

# SQL Injection Attacks (CS 915)

## 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 the 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. Your 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 types of attacks.

## Lab setup

- Download the lab setup files from [here](#) (save it at any folder under /home/seed **EXCEPT** the shared folder if you use Virtual).
- Read the manual on docker [here](#) (for background learning)

This lab is based on the [SQL Injection Attack lab](#) with some adaptations.

## Setting up the Lab environment

We have developed a web application for this lab, and we use containers to set up this web application. There are two containers in the lab setup, one for hosting the web application, and the other for hosting the database for the web application. The IP address for the web application container is 10.9.0.5, and The URL for the web application is <http://www.seed-server.com/>.

## Build and start the containers

Under the **Labsetup** folder, run the following

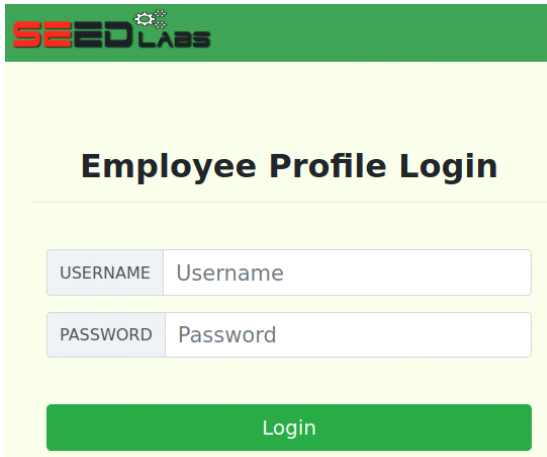
```
$ dcbuild
$ dcup
```

## Add the following entry to /etc/hosts

10.9.0.5      www.seed-server.com

## Testing

Visit the following URL. <http://www.seed-server.com/>. If the setup is correct, you should expect to see the following web page.



## Getting Familiar with SQL Statements

The data used by our web application is stored in a MySQL database, which is hosted on our MySQL container. We have created a database called **sqllab\_users**, which contains a table called **credential**.

## MySQL database

Database Name: "**sqllab\_users**"

Table Name: "**credential**"

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

## Logging into the MySQL database

First of all, log into the MySQL database as a root user. The following gives an example.

```
$ dockps
91a658341679 mysql-10.9.0.6
19f207327dbd www-10.9.0.5
$ docksh 91
root@91a658341679:/# mysql -u root -pdees
```

**Note** - In the above example, what follows docksh is the first two alphanumeric characters shown from dockps to log into the respective shell (you can use the first one, two, or three ... alphanumeric characters as long as they uniquely identify the container). The alphanumeric characters displayed on your terminal may be different.

Once you log into the MySQL database, you can practice the following commands.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sqllab_users |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> use sqllab_users;
Database changed

mysql> show tables;
+-----+
| Tables_in_sqllab_users |
+-----+
| credential |
+-----+
1 row in set (0.01 sec)

mysql> select * from credential;
```

After playing with the MySQL commands, you can quit.

```
mysql> quit
Bye
root@91a658341679:/#
```

## Overview of the lab tasks

- Task 1: SQL Injection on SELECT Statement

- Task 2: SQL Injection on UPDATE Statement (modify database)
- Task 3: Countermeasure by using Prepared statement

## Task 1: SQL Injection Attack on SELECT Statement

### Typical PHP Code

The following is an example of PHP code that is vulnerable to SQL injection.

```
<?php
$sql = "SELECT * FROM credential
      WHERE name='$input_name' AND password='$input_pwd'"
$result = $conn->query($sql);
?>
```

### Question

If you don't know Alice's password, can you get the database to return her records? Fill in the blanks below (this is an exercise; you will practice the attack later).

SELECT \* FROM credential WHERE name=' ' AND password=' ';

### A more challenging task

If you don't know anybody's name or password, can you get the database to return some records? Fill in the blanks below.

SELECT \* FROM credential WHERE name = ' ' AND password = ' ';

### Let's try the attack on the Login Page ([index.html](#))

The HTML page is located under [Labsetup/image\\_www/Code](#)

### Employee Profile Login

USERNAME

PASSWORD

## The Code for Login page ([unsafe\\_home.php](#))

```
$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'";
```

### Task 1.1: Attack Using Web Interface

Log in to Admin's account without knowing Admin's password (**you do know Admin's Name**, which is Admin).

### Task 1.2: Using Command Line Tool

Repeat the previous attack by using a command line tool **curl** (without using the webpage).

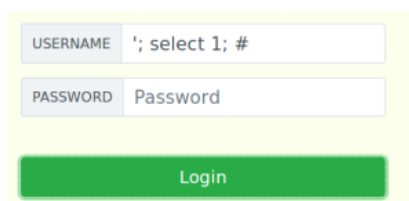
If you need to include special characters in the username or password fields, you need to encode them properly. Refer to the URL Encoding table below.

Character	UTF-8
'	%27
#	%23
space	%20

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

### Task 1.3: Append a New SQL Statement

(This task will fail due to the countermeasure, but you should observe the output)



A screenshot of a web application's login interface. It features two input fields: 'USERNAME' and 'PASSWORD'. The 'USERNAME' field contains the payload: `' ; select 1; #`. The 'PASSWORD' field contains the text: `Password`. Below the input fields is a green button labeled 'Login'.

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'select 1; #' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709' at line 3]\n

## Task 2: SQL Injection on UPDATE Statement (modify database)

Log into Alice's Account (username: alice, Password: seedalice).

Select "Edit Profile", You will see the following page (the PHP code is shown on the right).

**Alice's Profile Edit**

NickName	<input type="text" value="NickName"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

The Edit-Profile Program: `unsafe_edit_backend.php`

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

### Task 2.1: Modify Your Own Salary

Your boss did not increase your salary this year, so do it yourself (the salary field in the table is called "Salary"). Use the following work sheet to help you construct the SQL query.

nickname = '[redacted]', email='\$input\_email', ...

### Task 2.2: Modify Other People's Salary

You don't like your boss, and want to change his/her salary to -10000 dollars.

nickname = '[redacted]', email='\$input\_email', ... where ID='\$id'

### Task 2.3: Modify Other People's Password

The backend program uses a MySQL SHA1 function: `sha1()` to hash the password.

```
mysql> select sha1('seedboby');
+-----+
| sha1('seedboby') |
+-----+
| b78ed97677c161c1c82c142906674ad15242b2d4 |
+-----+
```

```
mysql> UPDATE credential SET Password=sha1('seedboby') where Name='boby';
```

Your task is to change Bobby's password to something that you know. You need to demonstrate that you have successfully changed Bobby's password by logging into Bobby's account using the new password.

```
nickname = ' ', email='$input_email', ...
```

## Task 3: countermeasure - prepared statement

The following is an example of vulnerable code.

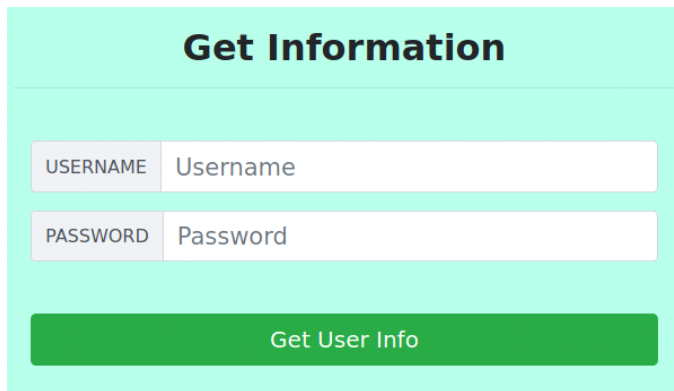
```
$sql = "SELECT name, local, gender
        FROM USER_TABLE
        WHERE id = $id AND password = '$pwd' ";
$result = $conn->query($sql)
```

A safer approach is to use prepared statement as shown below.

```
$stmt = $conn->prepare("SELECT name, local, gender
                        FROM USER_TABLE
                        WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

Patching a vulnerable program

Visit <http://www.seed-server.com/defense/>



**Get Information**

USERNAME Username

PASSWORD Password

Get User Info

Your task is to change `Labsetup/image_www/Code/defense/unsafe.php` to prevent SQL injections. Verify that your changes are effective by trying the attack again.

## After lab

- Type Ctrl-C to stop servers running in the container console.
- Under the Labsetup folder, run **dcdown** to shutdown the container properly
- Run the following (optional) to have a clean database: **sudo rm -rf mysql\_data**
- Complete the post-lab assignment.