic quotes for the first two lab tasks:

1. Go to /etc/php5/apache2/php.ini.
2. Find the line: `magic quotes gpc = On`.
3. Change it to this: `magic quotes gpc = Off`.
4. Restart the Apache server by running "`sudo service apache2 restart`".

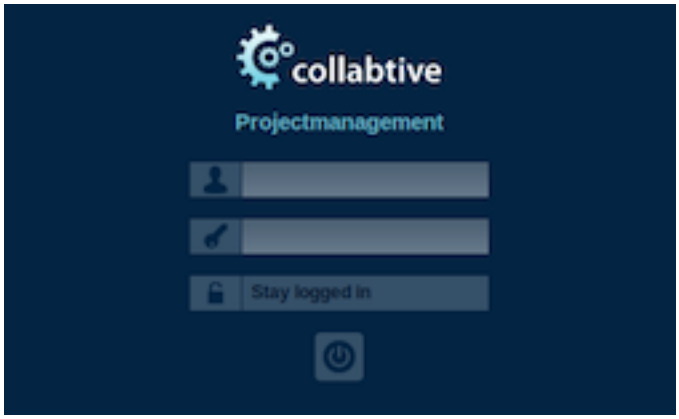
3. Lab Tasks

In this task, you need to log into Collabtive at www.sqllabcollabtive.com, *without* providing a password. You can achieve this using SQL injection. Normally, before users start using

Collabtive, they need to login using their user names and passwords. Collabtive displays the following login window asking the user to input a username and password.

Figure 2: The login window

User authentication is implemented in include/class.user.php under the Collabtive root directory (/var/www/SQL/Collabtive/). It takes the user-provided login data and determines whether it matches the user name and password fields of any record in the user database. If there is a match, the user has provided a correct user name and password combination and should be allowed to login. Like most web applications, PHP programs interact with their back-end databases using the standard SQL language. In Collabtive, the following SQL query is constructed in class.user.php to authenticate users:




```
$pass = sha1($pass);
$sel1 = mysql_query ("SELECT ID, name, locale, lastlogin,
gender,
    FROM  user
    WHERE (name = '$user' OR email = '$user') AND pass =
'$pass'");
```

In the above SQL statement, user is the name of the SQL table that holds the user data. The variable $user holds the string typed in the Username textbox, and $pass holds the string typed in the Password textbox. User's inputs in these two textboxs are placed directly in the SQL query string. If the SELECT statement returns at least one result, the user is allowed to login.

SQL Injection Attacks on Login: There is a SQL injection vulnerability in the above query. Can you take advantage of this vulnerability to achieve the following objectives?

Task 1.1: Can you log into another person's account without knowing the correct password?

Task 1.2: Can you find a way to modify the database (still using the above SQL query)? For

example, can you add a new account to the database, or delete an existing user account? Obviously, the above SQL statement is a query-only statement, and cannot update the database. However, using SQL injection, you can turn the above statement into two statements, with the second one being the update statement. Please try this method, and see whether you can successfully update the database.

To be honest, we are unable to achieve the update goal. This is because of a particular defense mechanism implemented in MySQL. In the report, you should show us what you have tried in order to modify the database. You should find out why the attack fails and what mechanism in MySQL has prevented such an attack. You may look up evidence (second-hand) from the Internet to support your conclusion. However, first-hand evidence will get more points (use your own creativity to find first-hand evidence). If you find ways to succeed in the attacks, you will be awarded bonus points.

What to submit:

1. A screenshot of completing task 1.1 and the command that you used. Include a short report of your thought process and experimentation.

2. Your thought process trying to update a password.

Useful Sources:

https://www.w3schools.com/sql/ SQL tutorial

https://www.w3schools.com/php/ PHP tutorial