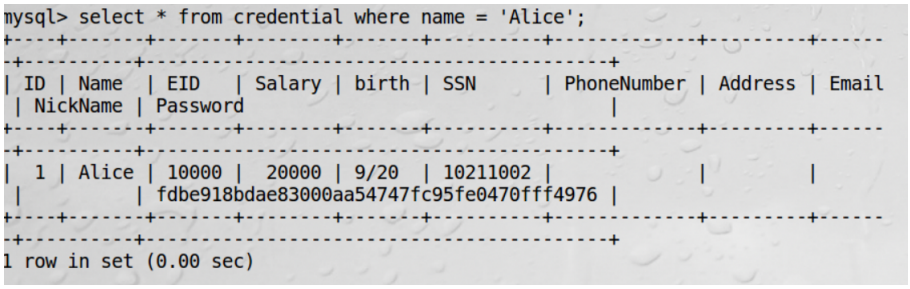
Task 1: Get Familiar with SQL Statements

The first thing we did for this lab is to open up the terminal and logged into MySQL using the userName root and password seedubuntu by using the command mysql -u root -pseedubuntu. Then we loaded the Users database, and used the command show tables and select * from credential where name = 'Alice'; to see the tables of the selected database for Alice.



```
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: SQL Injection Attack on SELECT Statement

Task 2.1 - SQL Injection Attack from webpage: For this task, we must figure out a way to login to the www.seedlabsqlinjection.com as an administrator while only knowing that the username is admin, but not knowing the password. We first analyze the pseudocode provided for how the authentication is implemented, and realized that if we use the '#' symbol at the end of the username provide, it will essentially comment out the rest of the WHERE clause constraints here: WHERE name= '$input_uname' and Password='$hashed_pwd'";. So we use the username admin'#, and that allows us to login as an administrator and see all the records for the employees.

Task 2.2- SQL Injection Attack from the command line:

For this task, we had to pretty much repeat task 2.1, but this time using the command line. We first exited out of MySQL on the terminal, then saw we could use the curl command for this task. In order to get the necessary parameters, we went to the website and analyzed the login form data using 'Inspect Element' and saw that it uses an HTTP 'get' method with the action unsafe_home.php and the parameters *username* and *password*. We can use the same method for commenting-out the need for a password by using the #' characters at the end of the username entry, but we must use the URL encoded version of these, where # = %27 and ' = %23. So our entire command becomes: curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%23&Password=' . This returns an HTML version of a table with all the values that can be seen by an

administrator and so we are successful.

```
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px
h: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item
Terminator
='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><
ss='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><butto
ick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></n
v class='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='ta
ble-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='
Id</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='
ickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th><
thead><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002<
d></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><
0</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>
/td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td></tr><tr><th s
row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></t
</td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td>
td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td><td
td><td>43254314</td><td></td><td></td><td></td><td></td><td></td></tr></tbody></table>      <br><br>
    <div class="text-center">
      <p>
        Copyright &copy; SEED LABs
      </p>
    </div>
  </div>
  <script type="text/javascript">
  function logout(){
    location.href = "logoff.php";
  }
  </script>
 </body>
```

Task 2.3 - Append a new SQL statement:

In this next task, we have to use the same vulnerability in the login page to modify the database. We will try to use the SQL injection attack to turn one SQL statement into two, with the second statement being the one that alters the database (in the case of our task, to delete a record). Since the semicolon is used to separate two SQL statements, we can run:

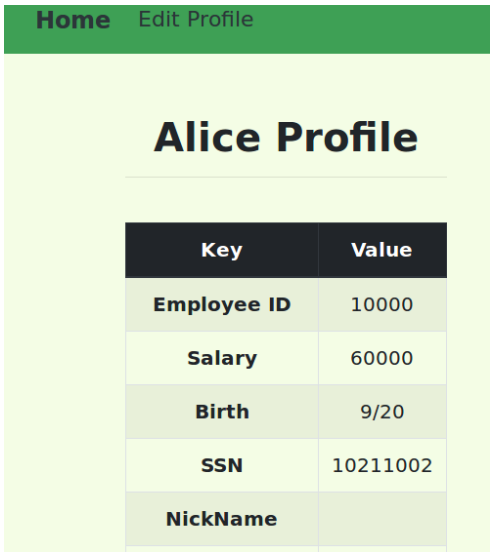
admin';Delete From credential Where Name = 'Alice';# . This did not work, as I kept getting the message that there was an error with my SQL syntax. I tried many different variations of this, but nothing was working.

> 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 'Delete From credential Where name='Alice';#' and Password='da39a3ee5e6b4b0d3255b' at line 3]\n

Task 3: SQL Injection Attack on UPDATE statement

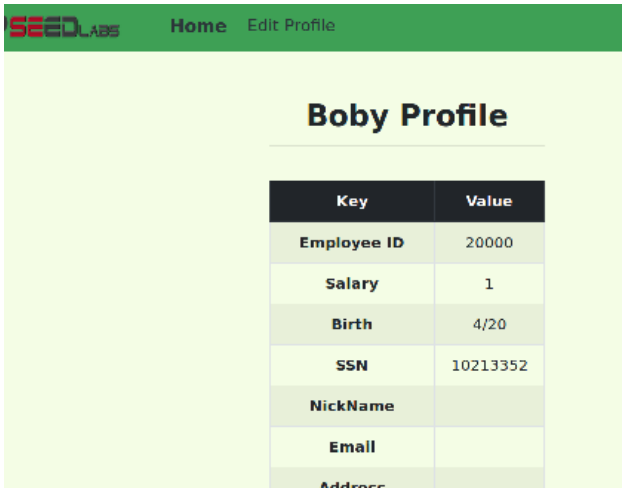
Task 3.1 - Modify your own Salary:

As shown in the Edit Profile page, employees are not authorized to change their salaries. I start by logging into Alice's employee profile with the username Alice and password seedalice, then navigate to the 'Edit Profile' section and saw that Alice's salary is listed as $20000. We ran the 'Inspect Element' feature on the 'Save' button and saw that it uses an HTTP 'get' method with the action "unsafe_edit_backend.php". We can pull up the unsafe_edit_backend.php that is stored in the /var/www/SQLInjection directory to further analyze the mysql code being used, which does not include a field for salary. We come up with a way to add this field by using ',salary='60000 command in the NickName field, which causes the Update statement to use the ' for nickname, while also using the forged salary information to update the salary field to 60000 (which comes right after the nickname field).



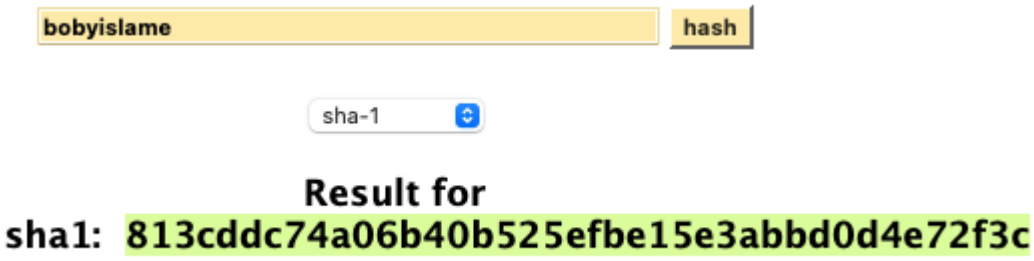
Task 3.2 - Modify other people's salary:

We can begin this attack by using the same approach to login to Boby's profile as we did for admin in Task 2.1: by using Boby'# (which effectively comments out the remainder of the Where clause which stipulates the need for a password to be entered, making it so that username is enough for authentication). This command allows us to get into Boby's profile. We can then use what we just did in Task 3.1, which is to enter the command ',salary='1 to put Boby's salary to $1.



Task 3.3 - Modify other people's password:

The first thing we can do for this task is take a look at the code snippet provided in the lab as to how passwords are stored. We see that passwords are stored in sha1 hash form, and so if we are to create a new password, we must convert it to that format. So we chose the alternate password 'bobyislame' to be used, and ran that through an sha1 hash generator, which gave us the following value:



Next, we logged into Alice's account using the username and passwords provided. Since the hashed password is still stored in the unsafe_edit_backed.pho file, we can enter a sql command in the nickname field of Alice's Edit Profile page. We can enter a command similar to what we used for changing salaries:

', Password='813cddc74a06b40b525efbe15e3abbd0d4e72f3c' WHERE name='Boby';#

Once we submitted this command, we logged out of Alice's profile, and tried the new credentials with Boby as a username and bobyislame as a password. We were able to access Boby's profile, and so we can say the attack was a success. As a way to double-check the validity of our attack, I pulled up the row for Boby's information on mysql in the terminal. We can clearly see that the new hash value shown above is

stored as Boby's new password, and so we validate that our attack was successful.

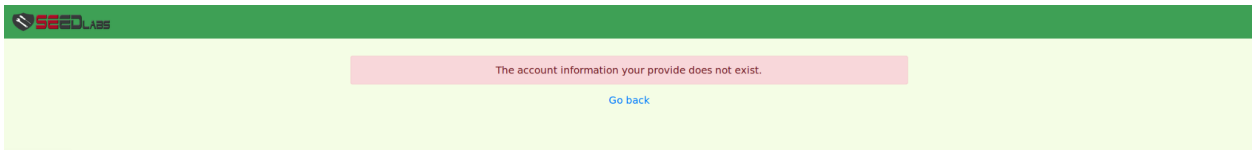
```
mysql> select * from credential where Name='Boby';
+----+------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------
------------------------+
| ID | Name | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
                       |
+----+------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------
------------------------+
|  2 | Boby | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | 813cddc74a06b40b5
25efbe15e3abbd0d4e72f3c |
+----+------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------
------------------------+
1 row in set (0.00 sec)

mysql>
```

Task 3.4: Countermeasure-Prepared Statement

For this task, We will edit the content of unsafe home.php to include the prepared statement technique. We send the SQL statement without the data first, and later the data to the database using bind_parm, preventing any SQL injection.

```
$stmt = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email, nickname, Password
FROM credential
WHERE name = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
$stmt->fetch();
```

When we attempt to replicate 2.1 Task, we see that it is no longer possible:



We can only see the user details if we use the correct user and password.