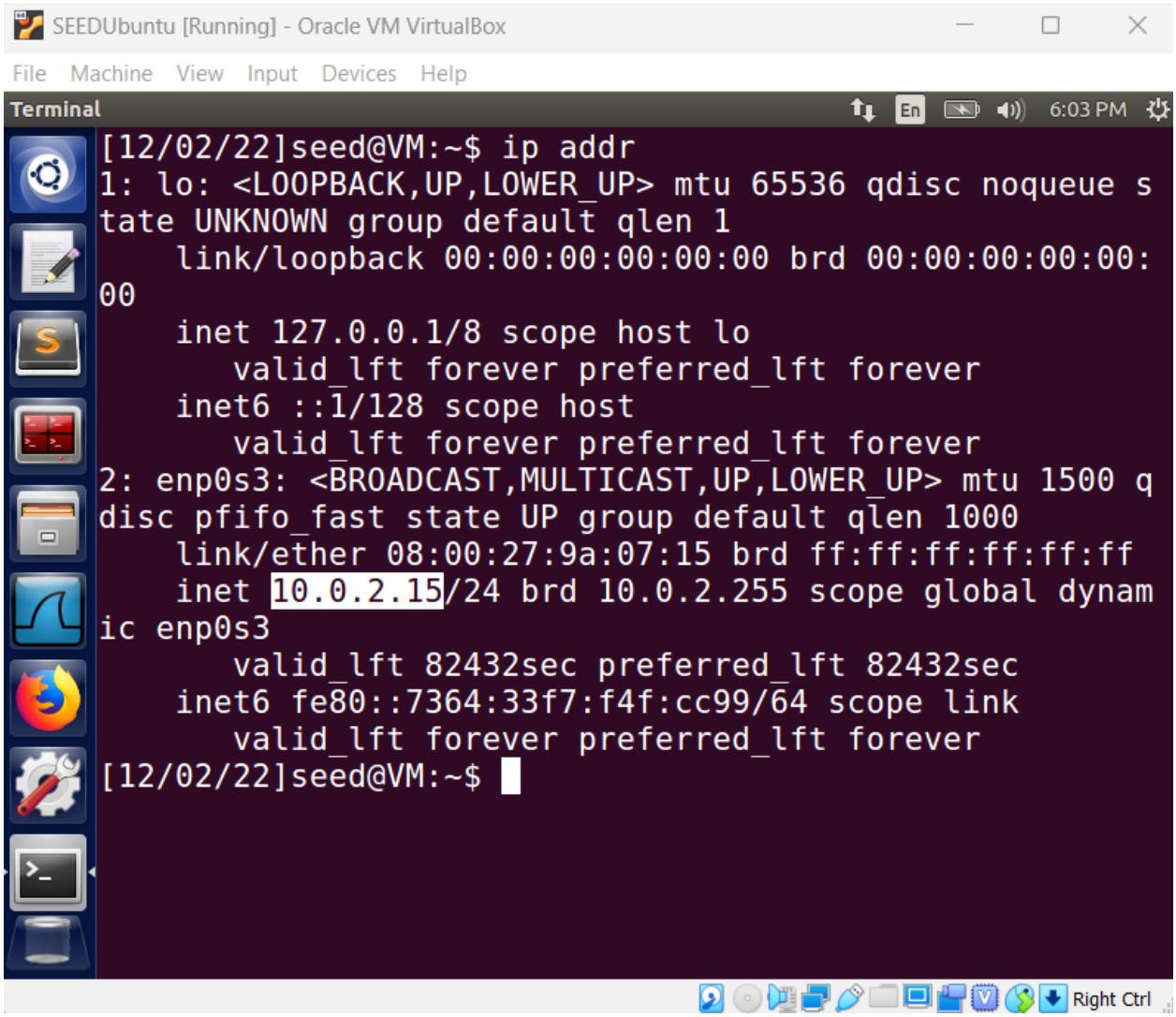
# Introduction to Information Security - CS 458 - Fall 2022 Lab 4 - SQL Injection Attack1

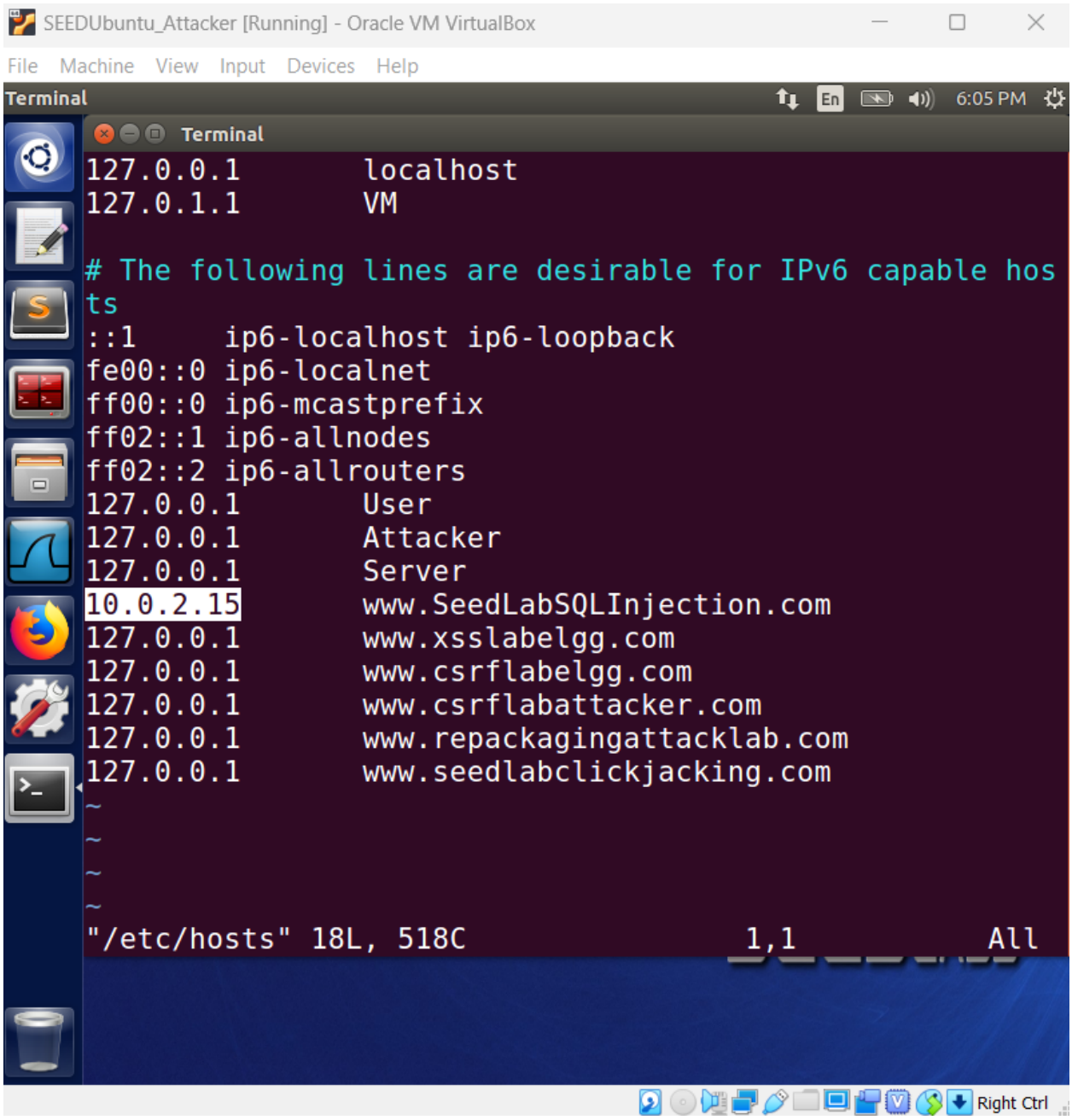
**By- Sagar Shekhargouda Patil(A20501427)**

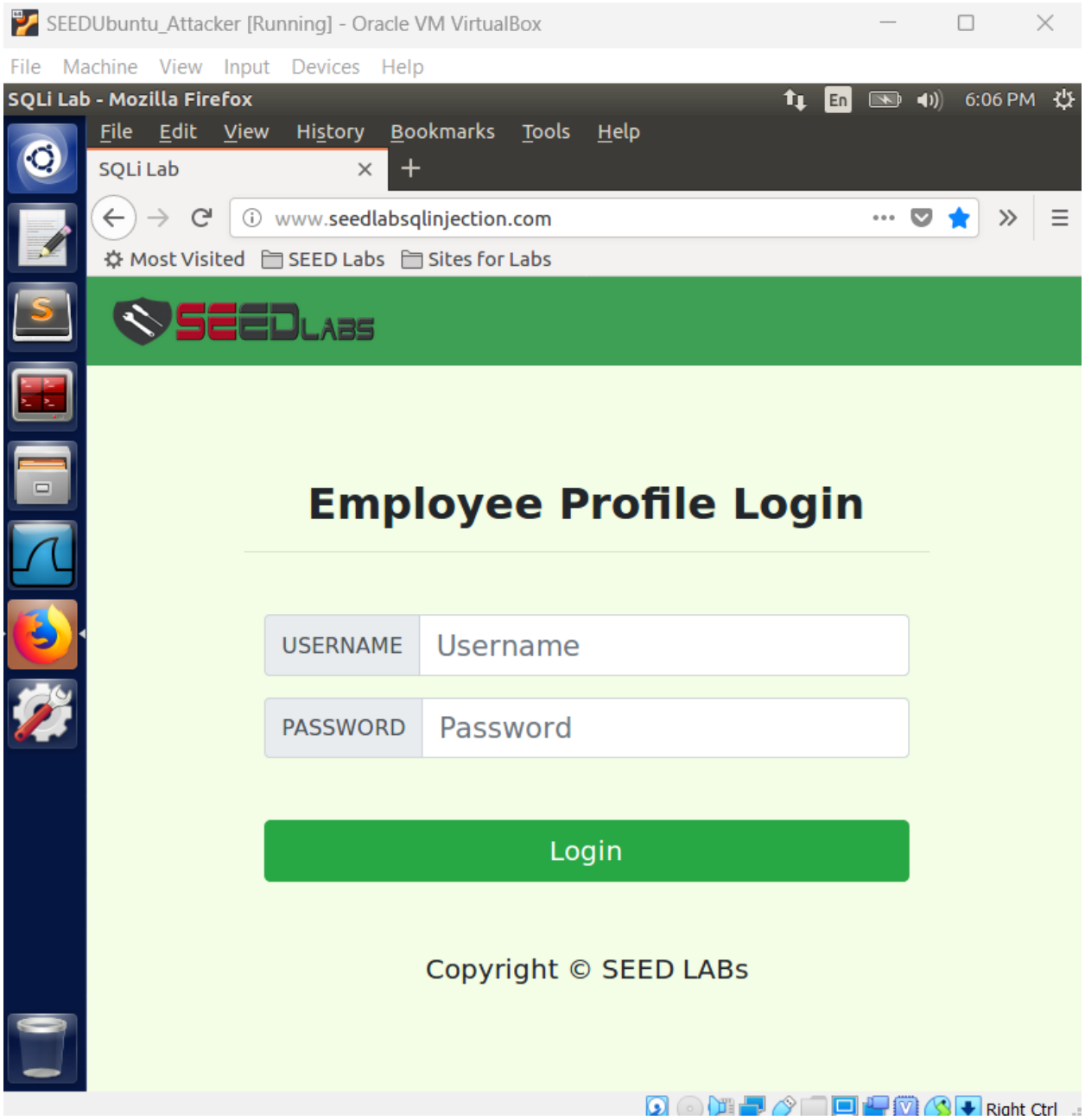
For this lab I am making use of two VMs one acting as server and the other acting as Attacker.
Attacker needs to have access to the /var/www/SQLInjection directory on the Server machine so I need to make changes in the /etc/hosts file on Attacker machine such that it points to the http://www.SEEDLabsSQLInjection.com URL points to the Server machine's IP address.

Server:



Attacker Machine:



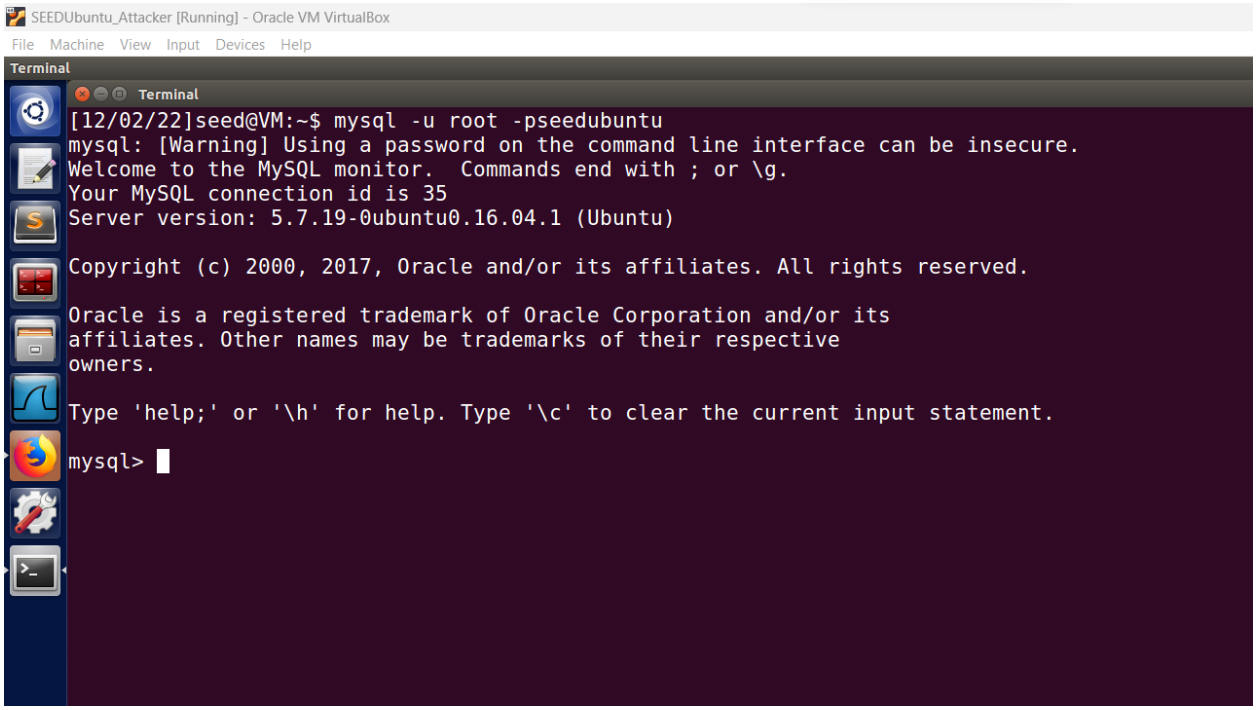



Now I am done with the setup.

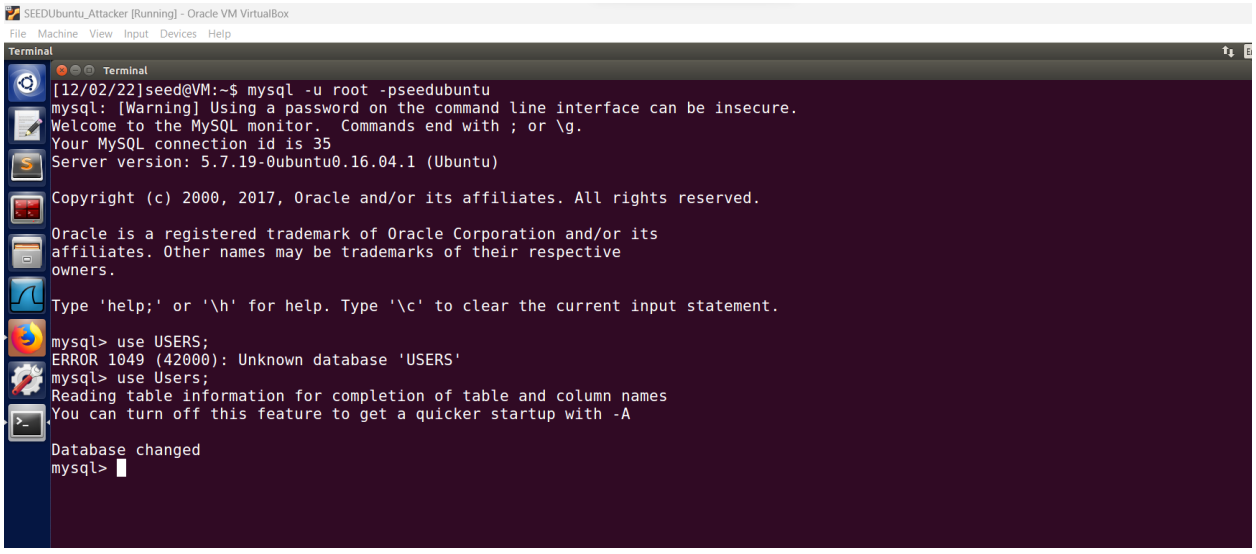
## 2.1 Task 1: Get Familiar with SQL Statements

The main focus of this work is becoming familiar with SQL commands. The SEED Labs virtual machine already has MySQL installed and a database made specifically for this lab (I'm using the Ubuntu 16.04 edition). The database is called Users and has a credential table in it.

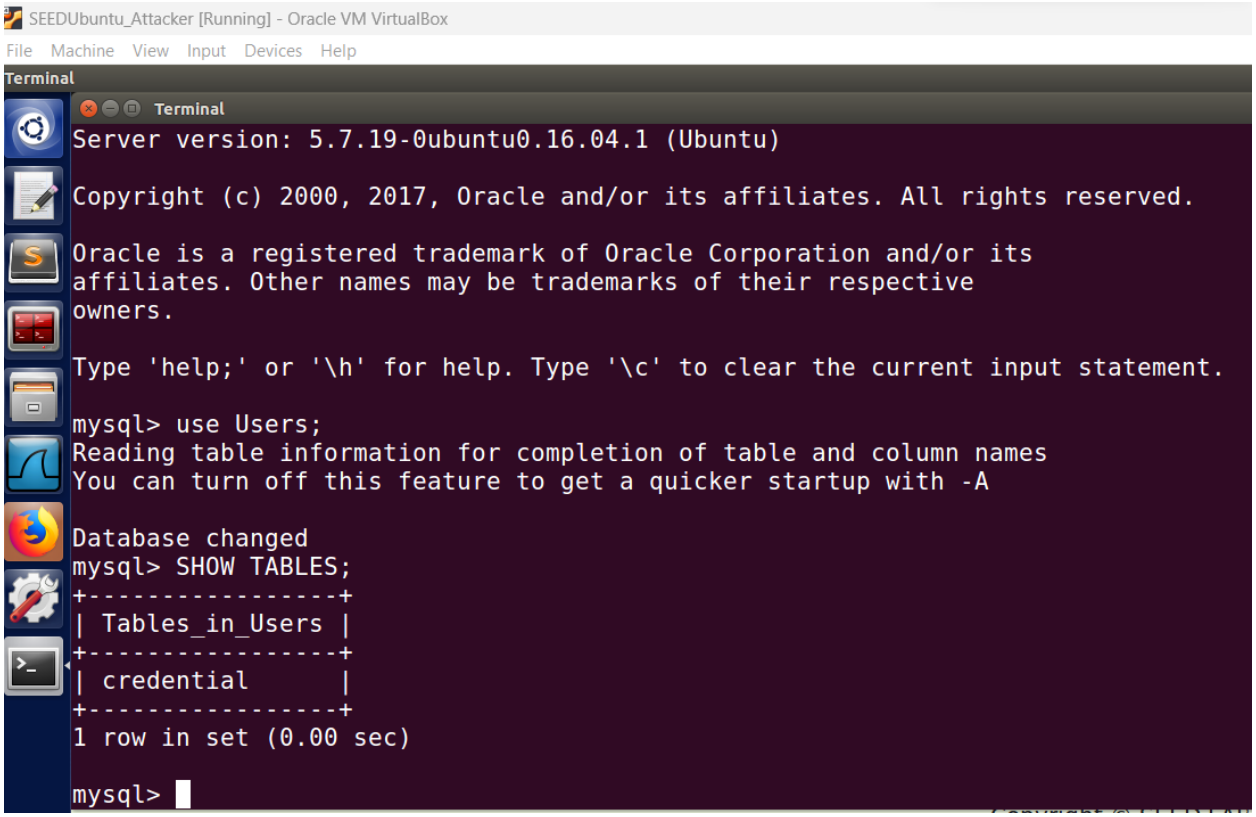
Firstly I logged into my MySQL using username root and password seedubuntu as shown below:




I am using already created table i.e. Users



The Users database consists of only one table as shown below:




Now I am first printing all the information of all users and then I will print only the information of ALICE.

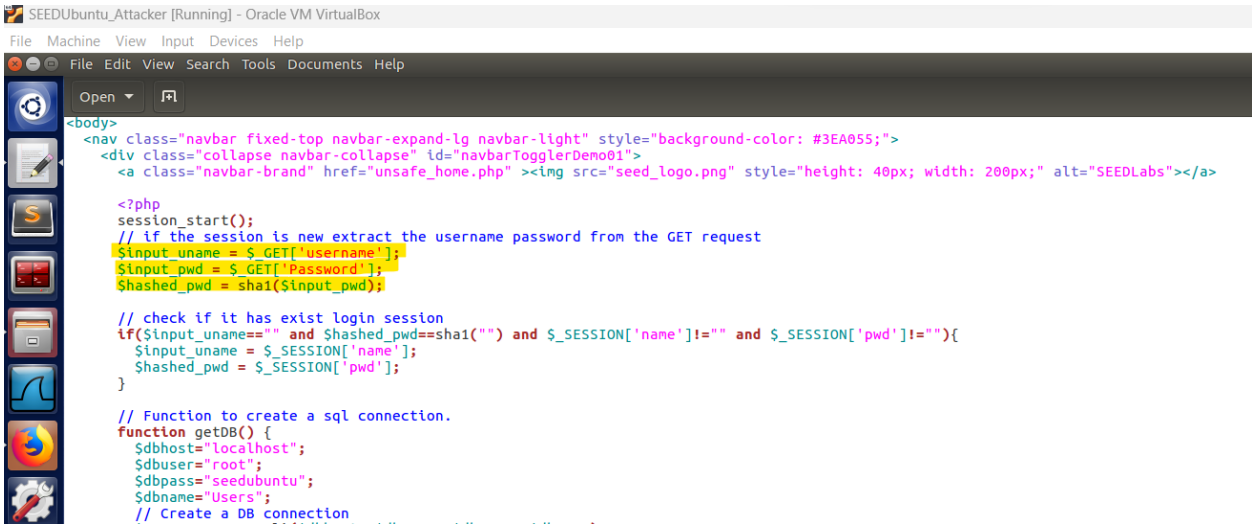
```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       | Samy     | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       | Samy     | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 | 100000 | 4/10  | 98993524 |             |         |       | Samy     | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |    100 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       | Samy     | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       | Samy     | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
6 rows 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 |             |         |       | Samy     | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)
```

(Note: I have already done this lab and I am doing it again to document for the assignment and that is the reason the Nickname is getting populated else nickname column will be empty)

In this task I am supposed to use various SQL attacks to access the information which I am not supposed to.

Below is the code that is used to authenticate the users:


(Above code snippet is taken from unsafe_home.php)



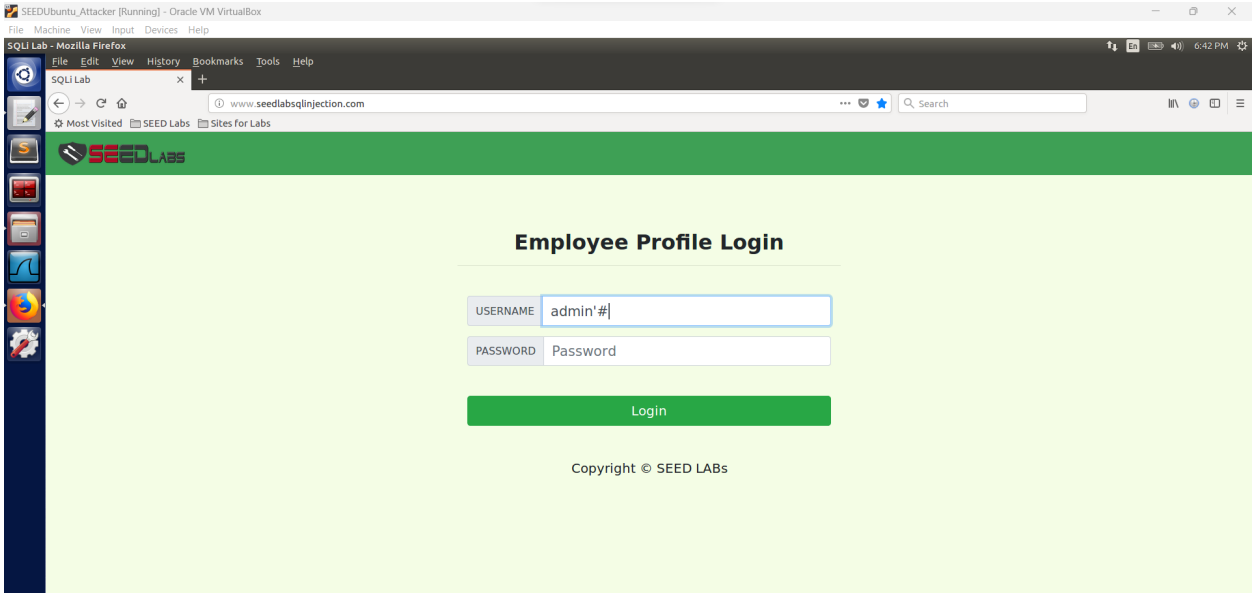

(Above code snippet is taken from unsafe_home.php)

From the above two code snippets we can observe that the password which the user inputs is hashed using sha1 and previously we saw that in the database also the password is hashed and stored.
So when the user enters the password it is first hashed using sha1 and then compared with the password present in the database which is also hashed using sha1 and if both match then user will be able to login else user will not be able to login.

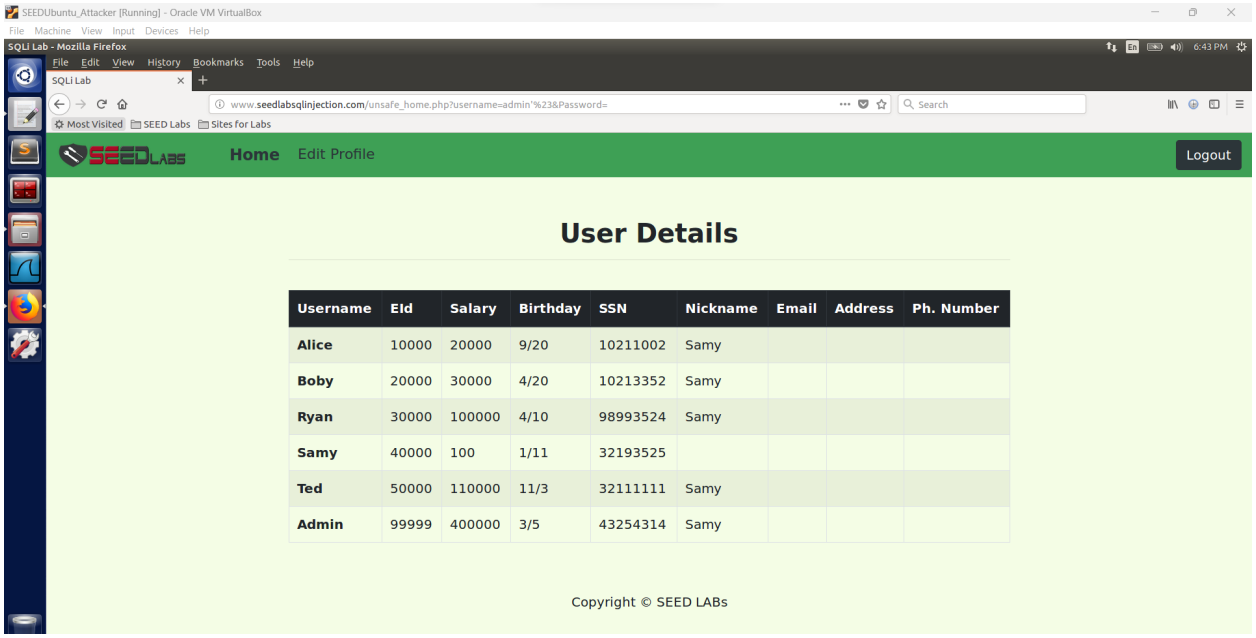
## 2.2 Task 2: SQL Injection Attack on SELECT Statement

Here in this task I need to login to the application  as admin so that I can see the information of all users. From the previous task we found that the username is ‘admin’ but I don't know the password as it is hashed. Now I need to decide what to type in the Username and Password fields so that I can succeed with the attack.

So in the previous screenshots I observed that select query is used after we enter the username and password so if I enter the username as admin’# and I am making use of ‘#’ because it will comment out the remaining part of the query (the password condition will be commented out) and I did the same as shown below:

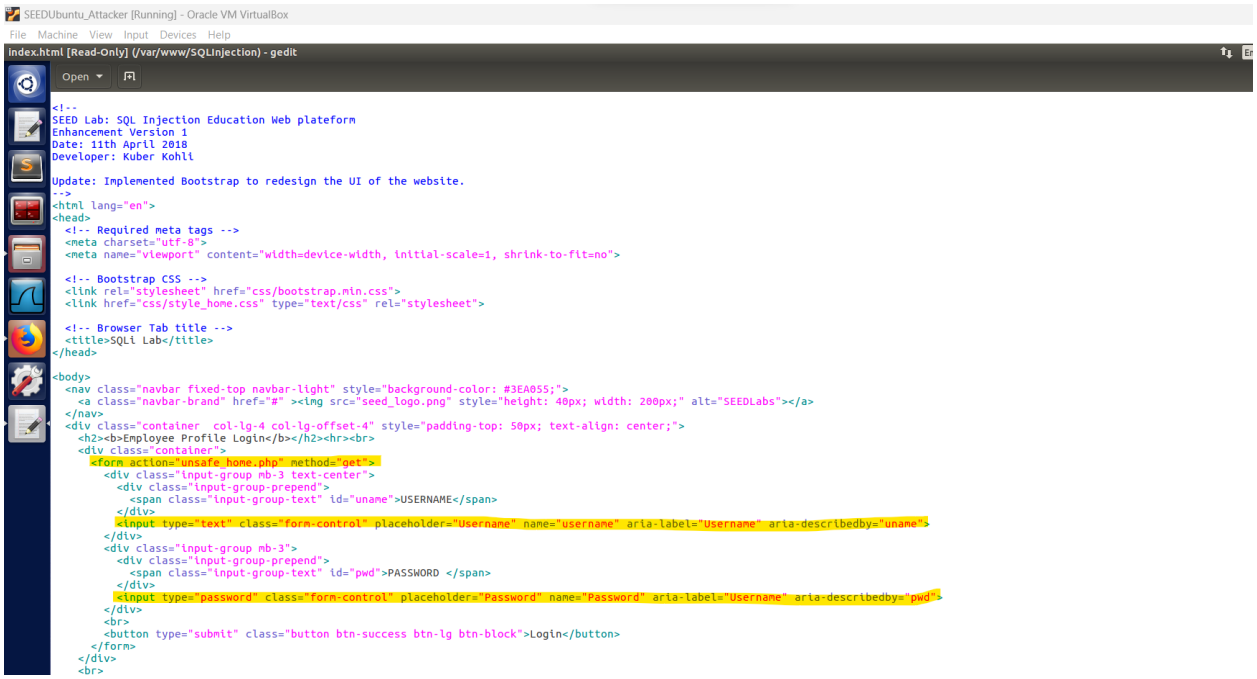



And after I clicked the Login button I was successfully able to login(as shown in the below screenshot) and this means my attack is successful.




## 2.2.2 Task 2.2: SQL Injection Attack from command line

In this task, I just need to repeat the same SQL injection attack but this time I need to make use of the command line using curl.
I first need to know what http method is getting called after we enter the username and password and then hit the login button.

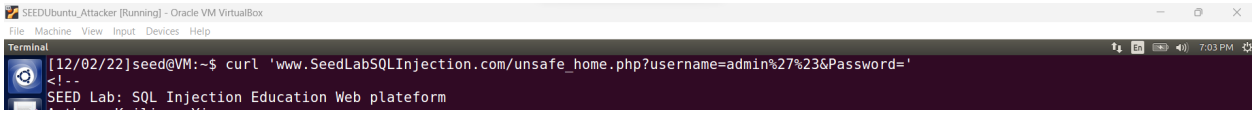


From the above screenshot of index.html it is clear that the http get method is getting called and it is sent to unsafe_home.php with the username and Password as the parameters.

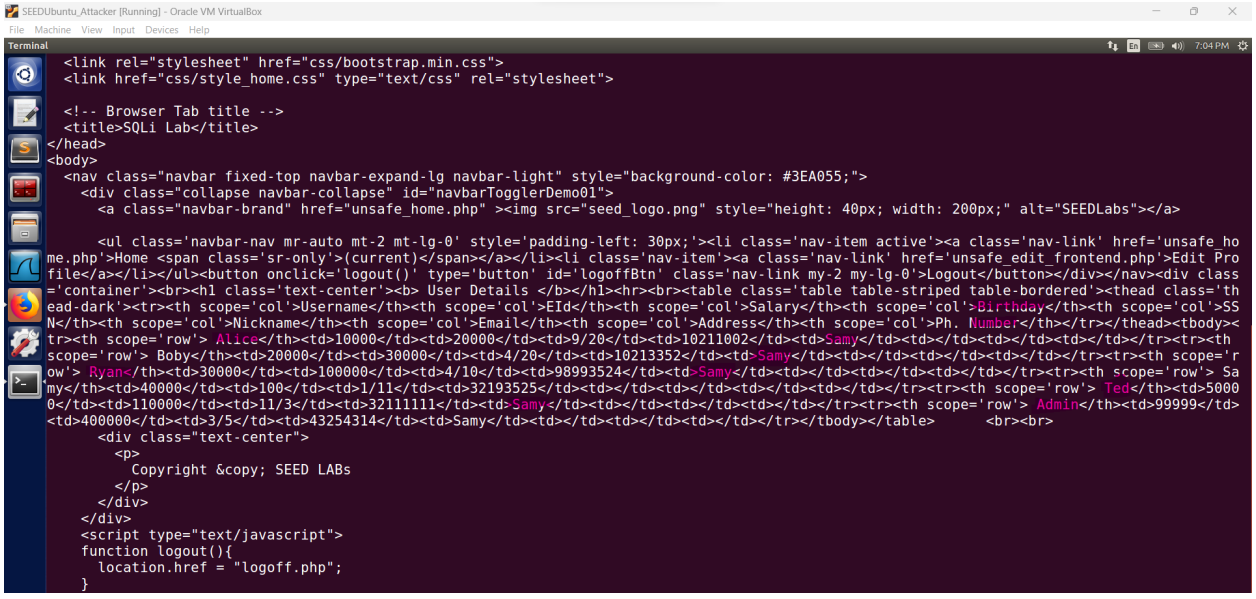
I need to use the following URL for the attack:
www.seedlabsqlinjection.com/unsafe_home.php?username=admin'#&Password=

And ' is encoded as %27 and # is encoded as %23 as mentioned in the task description.

So I used the curl command as shown below screenshot:




And the it displayed all the information as shown below:




And by this my attack succeeded.

## 2.2.3 Task 2.3: Append a new SQL statement

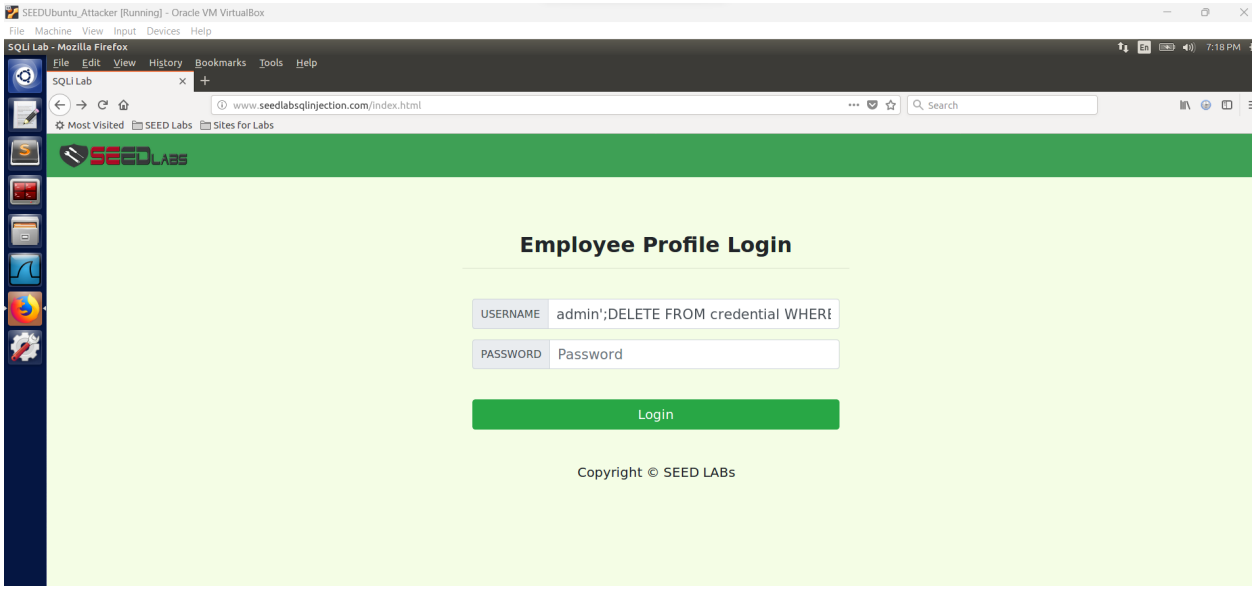
In this task, I need to make use of the same SQL injection vulnerability and add another delete or update statement with that so that I can update the data of the other users.

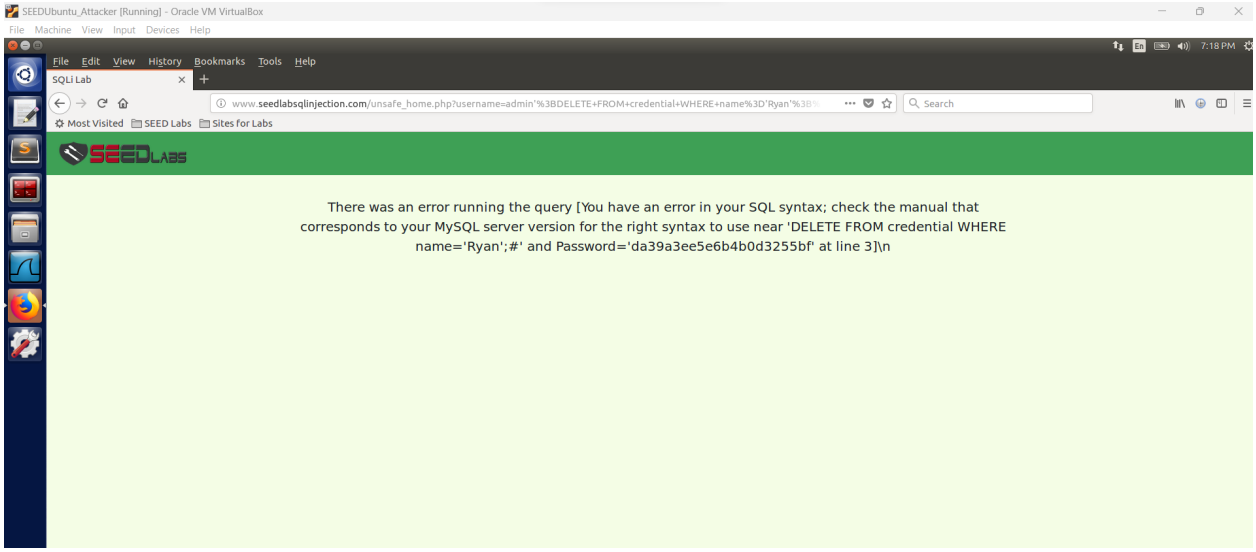
# User Details

| Username | EId | Salary | Birthday | SSN | Nickname | Email | Address | Ph. Number |
|---|---|---|---|---|---|---|---|---|
| **Alice** | 10000 | 20000 | 9/20 | 10211002 | Samy | | | |
| **Boby** | 20000 | 30000 | 4/20 | 10213352 | Samy | | | |
| **Ryan** | 30000 | 100000 | 4/10 | 98993524 | Samy | | | |
| **Samy** | 40000 | 100 | 1/11 | 32193525 | | | | |
| **Ted** | 50000 | 110000 | 11/3 | 32111111 | Samy | | | |
| **Admin** | 99999 | 400000 | 3/5 | 43254314 | Samy | | | |

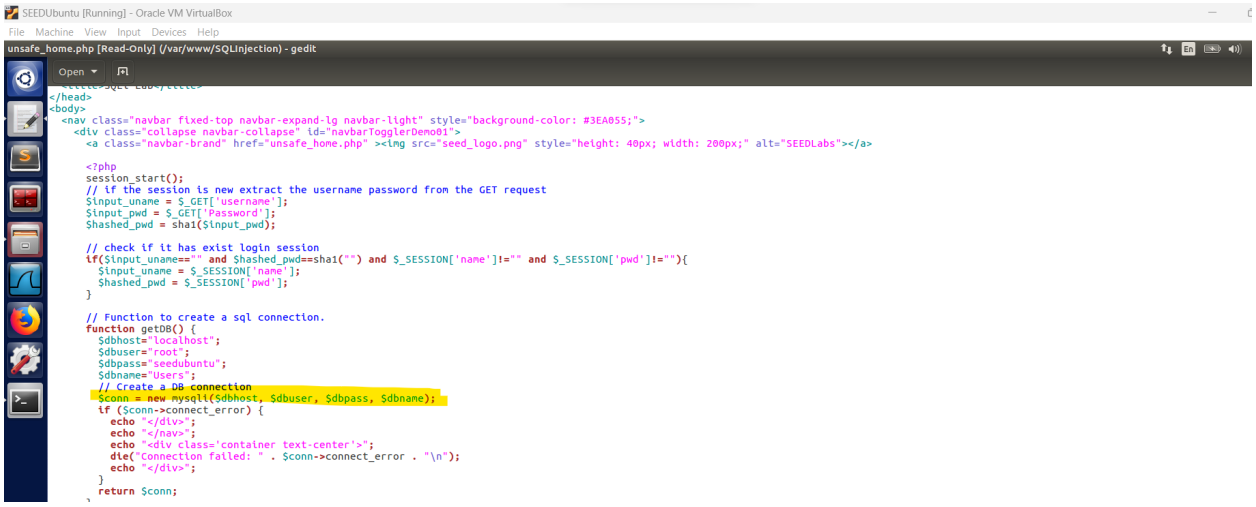


So I am planning to use:
admin';'DELETE FROM WHERE NAME='Ryan';#






From the above, it is clear that I wasn't able to succeed in the attack as such an attack does not work against MySQL, because PHP's mysqli extension, the mysqli::query() API does not allow multiple queries to run in the database server.
In our application, unsafe_home.php we are making use of the mysqli as shown in the below screenshot.



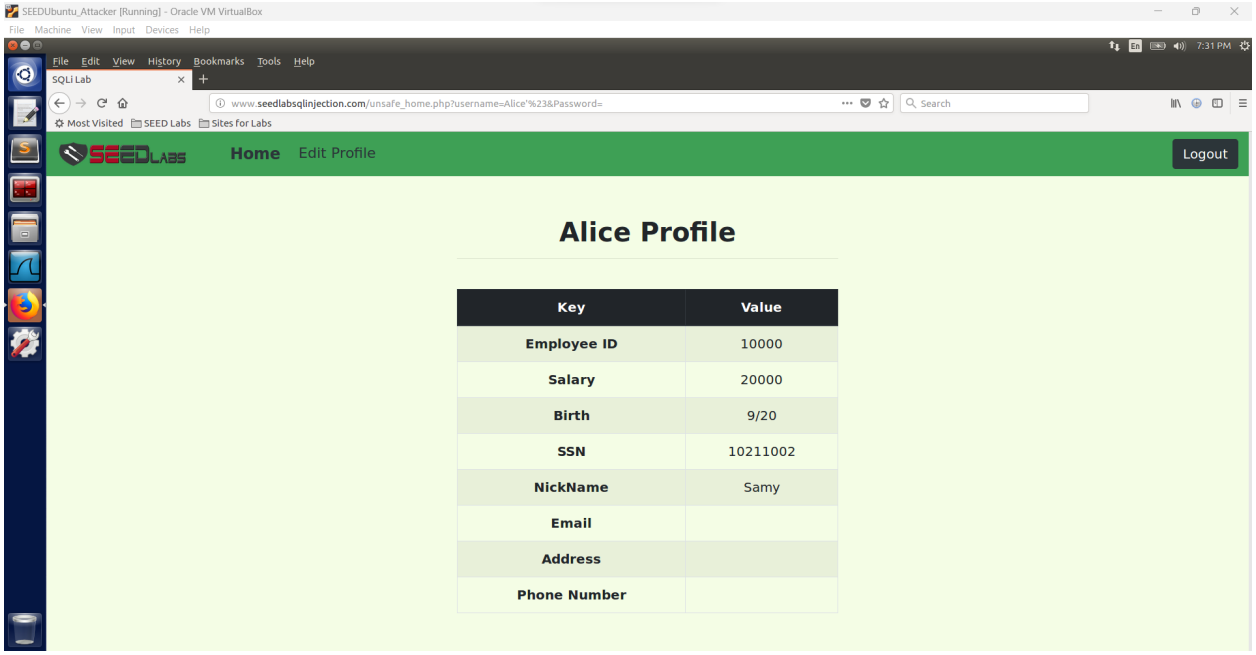

## 2.3 Task 3: SQL Injection Attack on UPDATE Statement

In our Employee Management application, there is an Edit Profile page that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to log in first. When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed.

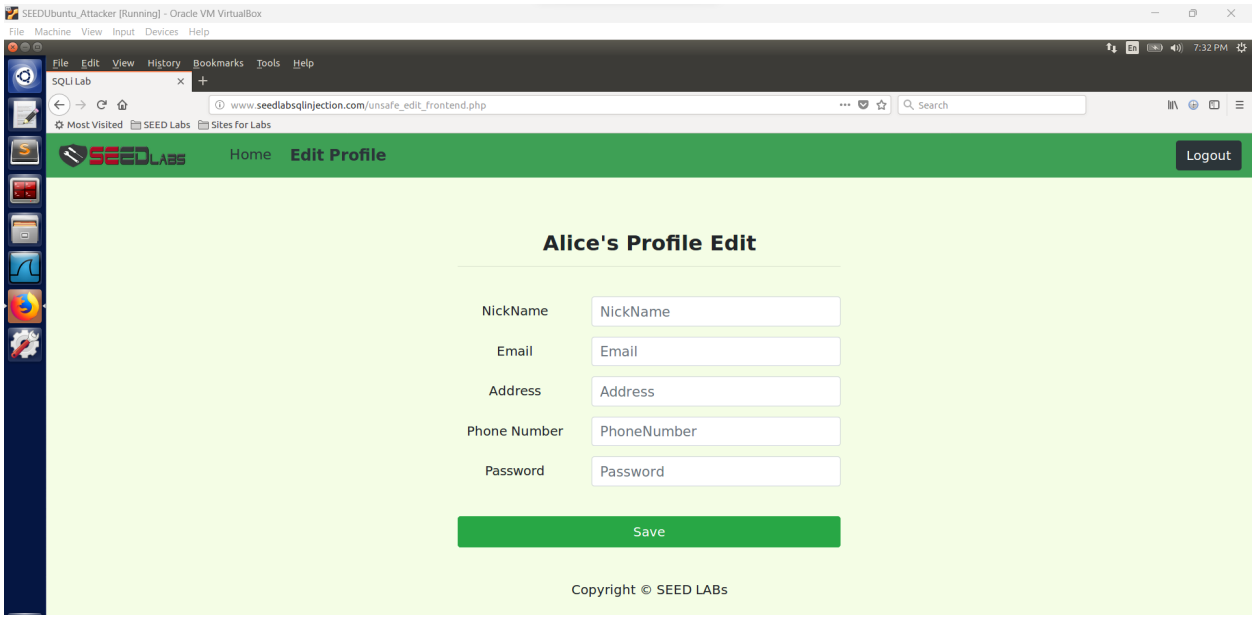
The PHP code implemented in unsafe edit backend.php file is used to update employee's profile information. The PHP file is located in the /var/www/SQLInjection directory.

### 2.3.1 Task 3.1: Modify your own salary.

Logged in as Alice using Alice'# in the login page




Going to the Edit Profile page,



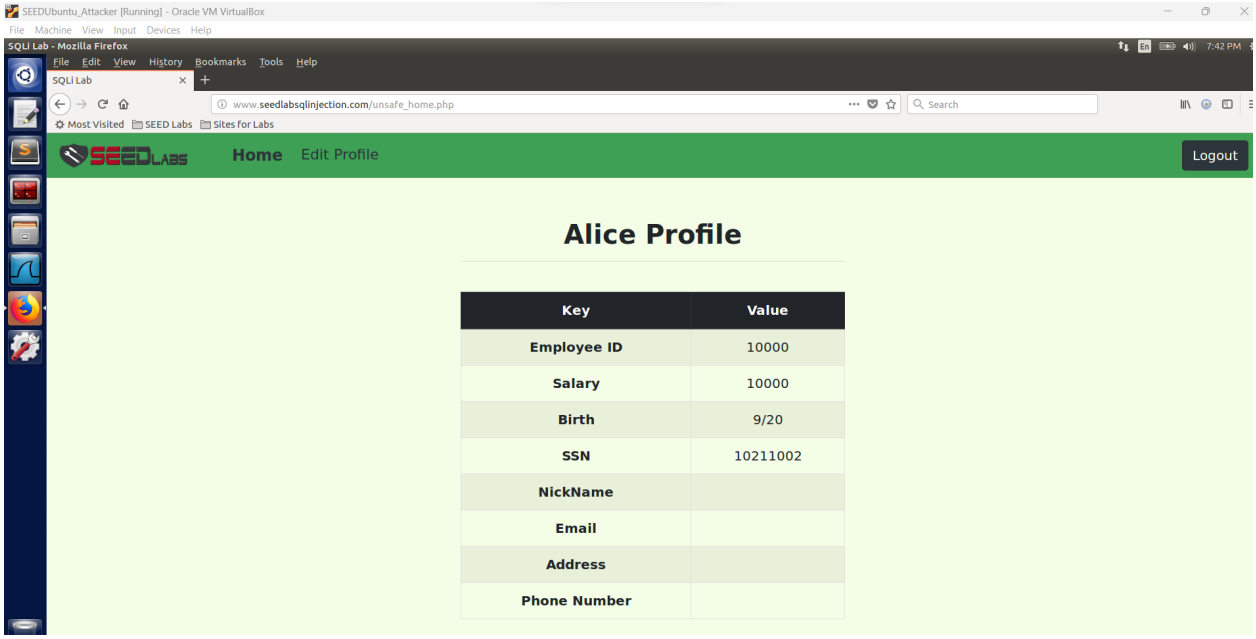
The unsafe edit backend.php file (found in the /var/www/SQLInjection directory) is used to update the employee's profile information in the database after a user completes the Profile Edit form and hits Save.

```
$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
}else{
  // if passowrd field is empty.
  $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
}
$conn->query($sql);
$conn->close();
header("Location: unsafe_home.php");
exit();
```

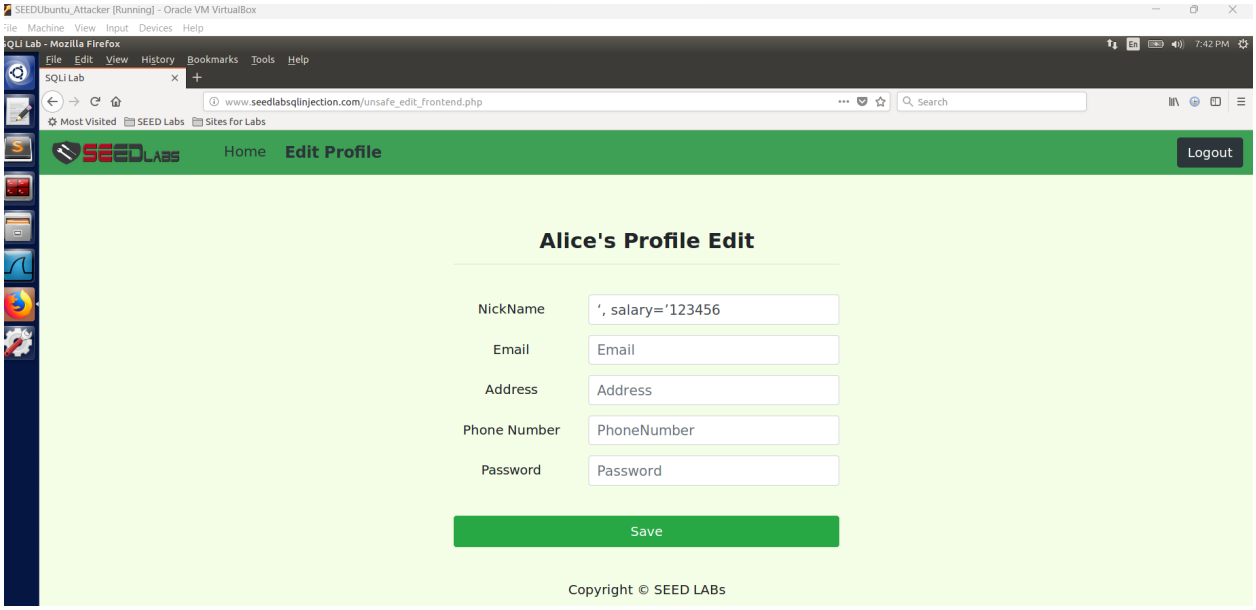
I first look at the SQL code that is executed. I think that I can enter a string into the nickname field that will allow me to add salary to the list of fields being updated. I will try entering:

‘, salary=’123456

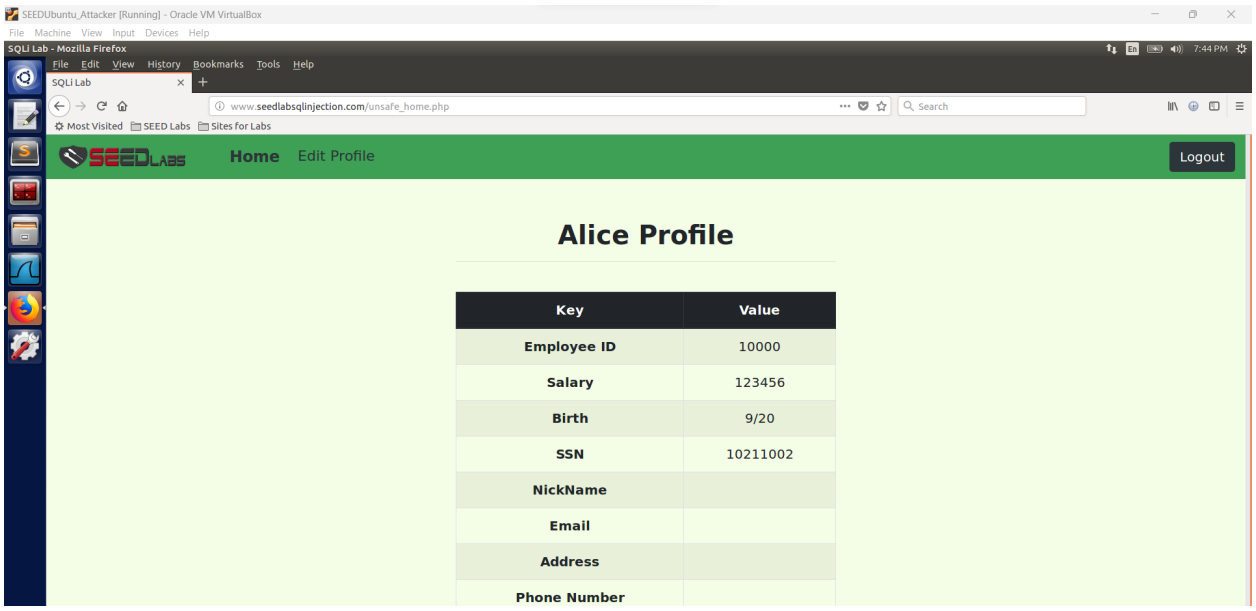
Alice’s salary before attack:




Inside Edit Profile page:




Once I clicked on save. Alice's salary got changed to 123456 as shown below:

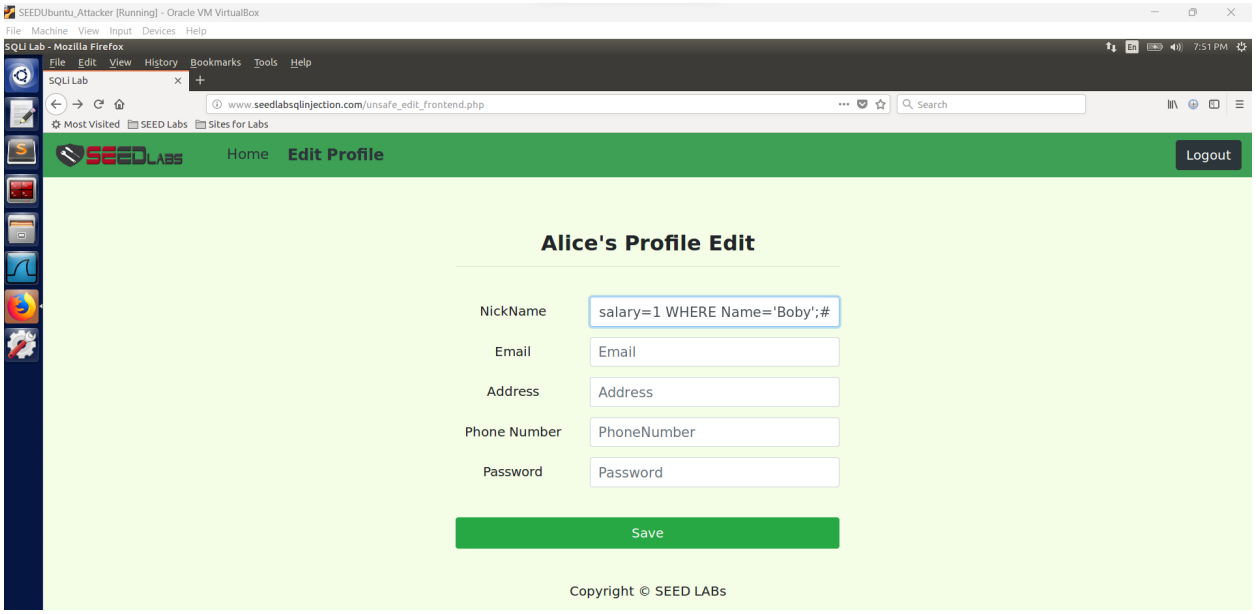



I was successfully able to update my own(Alice) salary to 123456. Thus I succeeded in the attack.

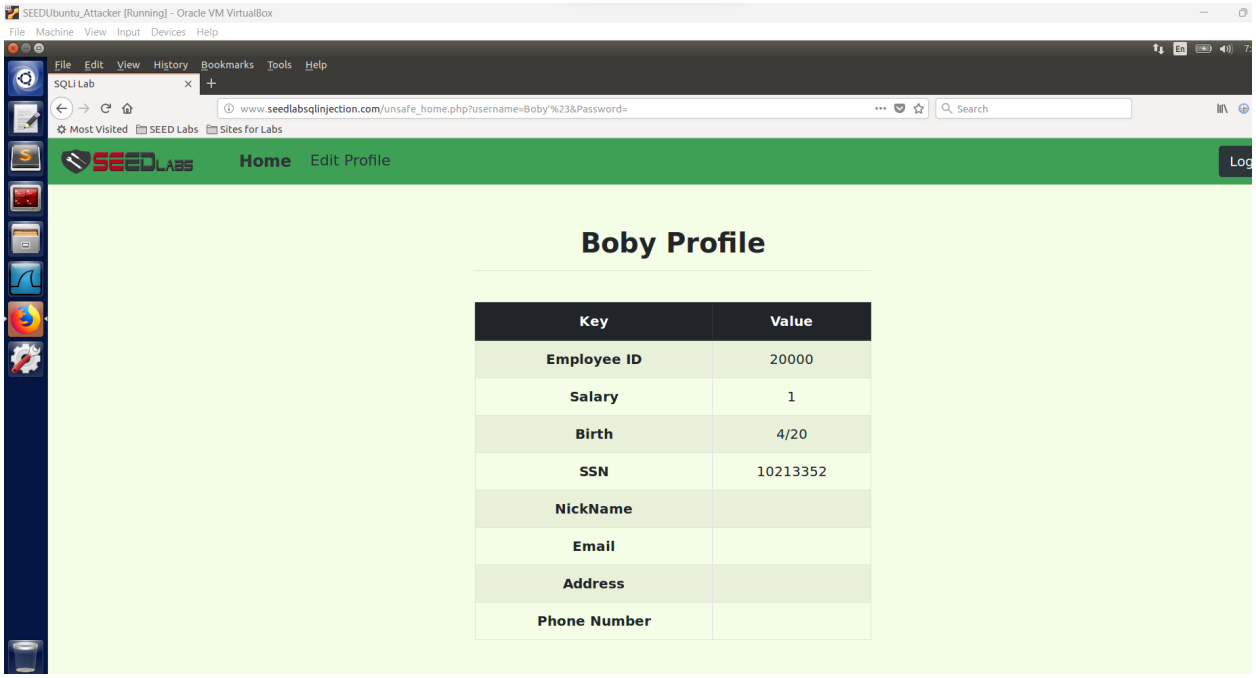
## 2.3.2 Task 3.2: Modify other people' salary

Now In this task, I need to update my boss Boby's salary to $1.

In order to do this, I will to my(Alice) edit profile page as I did in my previous task and then instead of ', salary=1 I will make use as shown below:
'salary=1 WHERE Name='Boby';#




Now log in as Boby and let's see if the salary has got updated to 1.



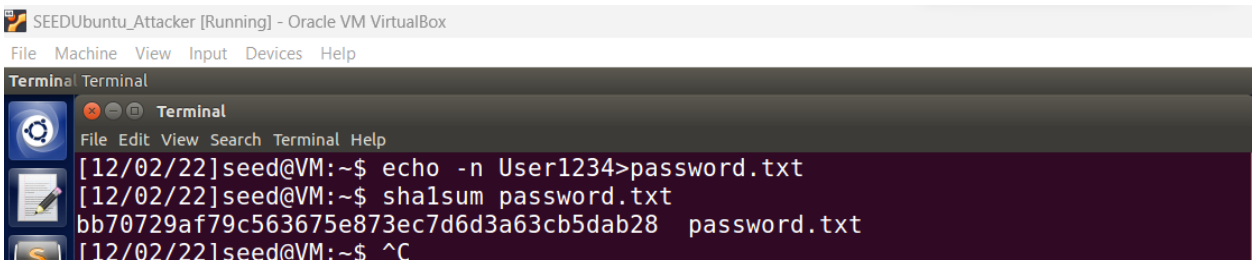

From the above screenshot, it is clear that I was successfully able to update the Boby salary and thus succeeded in my attack.

### 2.3.3 Task 3.3: Modify other people' password

In this task, I should change the password of Boby(my boss). So we could have used the same statement as in the previous task but the catch here is that the password is hashed. So we need to first hash the text whichever we want to update and then add it in the where condition.

In the previous tasks we have observed that sha1 is used to hash the password.
This means that I will need to use SHA1 hashing on the password I choose and use that hashed version in the SQL injection attack.

I will make use of sha1sum to generate the hash as shown below:

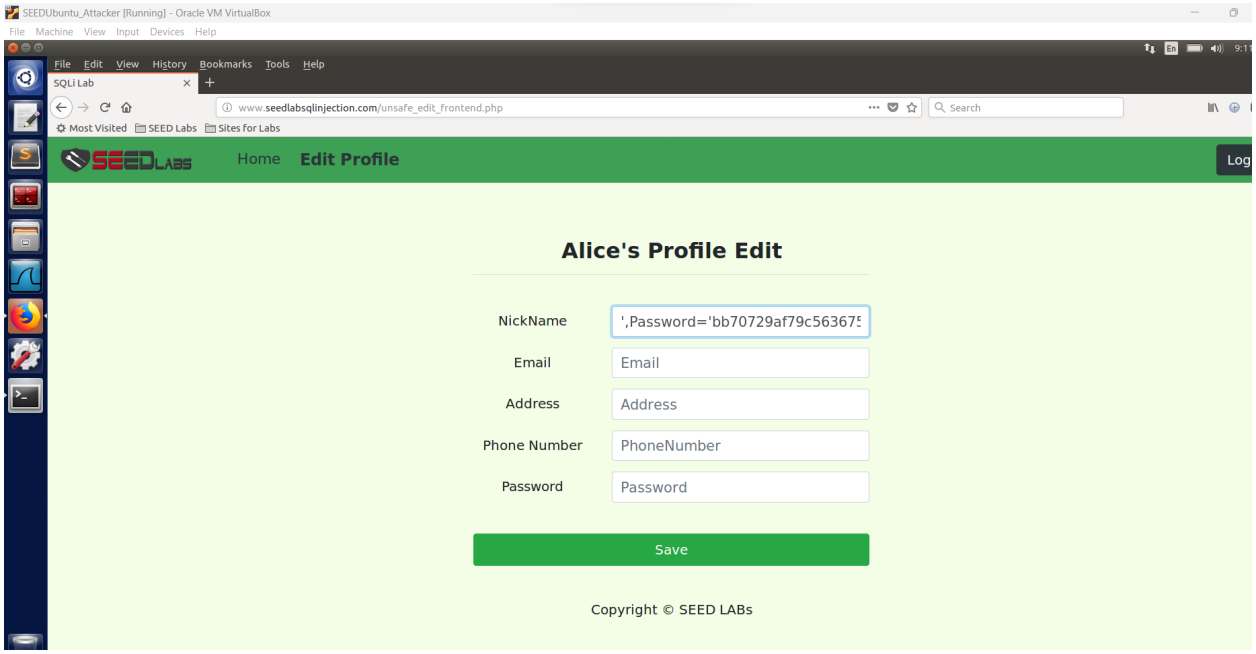



Now I will make use of the hash I got after running sha1sum password.txt command.

I will enter the below in the Nickname field of the Alice's edit profile page.

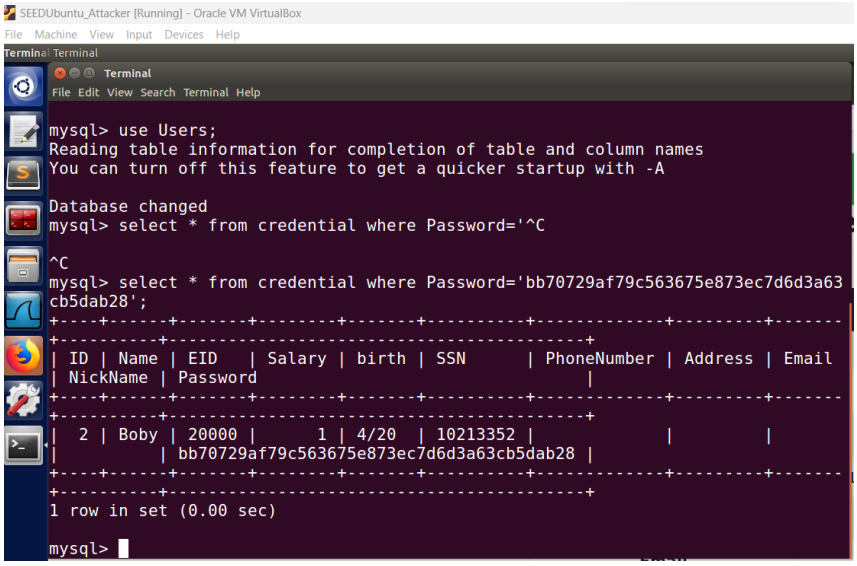
I enter:

', Password='Hash generated after I ran sha1sum password.txt command ' WHERE Name='Boby';#

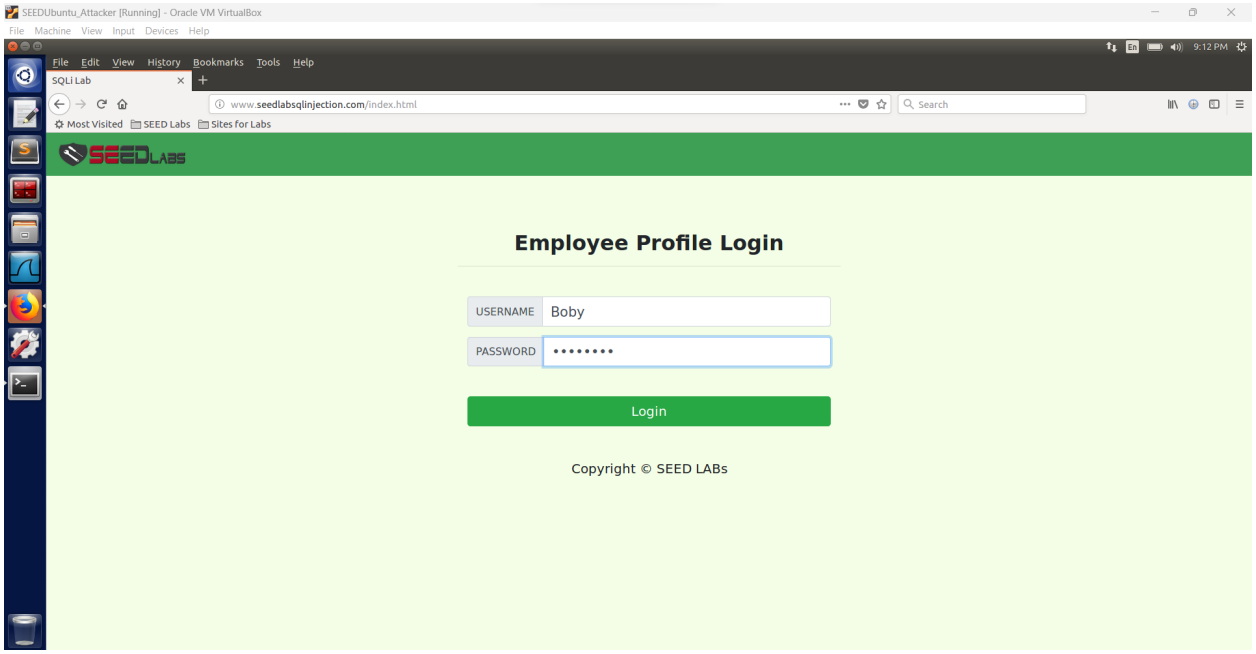


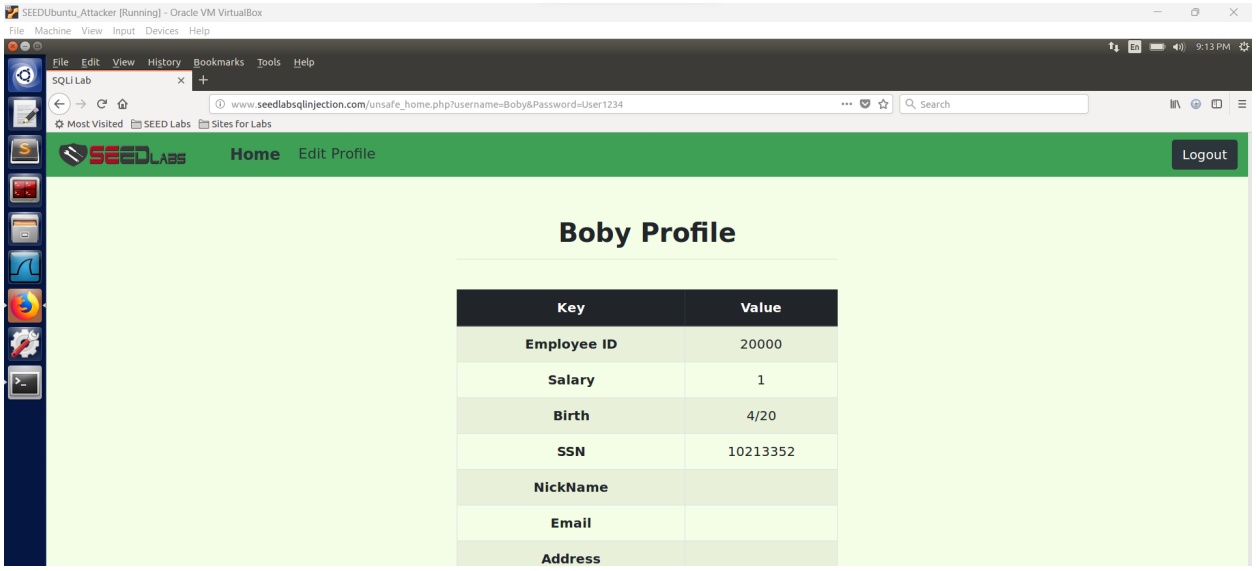

Now I hit save and then logout and

Now I need to check in the database first that hash corresponds only to Boby as shown below which confirms that.



Now I login as Boby with the password ‘User1234’.



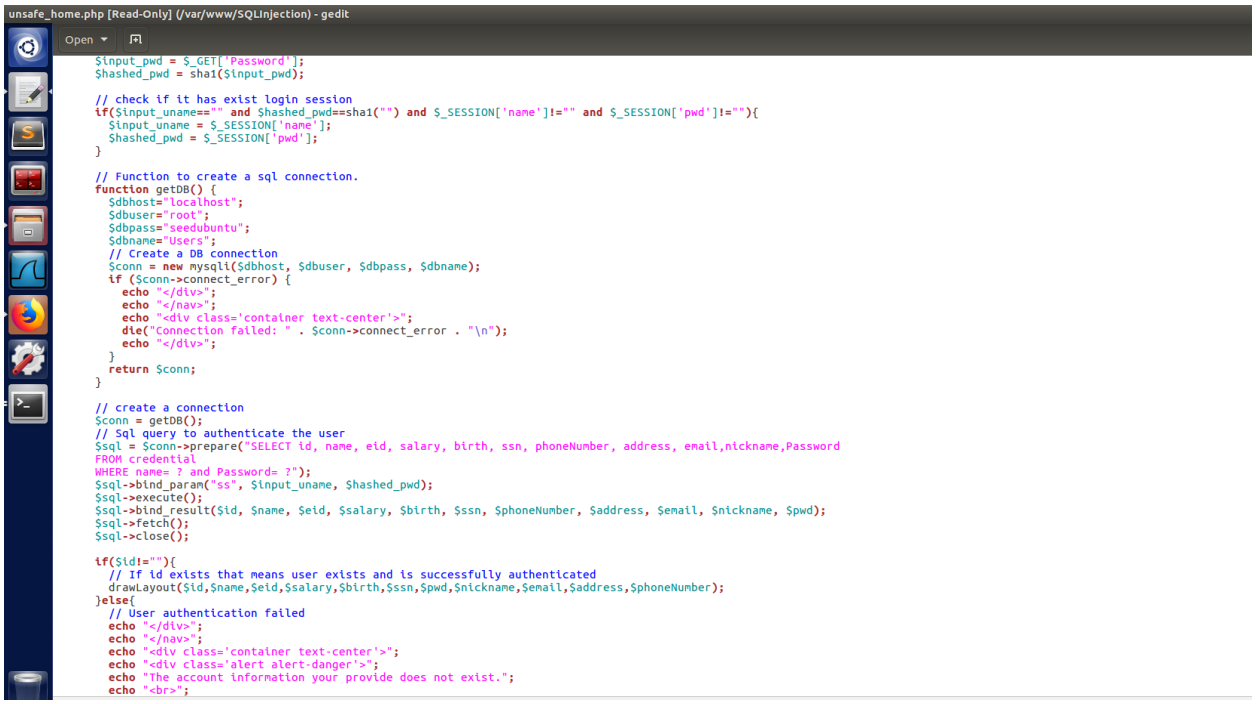




I was able to login to Boby’s account using the new password. The SQL injection attack was a success.

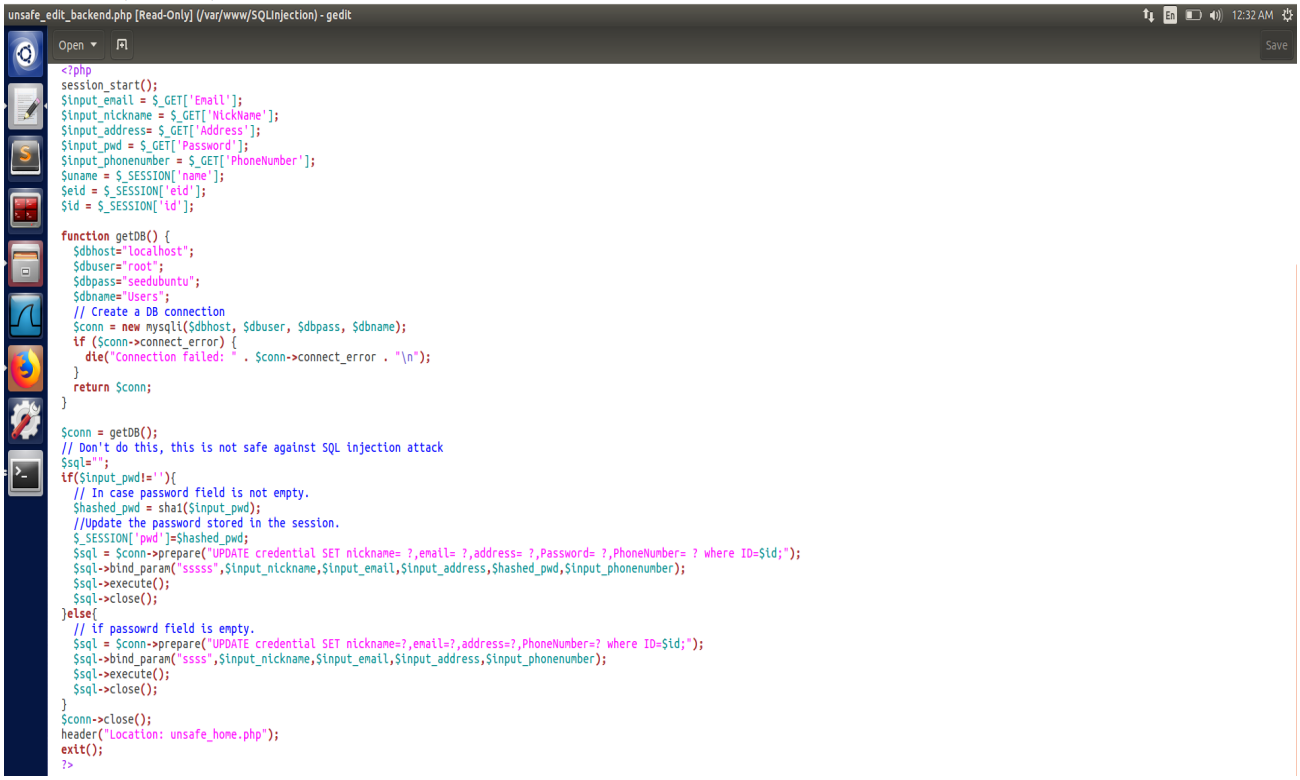
## Task 4: Countermeasure – Prepared Statement

In this task, I need to edit the source code of both unsafe_home.php and unsafe_edit_backend.php to have them use the prepared statement mechanism as shown below:

unsafe_home.php

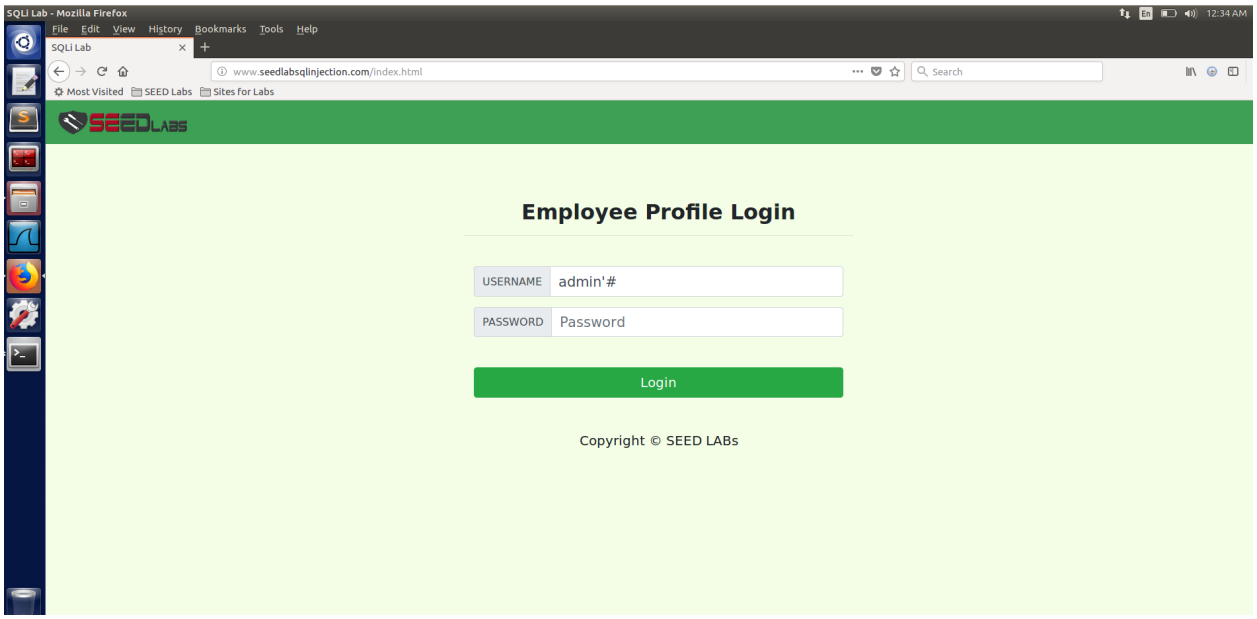


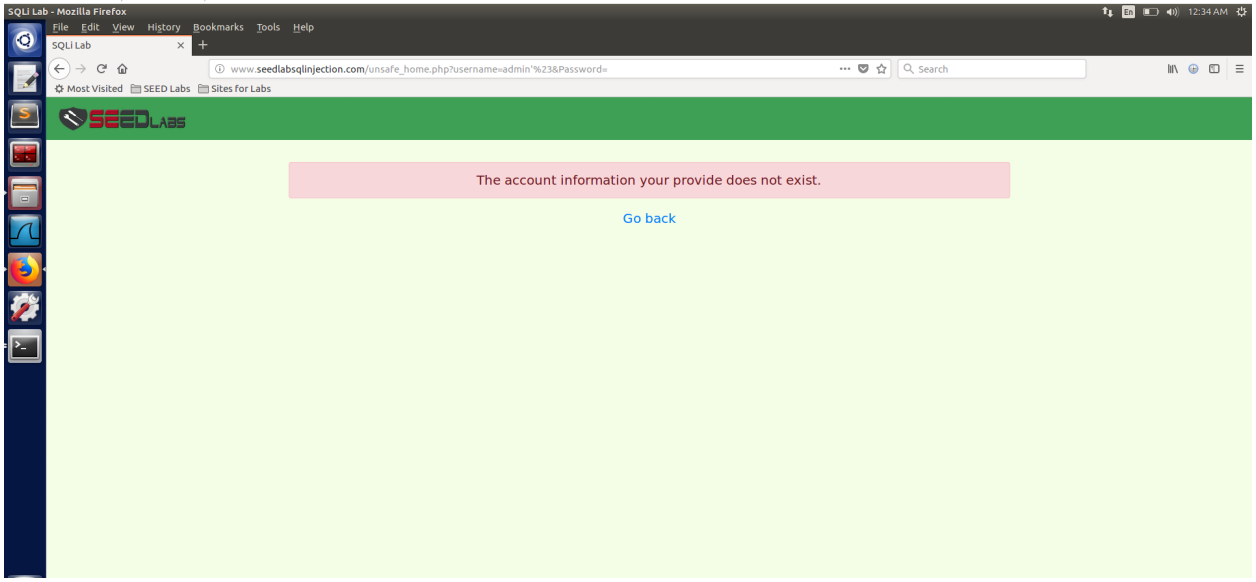

Unsafe_edit_backend.php




```php
<?php
session_start();
$input_email = $_GET['Email'];
$input_nickname = $_GET['NickName'];
$input_address= $_GET['Address'];
$input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$uname = $_SESSION['name'];
$eid = $_SESSION['eid'];
$id = $_SESSION['id'];

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) {
    die("Connection failed: " . $conn->connect_error . "\n");
  }
  return $conn;
}

$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $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();
}else{
  // if passowrd field is empty.
  $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();
}
$conn->close();
header("Location: unsafe_home.php");
exit();
?>
```

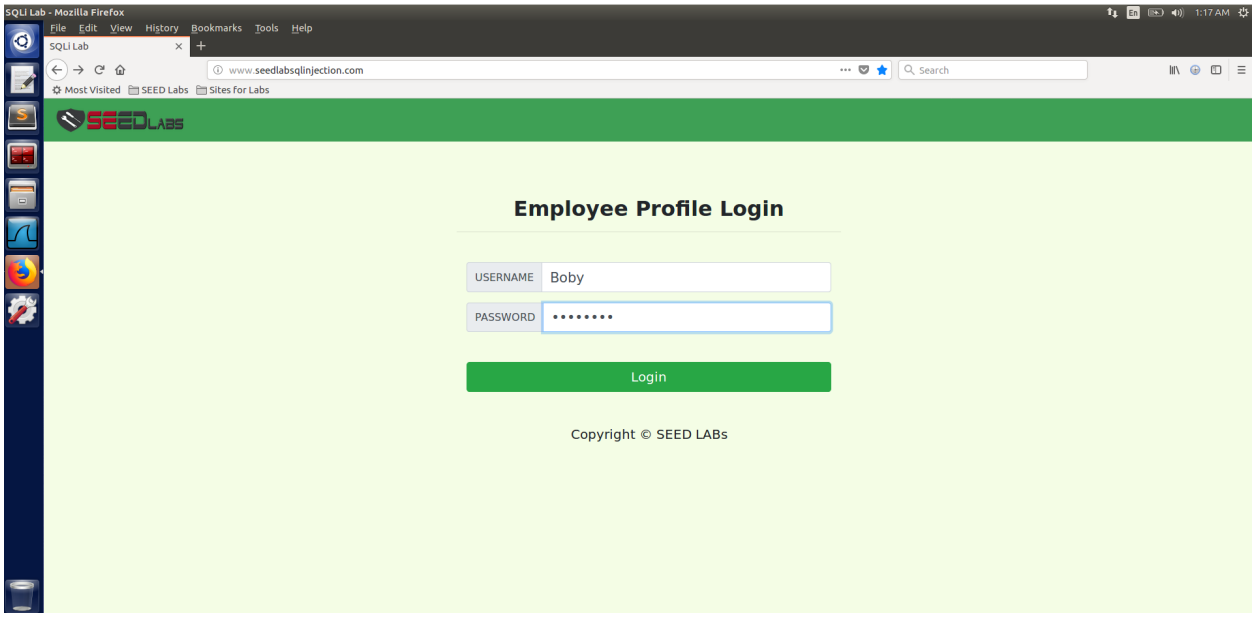
Then first let us try to login using the SQL inject attack which we used earlier:

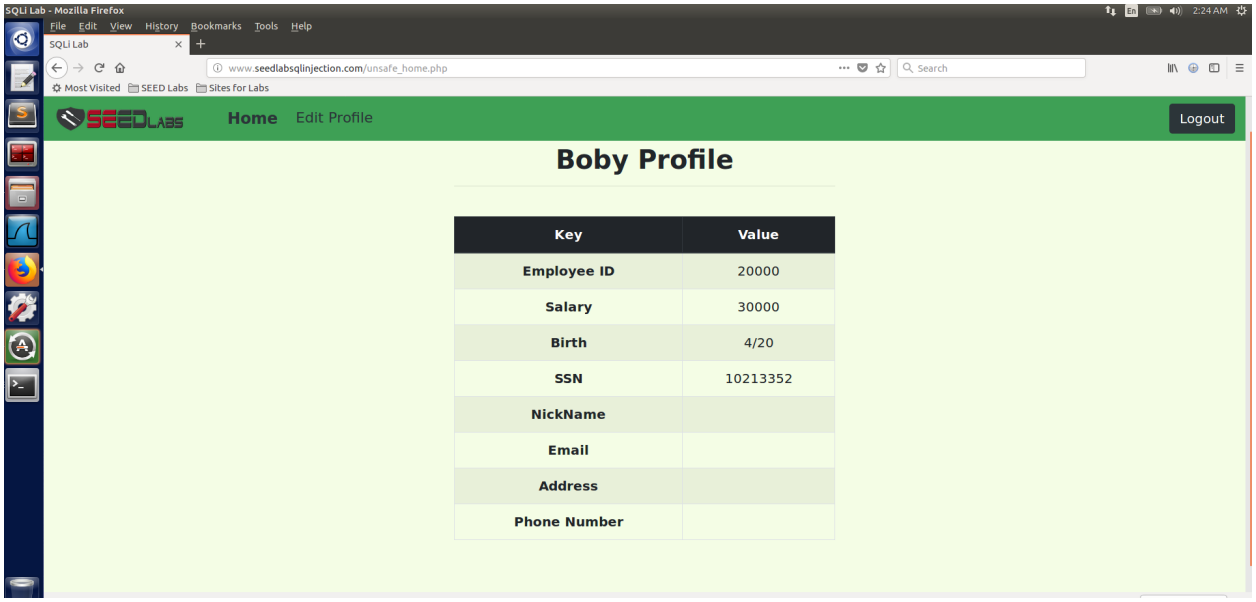


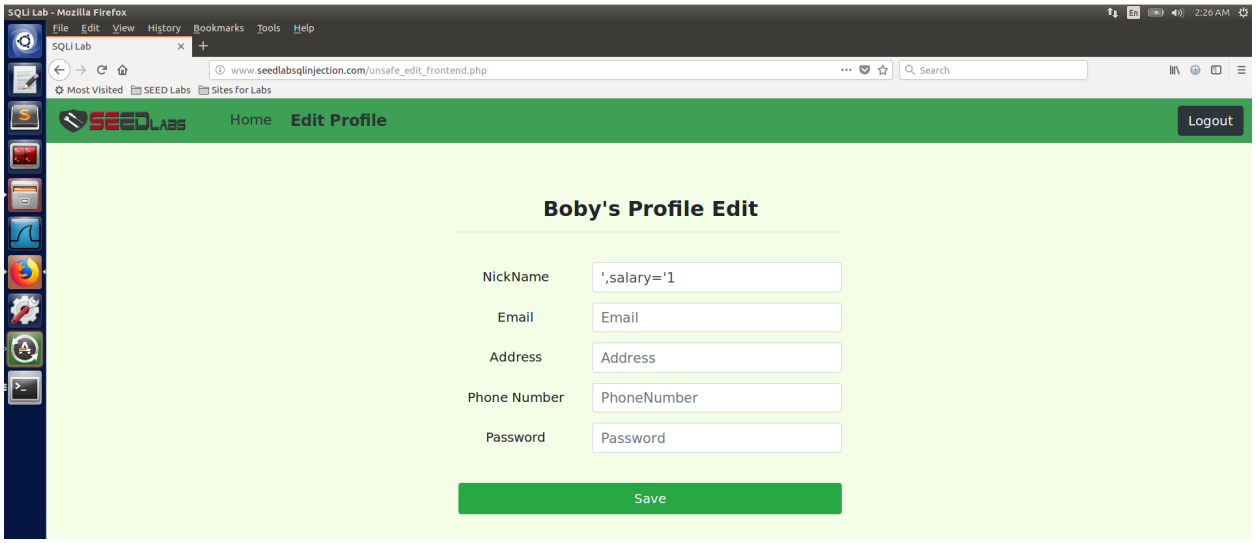


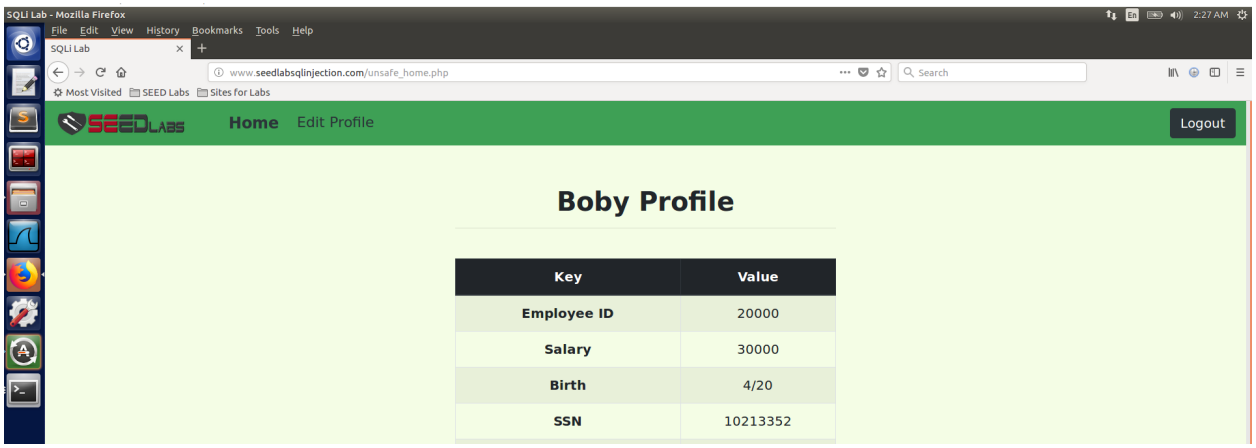

As you can see the SQL injection attack failed and thus we have successfully defended by using prepared statements.

Now lets login in as Boby and try to do a SQL injection attack in the edit profile section as we did earlier:




SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php
Most Visited SEED Labs Sites for Labs
SEEDLABS
Home
Edit Profile
Logout
Boby Profile
Key
Value
Employee ID
20000
Salary
30000
Birth
4/20
SSN
10213352
NickName
Email
Address
Phone Number


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_edit_frontend.php
Most Visited SEED Labs Sites for Labs
SEEDLABS
Home
Edit Profile
Logout
Boby's Profile Edit
NickName
',salary='1
Email
Email
Address
Address
Phone Number
PhoneNumber
Password
Password
Save


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php
Most Visited SEED Labs Sites for Labs
SEEDLABS
Home
Edit Profile
Logout
Boby Profile
Key
Value
Employee ID
20000
Salary
30000
Birth
4/20
SSN
10213352

From the above screenshots, it is clear that we defended the SQL injection attack on the Edit Profile Section also with the help of prepared statements.