CSE: 5382-001: SECURE PROGRAMMING

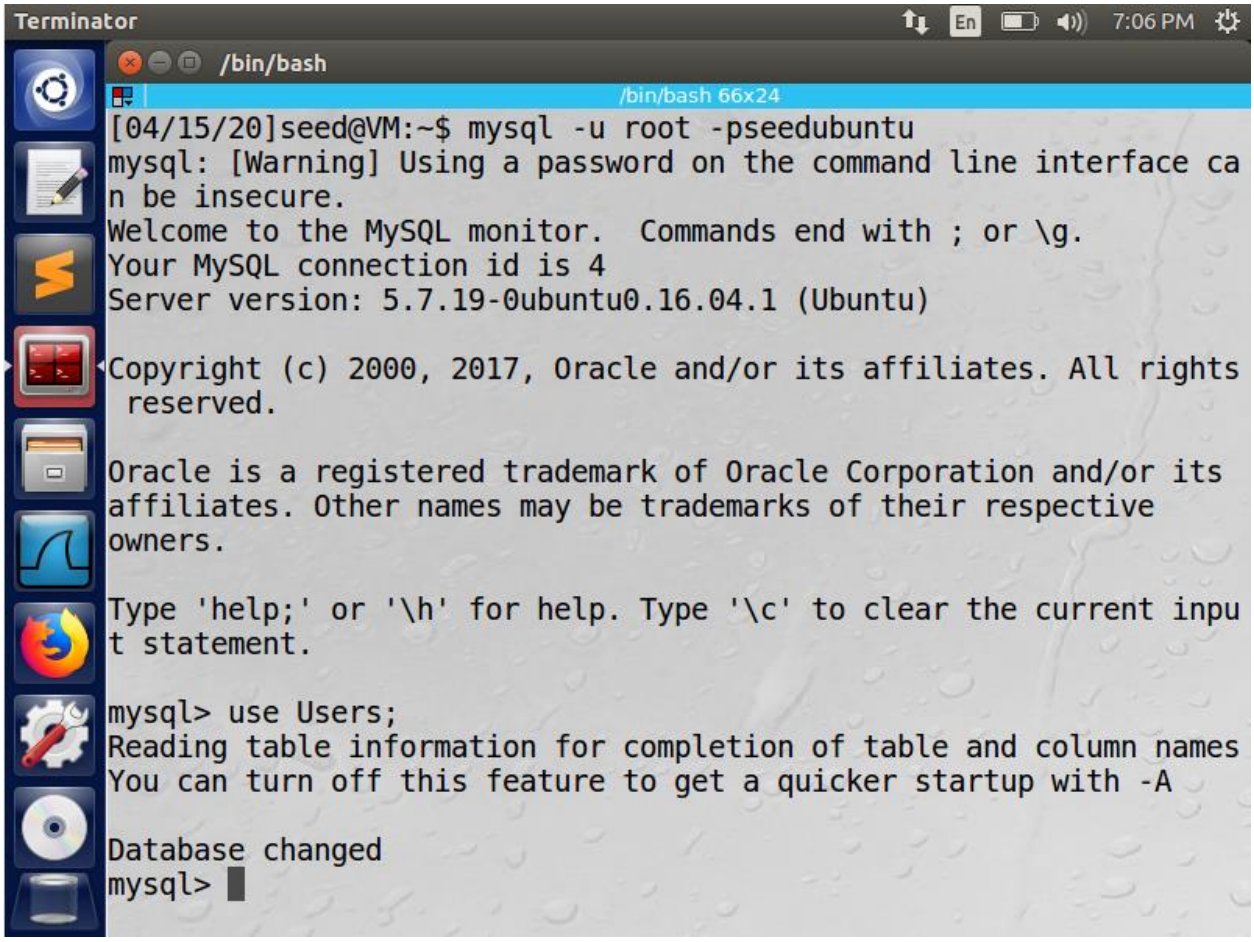
ASSIGNMENT 9

Tharoon T Thiagarajan

1001704601

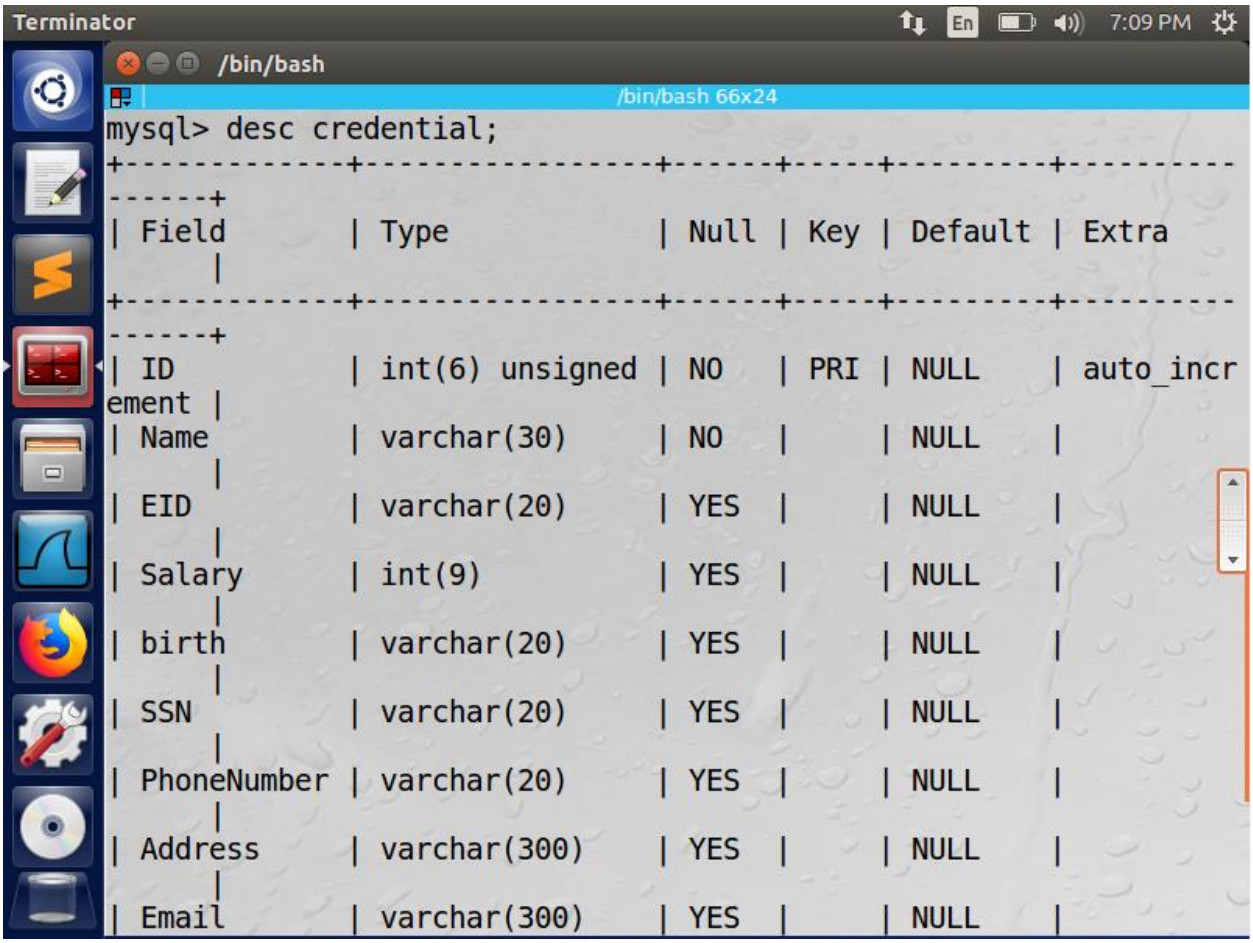
3.1 Task 1: Get Familiar with SQL Statements:

Before starting with the task, I opened the terminator on my VM and gave the command mysql -u root -pseedubuntu to open MySQL where root and seedubuntu are the credentials for the database. After logging into the database using the given command, I loaded the existing database using the SQL command “use”. I used the command “use Users;” to load the existing database.

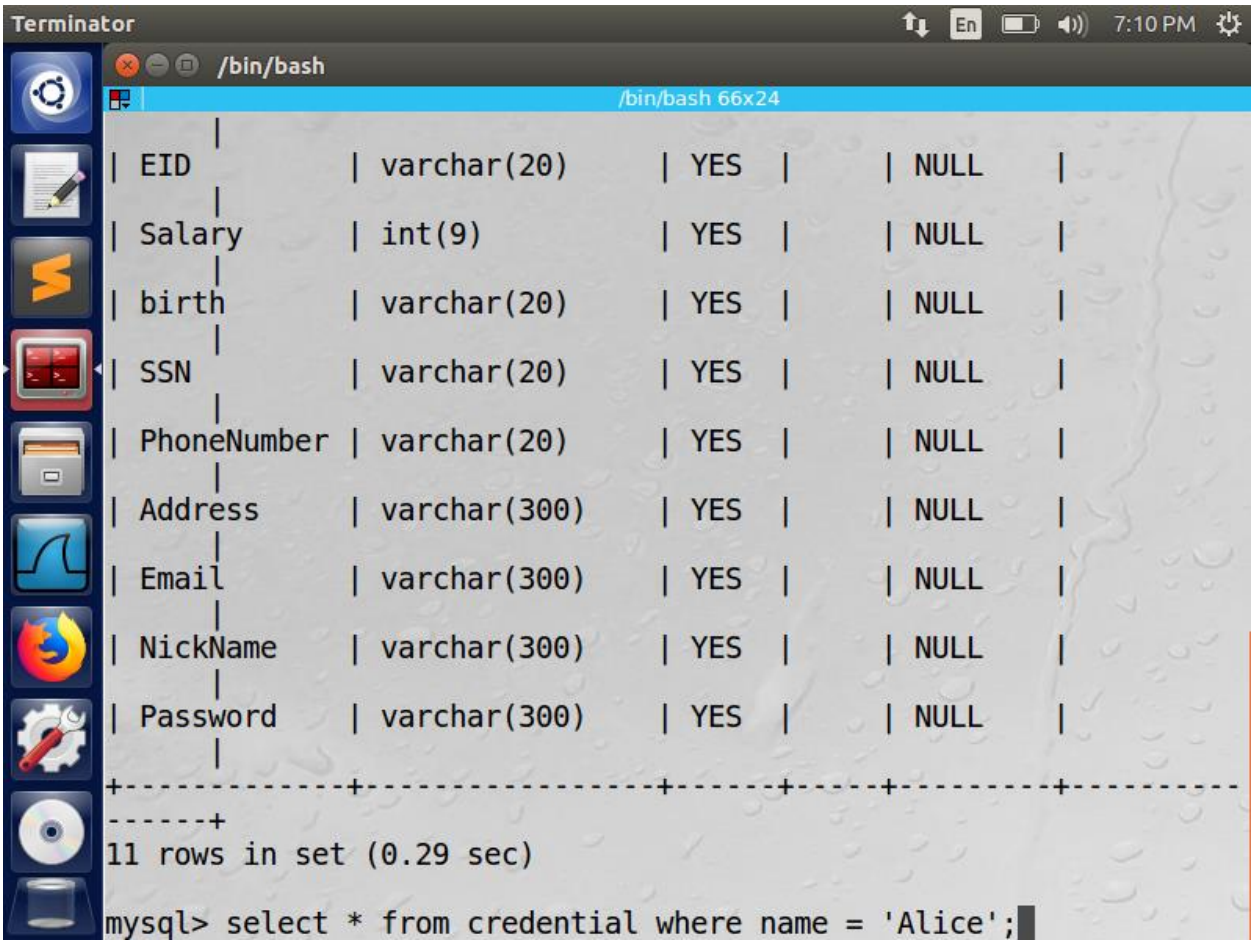


After loading the users database, I used the show tables; SQL command to list the tables that are present in the database. I was able to see one table getting displayed.

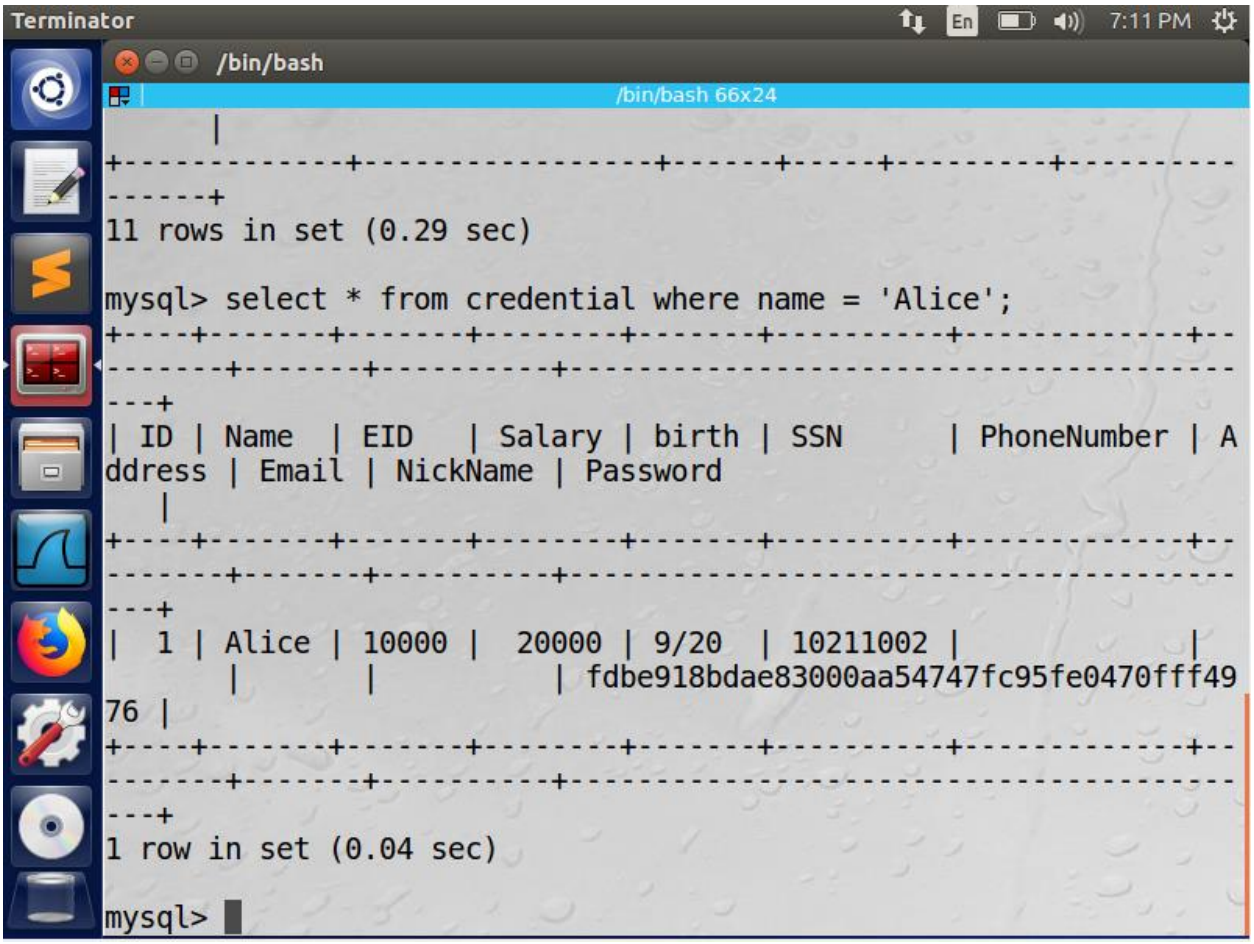
Then I used the desc command to view the structure of the table, so that I can know the column names and their data types.



After getting the structure of the table, I fetched the profile information of Alice using the SELECT statement in SQL query.



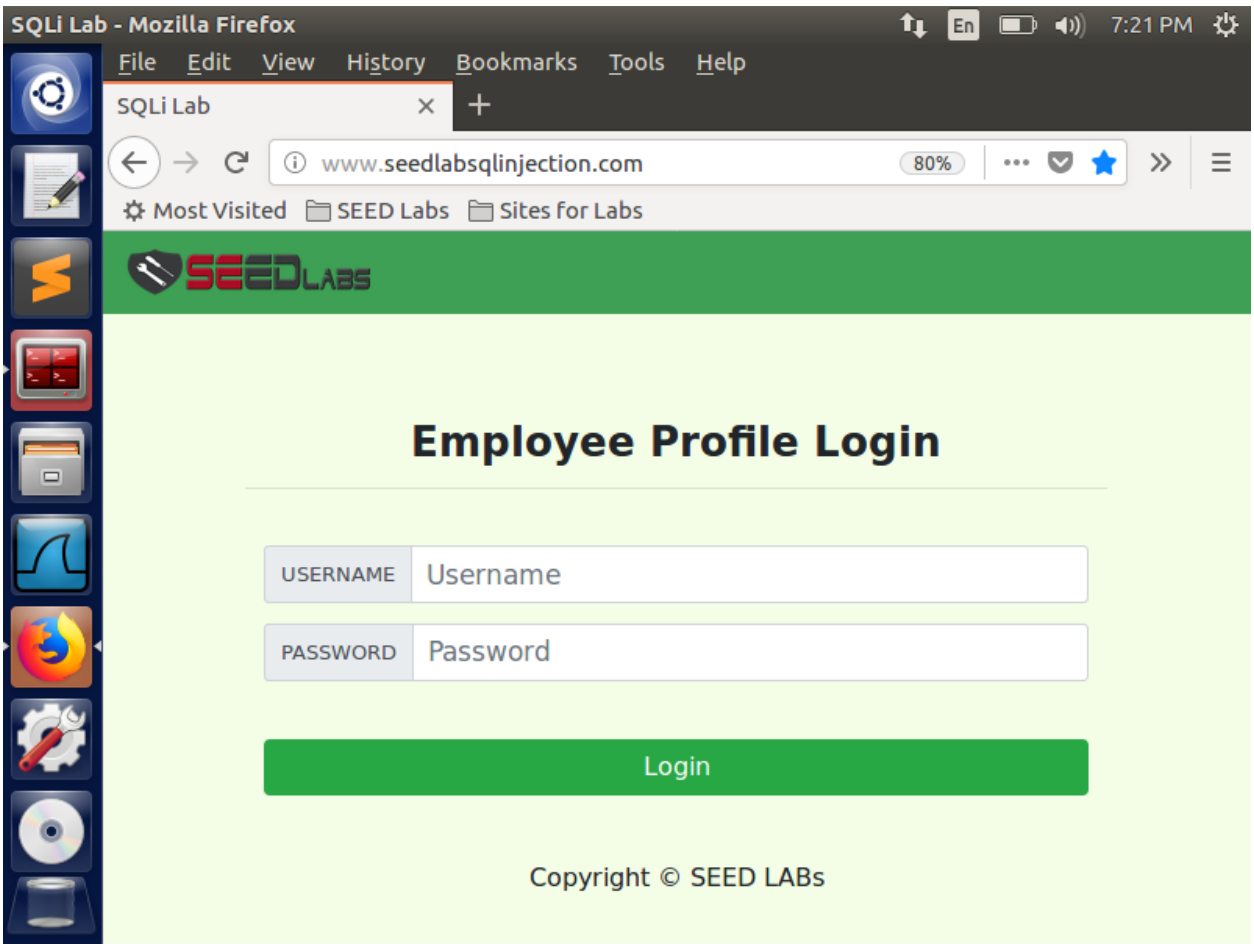
After running the SELECT SQL query, I was able to get the profile information of Alice from the credential table.



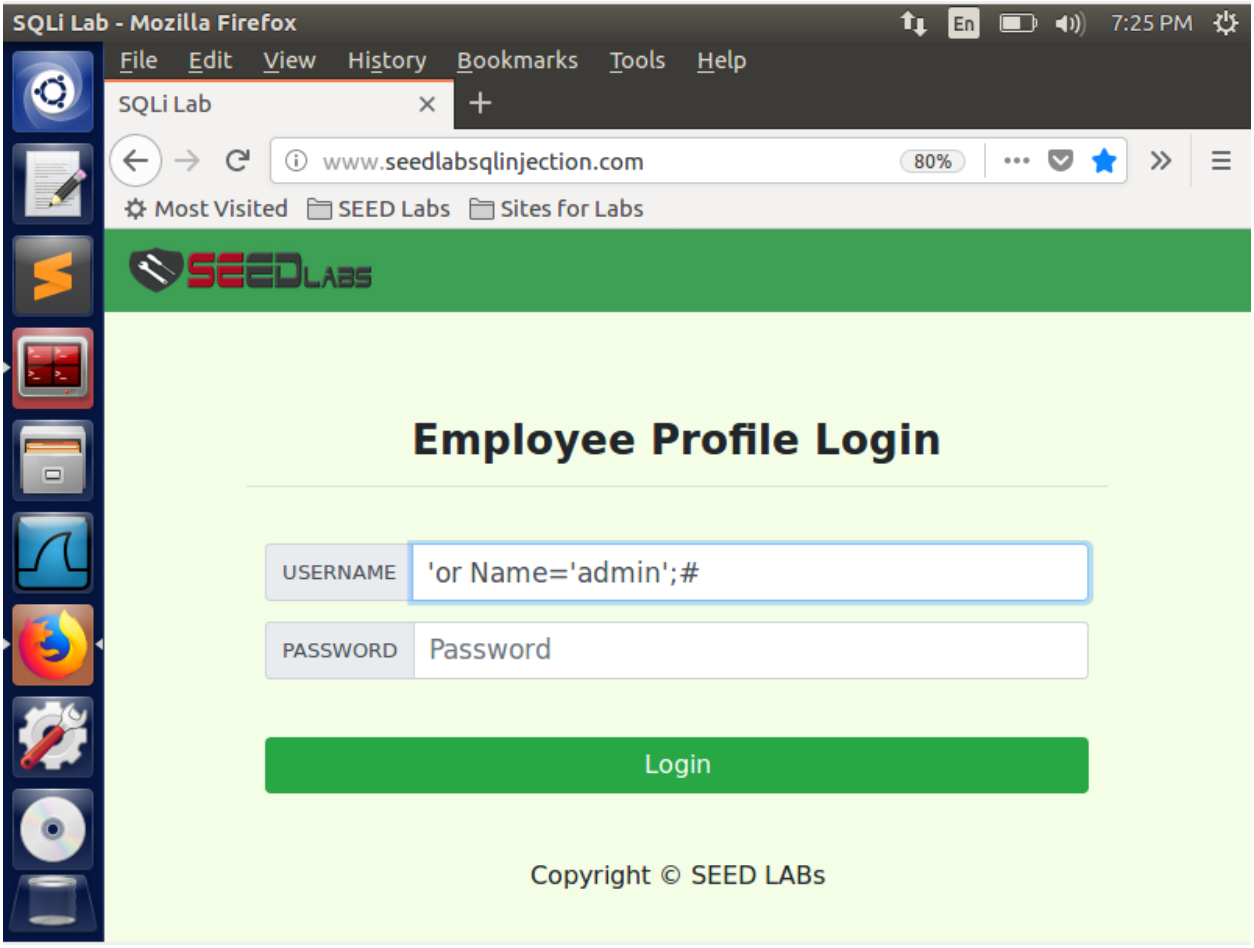
3.2 Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage:

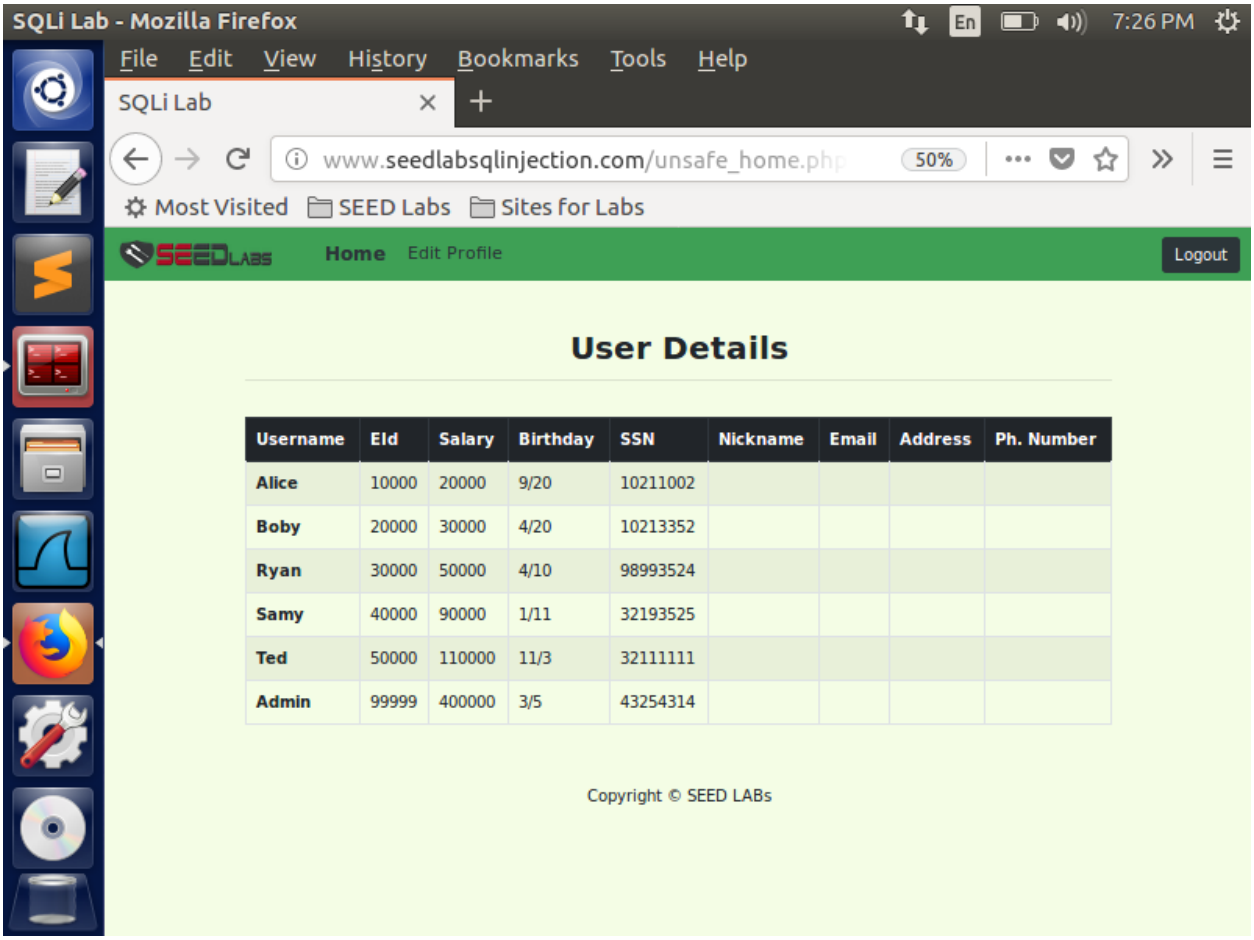
Before starting with the task, I opened the given website www.SEEDLabSQLInjection for the SQL Injection attack. This is the home page of the website.



To login as admin, without knowing the password of the admin, I used SQL Injection attack to login as admin without knowing the password for admin. I used the code ‘ or Name = ‘admin’;# in the input filed of the username. I used the mentioned code because the username and the password are the inputs to the where clause of the SELECT SQL statement. I try to modify the SELECT statement in the backend of the website by giving the single quotes that closes the input for the input id, then I used OR in the code so that we insert name as admin, after we are logged in as admin. I gave the # in the code, so that it comments out the rest of the SQL SELECT statement such that it skips the password input.

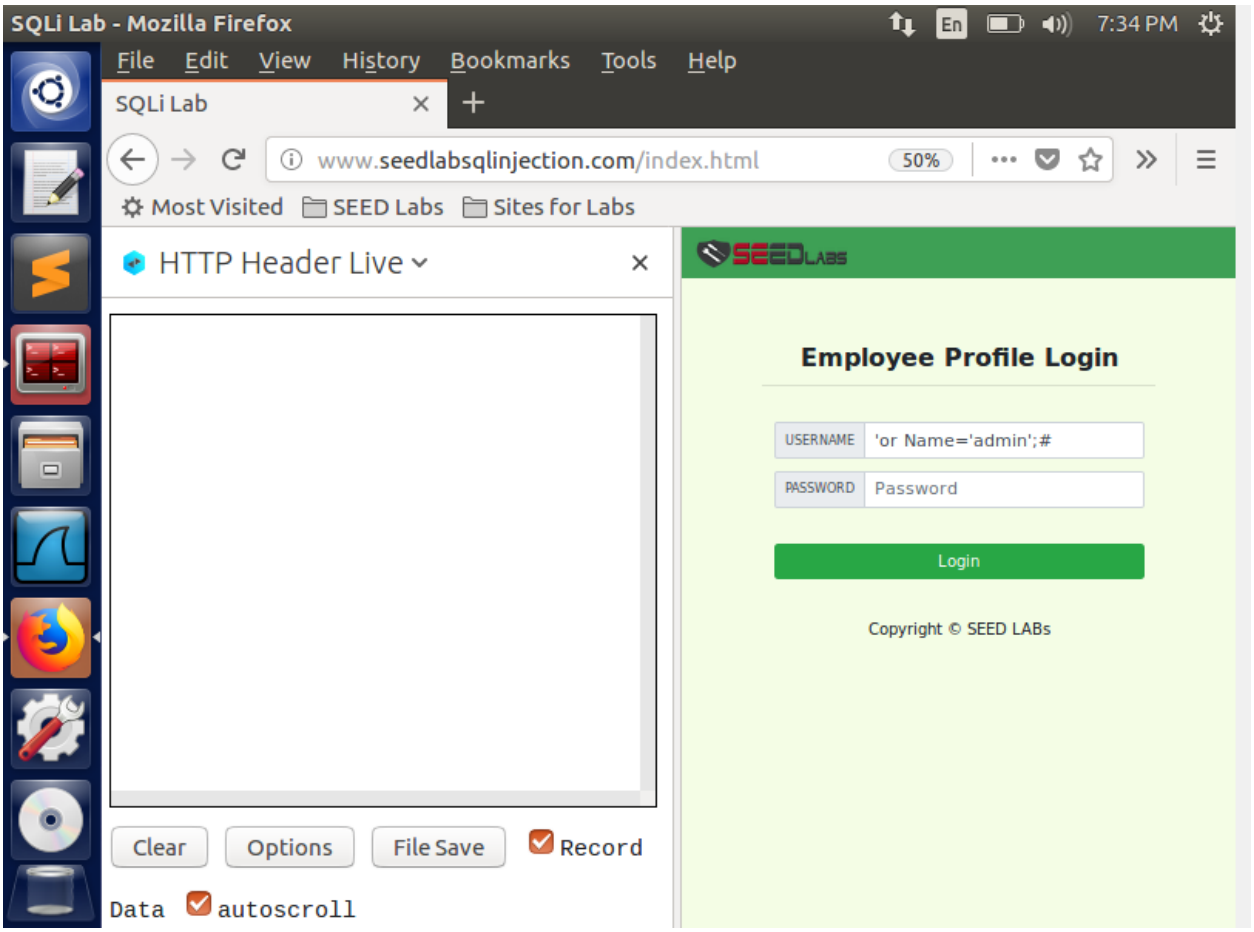


After performing the SQL Injection in the username field, I was able to log in as admin without entering the password for the admin and view the details of other users who are members of the website.

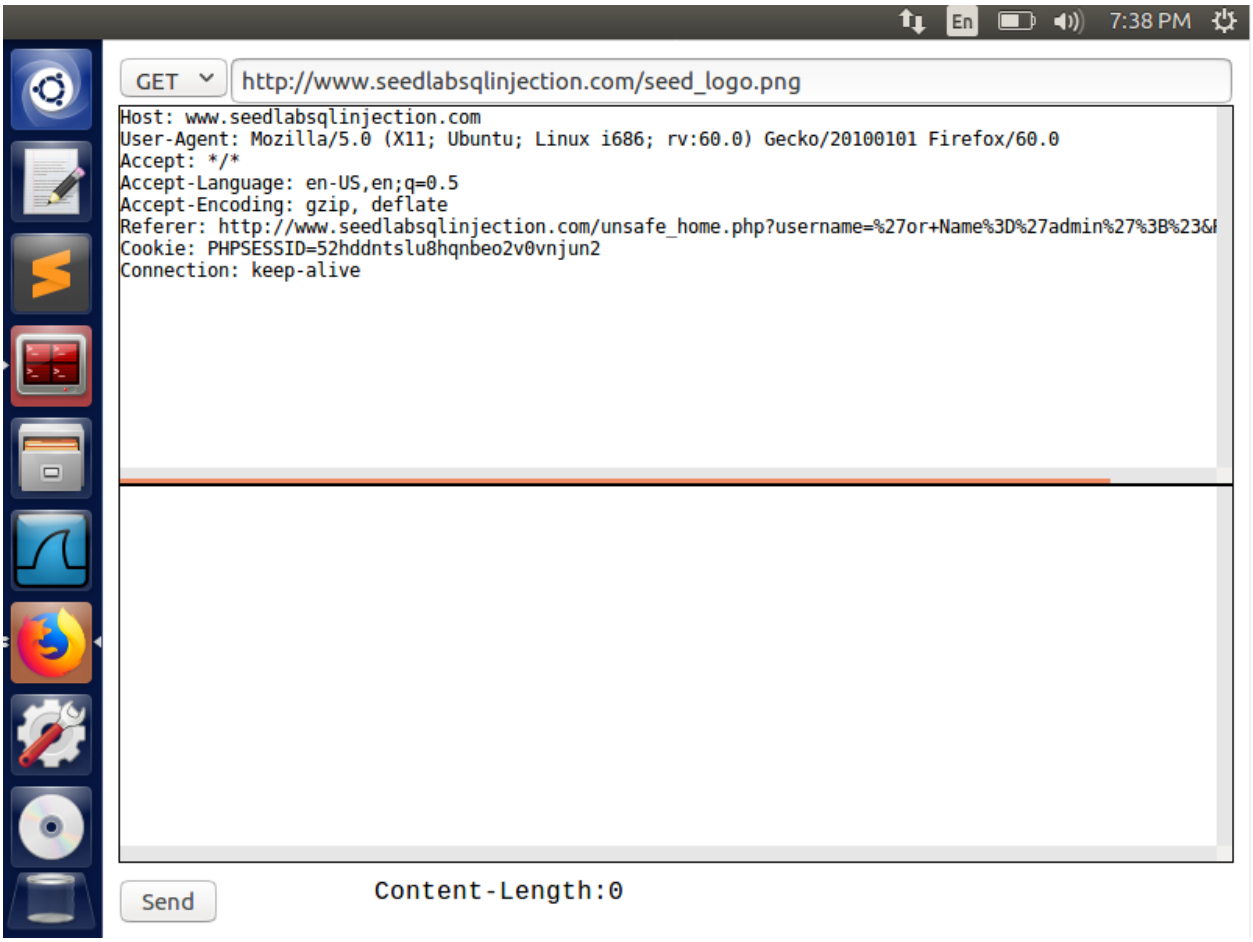


Task 2.2: SQL Injection Attack from command line:

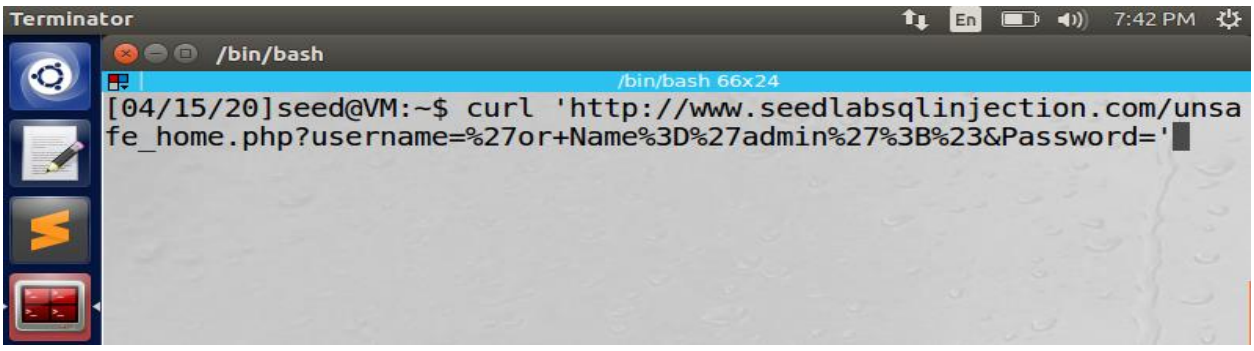
Before doing the SQL injection attack from the command line, I used the “HTTP Live Headrer” firefox add on to capture the complete structure of the HTTP request for the login of a user.



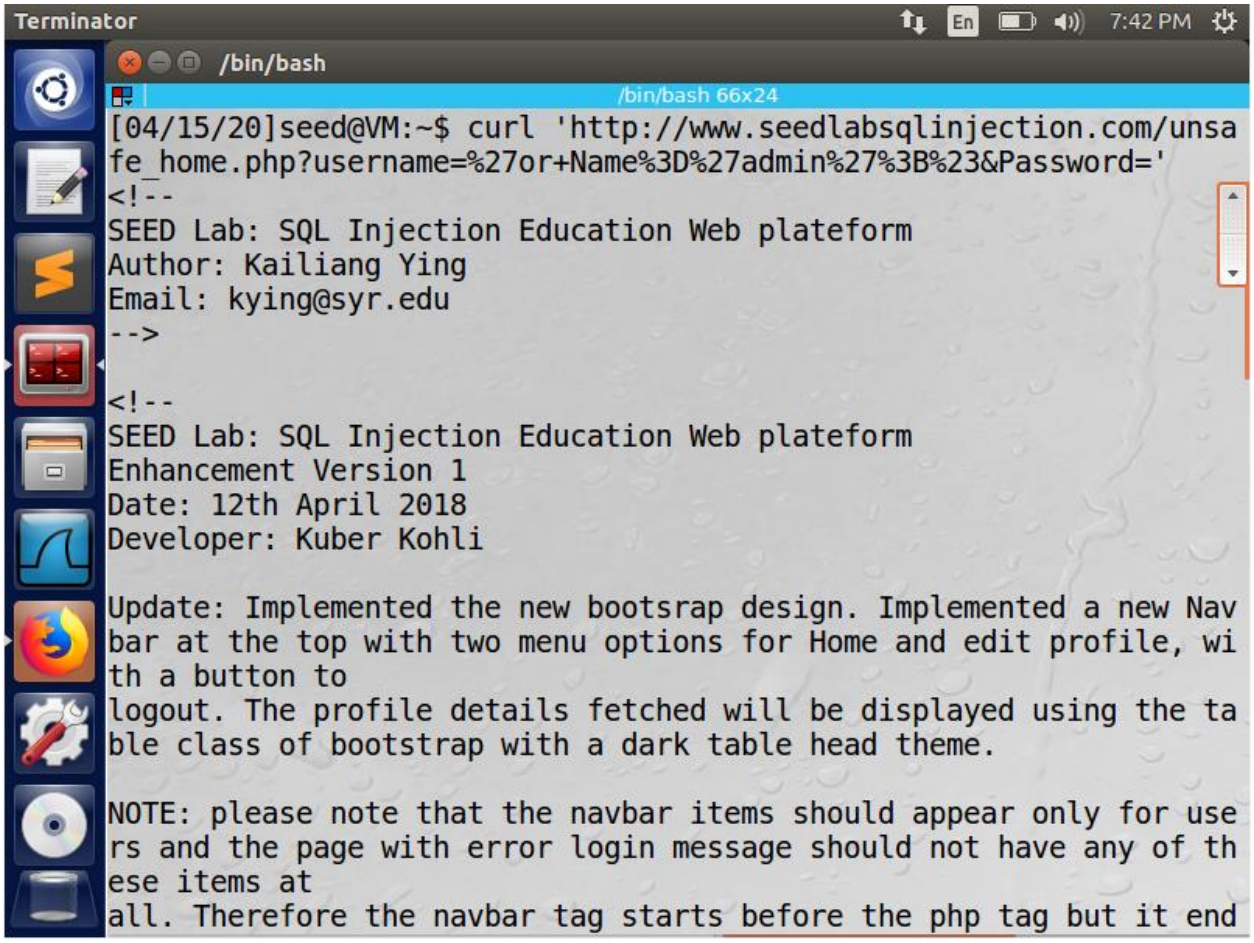
This was the complete structure of the HTTP GET request for the logging in of a user. From this I was able to get the complete HTTP request so that I can encode the special characters required for the attack. I copied the link from the Referer parameter and saved it for later purpose.



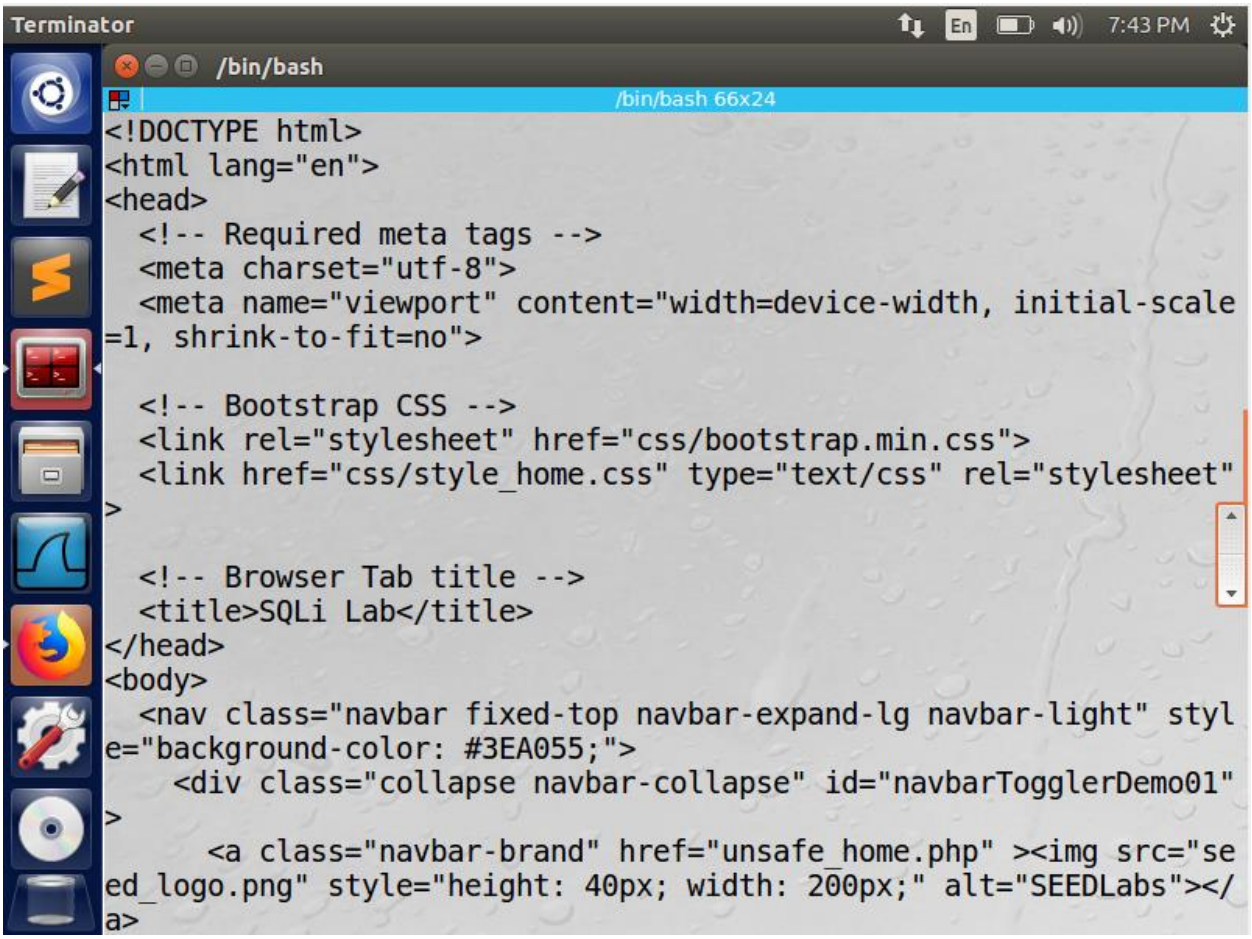
To perform the SQL injection from the command line, I opened the terminator and used the curl command so that I can listen from the URL request on my command line. I used the curl command along with the complete HTTP URL of the website which I got from the firefox add on.



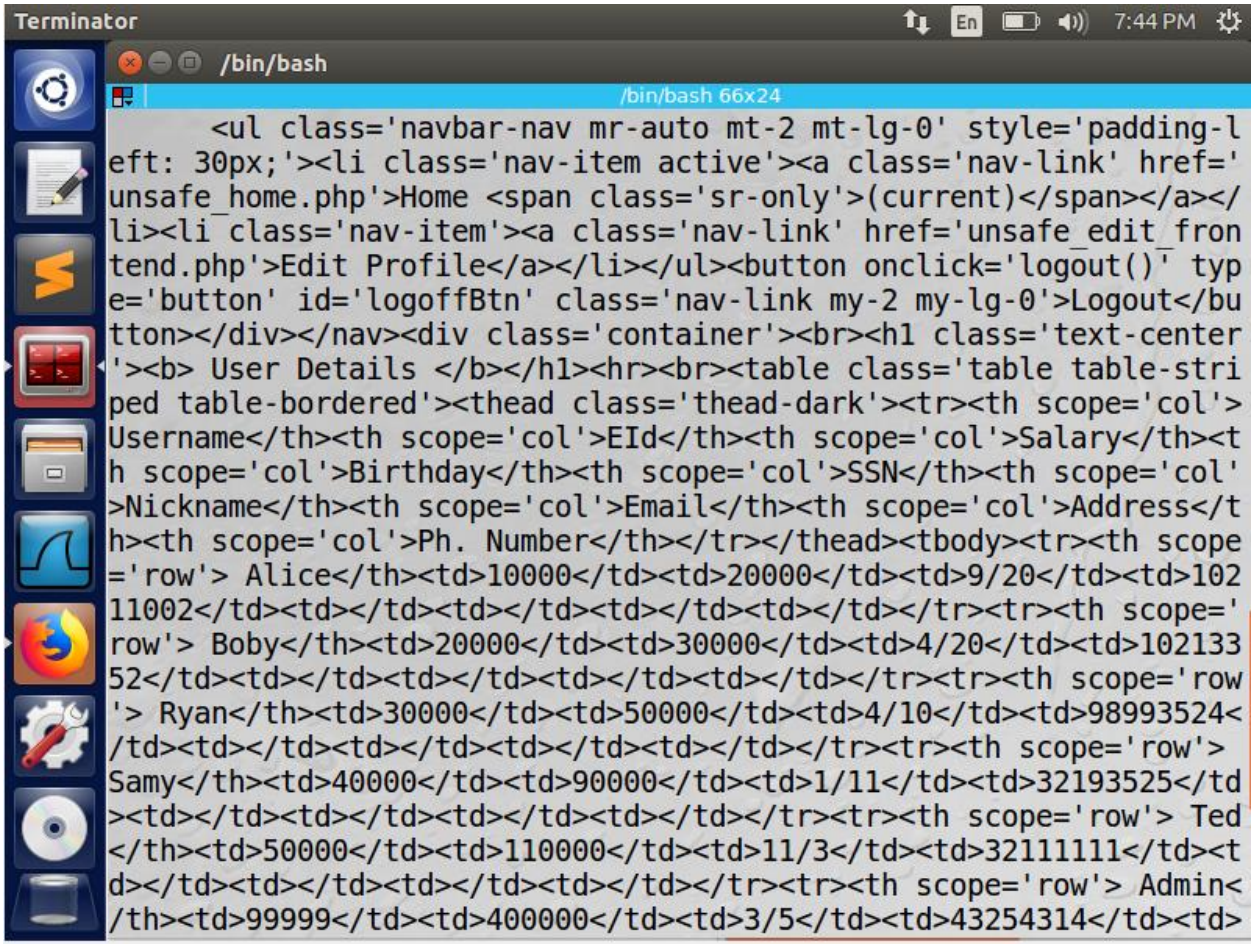
After using the curl command from the terminator, I got the entire website in the form of the HTML code. From the HTML code I was able to see that I got the all the information from the admin page.

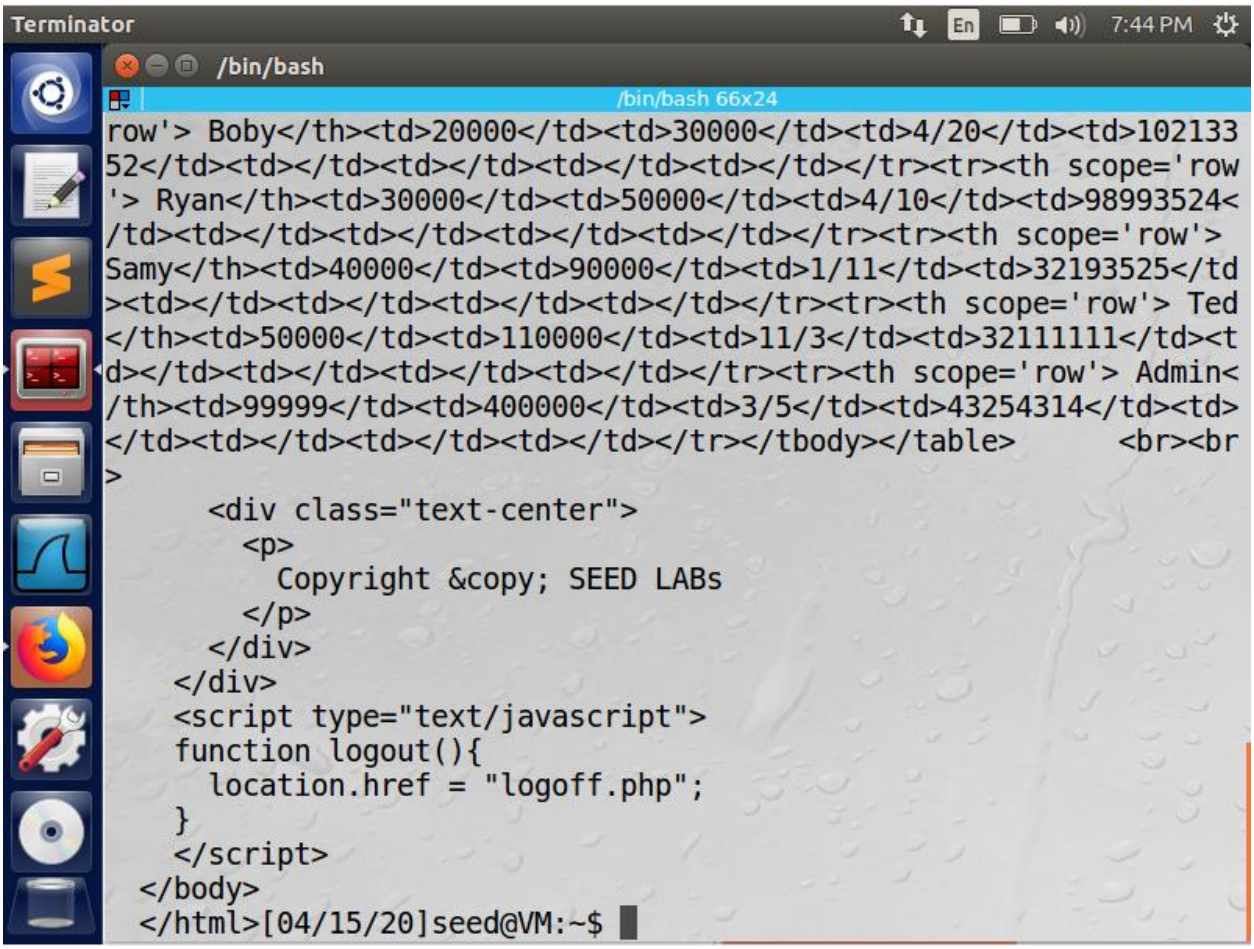


This is the HTML code that I got using the curl command. This is the webpage of the admin's home page.



I was able to get the profile information of other users from the admin's home page in the form HTML table. Since the output is in the form of HTML code, I was able to get the entire profile information of the other users in the form HTML table.



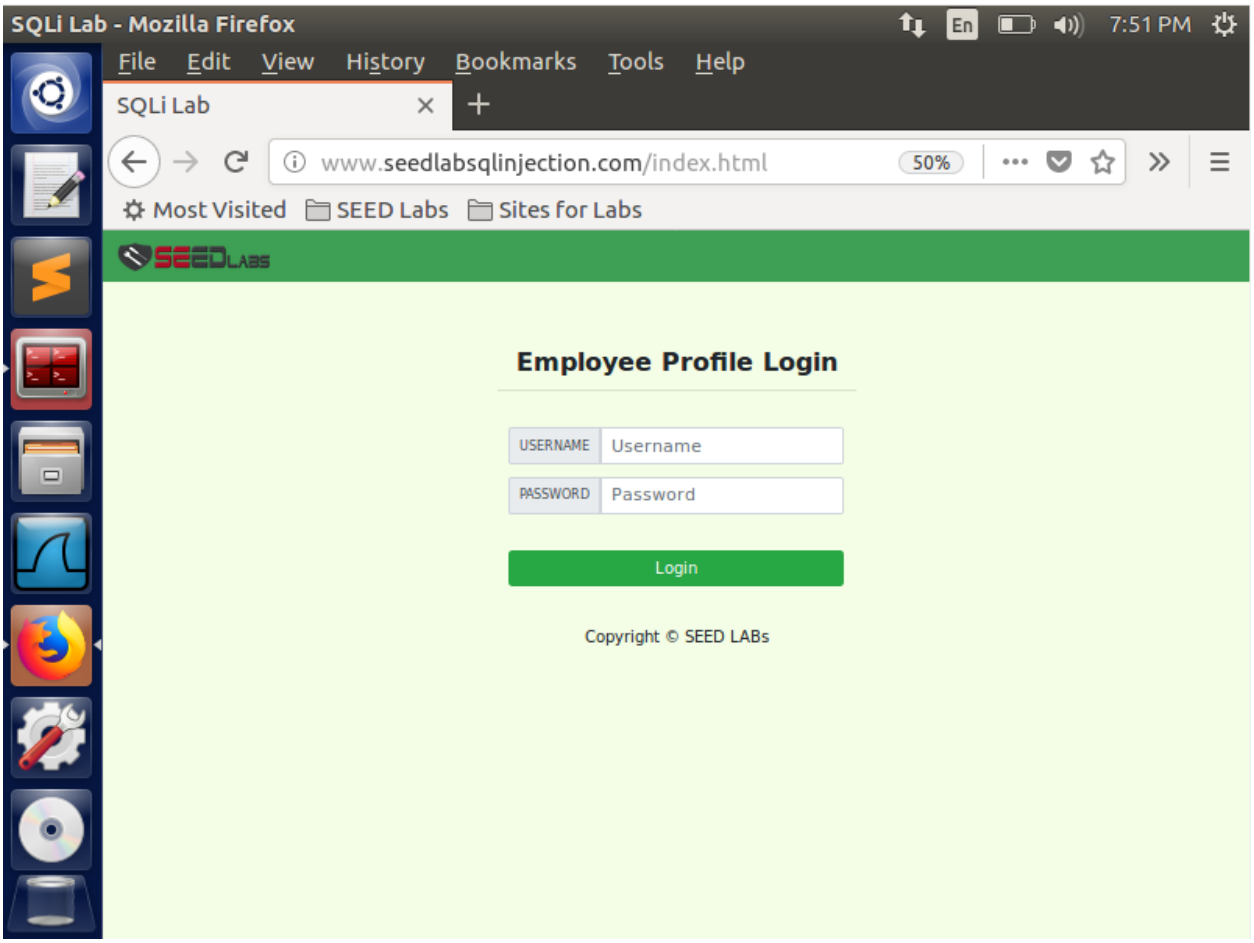

Terminator
En
7:44 PM
/bin/bash
/bin/bash 66x24
row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>102133
52</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row
'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524<
/td><td></td><td></td><td></td><td></td><td></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><td></td></tr><tr><th scope='row'> Ted
</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><t
d></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin<
/th><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>

Copyright © SEED LABs

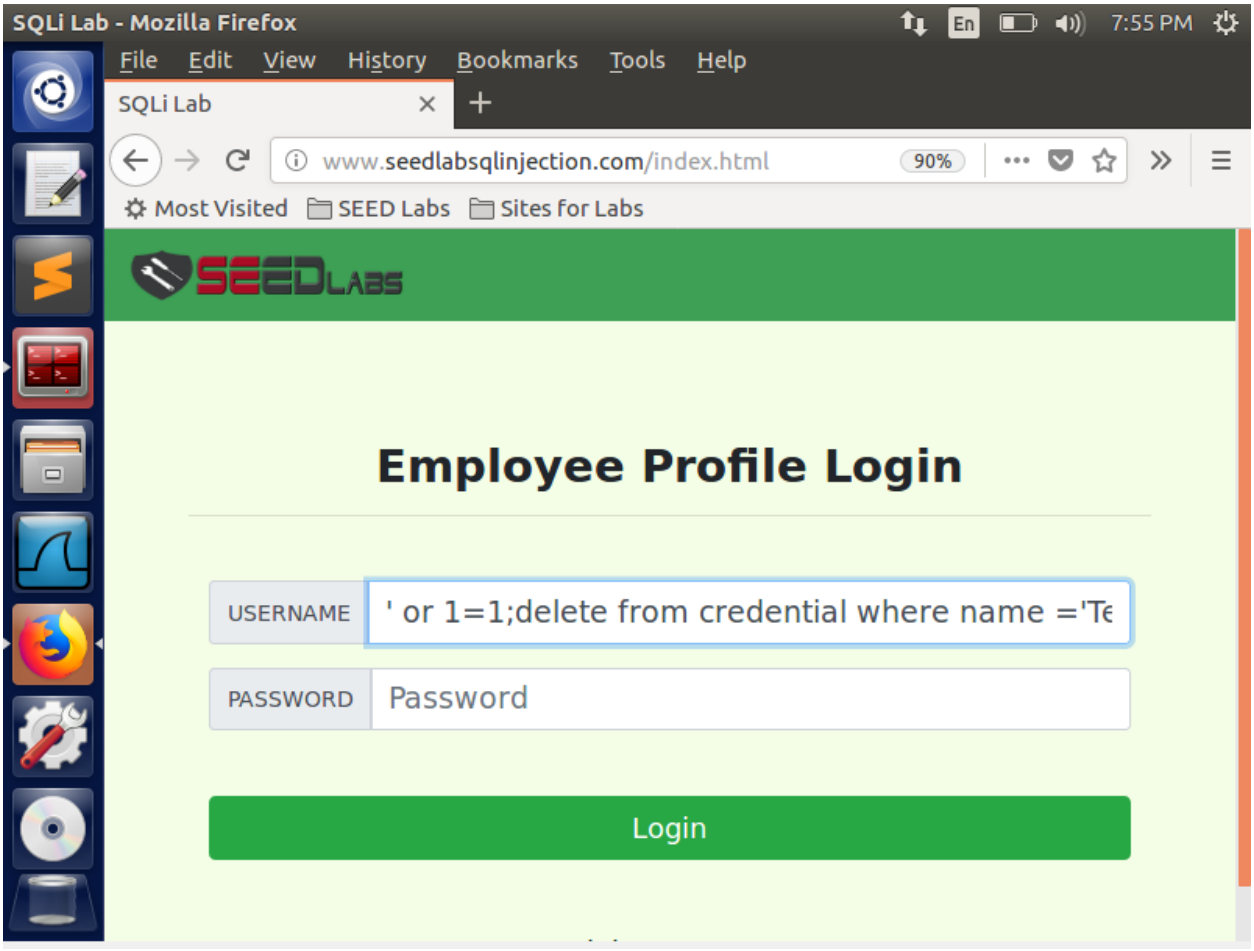
<script type="text/javascript">
function logout(){
location.href = "logoff.php";
}
</script>
</body>
</html>[04/15/20]seed@VM:~$

Task 2.3: Append a new SQL statement:

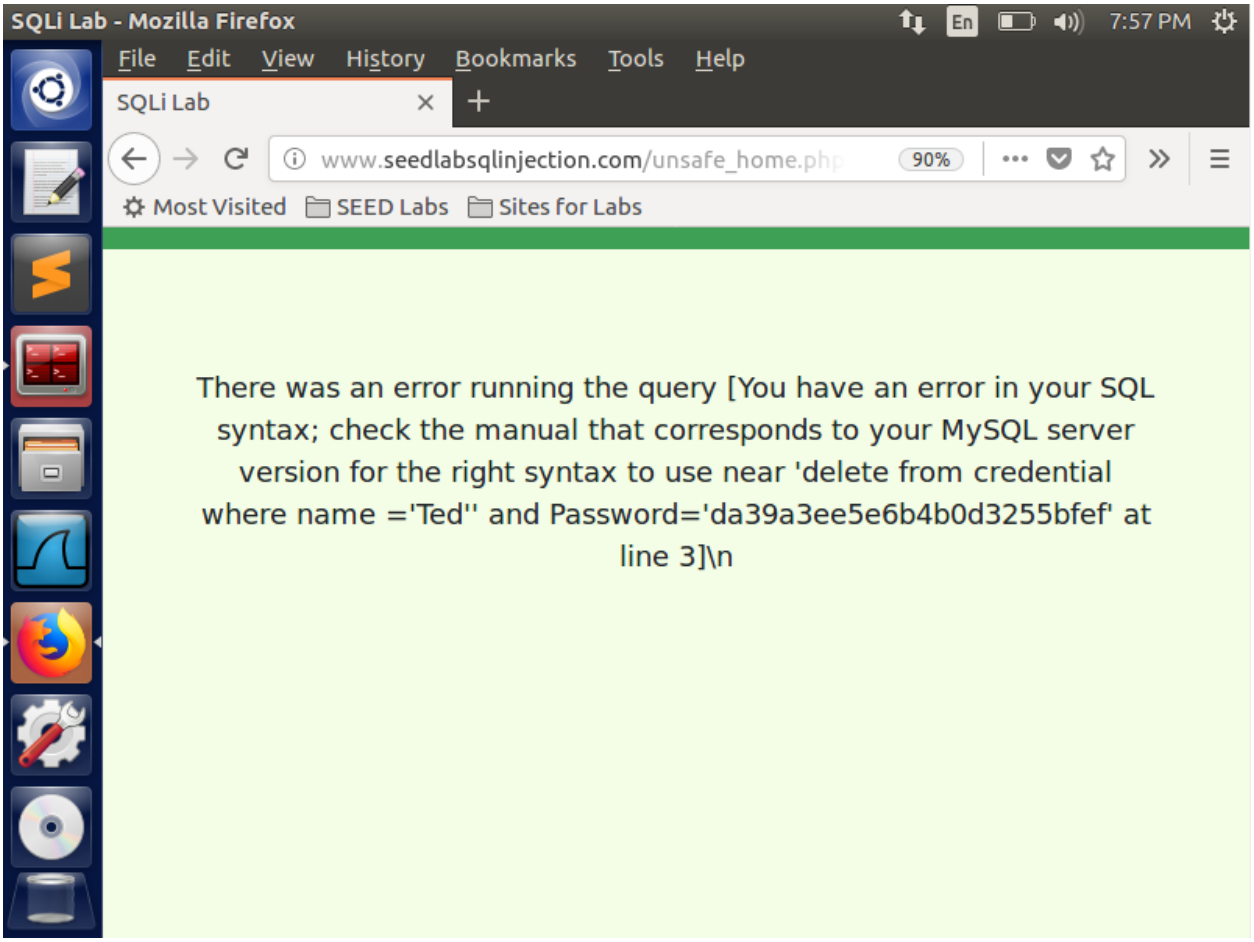
Before starting with the task of appending a new SQL statement , I opened the given website www.SEEDLabSQLInjection for the SQL Injection attack. This is the home page of the website.



I tried to delete the user Ted from the database by performing the SQL injection attack by appending the DELETE SQL command along with the existing SELECT statement in the backend of the website. I gave the code ‘ or 1=1; delete from credential where name=’Ted’; as input to the username field. I used the single quotes that closes the input for the input id and I ended the first statement with s semicolon so that I can append the DELETE statement along with the first SQL query.



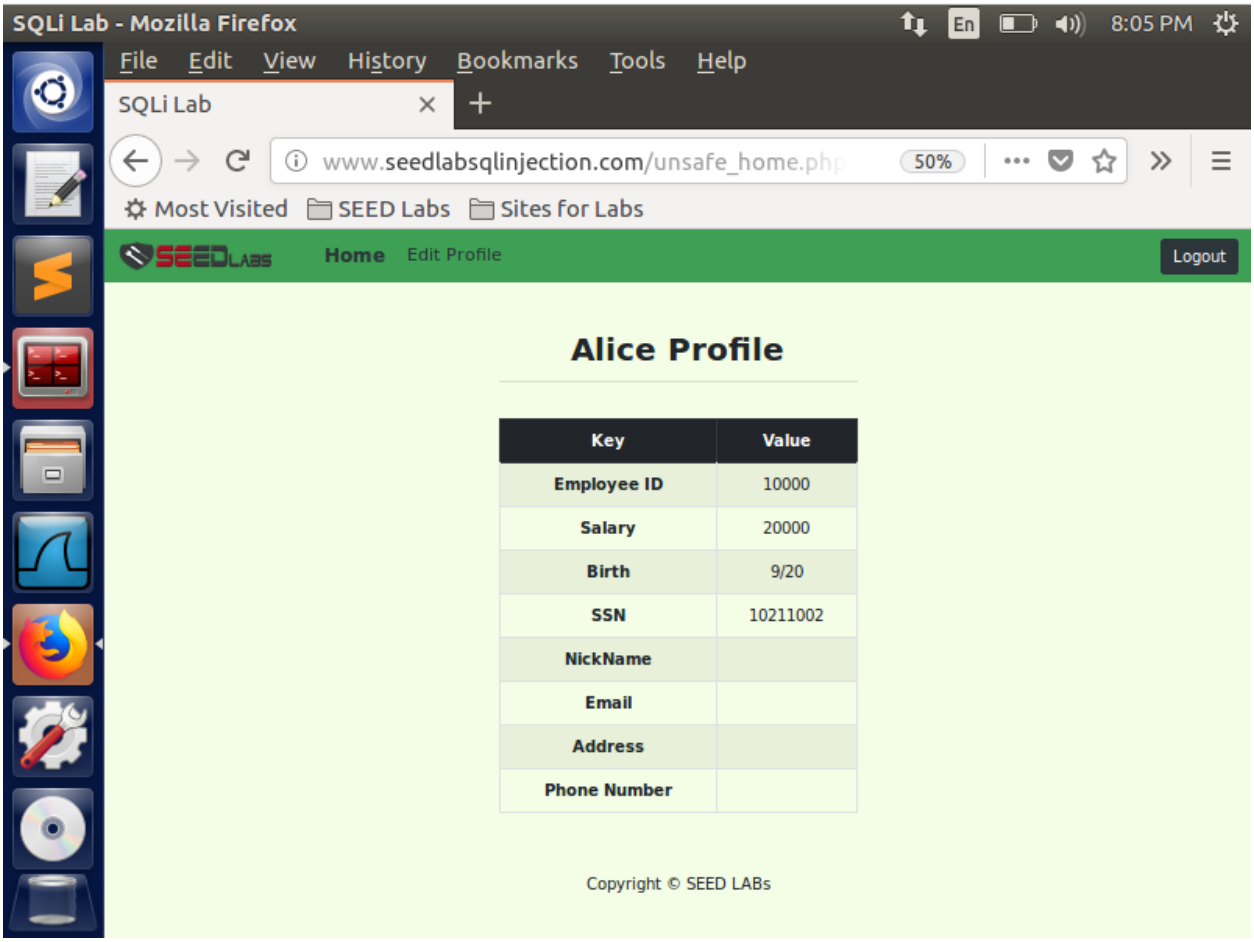
When I gave the SQL injection code in the username input field and clicked login without entering the password, I was able to see an error message displaying in the webpage saying that an error has been occurred while running the query. The attack was failed, and this is because of the countermeasure in MySQL that prevents appending multiple SQL statements from executing when invoked from the backend PHP code.



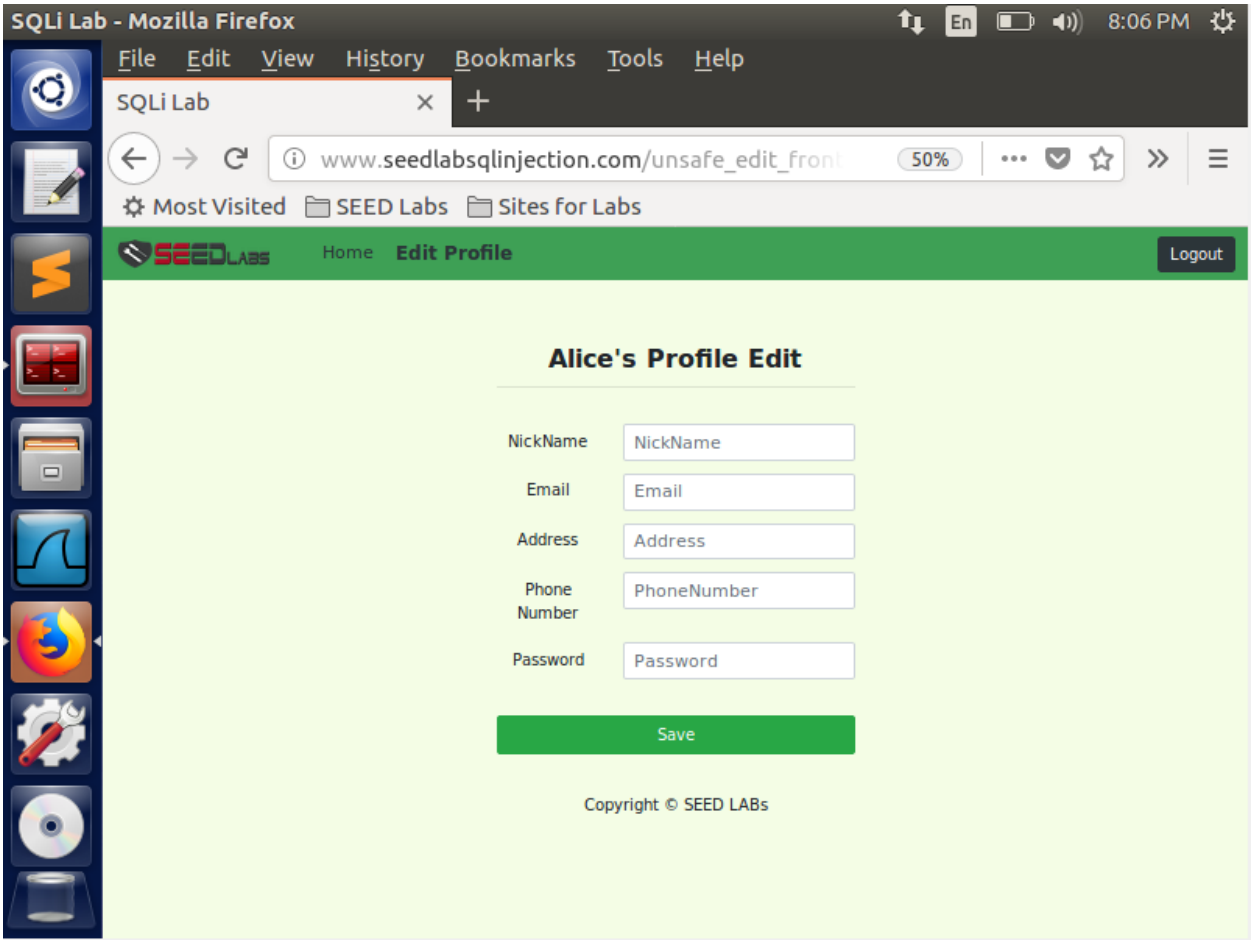
3.3 Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary:

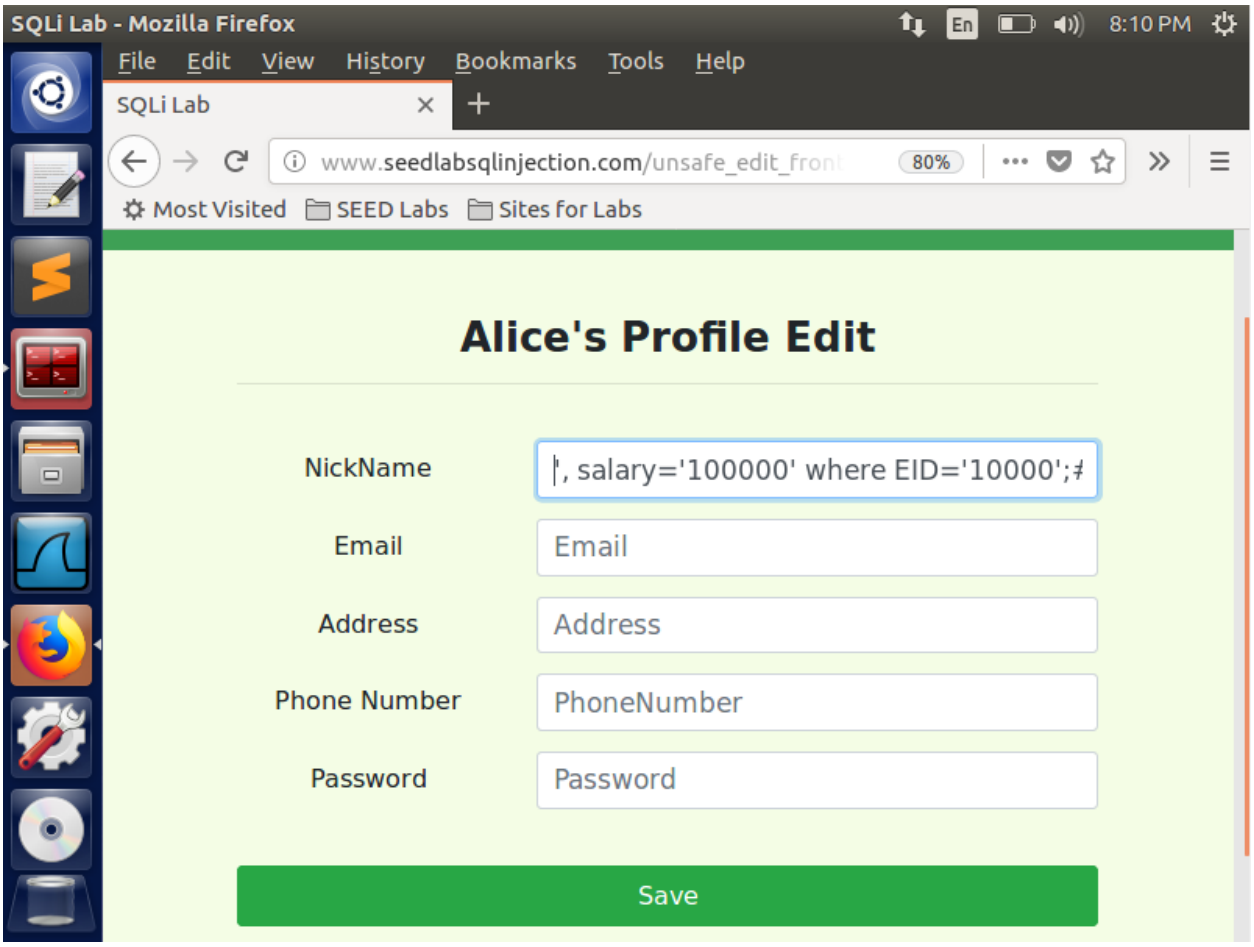
Before doing the task of modifying the own salary, I opened the given website and logged in as Alice using her given credentials and navigated to the home page of Alice. This is the home page of Alice where I was able to see the basic information about Alice.



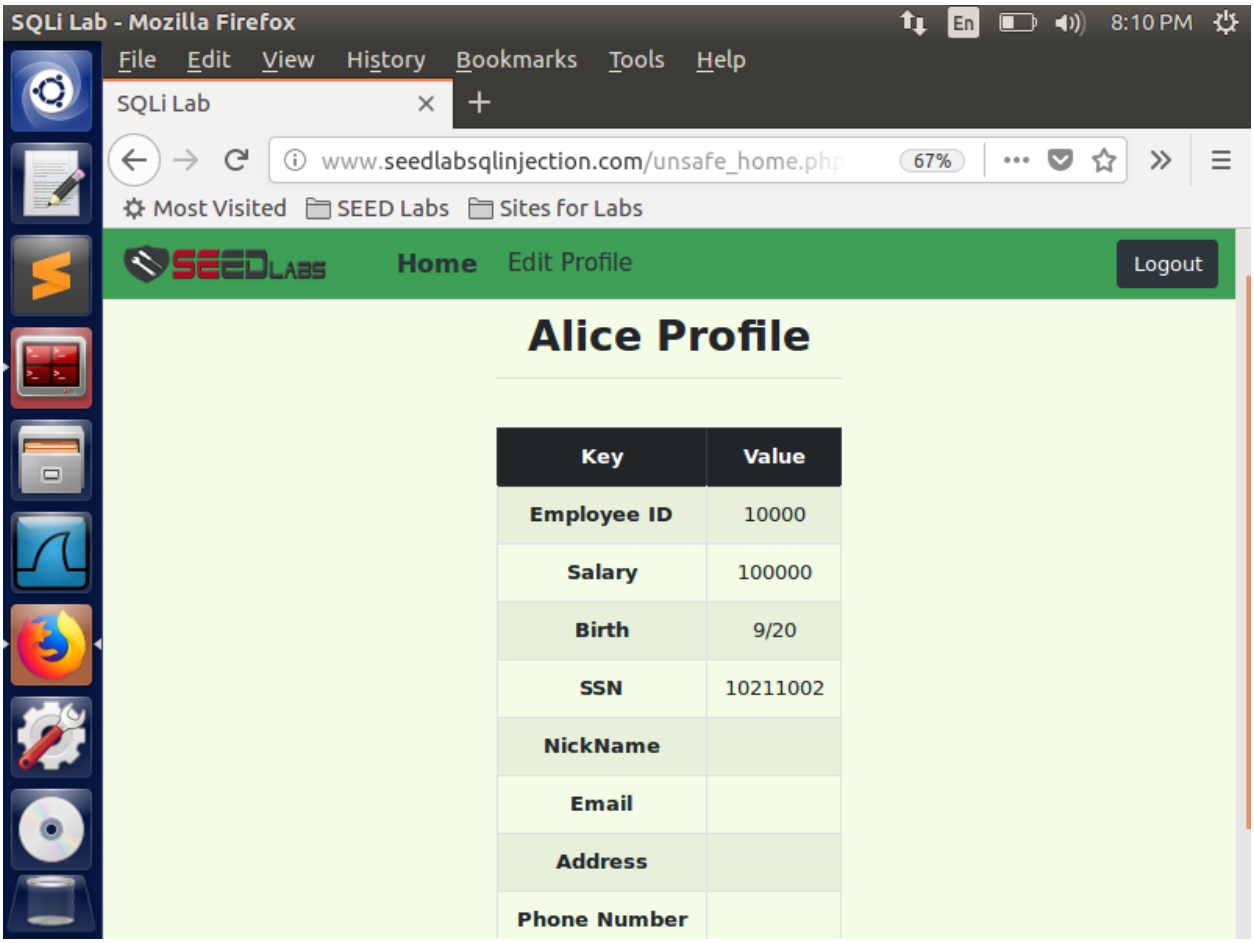
Then I navigated to the Edit Profile page where I can edit the basic information about Alice. One thing I noticed in the Edit profile page was that I was not able to see an editable field for Salary.



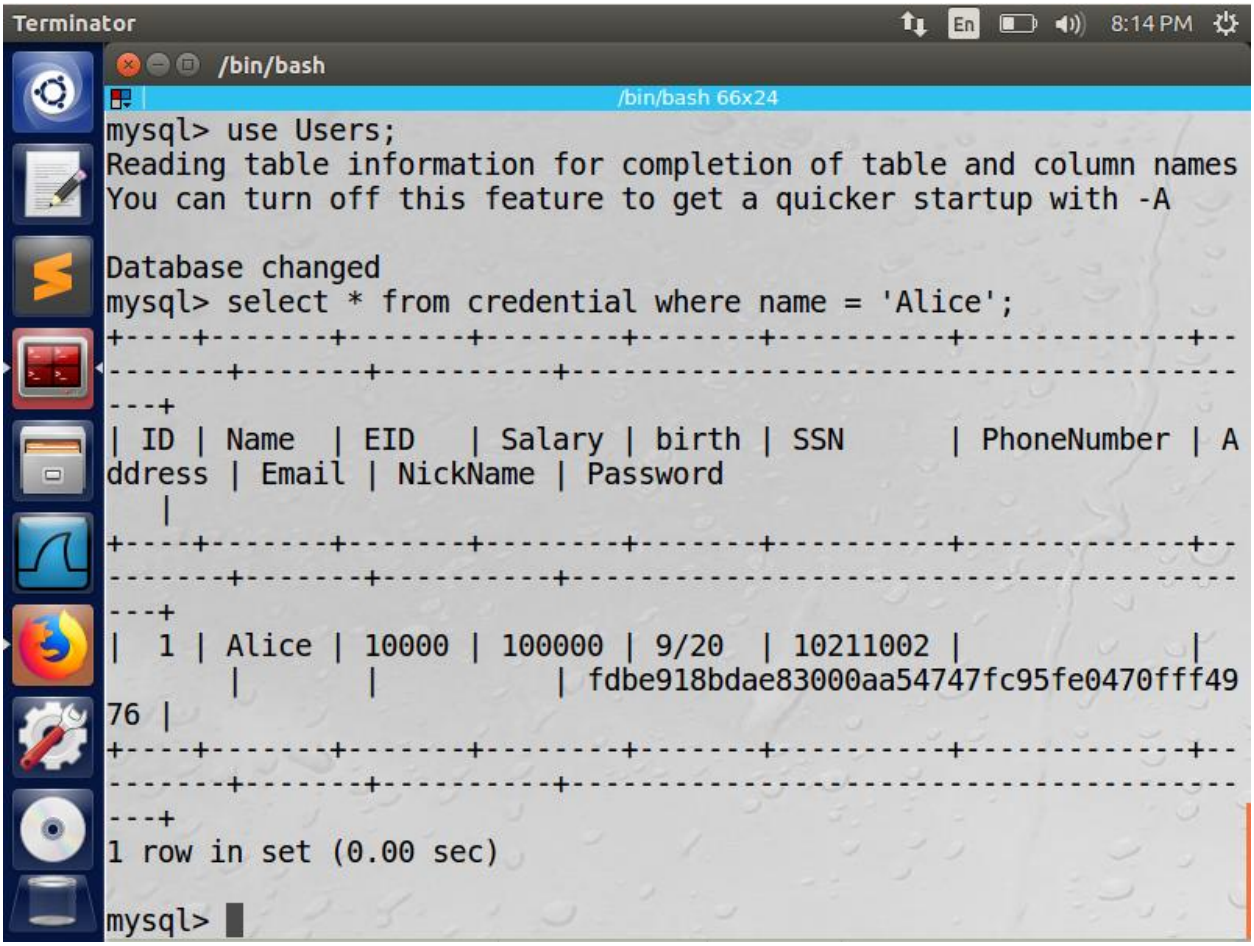
After noticing that there is no editable field for salary, I used the nickname field to perform the SQL Injection attack. I gave the code ‘, salary=’100000’ where EID = ‘10000’;#, so that I can update the salary of Alice from 20000 to 100000. I gave the query to edit the salary of Alice along with # in the end of the code so that I am commenting the rest of the SQL query used to update the nickname of Alice. Basically, I am overwriting the query to update the nickname of Alice to updating the salary of Alice by using the # so that I am commenting out the rest of the query. I am doing this because salary is only editable by the admin.



After performing the SQL Injection attack on the salary of Alice, I came back to the home page of Alice and I was able to see that the salary has been updated to 100000 from 20000. This is because of the SQL Injection attack, and the attack was successful.

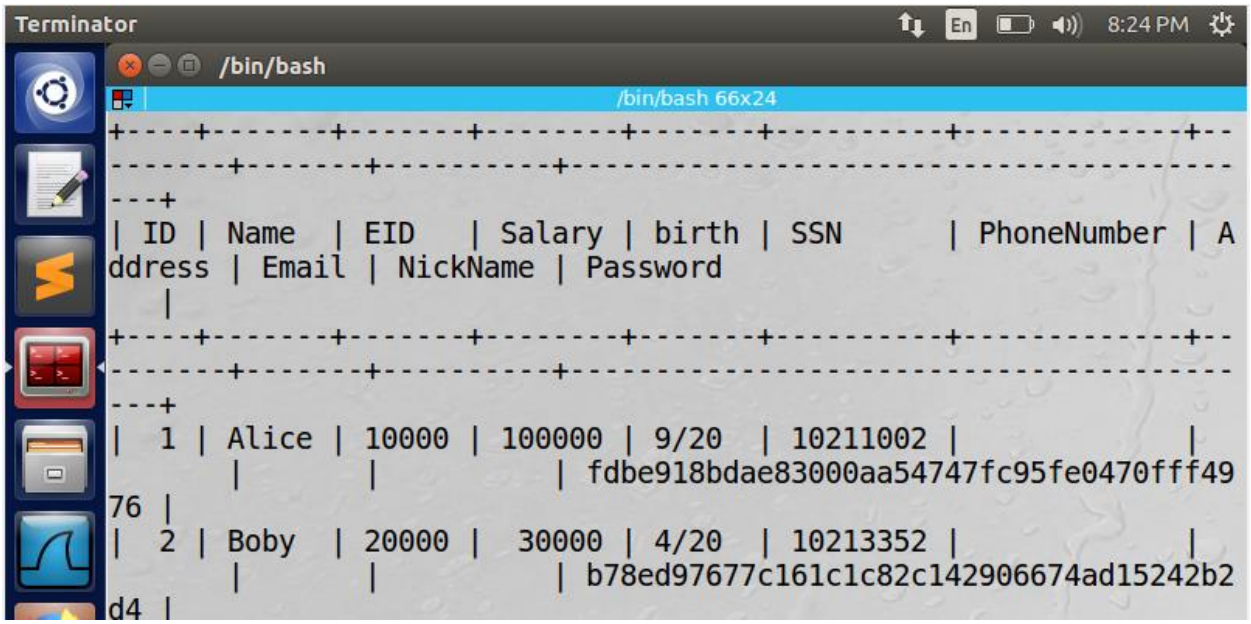


Then I opened the MySQL database to check if the salary has been updated using the SQL Injection attack has been reflected in the backend database also. I used the SELECT statement to fetch the profile information of Alice and I was able to see that the salary of Alice has been updated in the database also. This is because of the SQL Injection attack.

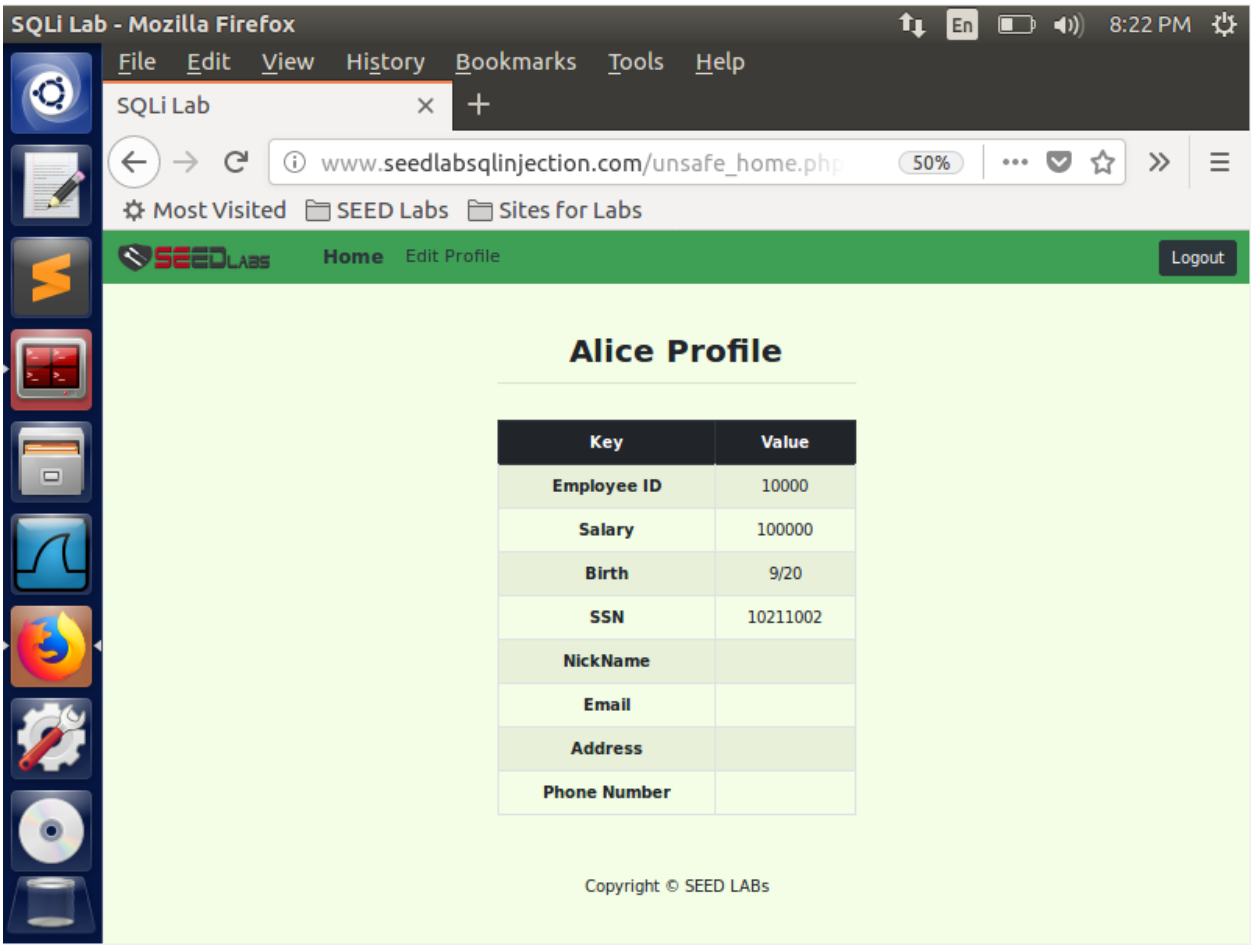


Task 3.2: Modify other people' salary:

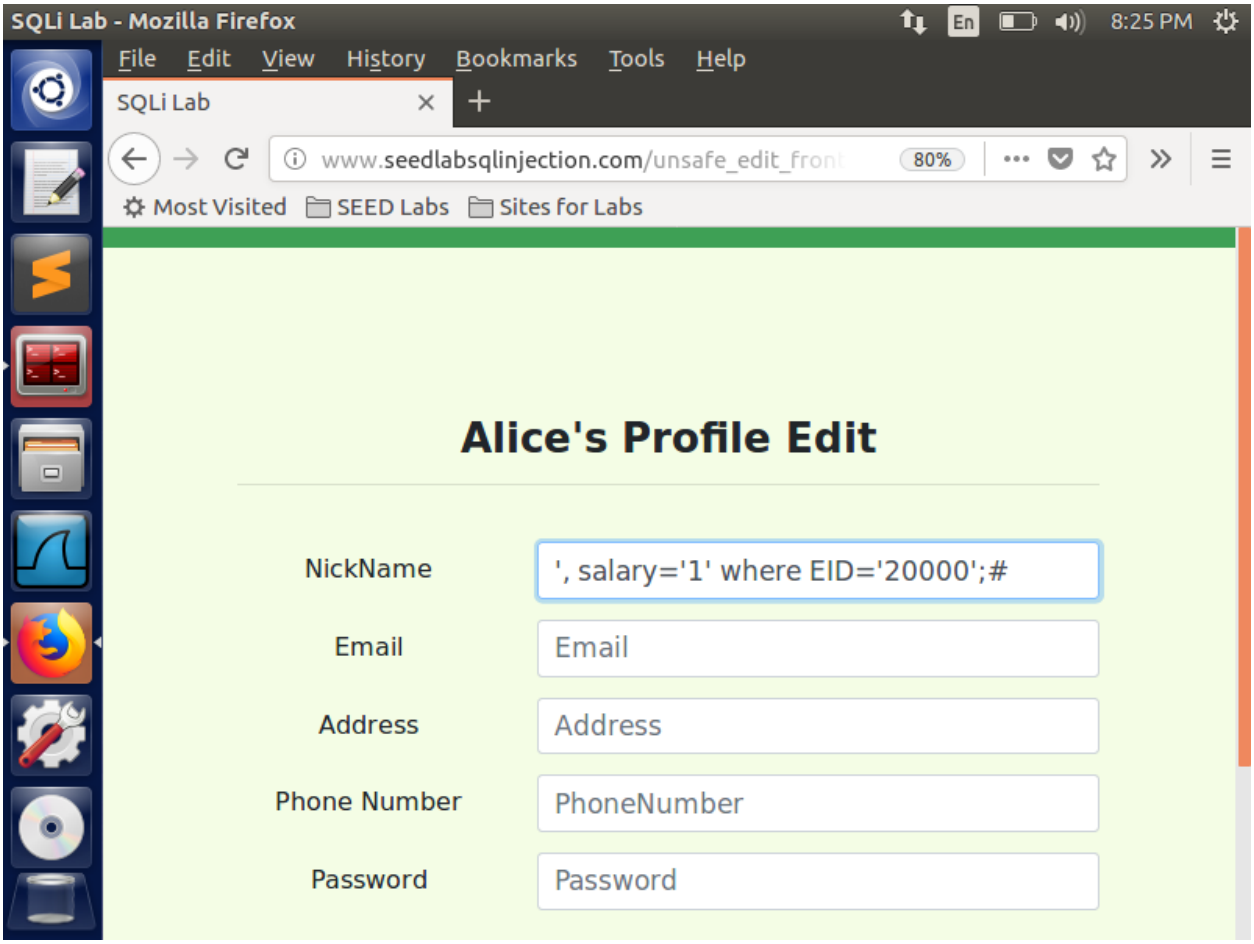
Before performing the task of modifying other people's salary, I chose the salary of Boby to be modified. I opened MySQL from the terminator and opened the Users database and fetched the details of the profile information of all users in the database. I was able to see the salary of Boby was 30000. This shows the snapshot of the database table.



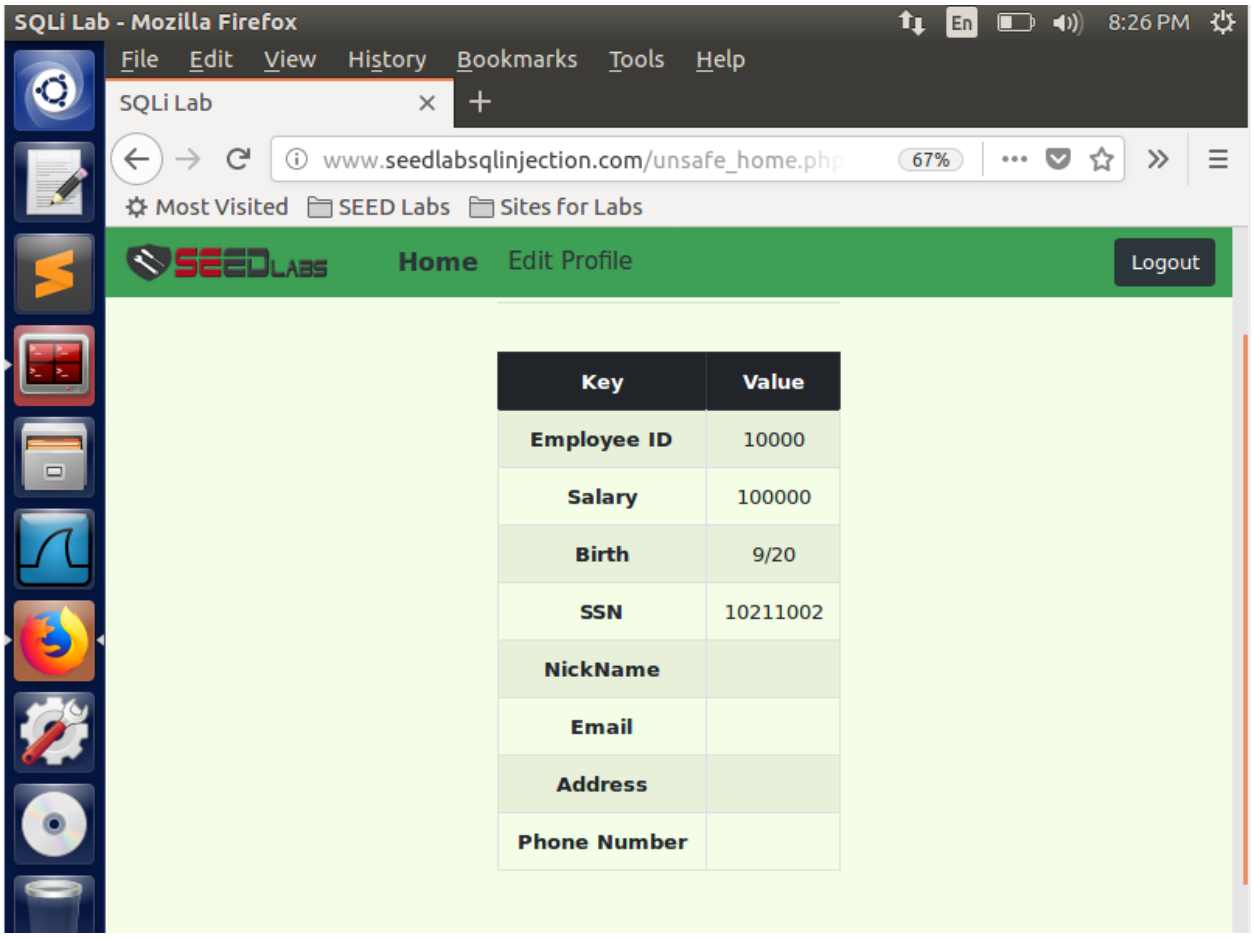
Then I opened the given URL and logged into Alice profile. This is the home page of Alice from where we are going to perform the SQL Injection attack of updating Boby's salary.



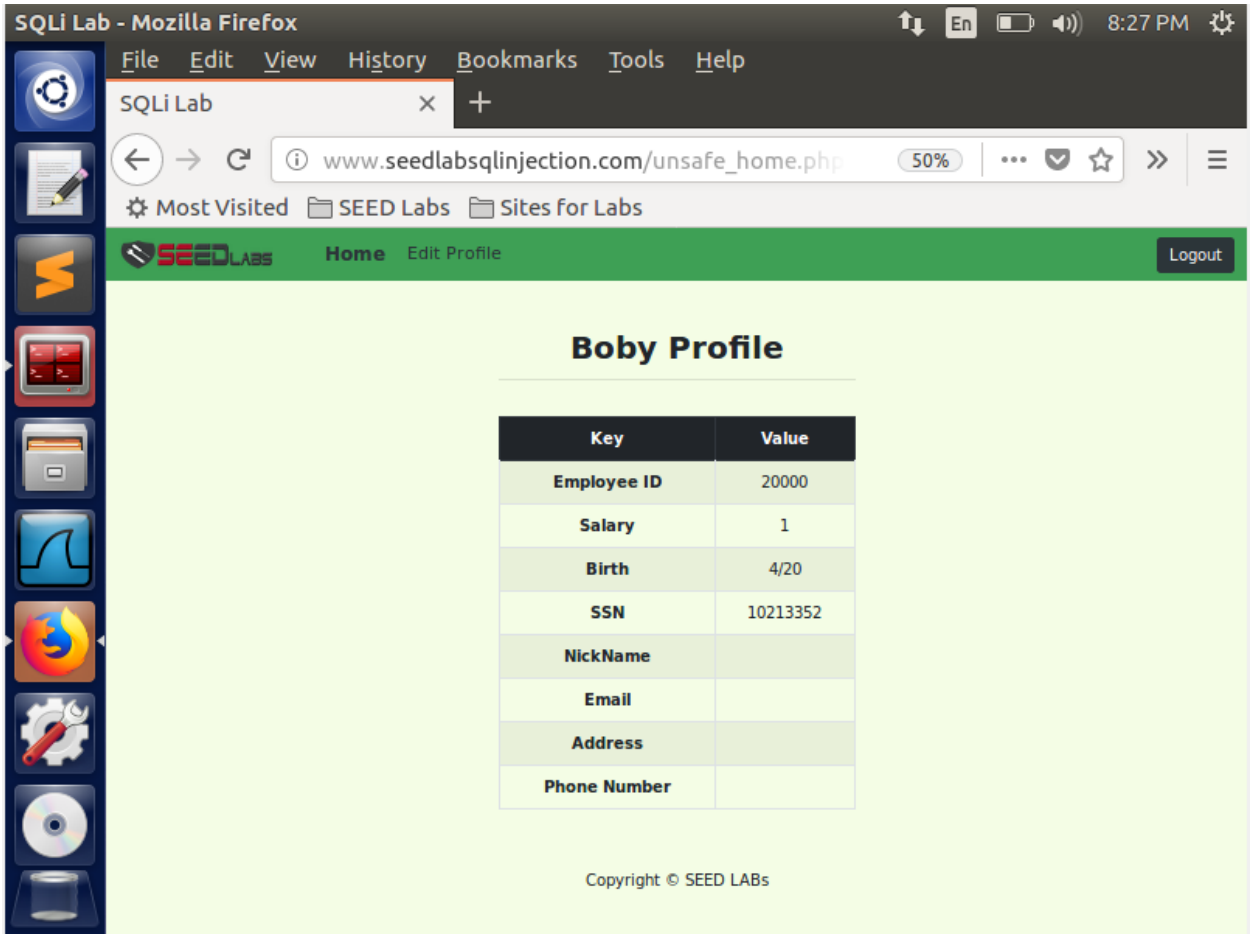
I used the nickname field to perform the SQL Injection attack. I gave the code ‘, salary=’1’ where EID = ‘20000’;#, so that I can update the salary of Boby from 30000 to 1. I gave the query to edit the salary of Boby along with # in the end of the code so that I am commenting the rest of the SQL query used to update the nickname of Alice. Basically, I am overwriting the query to update the nickname of Alice to updating the salary of Boby by using the # so that I am commenting out the rest of the query. I am doing this because salary is only editable by the admin.



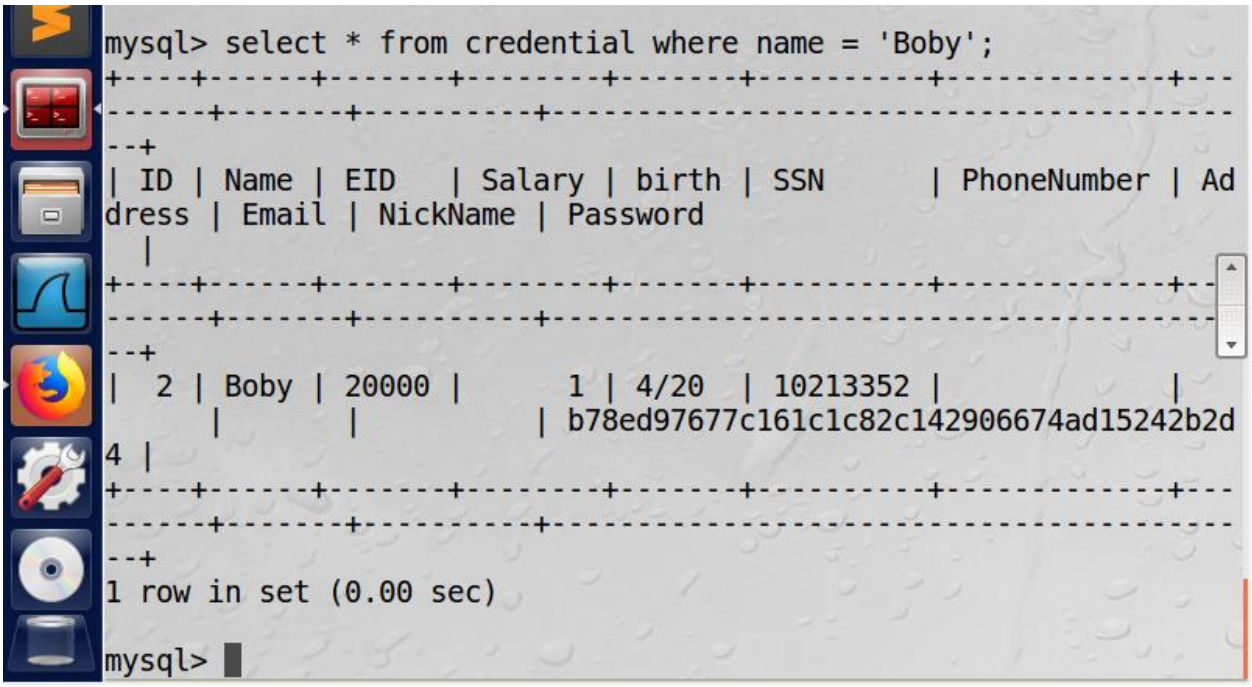
After performing the SQL Injection in the Edit Profile Page of Alice I clicked the save button redirected back to the Home Page of Alice.



Then I logged into Boby's profile using his credentials to check if the salary of Boby has been updated. I was able to see that the salary of Boby has been updated because of the SQL Injection attack performed from Alice's profile.

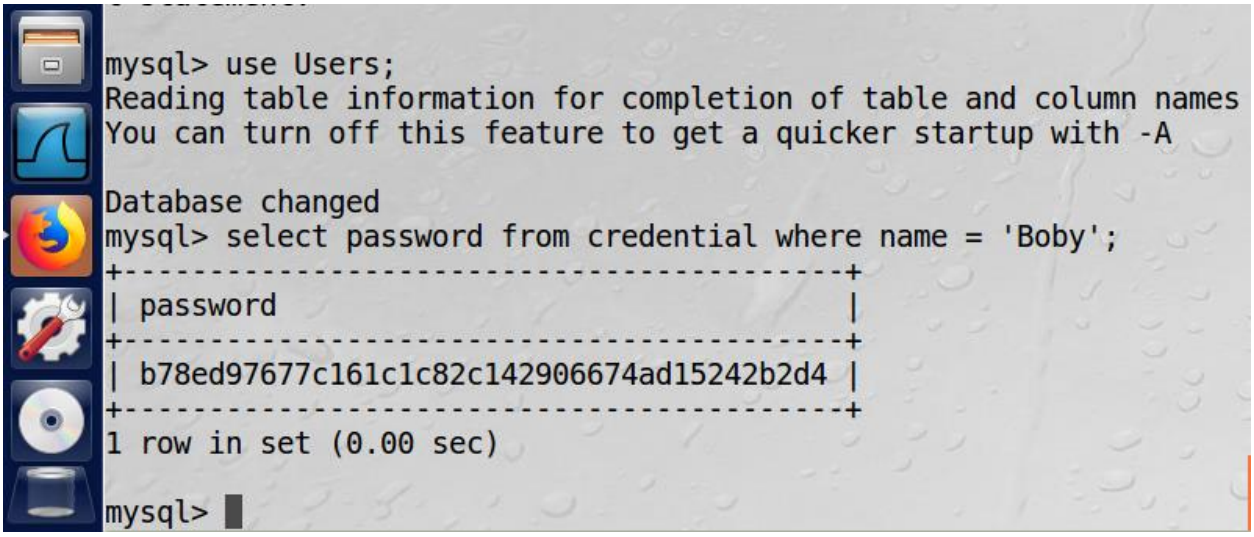


Then I opened the MySQL database to check if the salary has been updated using the SQL Injection attack has been reflected in the backend database also. I used the SELECT statement to fetch the profile information of Boby and I was able to see that the salary of Boby has been updated in the database also. This is because of the SQL Injection.

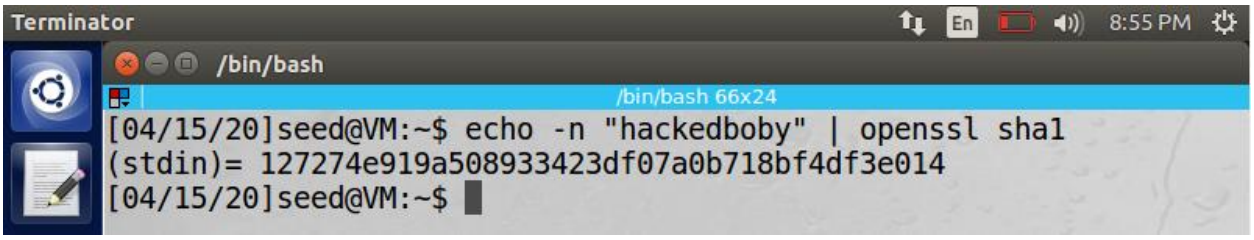


Task 3.3: Modify other people' password:

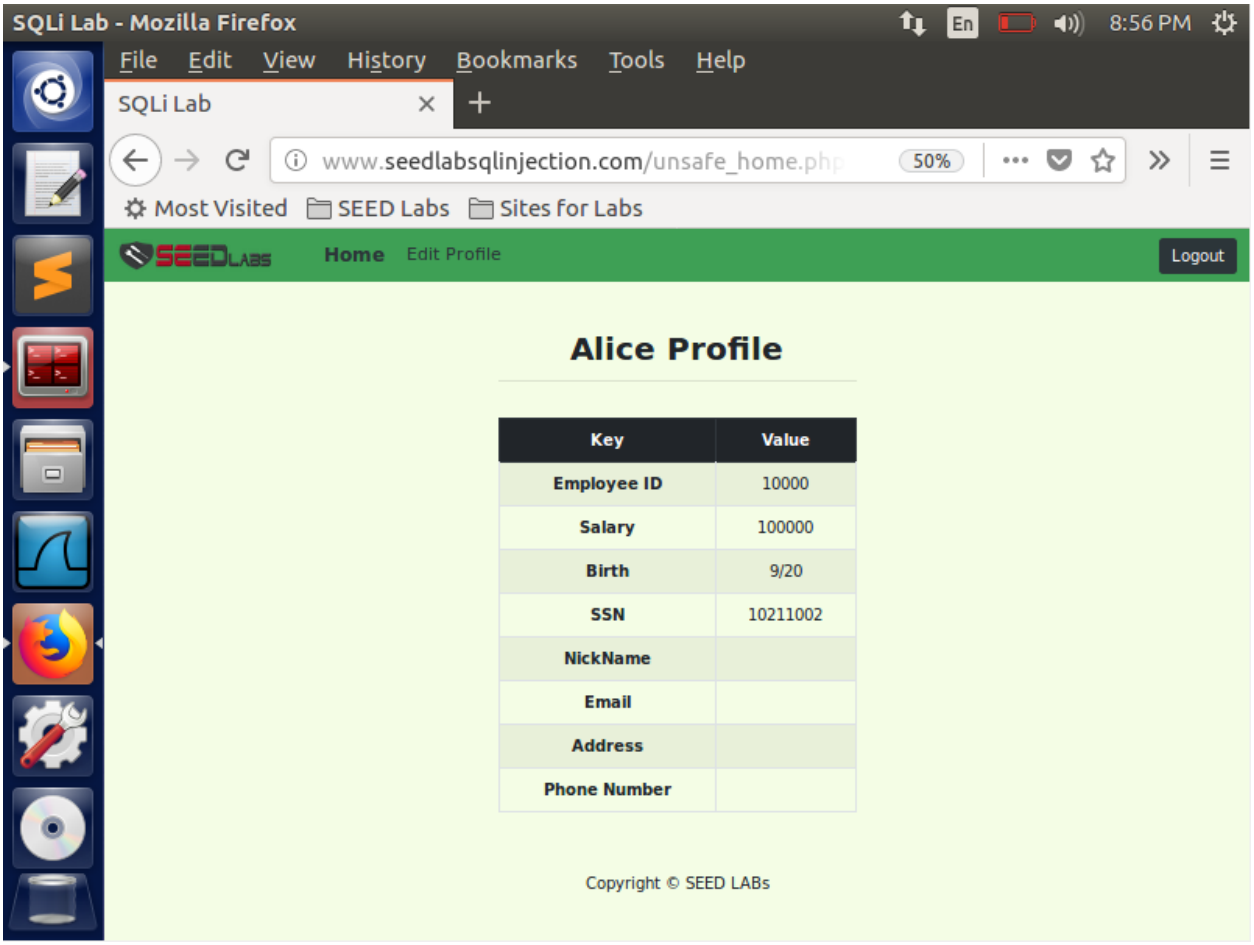
Before doing the task of Modifying other people's password I chose Boby as victim whose password is going to be changed. I opened the MySQL database to view the password of Boby. Then I used the SELECT statement to fetch the password of Boby. I was able to see that the password has been generated using the SHA1 hash function to generate the hash value of the password.



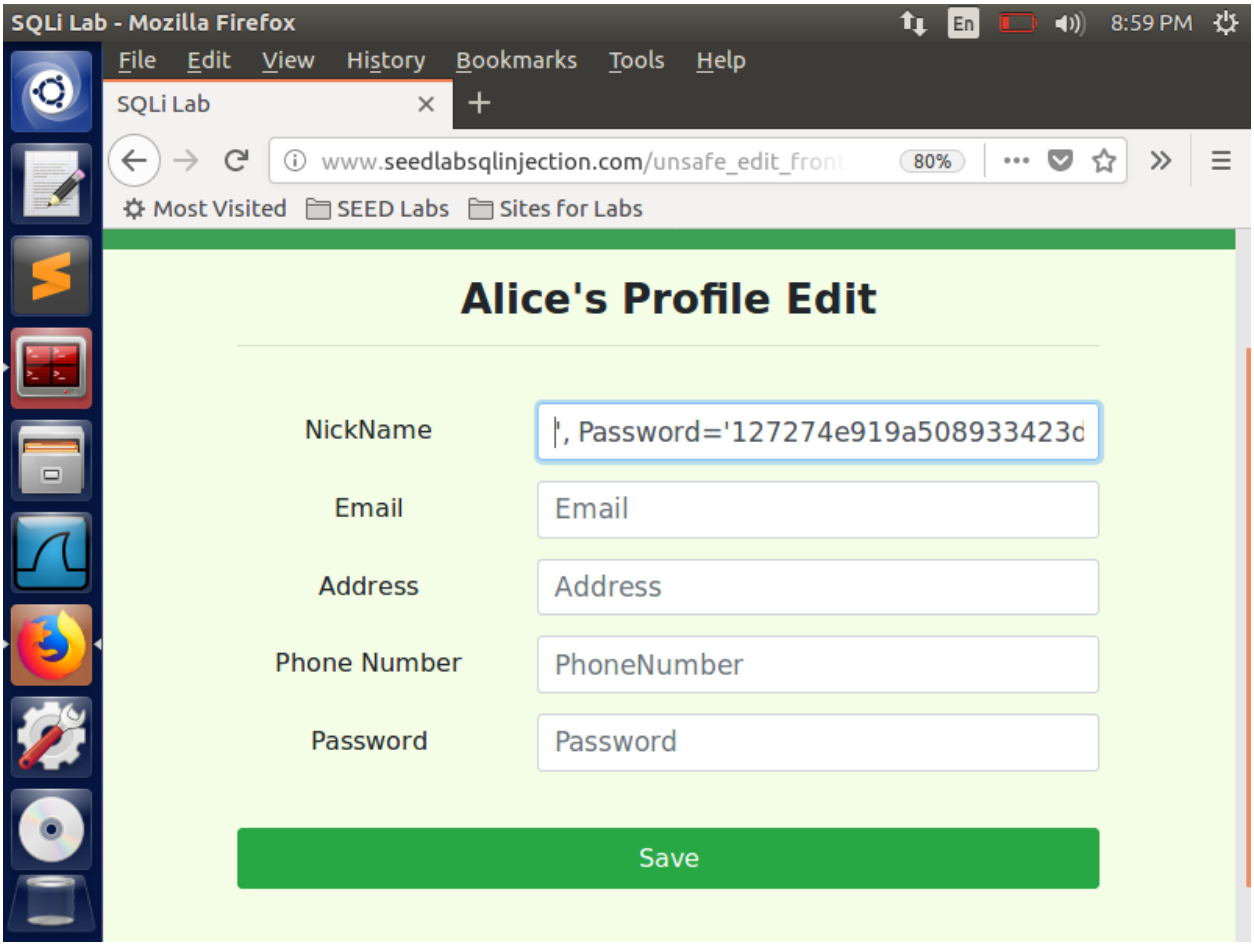
Then I generated the hash value of the new password which I am going to update as the new password of Boby. The value of the new password is "hackedboby" and I got its corresponding hash value using the openssl sha1 command. I am going to use this hash value of the new password the for SQL Injection attack.



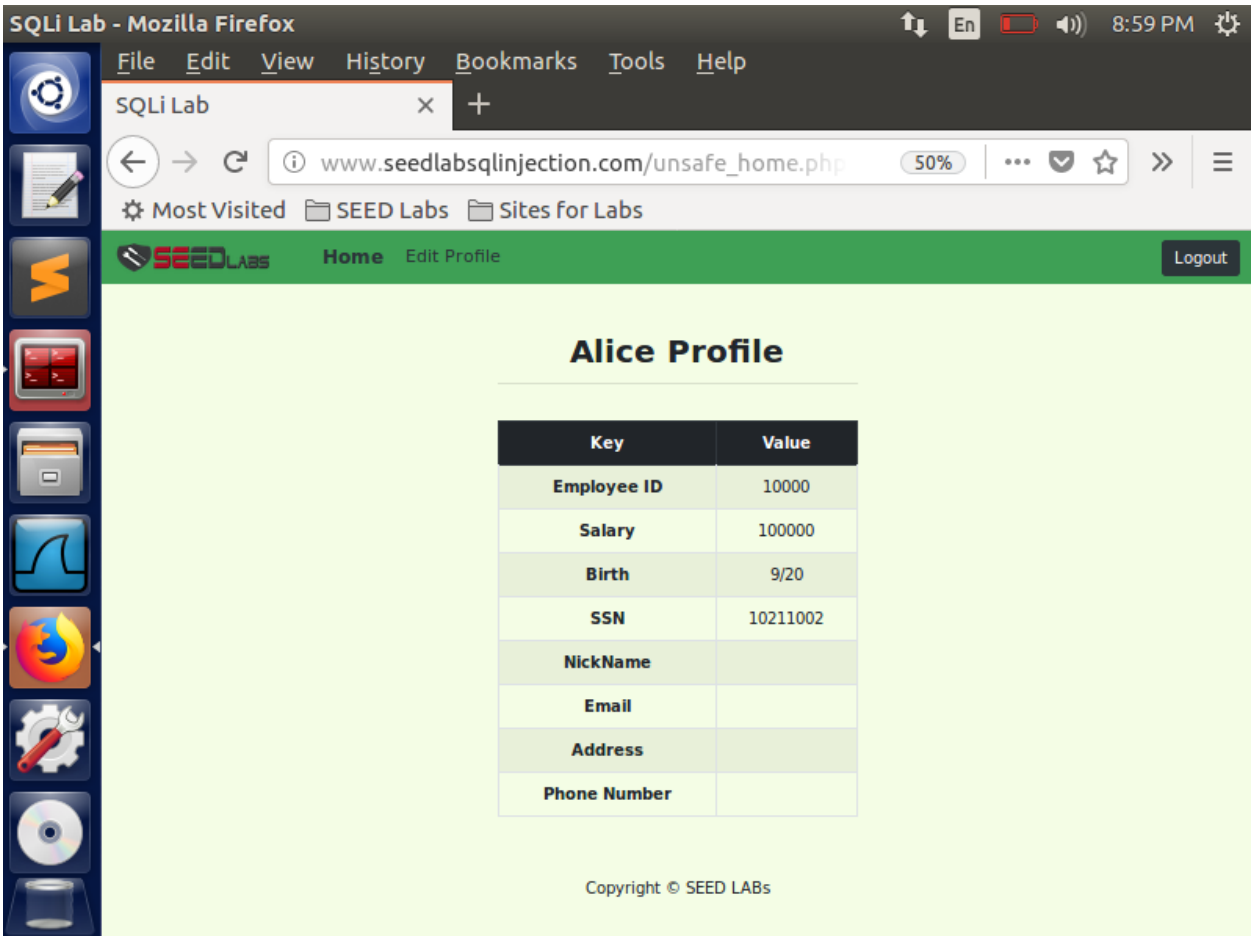
Then I opened the given URL and logged into Alice profile. This is the home page of Alice from where we are going to perform the SQL Injection attack of updating Boby's password.



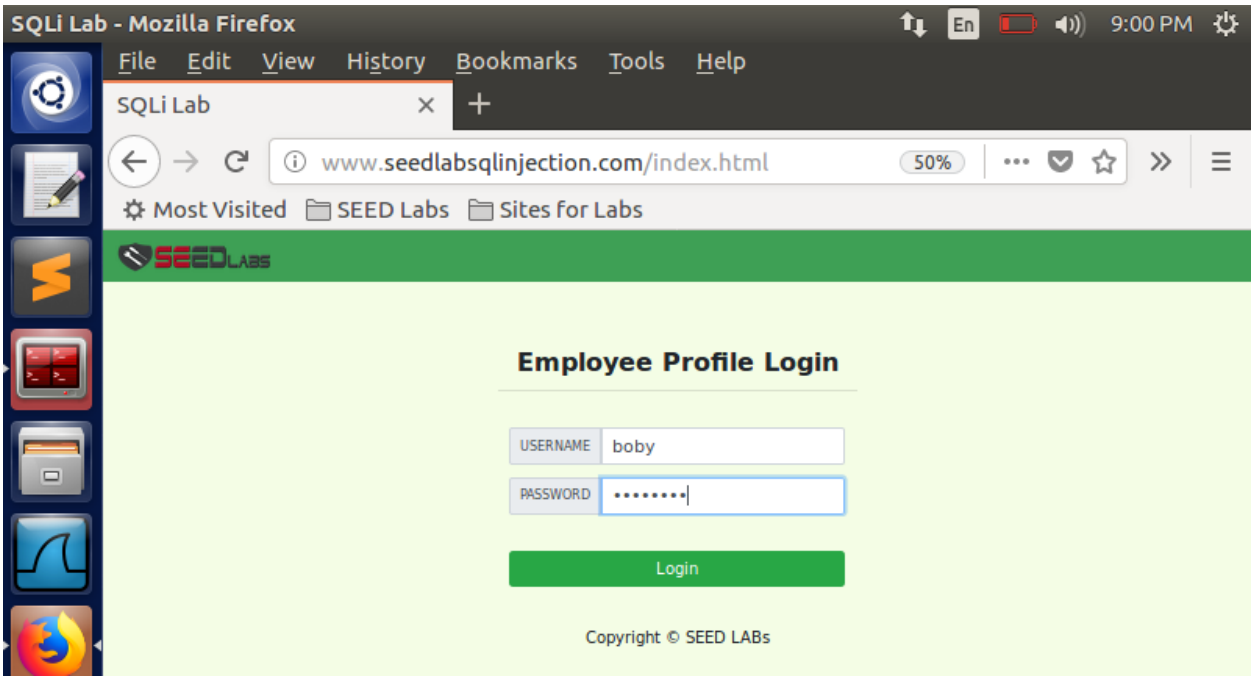
Then I navigated to Edit Page profile Alice where, I will performing the SQL Injection attack. I gave the code ‘, Password = 127274e919a508933423df07a0718e014 where name = ‘Boby’;# in the nickname field of the Edit Profile page. Here I am updating the password of Boby by giving the new password along with a # which is used to comment out the rest of the query used in the backend code of the webpage. Since the Edit Profile page uses the Update Query in all the fields we use the SQL Injection of updating other people’s password in Edit Profile page.



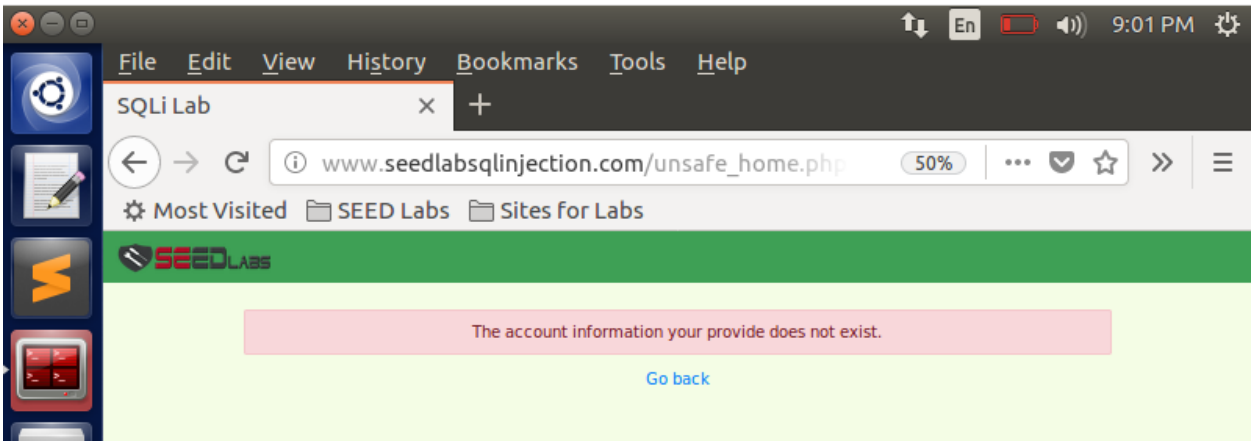
After giving the SQL Injection code in the Edit Profile page of the Alice, I clicked save button and I navigated back to Alice home page.



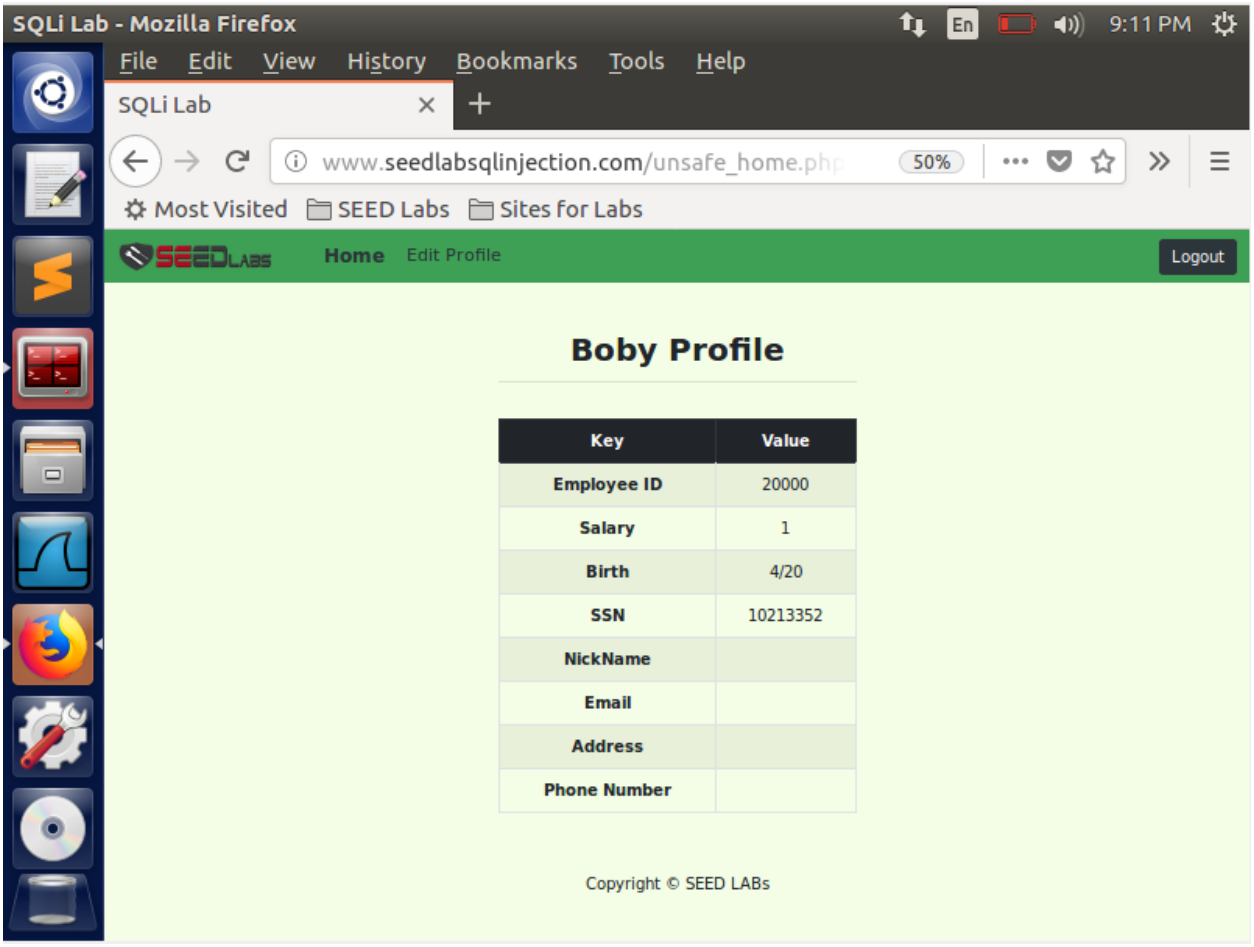
Then I tried to login to Boby's profile using the old password and I was not able to login since Boby's password has been changed due to the SQL Injection attack.



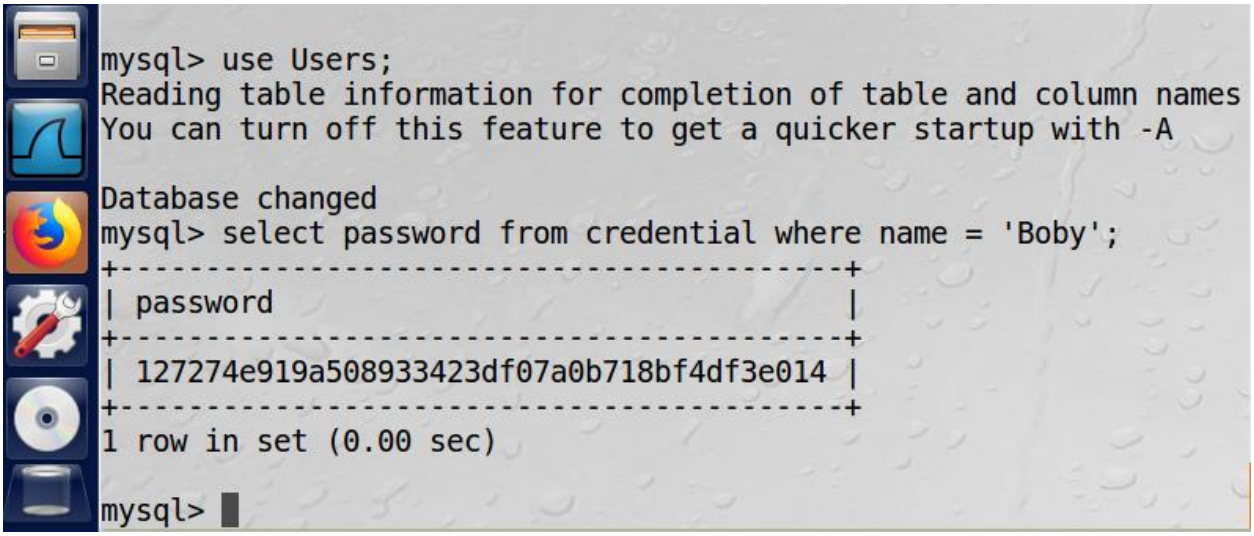
After trying to login into Boby's profile using the old password I was able to get the error saying that the account information did not exist. This is because of the SQL Injection attack.



This is the Home Page of Boby after logging in using the updated password.

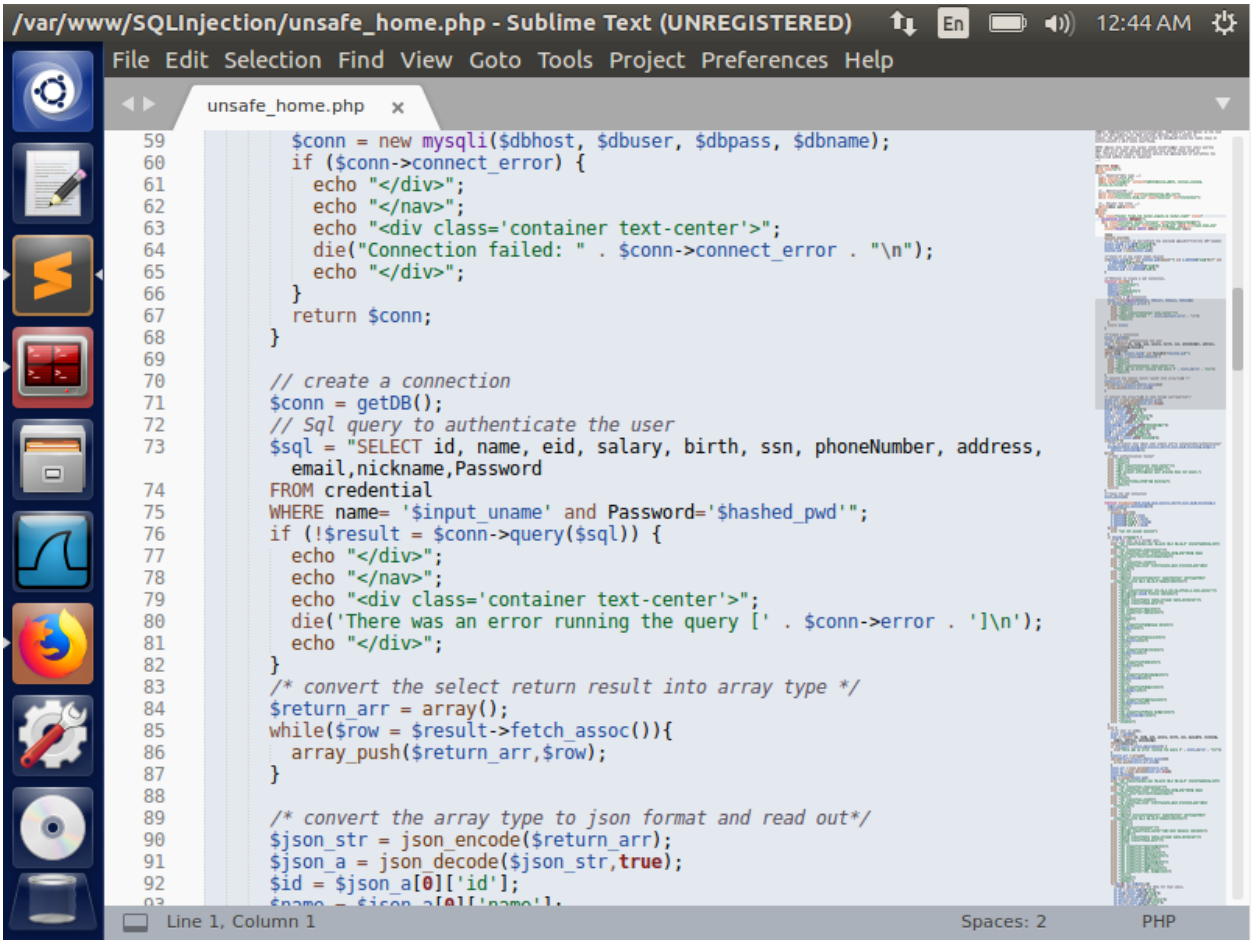


Then I opened the MySQL database to see if the password of Boby has been updated in the database also. I was able to see that the password of Boby has been changed to the new password which Alice changed using the SQL Injection attack.

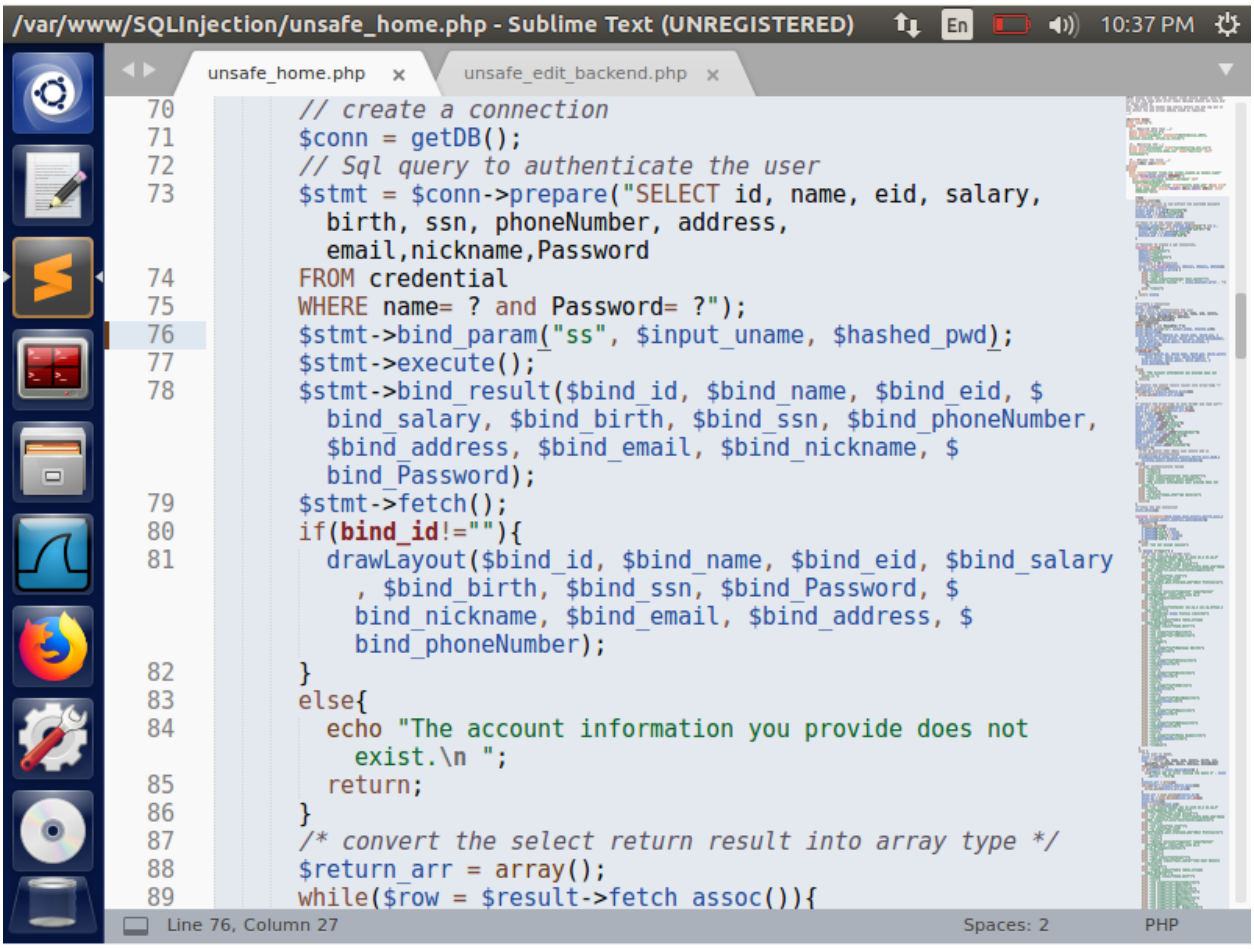


Task 4: Countermeasure — Prepared Statement:

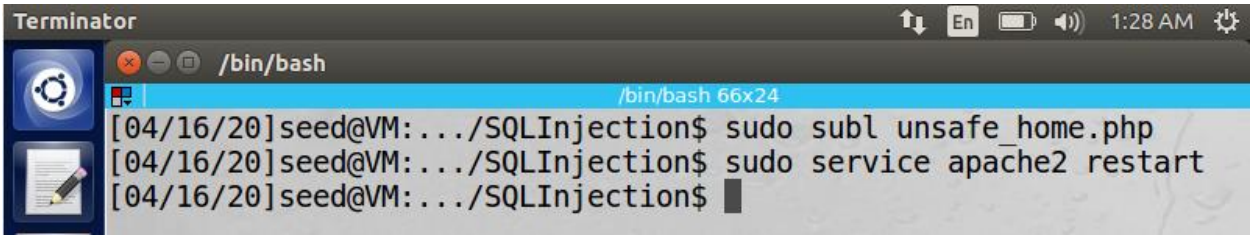
Before enabling the countermeasure, I opened the unsafe_home.php file to check if the countermeasure is enabled. The file is located in the path /var/www/SQLInjection/. I was able to see that the SELECT SQL statement in the php code used for log in has no countermeasure enabled. This is the snapshot of the php code before changing the SQL Statement to prepared statement.



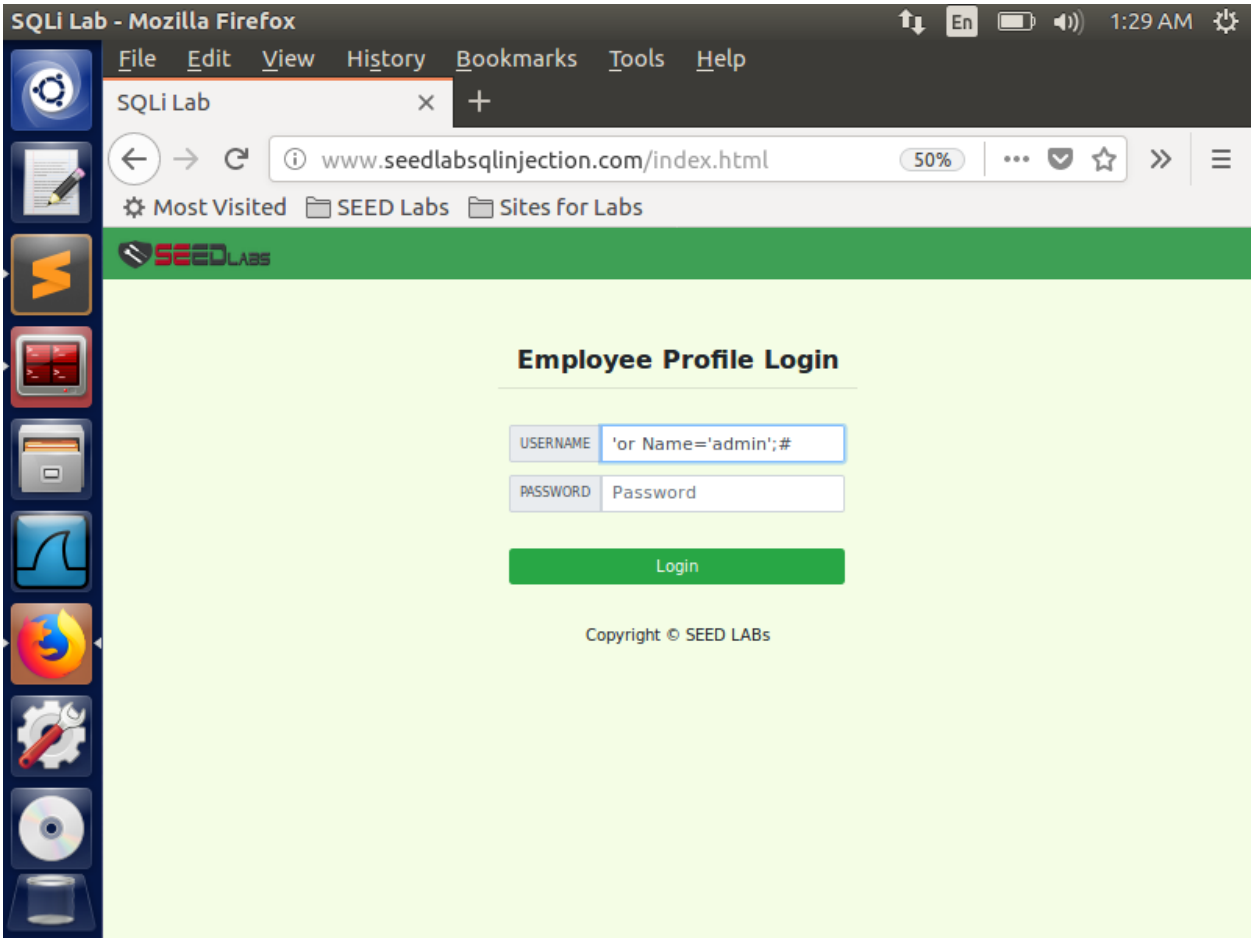
Then I modified the SELECT SQL statement in the unsafe_home.php by using the prepare statement. I replaced the code and the data in the SQL query by placing the ? in the SQL query. After that I send the data to the database using the bind_param() function. After sending the data to the database using the bind_param() function , I then executed the SQL Query and fetched its result and displayed the data into the webpage as per the results fetched. This is the snapshot of the php code after enabling the countermeasure using the prepare statement.



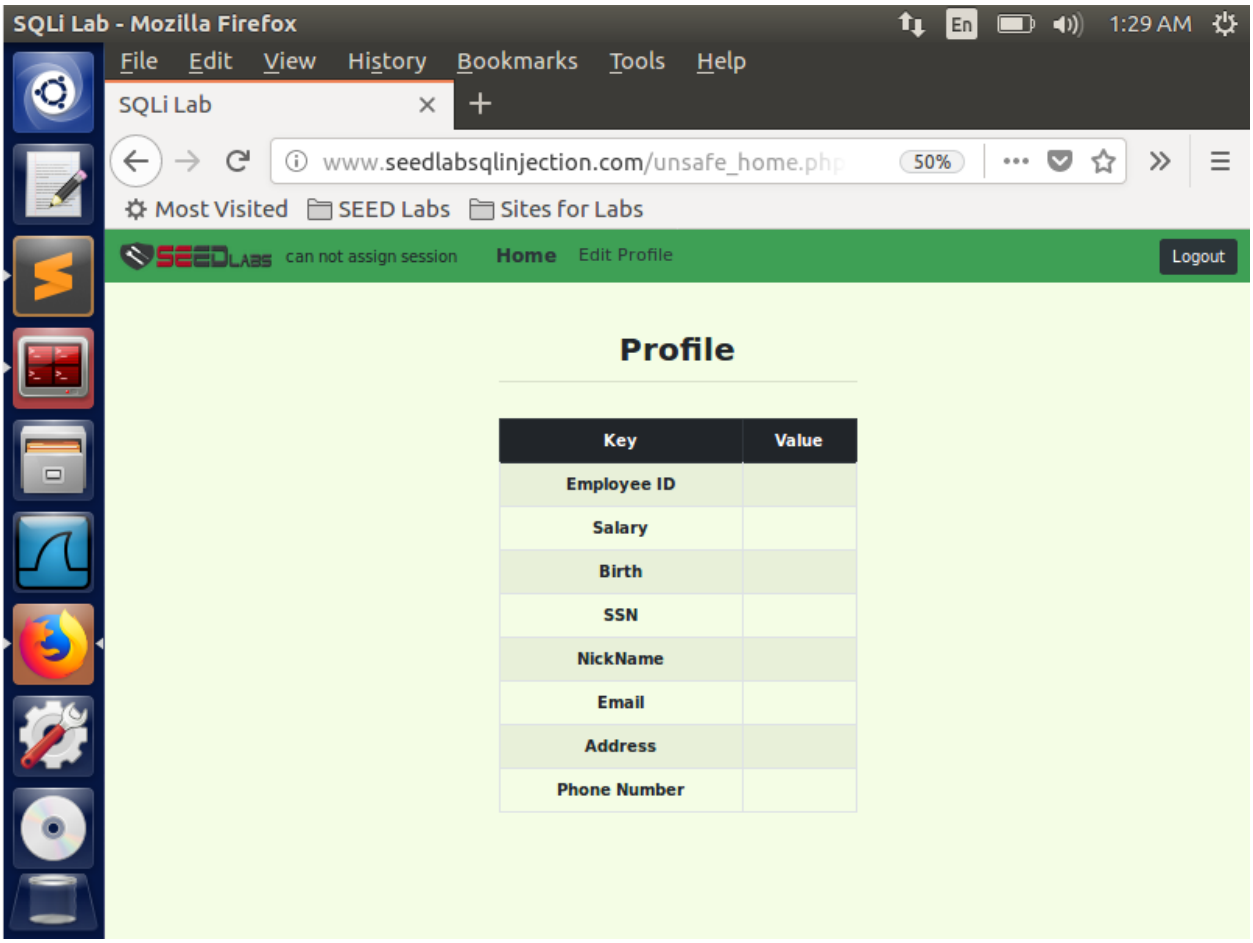
After modifying the php code I just restarted the apache server using the sudo service apache2 restart command which hosts all the webpages for the given website.



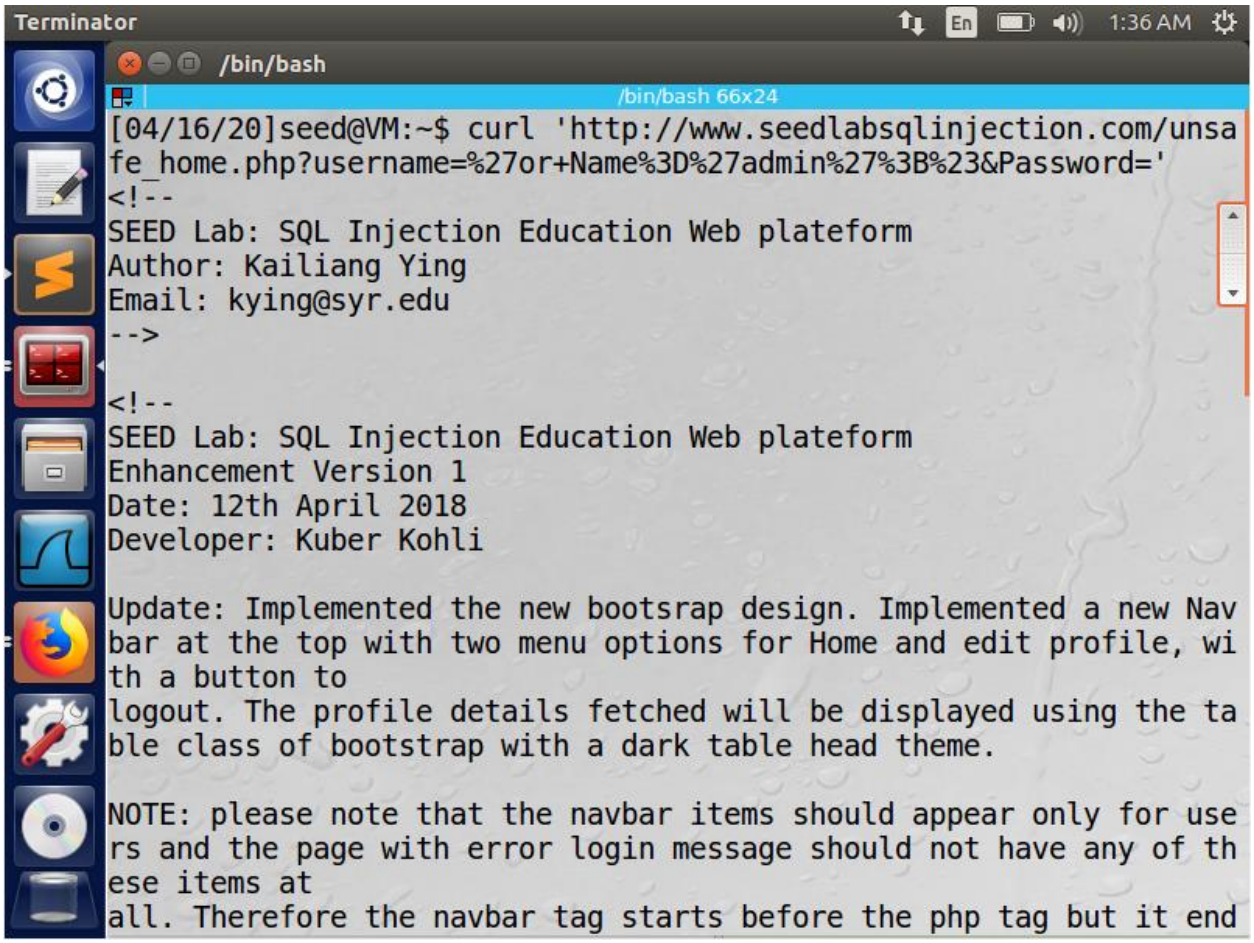
Now I try to login as admin without knowing the password of admin by giving the SQL Injection code in the username field of the login page. I gave the code ‘ or Name=’admin’;# and tried to do the attack which was done in task2.

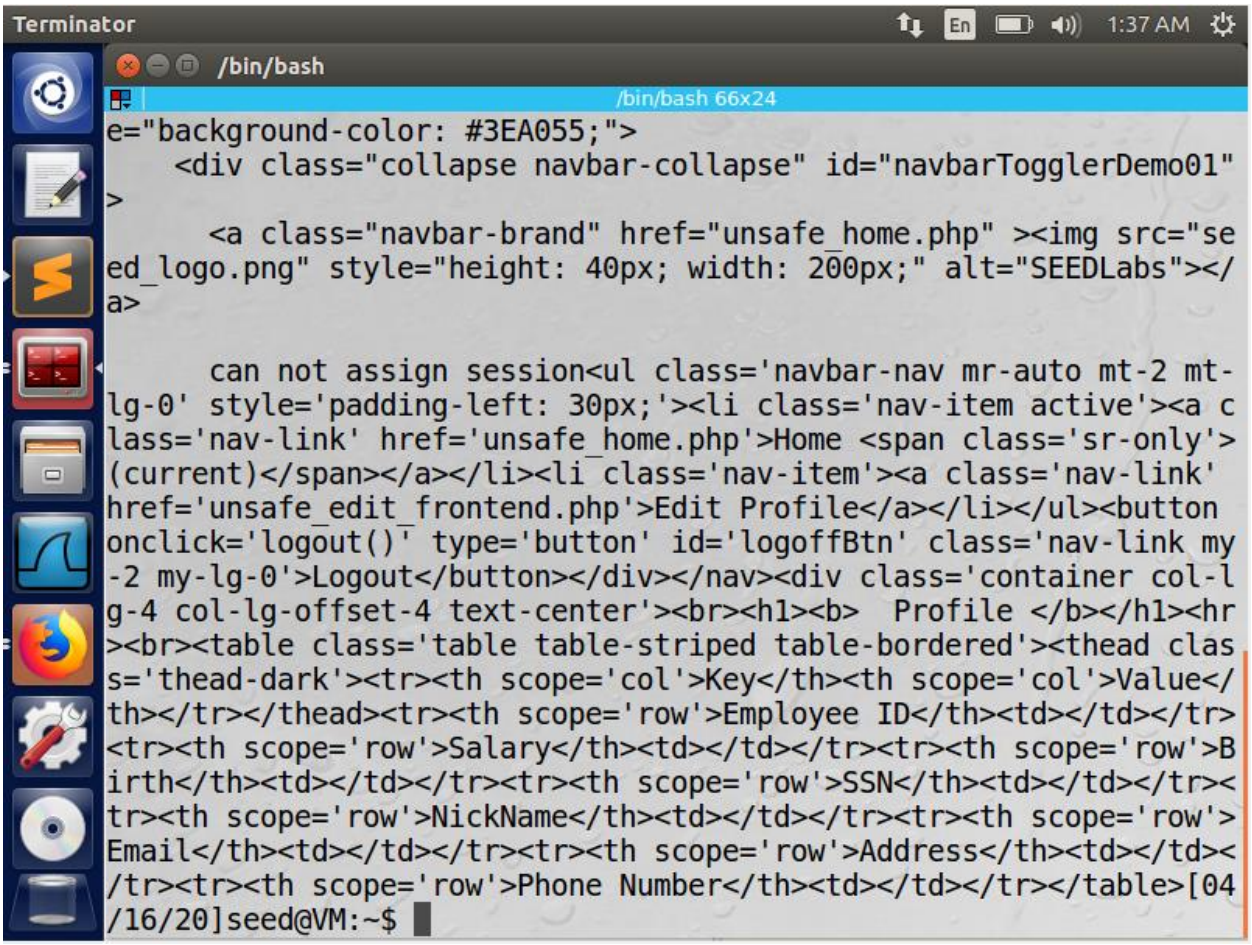


Now when I gave the SQL Injection code in the username field and clicked the login button, I was able to see a page with blank table and a message saying cannot assign session. The attack fails because of the use of the prepare statement in the php code. By using prepare statement I separate the code from the data. The prepare statement executes the SQL query without the data. The data is provided only after the query is complied and executed. By using the prepare statement if there is any code in the data it will only be treated as data to query and not as SQL code. Hence the attack fails.

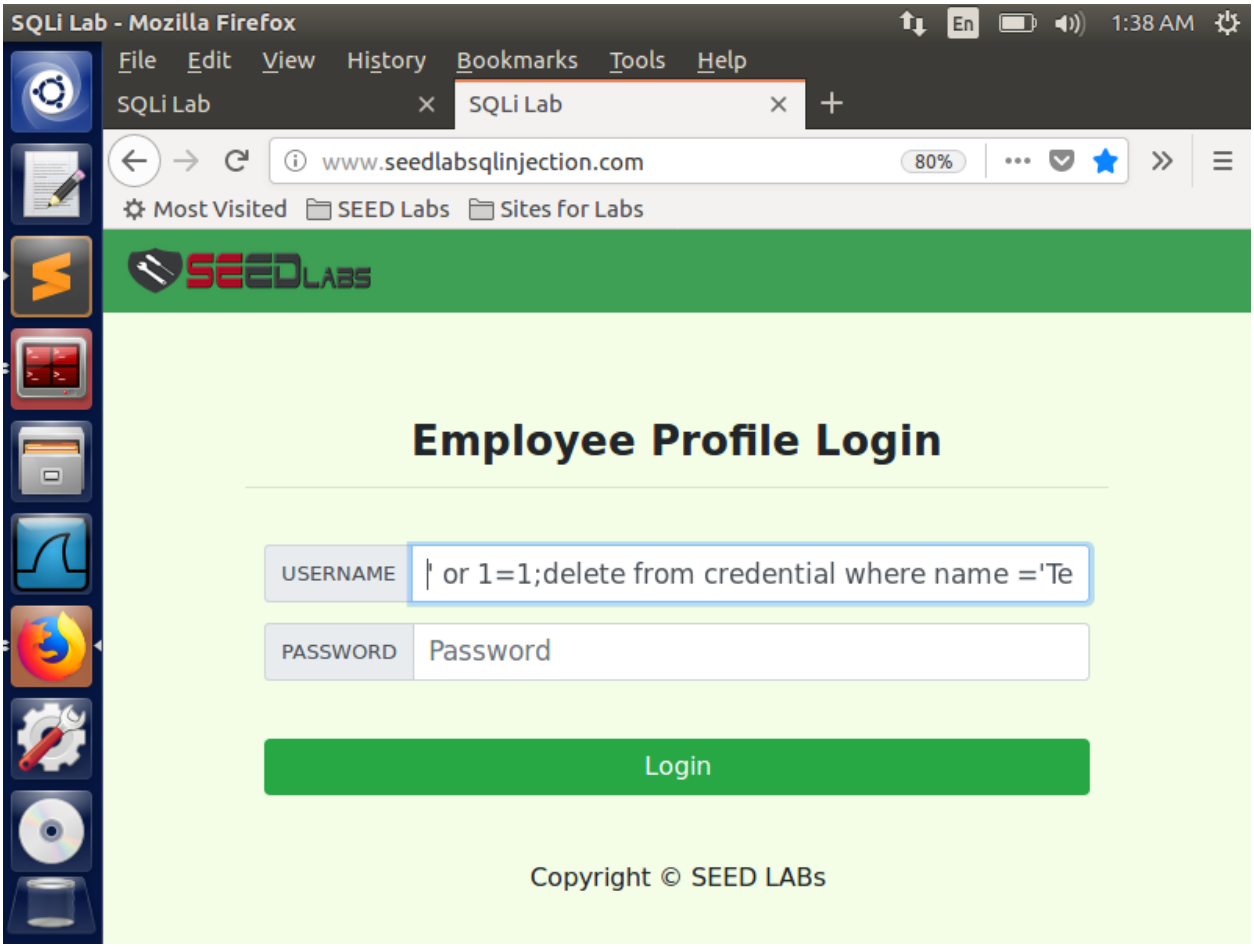


I tried the same attack using the command line by using the curl command followed by the complete HTTP URL. I was able to see that the attack got failed and I was not able to see the profile information of other members. This is because of the prepare statement in the php code.

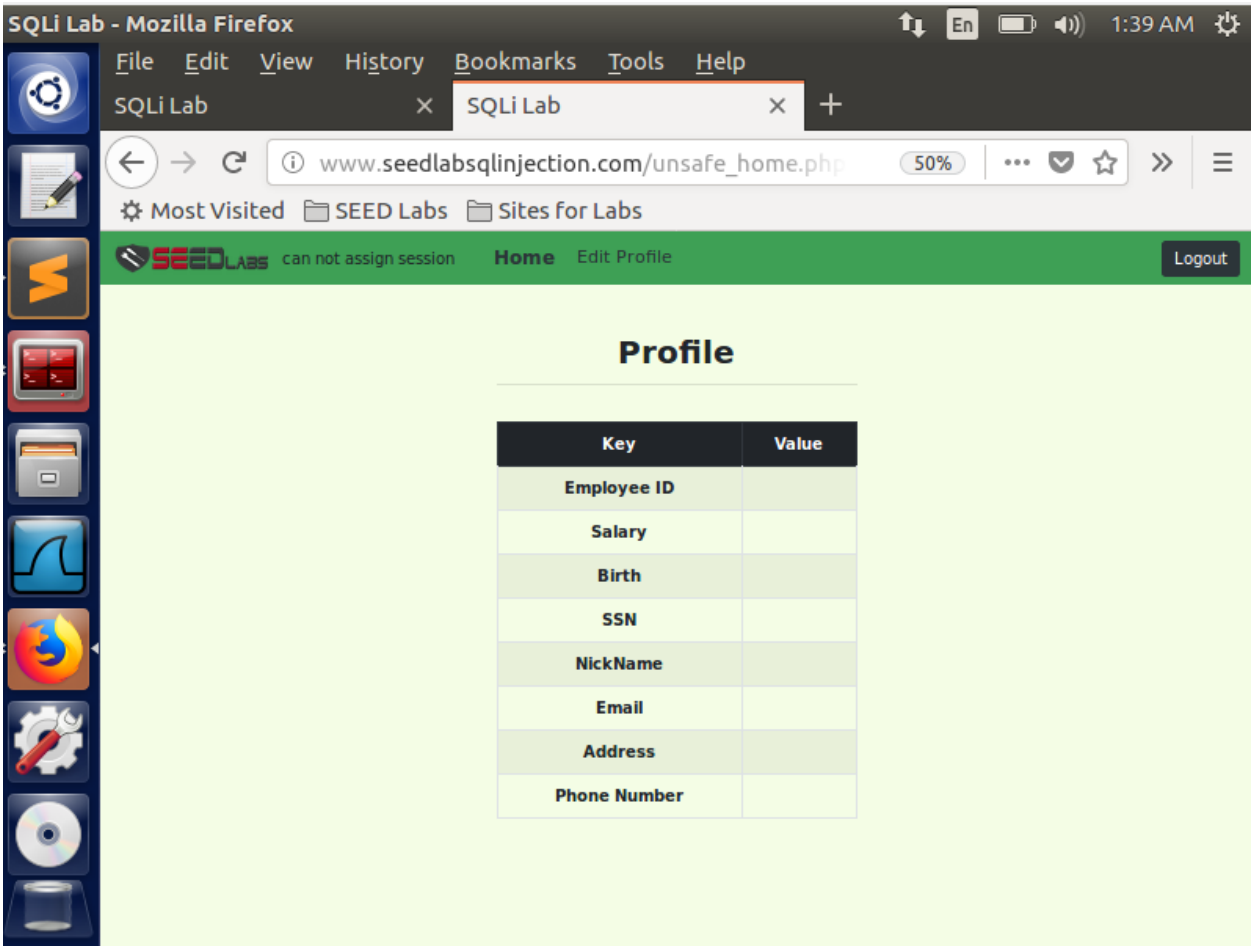



Terminator
En
1:37 AM
/bin/bash
/bin/bash 66x24
e="background-color: #3EA055;">
<div class="collapse navbar-collapse" id="navbarTogglerDemo01"
>
<img src="se
ed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></
a>
can not assign session<ul class='navbar-nav mr-auto mt-2 mt-
lg-0' style='padding-left: 30px;'><li class='nav-item active'><a c
lass='nav-link' href='unsafe_home.php'>Home
(current)
<li class='nav-item'><a class='nav-link'
href='unsafe_edit_frontend.php'>Edit Profile
<button
onclick='logout()' type='button' id='logoffBtn' class='nav-link my
-2 my-lg-0'>Logout</button> </nav><div class='container col-l
g-4 col-lg-offset-4 text-center'>
<h1> Profile </h1><hr
>
<table class='table table-striped table-bordered'><thead clas
s='thead-dark'><tr><th scope='col'>Key</th><th scope='col'>Value</
th></tr></thead><tr><th scope='row'>Employee ID</th><td></td></tr>
<tr><th scope='row'>Salary</th><td></td></tr><tr><th scope='row'>B
irth</th><td></td></tr><tr><th scope='row'>SSN</th><td></td></tr><
tr><th scope='row'>NickName</th><td></td></tr><tr><th scope='row'>
Email</th><td></td></tr><tr><th scope='row'>Address</th><td></td><
/tr><tr><th scope='row'>Phone Number</th><td></td></tr></table>[04
/16/20]seed@VM:~$

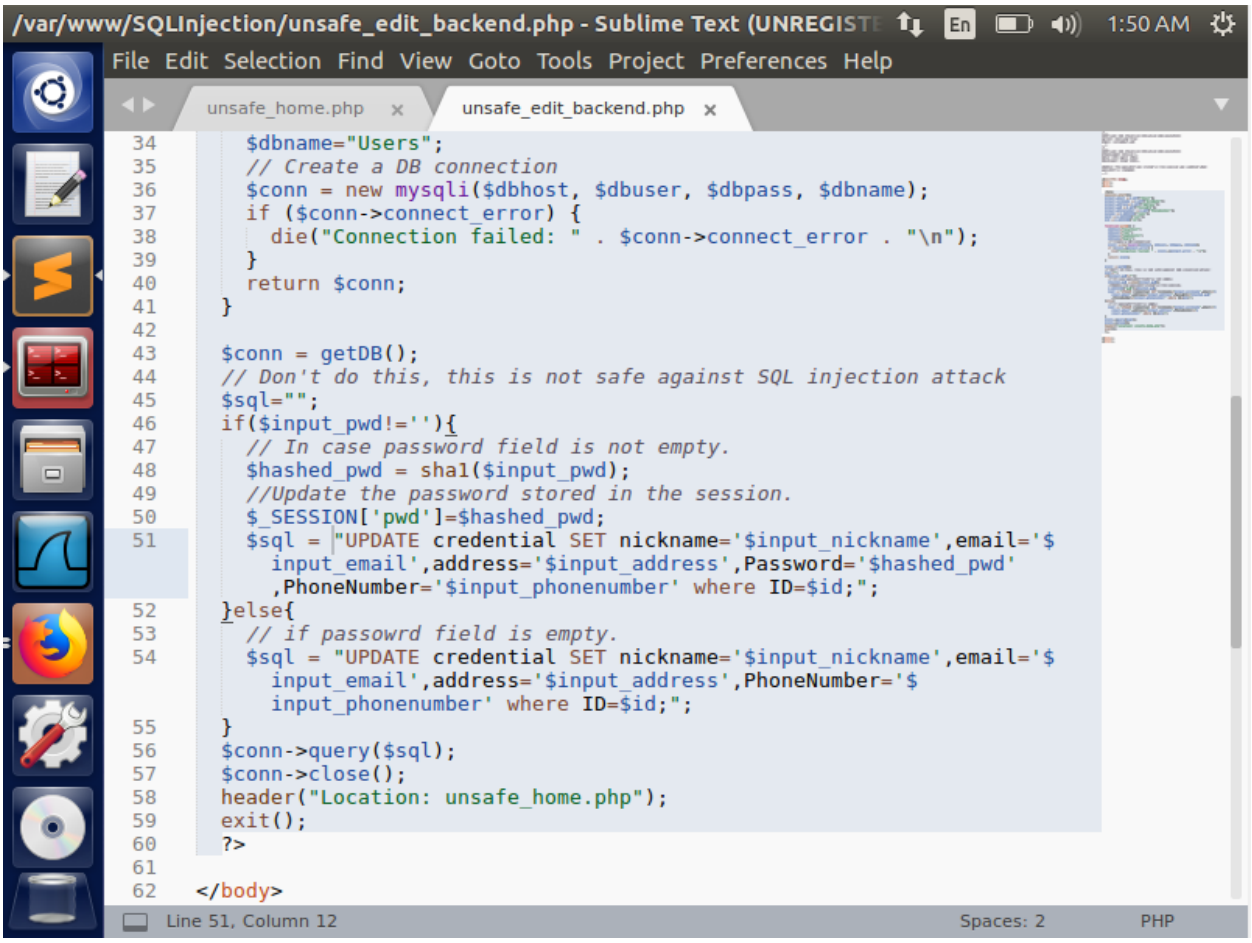
Now I tried to delete a member in the database by appending the SQL statement along with SELECT statement. After entering the SQL Injection code by appending the SQL Delete Statement and I clicked the login button.



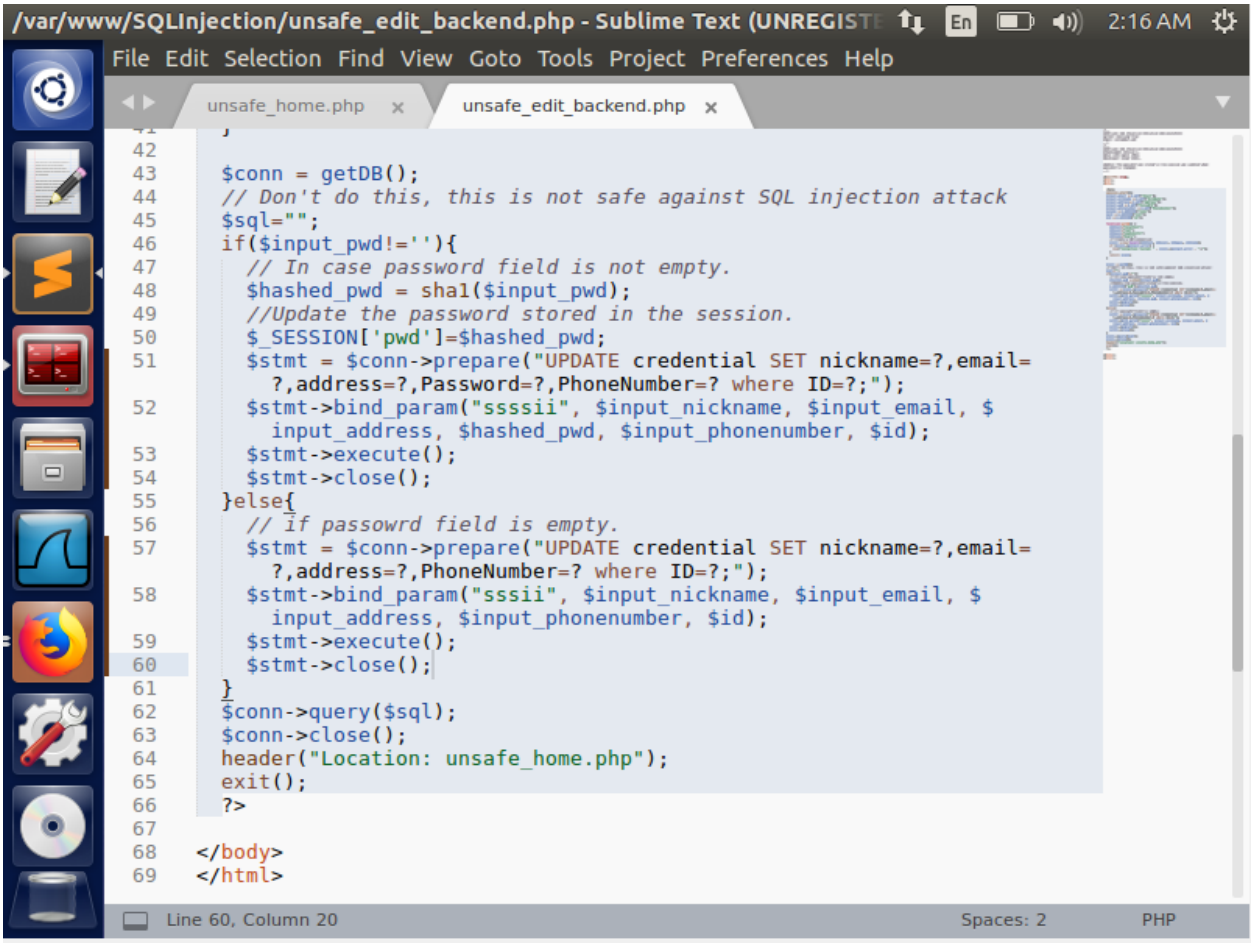
After clicking the login button, I was able to see that a webpage with empty table getting displayed along with the message cannot assign session. This is because of the prepare statement in the unsafe_home.php.



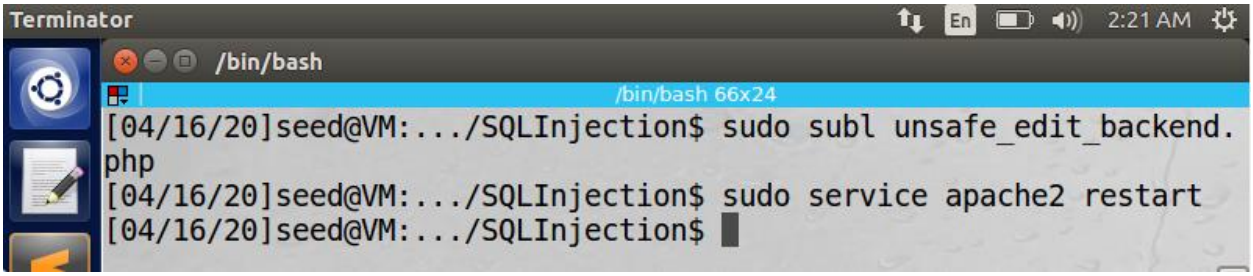
Then I opened the unsafe_edit_backend.php to check if the countermeasure is enabled. The file is located in the path /var/www/SQLInjection/. I was able to see that there was no countermeasure enabled. Now I am checking the unsafe_edit_backed.php file so that I can enable the counter measure for the update query. This is before modifying the the php code.



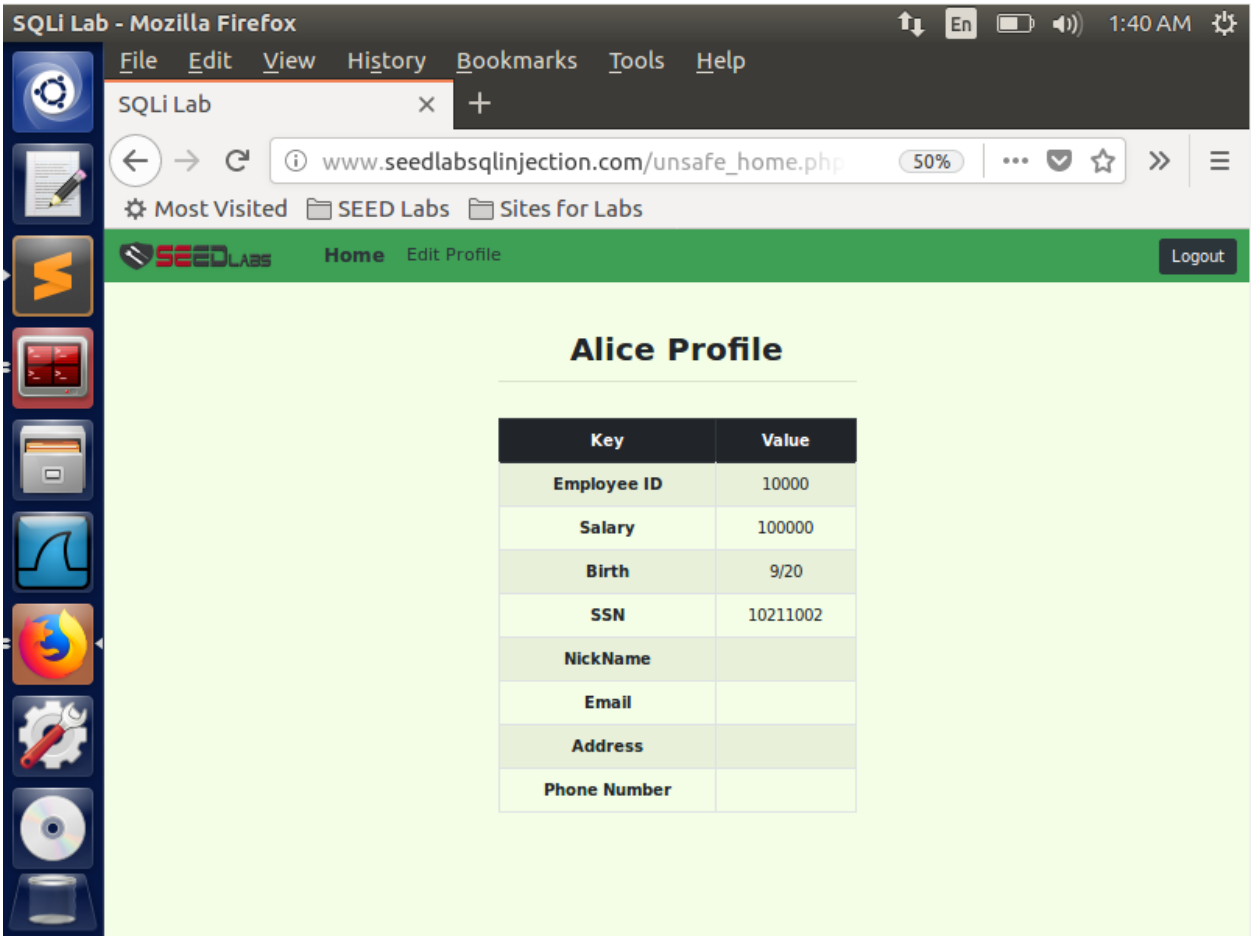
Then I used the prepare statement in the unsafe_edit_backend.php to enable the countermeasure for the UPDATE SQL Statement. I replaced the data values with the ? in the SQL query. After that I send the data to the database using the bind_param() function. After sending the data to the database using the bind_param() function , I then execute the SQL Query and fetch its result and display the data into the webpage as per the results fetched. This is the snapshot of the php code after enabling the countermeasure using the prepare statement.



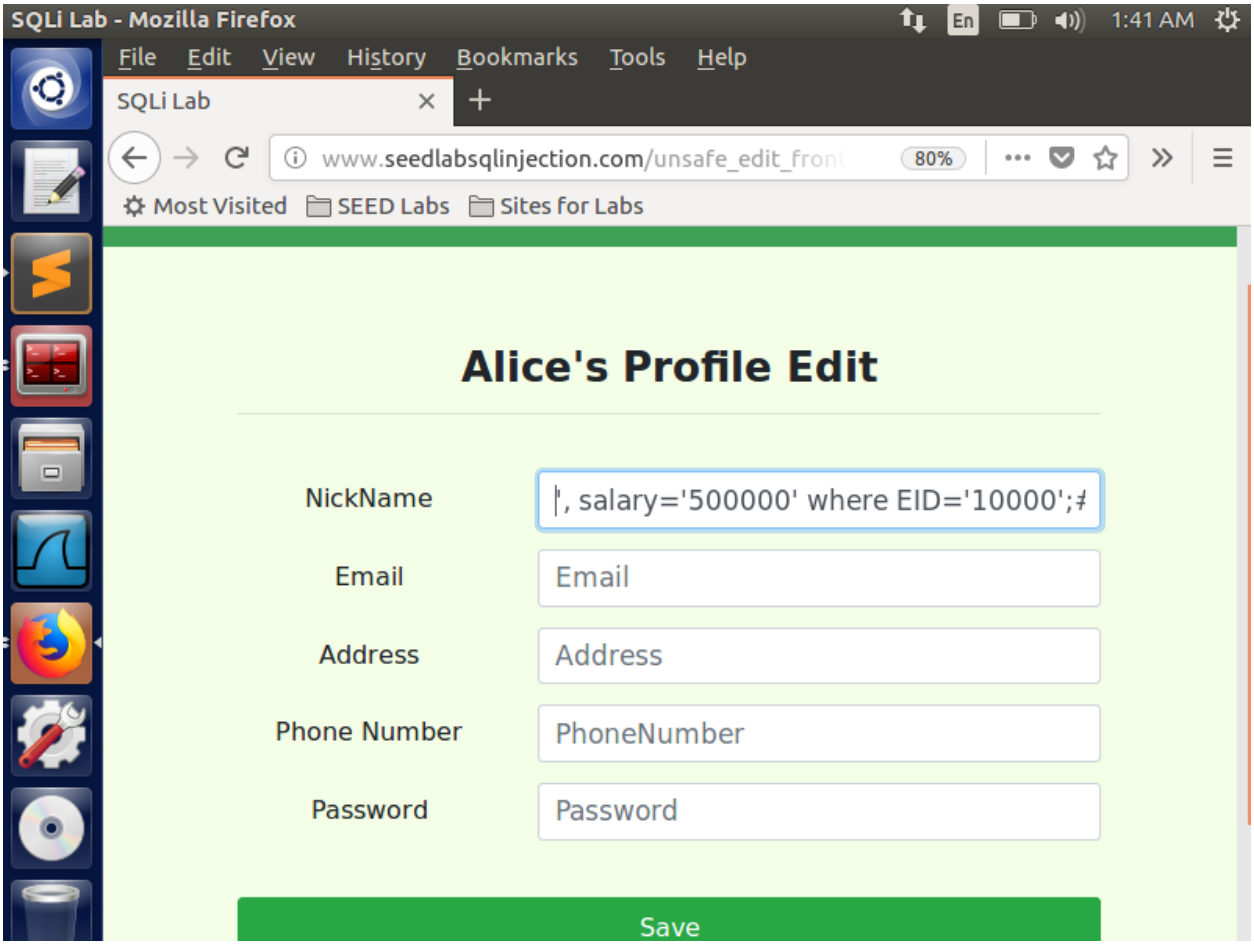
After modifying the unsafe_edit_backend.php by enabling the countermeasure using the prepare statement, I restarted the apache server using the sudo service apache2 restart command, which hosts the webpages for the website.



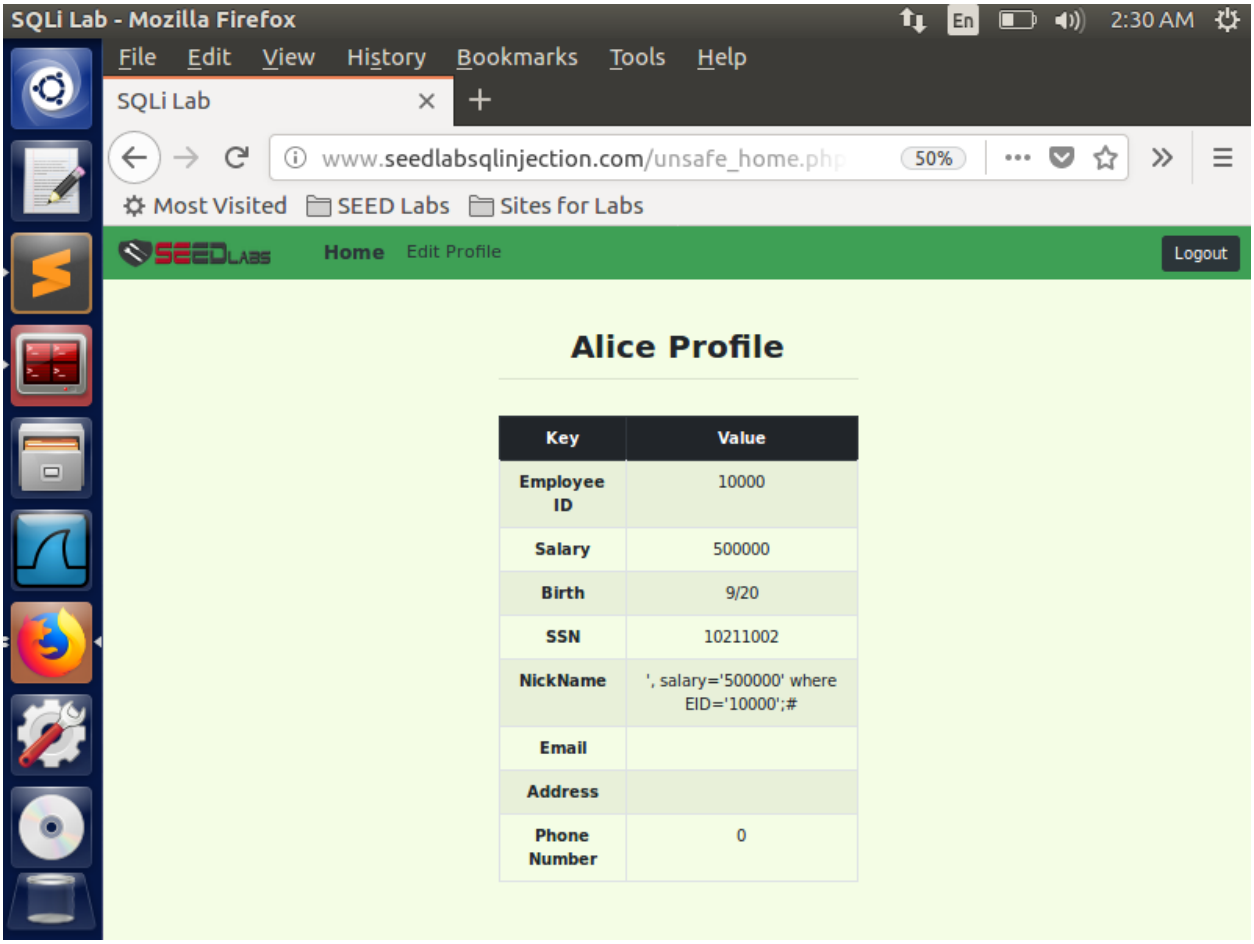
After enabling the countermeasure, I logged into Alice profile using her credentials so that I can modify the salary of Alice by using the SQL Injection attack. Then I navigated to the Edit Profile so that I can perform the SQL Injection attack.



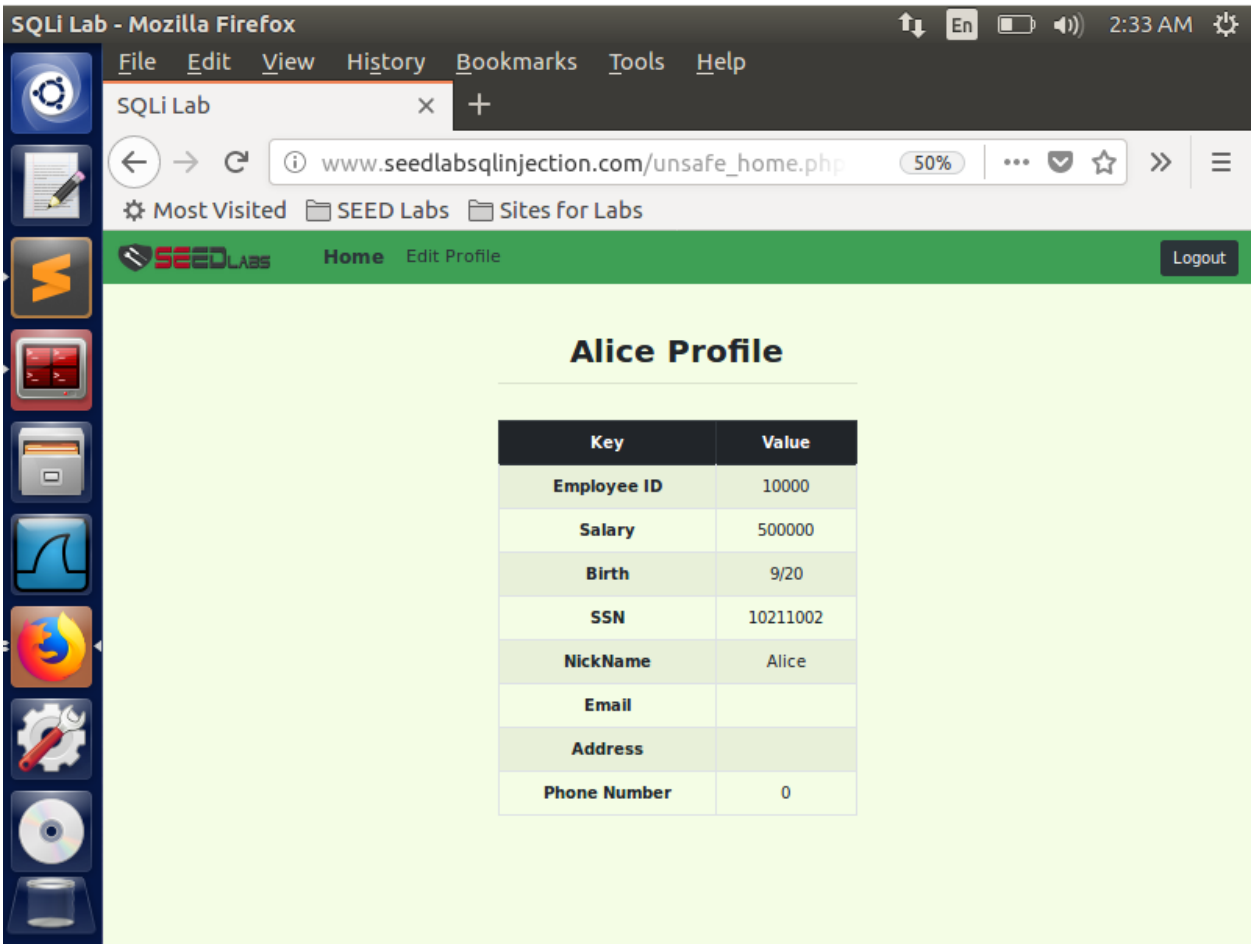
Now I tried to enter the SQL Injection code in the nickname field of the Edit profile and tried to do the same attack as in task3.1. After entering the SQL Injection code, I clicked the save button.



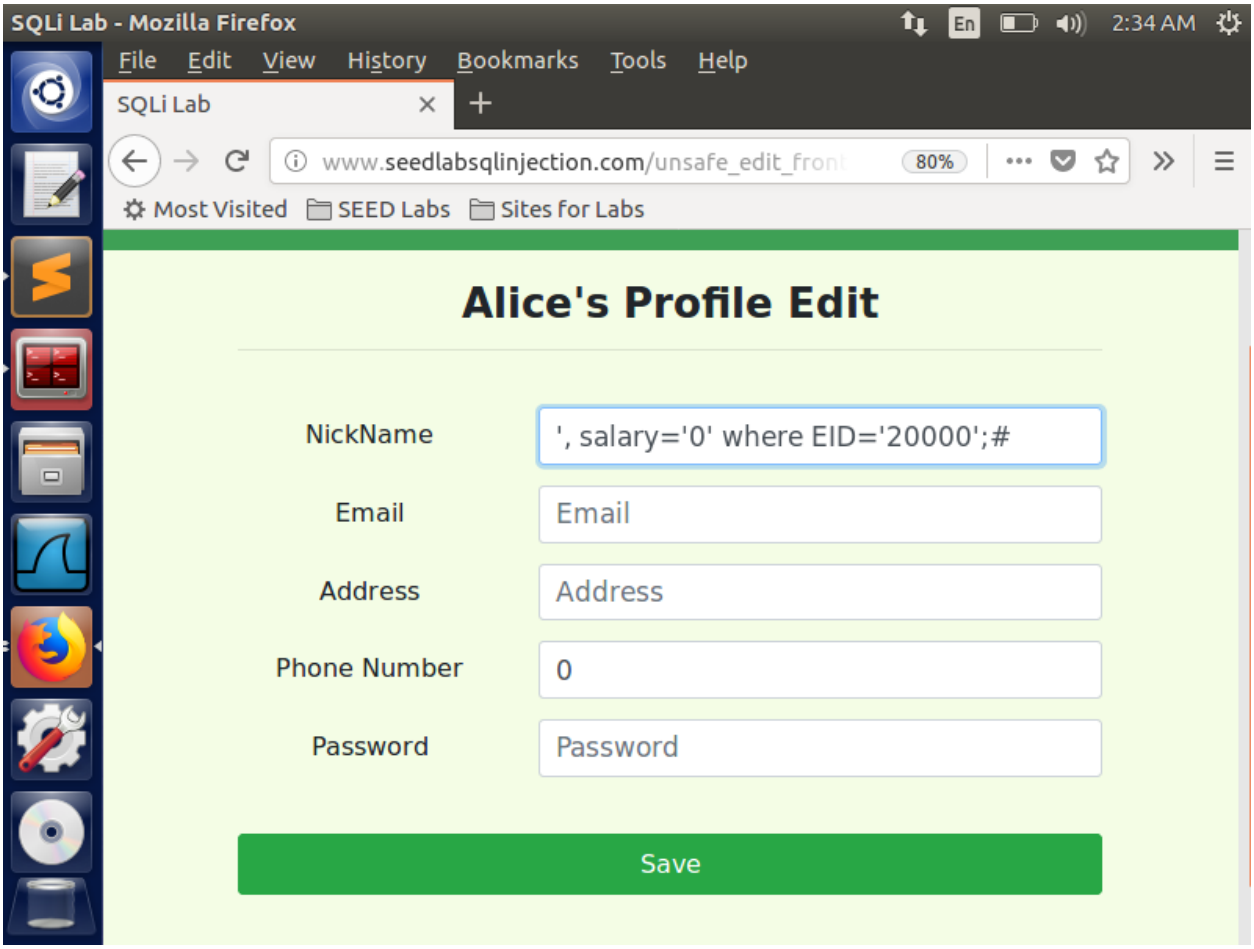
Then I navigated back to the home page of Alice to see if the salary of Alice has been updated. I was able to see that the SQL Injection code being displayed in the nickname field of Alice profile. This is because of the prepare statement in the php code which acts as the countermeasure for the update statement. The attack is failed and this is why we are able to see the SQL Injection code getting displayed in the webpage. The data and the code are separated and this is why the attack gets failed.



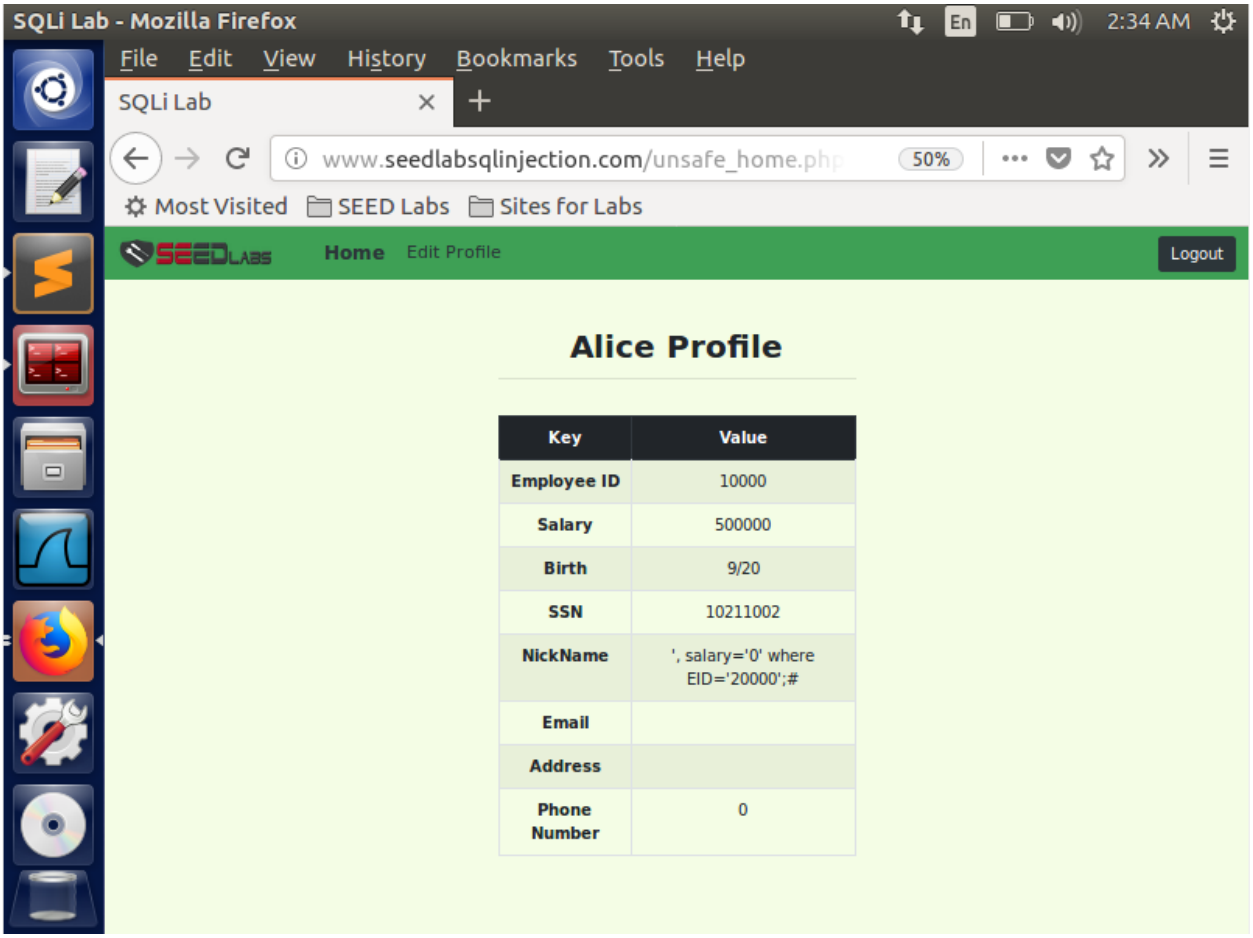
I logged into Alice profile using her credentials so that I can modify the salary of Boby by using the SQL Injection attack. Then I navigated to the Edit Profile so that I can perform the SQL Injection attack.



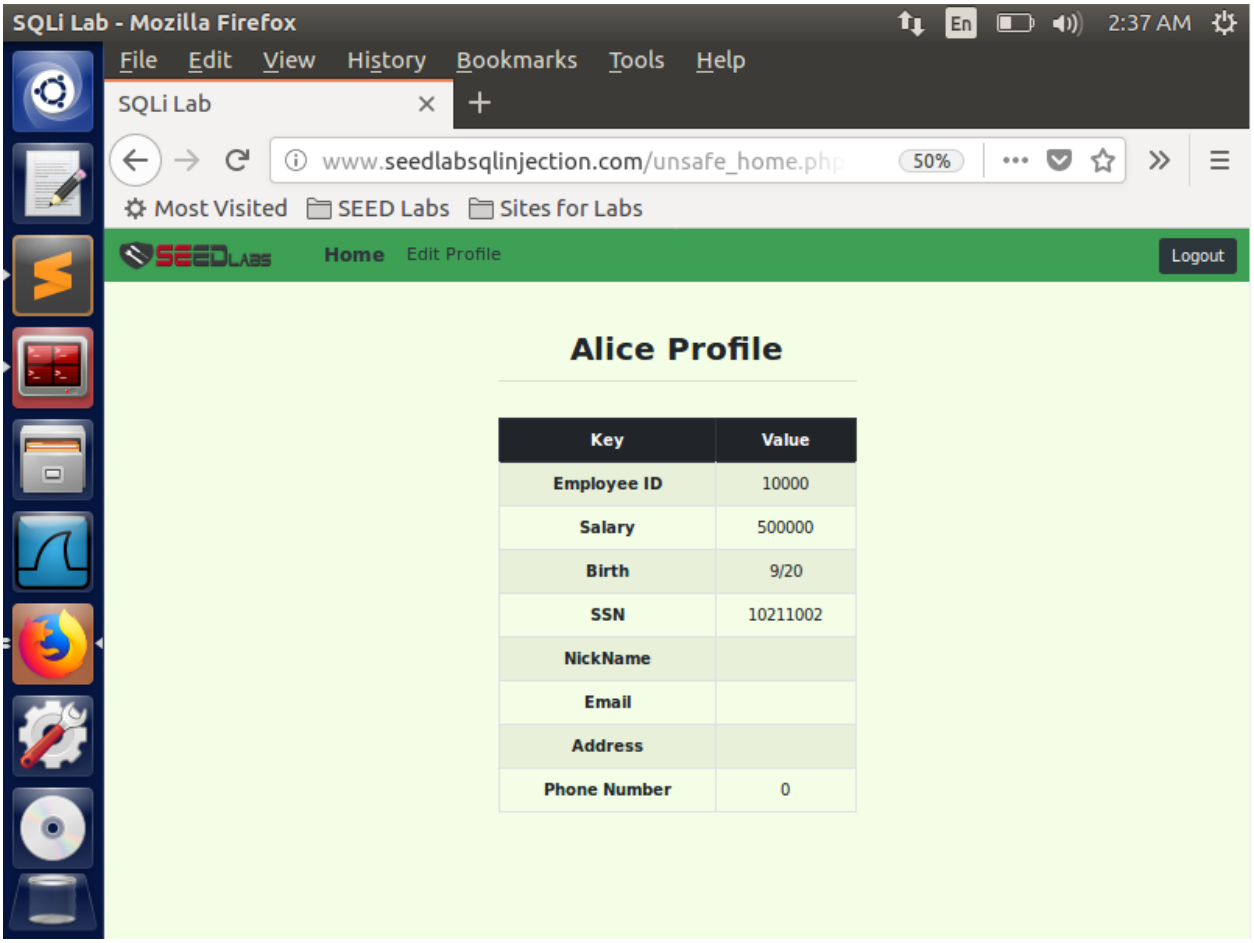
Now I tried to enter the SQL Injection code in the nickname field of the Edit profile and tried to do the same attack as in task3.2 which is to modify Boby's salary. After entering the SQL Injection code, I clicked the save button.



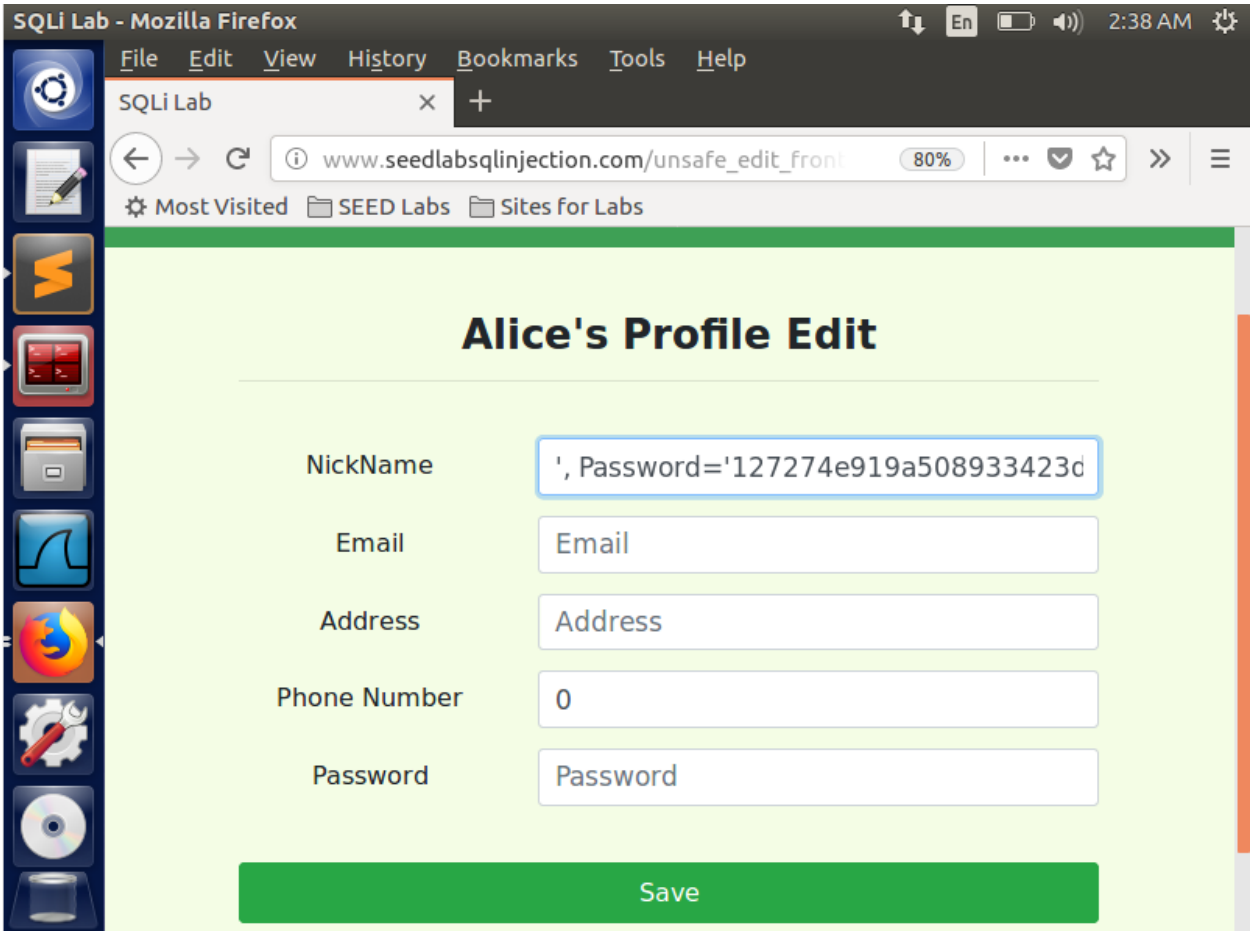
Then I navigated back to the home page of Alice. I was able to see that the SQL Injection code being displayed in the nickname field of Alice profile. This is because of the prepare statement in the php code which acts as the countermeasure for the update statement. The attack is failed and this is why we are able to see the SQL Injection code getting displayed in the webpage. The data and the code are separated and this is why the attack gets failed.



I logged into Alice profile using her credentials so that I can modify the password of Boby by using the SQL Injection attack. Then I navigated to the Edit Profile so that I can perform the SQL Injection attack.



Now I tried to enter the SQL Injection code in the nickname field of the Edit profile and tried to do the same attack as in task3.2 which is to modify Boby's password. After entering the SQL Injection code, I clicked the save button.



Then I navigated back to the home page of Alice. I was able to see that the SQL Injection code being displayed in the nickname field of Alice profile. This is because of the prepare statement in the php code which acts as the countermeasure for the update statement. The attack is failed and this is why we are able to see the SQL Injection code getting displayed in the webpage. The data and the code are separated and this is why the attack gets failed.