SQL Injection Attack Lab

Table of Contents:

1 Abstract:

A SQL injection attack involves inserting or "injecting" a SQL query into the program via the client's input data. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), perform database administration operations (such as shutting down the DBMS), recover the content of a given file present on the DBMS file system, and in some cases issue commands to the operating system. SQL injection attacks are a sort of injection attack in which SQL instructions are injected into data-plane input to influence the execution of predetermined SQL commands.

2 Introduction:

SQL injection attacks occur when hackers use these flaws to breach the target system's security. They are one of the most ancient and widely used hacking methods. If they are effective, they may be quite devastating. If a prominent website holds credit card information for its customers in its databases and is the target of a successful assault, the attackers might obtain access to the financial information of millions of people. Attackers may also obtain access to users' personal information. If these attacks can go unnoticed, they have a far better chance of succeeding. Because of the potential for massive damage, it is critical that these flaws are identified and addressed.

If queries that obtain and change information in the database are not utilized securely, they may render the applications susceptible. Retrieval queries, for example, might be used to obtain information from the database that users are not supposed to view. Commands can also be used to get database administrator rights. This may allow them to obtain access to data that only administrators have.

When the **WHERE** clause is utilized in **SELECT** and **UPDATE** queries, the majority of injection vulnerabilities arise. If programmers opt to dynamically build queries by concatenating user input with queries, injection issues are probable. This is a concern since user input cannot be relied on. Using prepared statements is the simplest technique to reduce the risk of SQL assaults occurring (parametrized queries). This prevents user input from being used to execute illegal database operations. When developing a web application, make important to test for injection vulnerabilities to protect the security of your data.

This lab report covers the following topics:

1 – Get familiar with SQL statements.

2 – SQL Injection attack on SELECT statement:

• SQL Injection attack from webpage • SQL Injection attack from command line • Append a new SQL statement

3 – SQL Injection Attack on UPDATE statements:

• Modify your own salary • Modify other people salary • Modify other people password

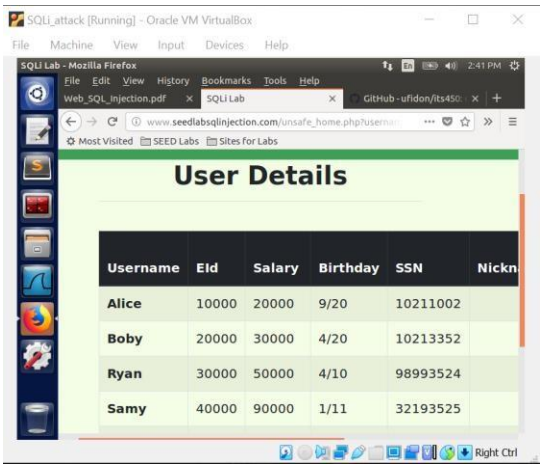
4 – Countermeasure – Prepared statement.

3 Objective:

- The objective is to uncover ways to exploit SQL injection vulnerabilities, illustrate the harm that the attack may do, and grasp the tactics that can assist protect against such assaults.

4 Tasks:

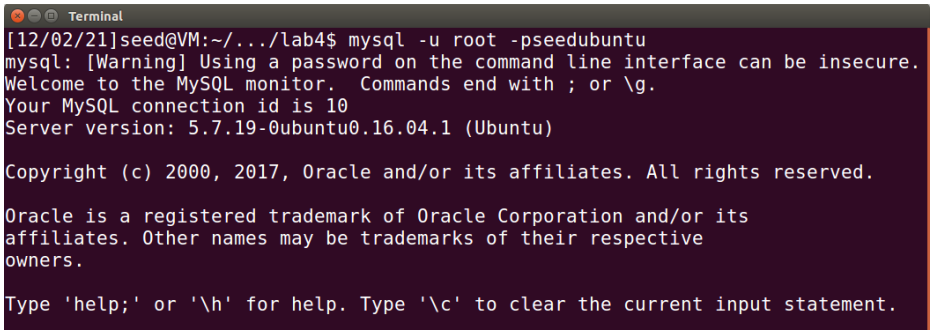
The lab's environment are as follows:



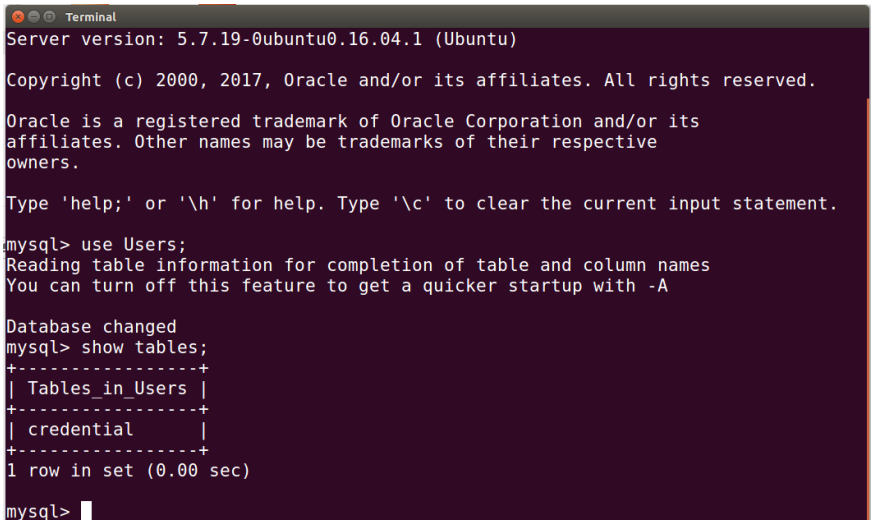
4.1 Task 1: Get Familiar with SQL Statements.

This first task is all about being acquainted with SQL commands. The SEED Labs virtual machine (I'm using Ubuntu 16.04) already has MySQL installed and a database set up for this experiment. The database is named Users, and it has a credential table.

First, we log in to the MySQL console and change the database in use to Users:



When we list all of the tables, we notice that there is just one table entitled credential:



SQL commands are not case sensitive; however, it is usual practice to type commands in all capital letters. SQL commands are readily distinguished from non-commands, which are lowercase.

Printing all of the employee 'Alice's information:

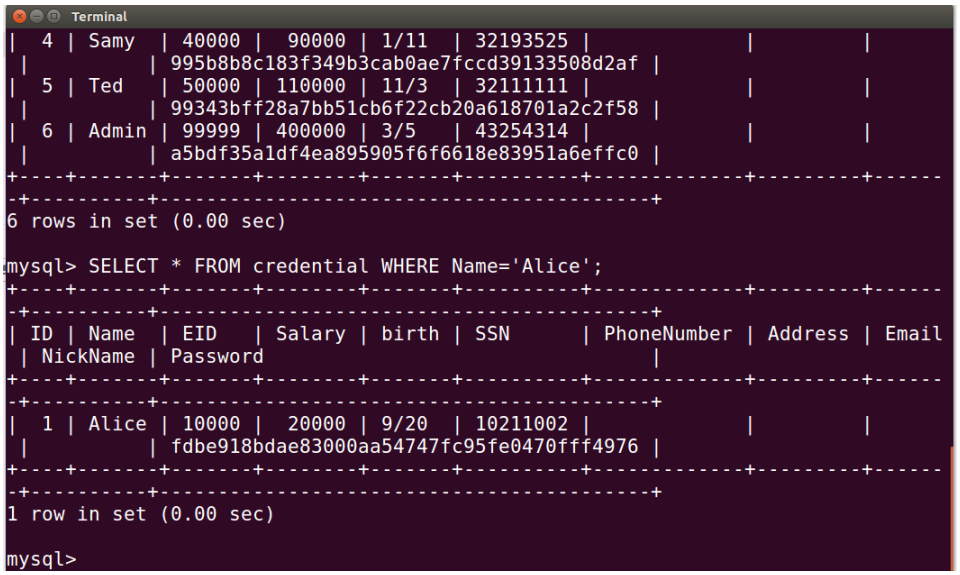
There's 2 ways to get information: **first one**

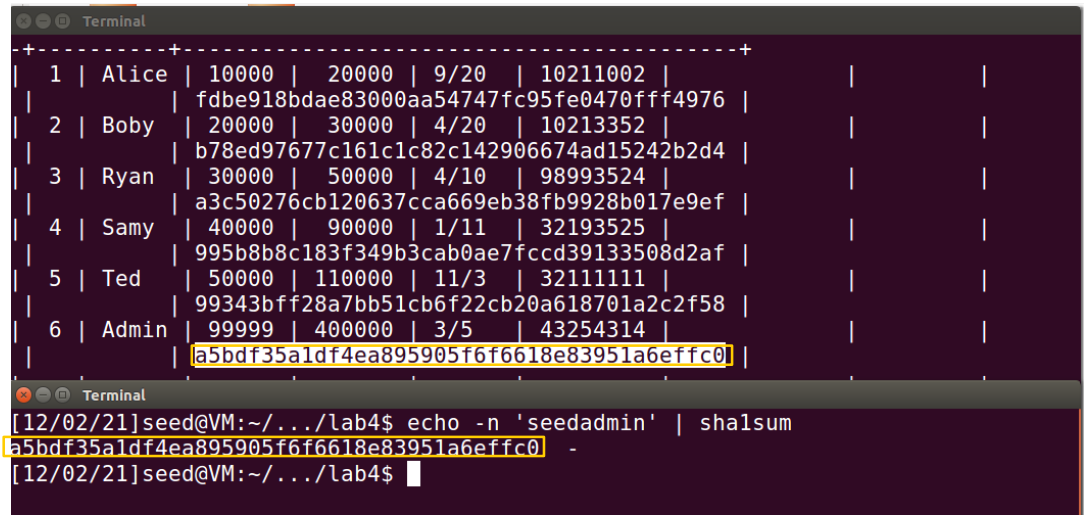
```
Terminal
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> describe credential;
+-------------+-----------------+------+-----+---------+----------------+
| Field       | Type            | Null | Key | Default | Extra          |
+-------------+-----------------+------+-----+---------+----------------+
| ID          | int(6) unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)     | NO   |     | NULL    |                |
| EID         | varchar(20)     | YES  |     | NULL    |                |
| Salary      | int(9)          | YES  |     | NULL    |                |
| birth       | varchar(20)     | YES  |     | NULL    |                |
| SSN         | varchar(20)     | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)     | YES  |     | NULL    |                |
| Address     | varchar(300)    | YES  |     | NULL    |                |
| Email       | varchar(300)    | YES  |     | NULL    |                |
| NickName    | varchar(300)    | YES  |     | NULL    |                |
| Password    | varchar(300)    | YES  |     | NULL    |                |
+-------------+-----------------+------+-----+---------+----------------+
11 rows in set (0.02 sec)

mysql>
```

```
Terminal
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
 |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
 |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  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>
```

The single table in the Users database is the credential table. The operation requires the usage of a SQL statement to print all of the employee Alice's profile information. First, I'll use the 'SELECT * FROM credential;' statement to output all of the data in the table:

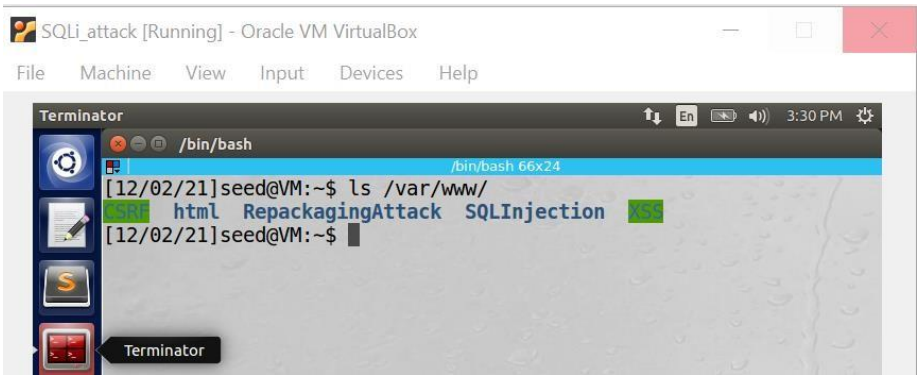
Or by write: SELECT * FROM credential WHERE Name = 'Alice';

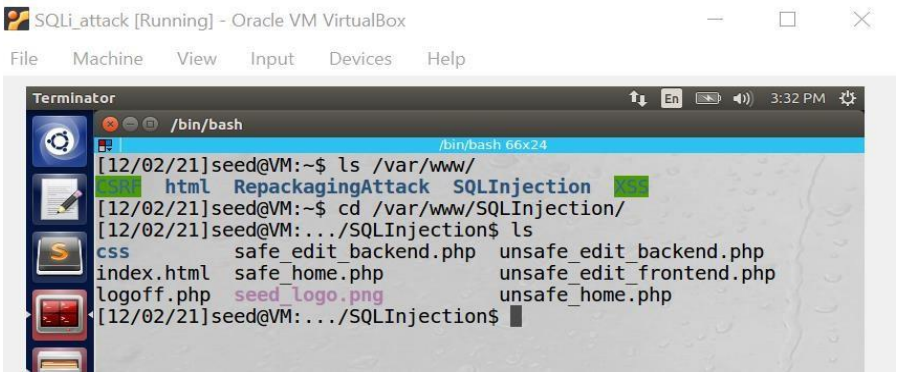


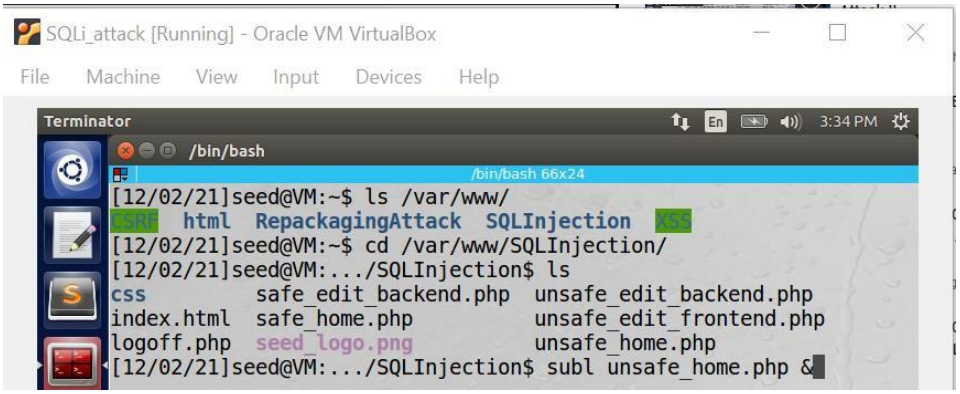


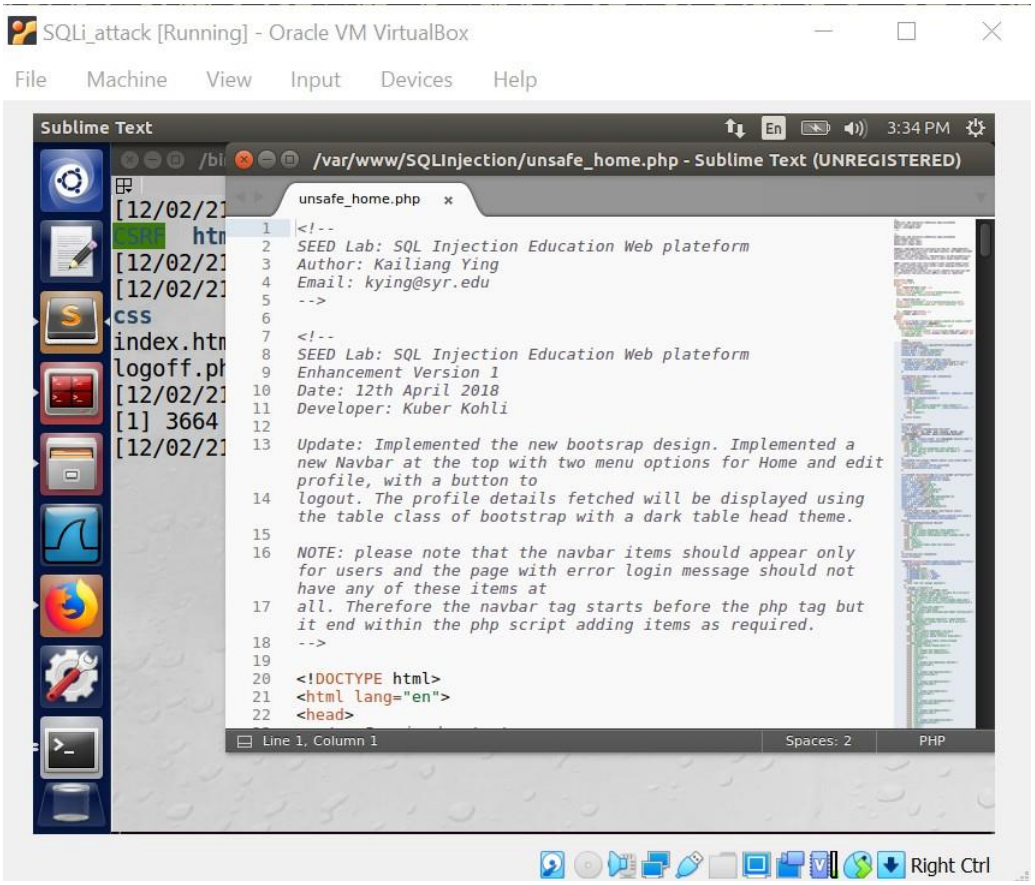
Notes: We connect to MySQL using the command *"mysql -u root -pseedubuntu."* The command *"use Users"* is then used to access the database Users. To extract all of Alice's information, we execute the command *"select * from credential where name="Alice";".*

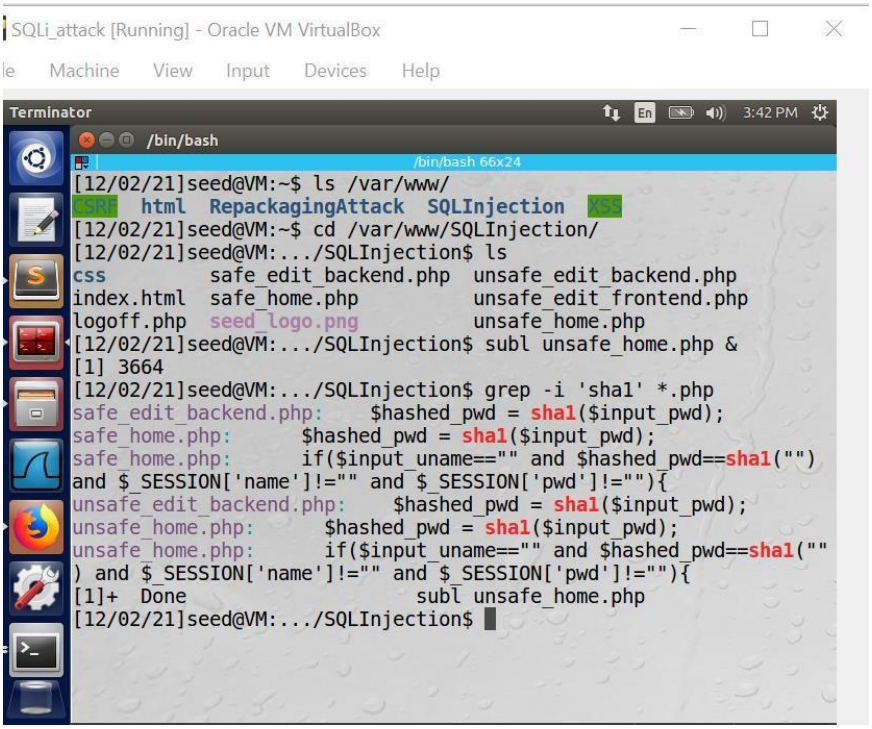
4.2 Task 2: SQL Injection Attack on SELECT Statement.




SQLi_attack [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
3:32 PM
/bin/bash
/bin/bash 66x24
[12/02/21]seed@VM:~$ ls /var/www/
CSRF html RepackagingAttack SQLInjection XSS
[12/02/21]seed@VM:~$ cd /var/www/SQLInjection/
[12/02/21]seed@VM:.../SQLInjection$ ls
css safe_edit_backend.php unsafe_edit_backend.php
index.html safe_home.php unsafe_edit_frontend.php
logoff.php seed_logo.png unsafe_home.php
[12/02/21]seed@VM:.../SQLInjection$


SQLi_attack [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
3:34 PM
/bin/bash
/bin/bash 66x24
[12/02/21]seed@VM:~$ ls /var/www/
CSRF html RepackagingAttack SQLInjection XSS
[12/02/21]seed@VM:~$ cd /var/www/SQLInjection/
[12/02/21]seed@VM:.../SQLInjection$ ls
css safe_edit_backend.php unsafe_edit_backend.php
index.html safe_home.php unsafe_edit_frontend.php
logoff.php seed_logo.png unsafe_home.php
[12/02/21]seed@VM:.../SQLInjection$ subl unsafe_home.php &


SQLi_attack [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sublime Text
3:34 PM
/var/www/SQLInjection/unsafe_home.php - Sublime Text (UNREGISTERED)
unsafe_home.php
<!--
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 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>
Line 1, Column 1
Spaces: 2
PHP
Right Ctrl

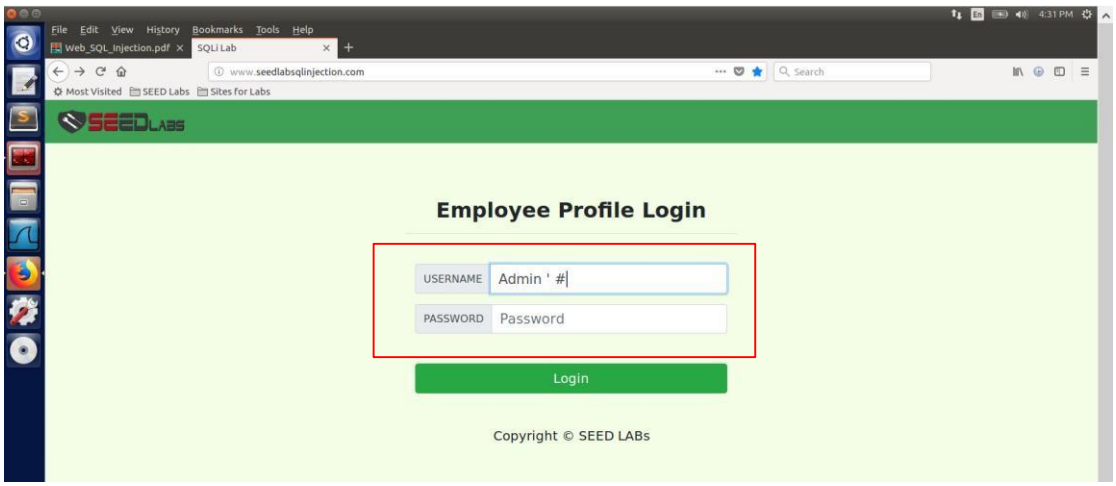


It shows all sha1 functions.

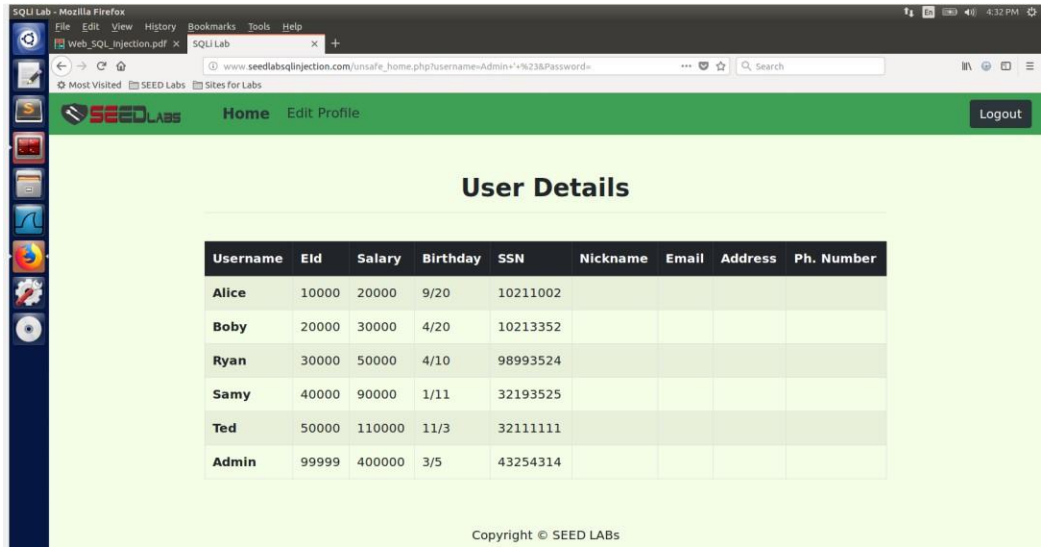
2.1: SQL Injection Attack from webpage.

I have to log in as the administrator to the online program in order to access all of the employees' data. I know the account's username is admin, but I don't know the password. To succeed in the assault, I must determine what to enter in the Username and Password areas.

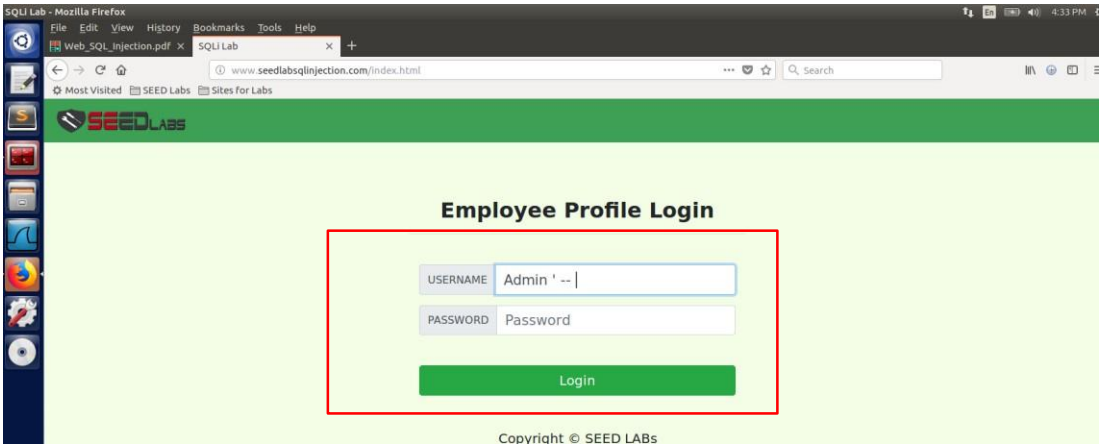
I notice that the $input uname variable stores the value entered into the Username input field on the login form. That value is also utilized in the WHERE clause of the SQL query. This implies that I may be able to input a Username value that alters the meaning of the SQL query. **WHERE name='admin'#' and Password='$hashed_pwd'";**



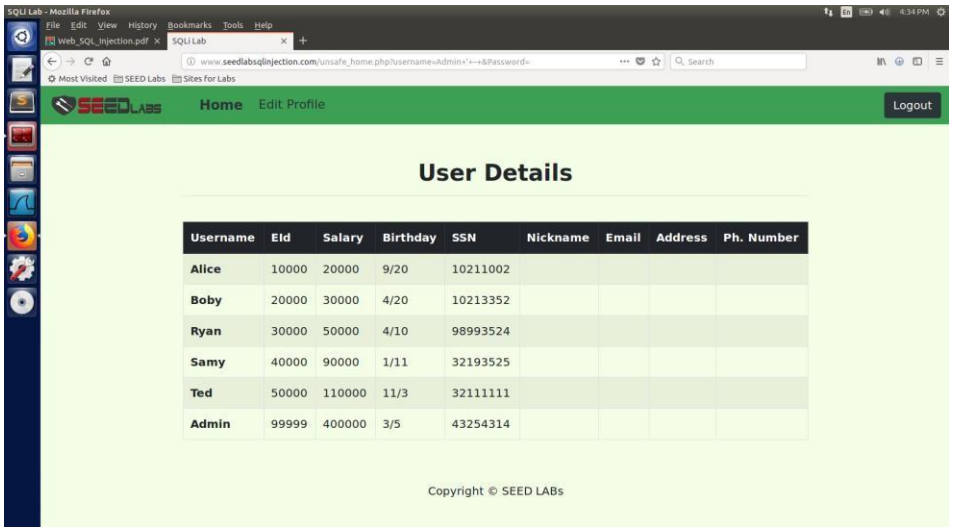
Notes: We are attempting to exploit a susceptible website to SQL Injection attacks by logging in as admin. Given that we know there is an administrator account called admin, we inject our code as described above to login without knowing the administrator's id and password.



other way to do this step:



I get the same result I can Entered as an Admin just by add to **username + ' --**



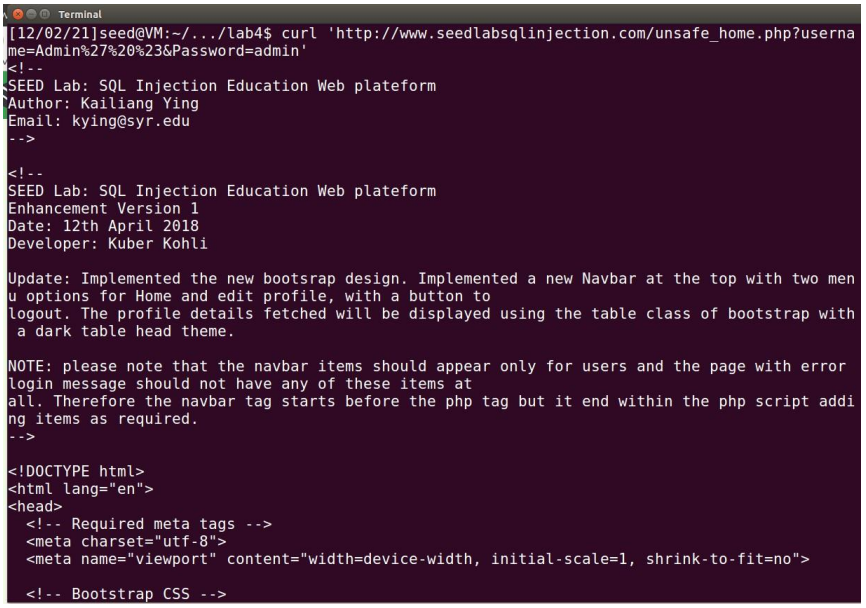
Notes: The following picture demonstrates that the attack was effective, and we signed in as admin despite not knowing the admin user's ID or password.

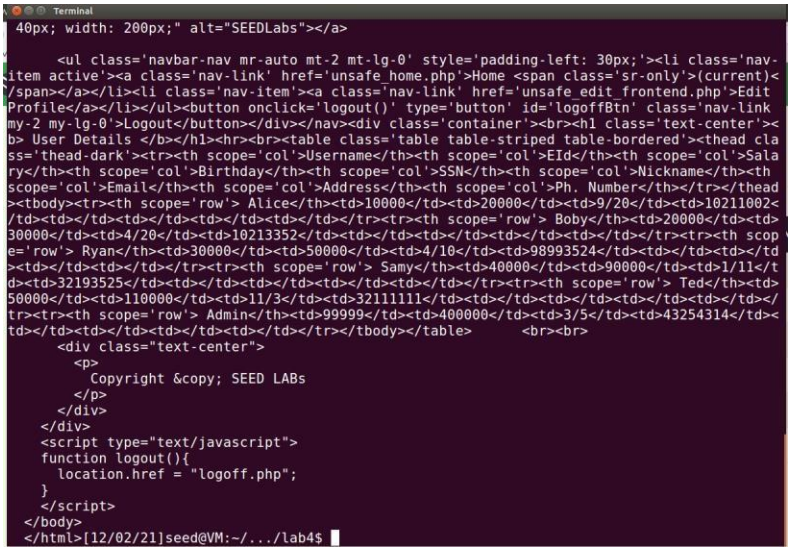
Description: The where clause is filled in with the employee ID and password fields. So, everything we type into these fields is included into the query. So, in order to take advantage of the SQL Injection attack, we inject the following code: Alternatively, ' Name='admin';# .

The single quotation closes the argument for the input id, and the OR statement that follows permits us to login as admin. The # is added at the end to comment out anything that follows, allowing the password input to be bypassed.

2.2: SQL Injection Attack from command line.

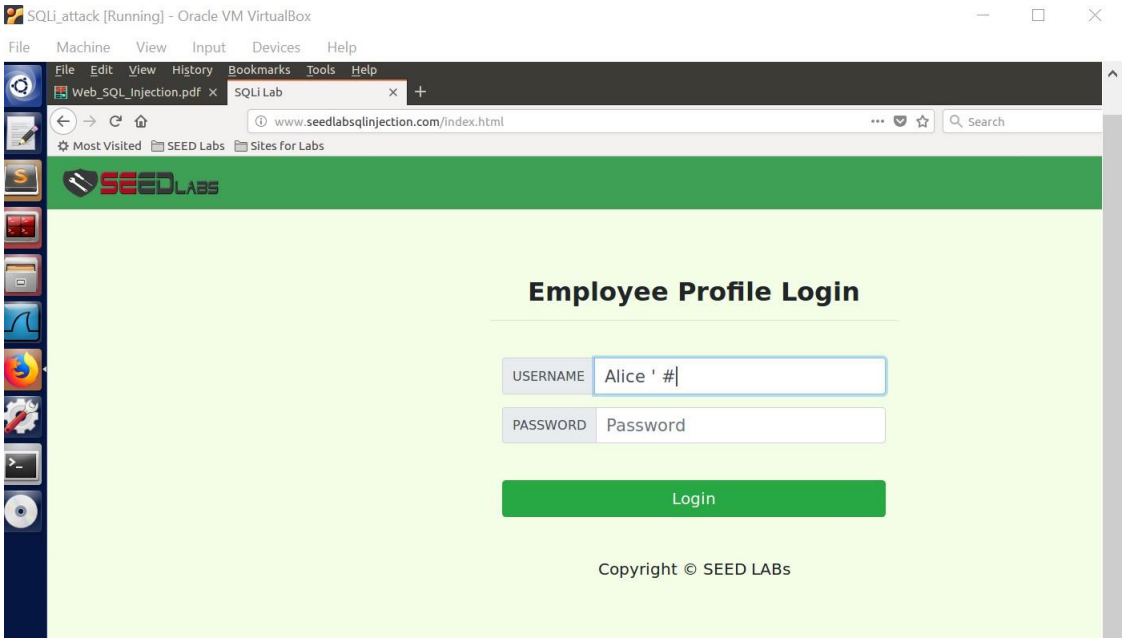
I use the curl command to make an HTTP request to the website and login again in the same way as previously, and we observe that we get the HTML page in return:



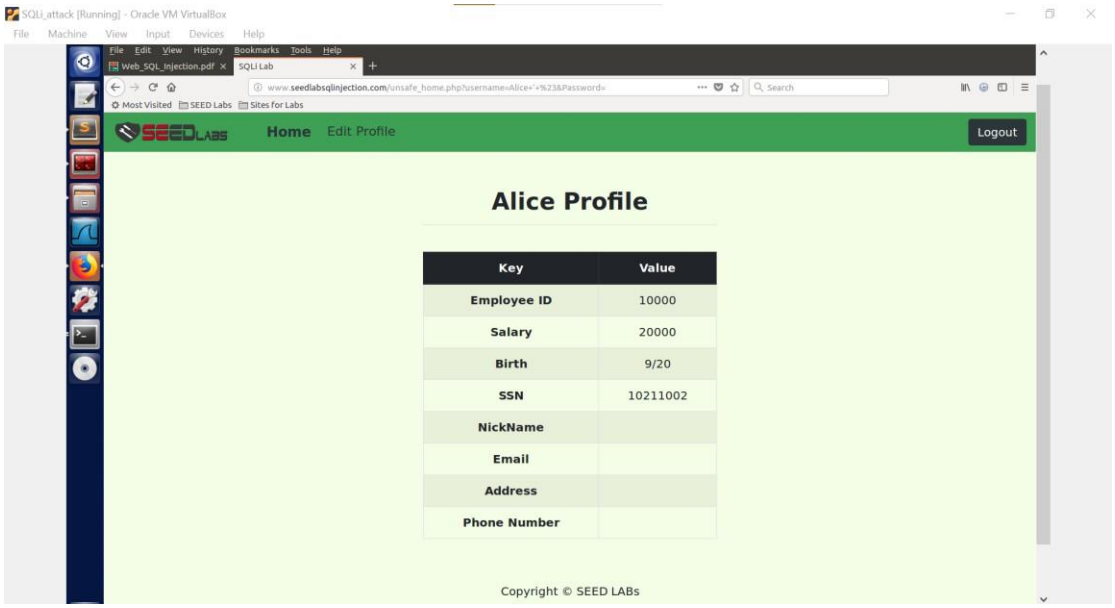


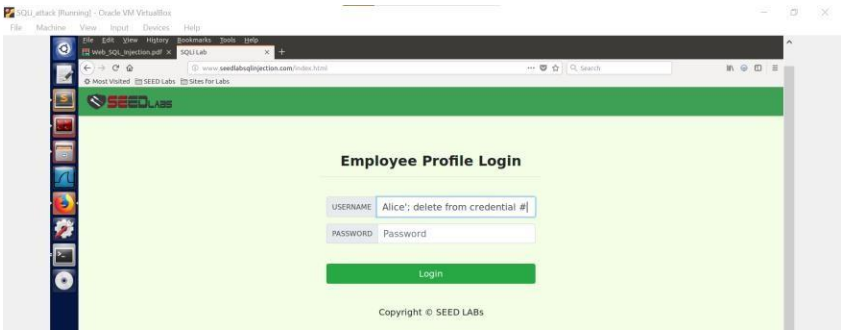
I notice that all of the employee's information is delivered in an HTML tabular format. As a result, I was able to execute the identical assault as in Task 2.1. Whereas the Web UI does not assist in automating the assault, CLI commands can. The curl command encodes the special characters in the HTTP request, which was a significant departure from the web UI. I utilize the following: Space - %20; Hash (#) - %23; and Single Quote (') - %27.

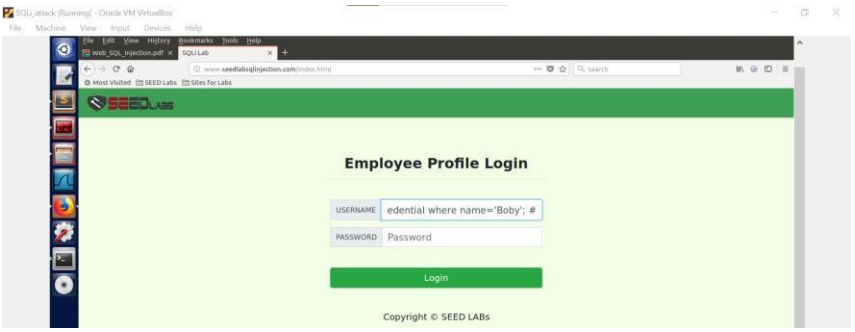
2.3: Append a new SQL statement.



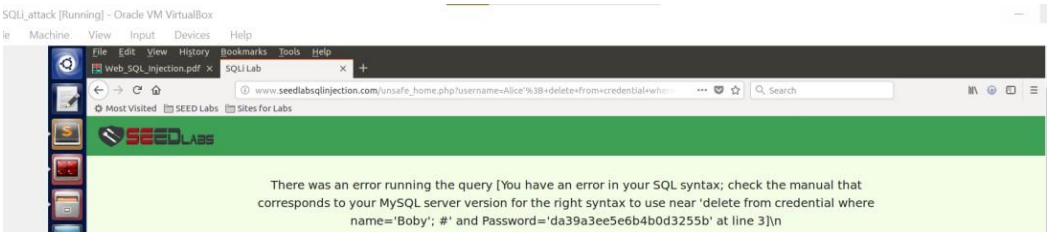
When I entered to Alice profile by use the same way I use it task 2.1

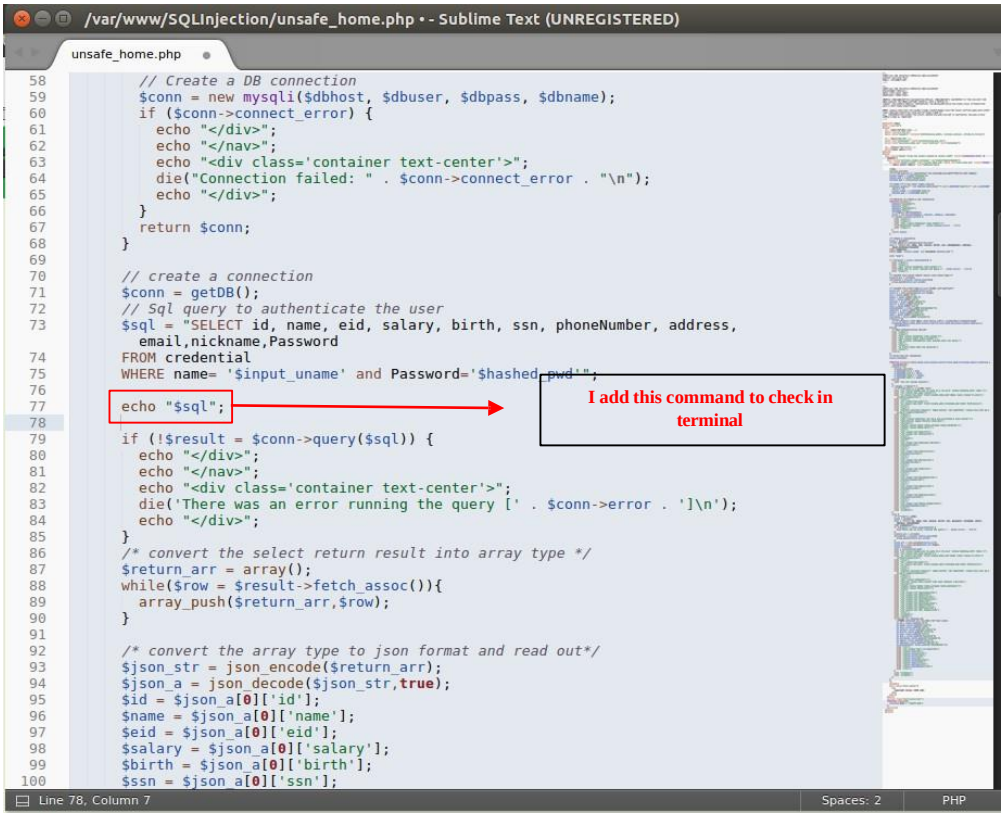




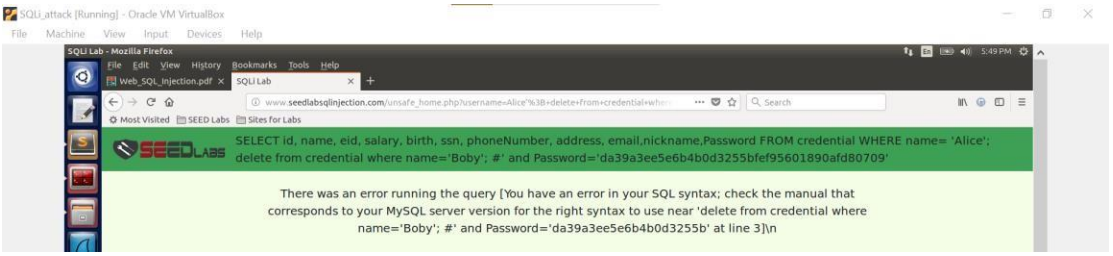


To add an entry to the current database, I need to remove the following statement: *(DELETE credential where name='Boby';)*

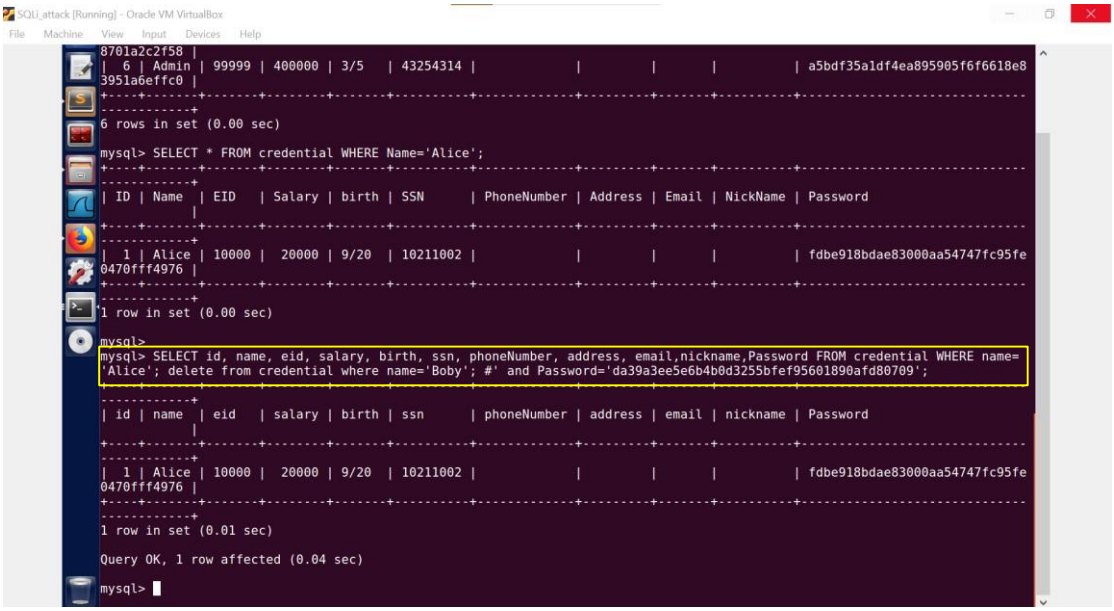




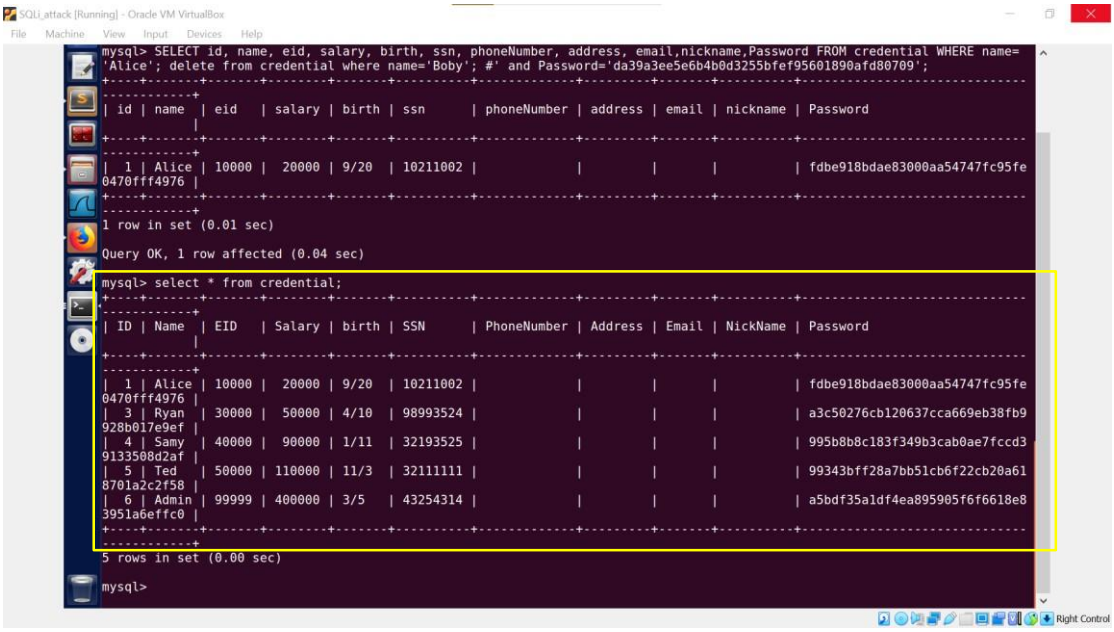
When I re-check in the website it shows same statement



I coped the statement and past it in terminal



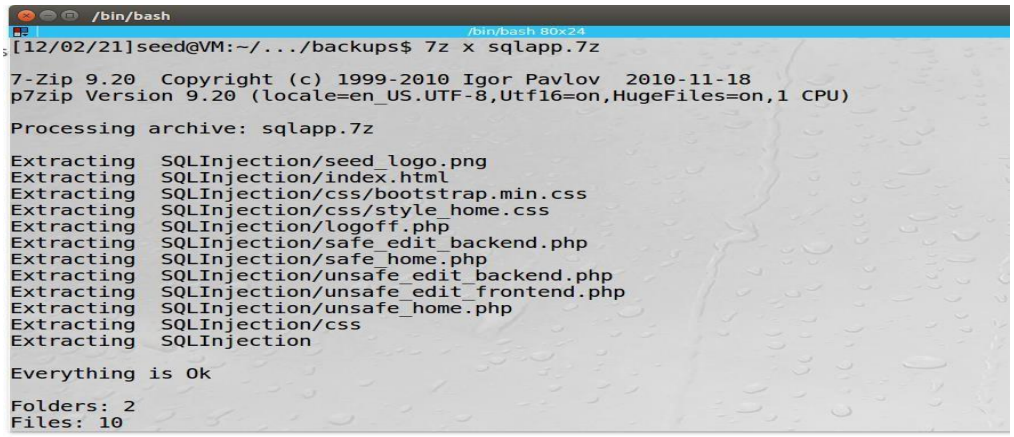
I get the result!

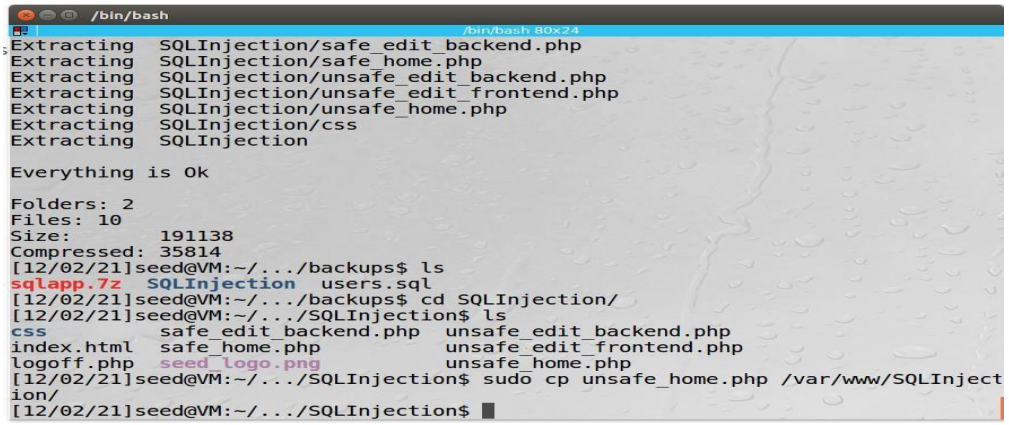


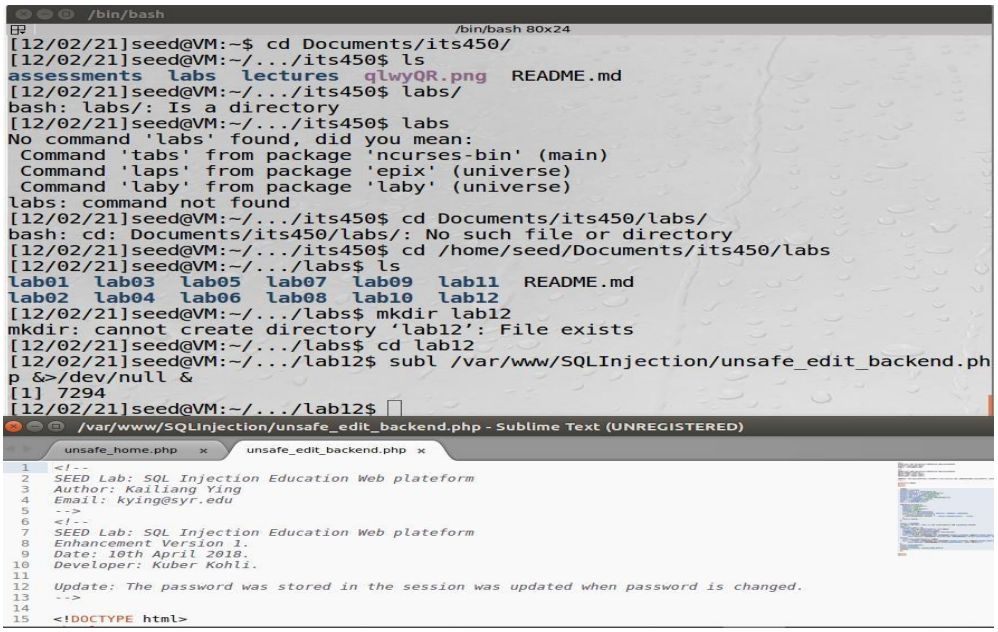
Boby removed successfully.

4.3 Task 3: SQL Injection Attack on UPDATE Statement.

All of task 3's subtasks will be focused on launching SQL injection attacks utilizing Alice's profile's Edit Profile Page.

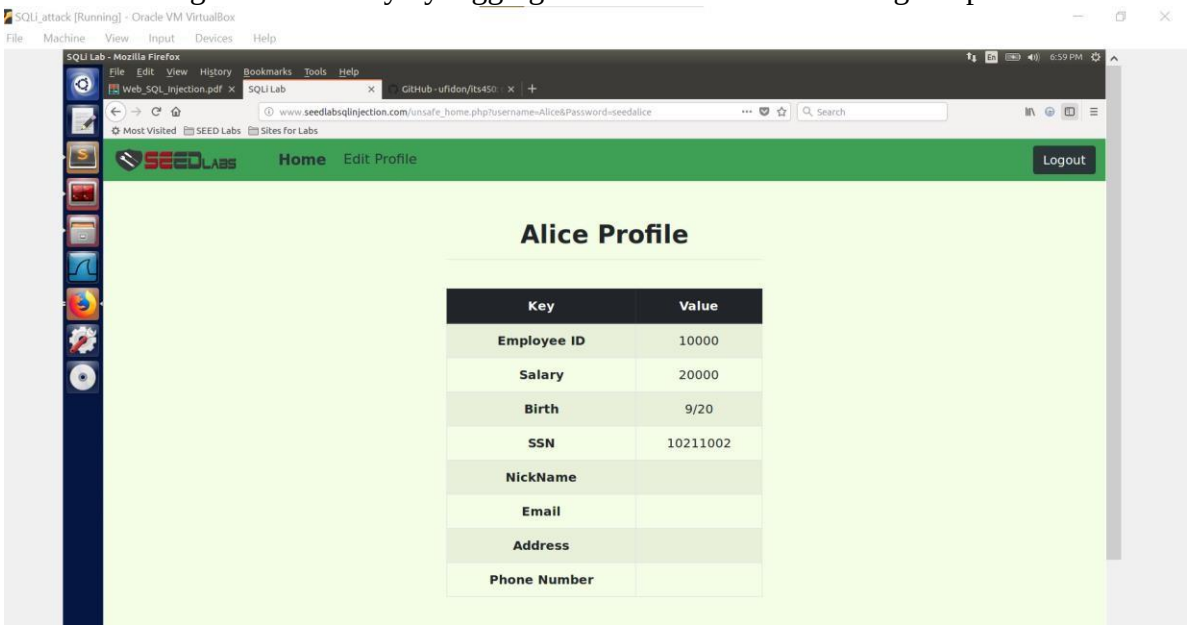




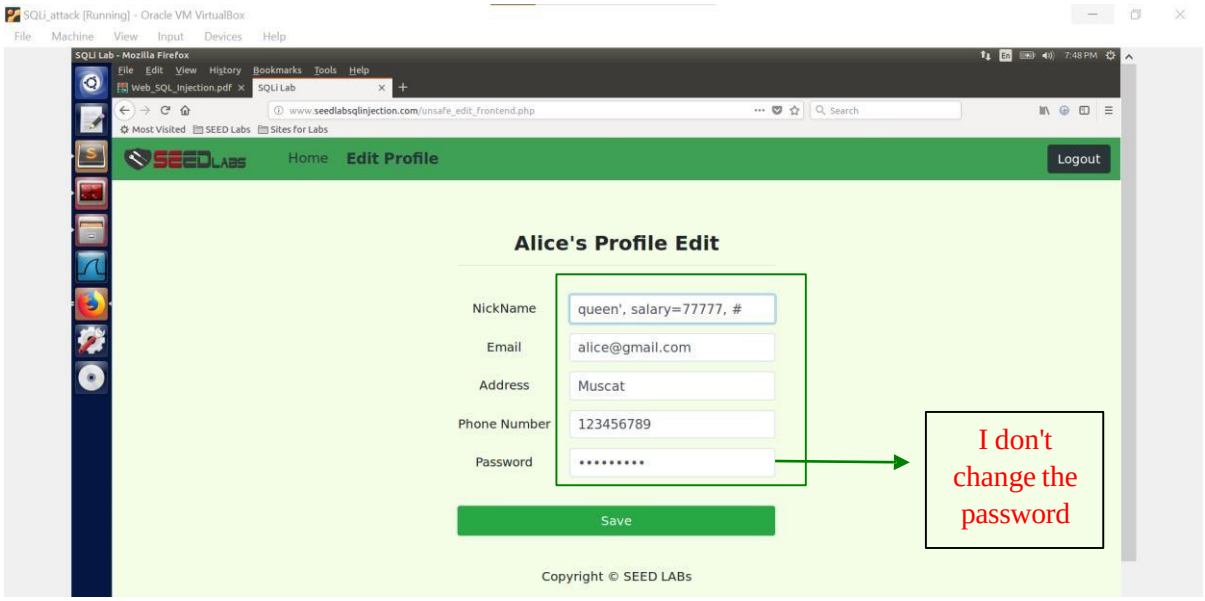


3.1: SQL Injection Attack on UPDATE Statement — modify salary.

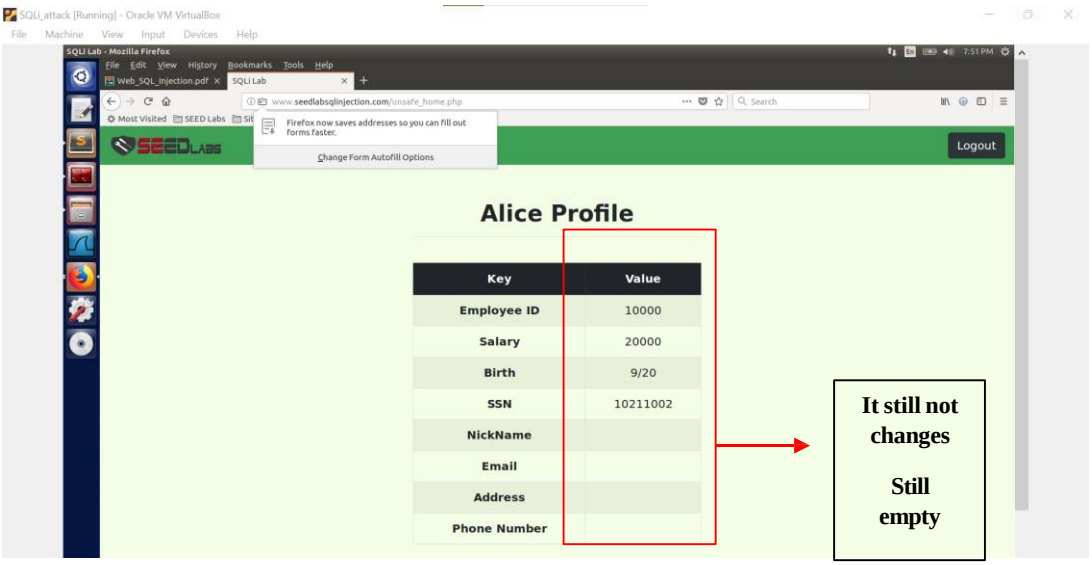
I can change Alice's salary by logging into her account and editing her profile.



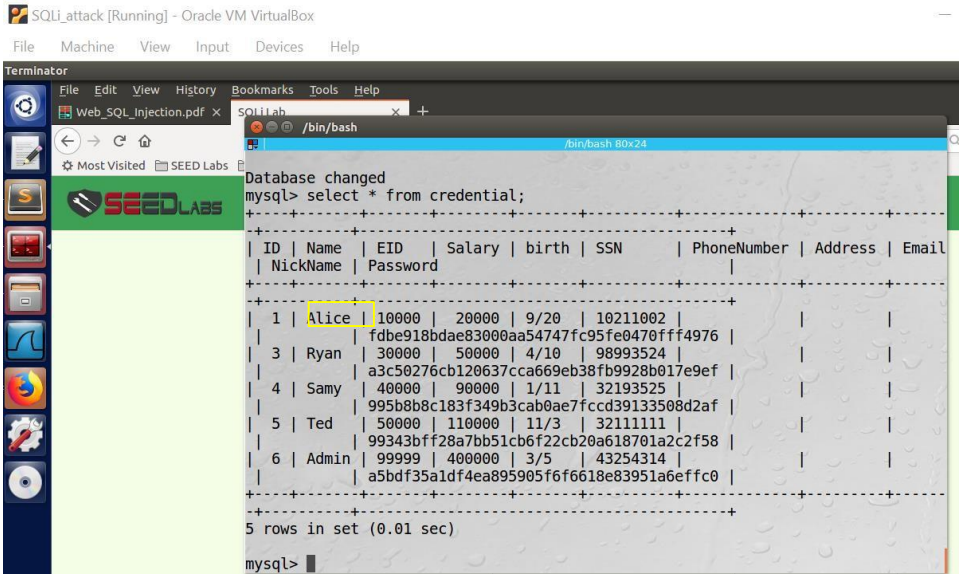
I fill out the form with the following information



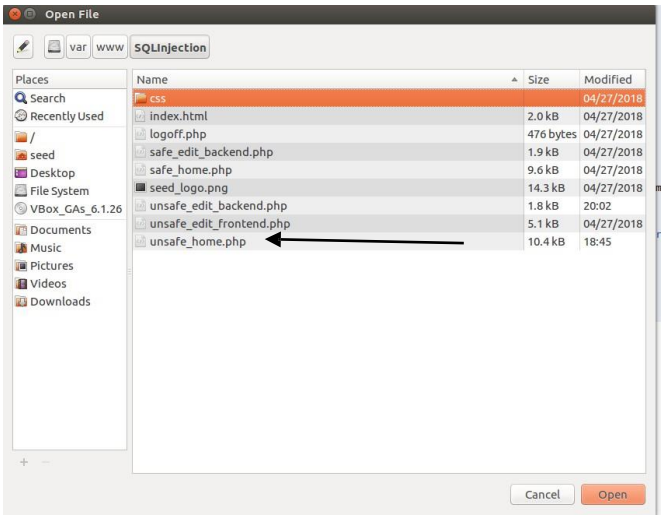
Then I click save to check if it changes

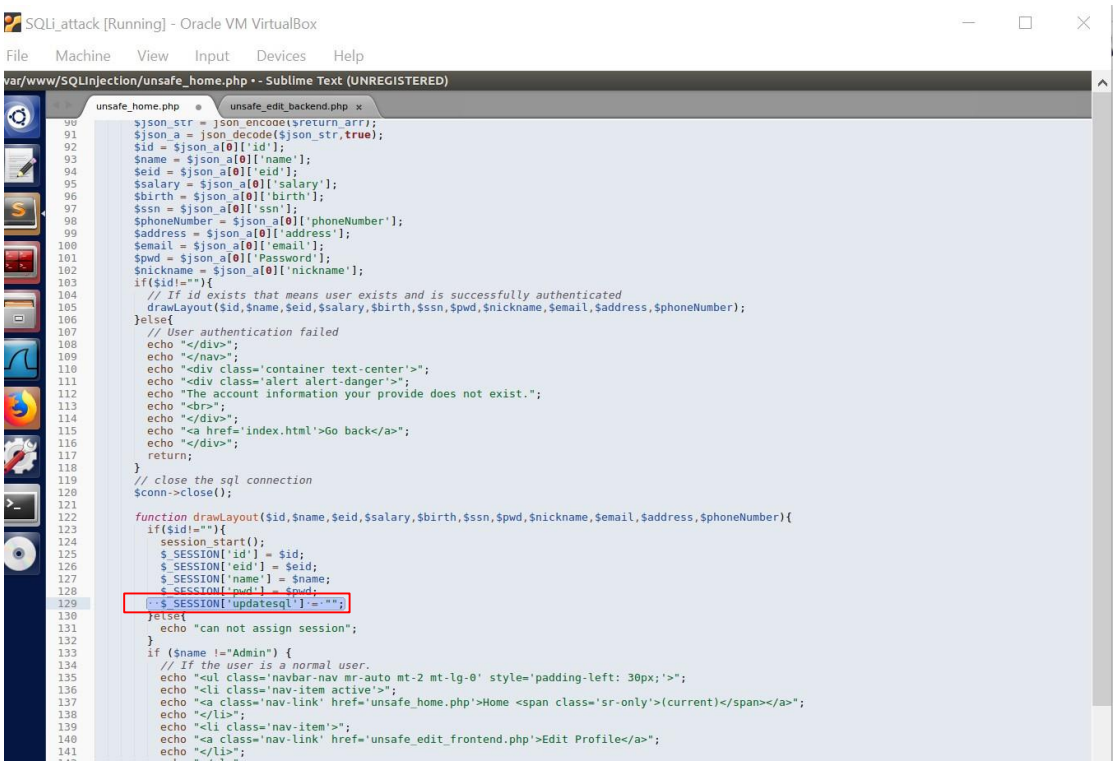


now I will check by terminal

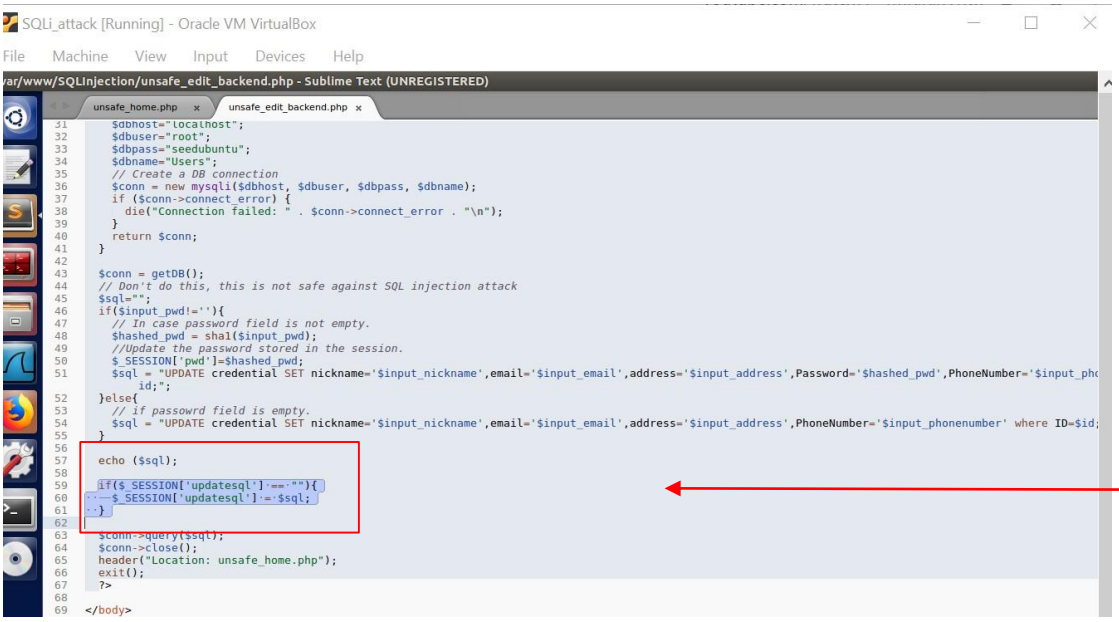


still not modify. so, I will check & add some statement in php code.

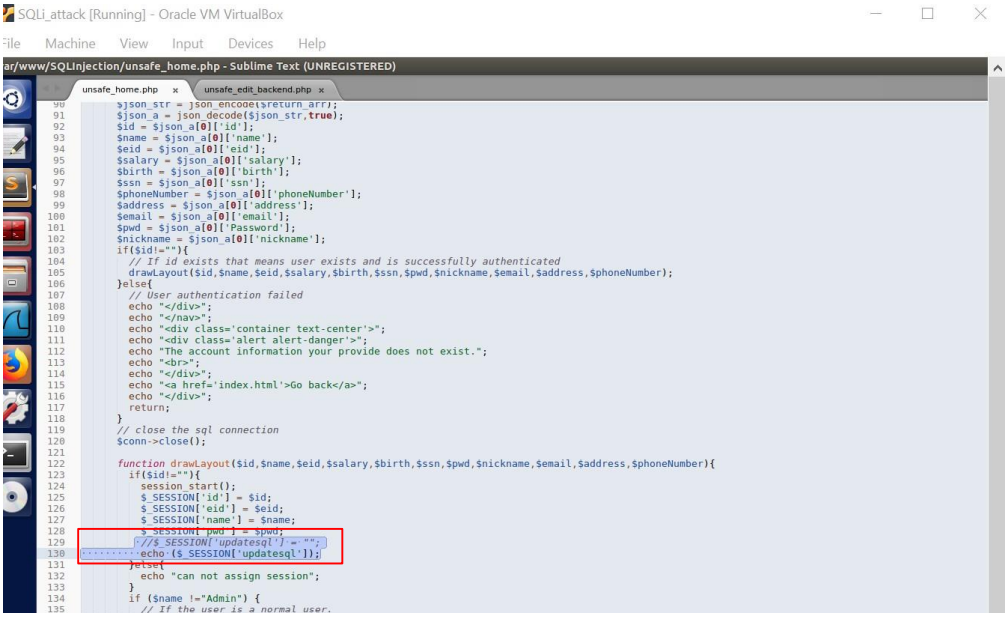




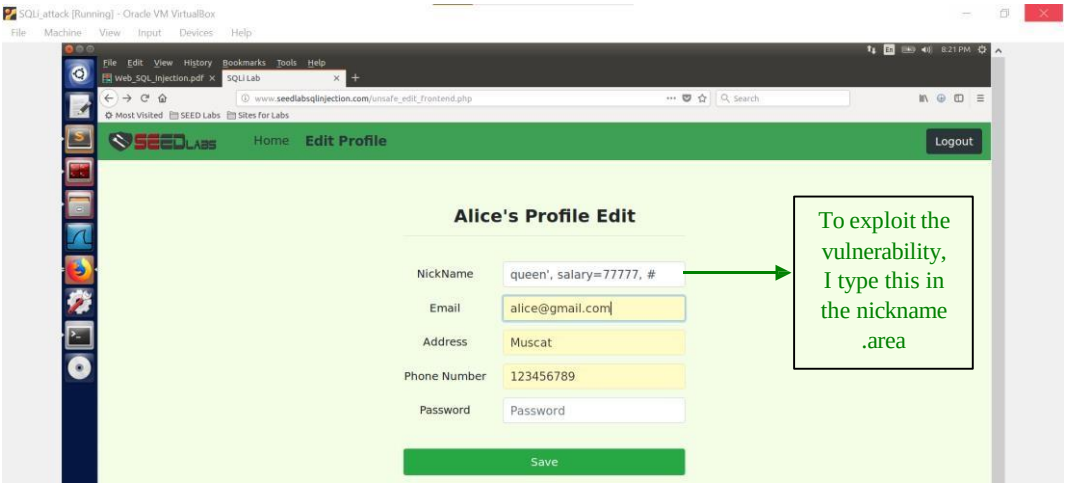
I add this statement in unsafe_home.php and I add it in unsafe_edit_backend.php as a conditional statement.



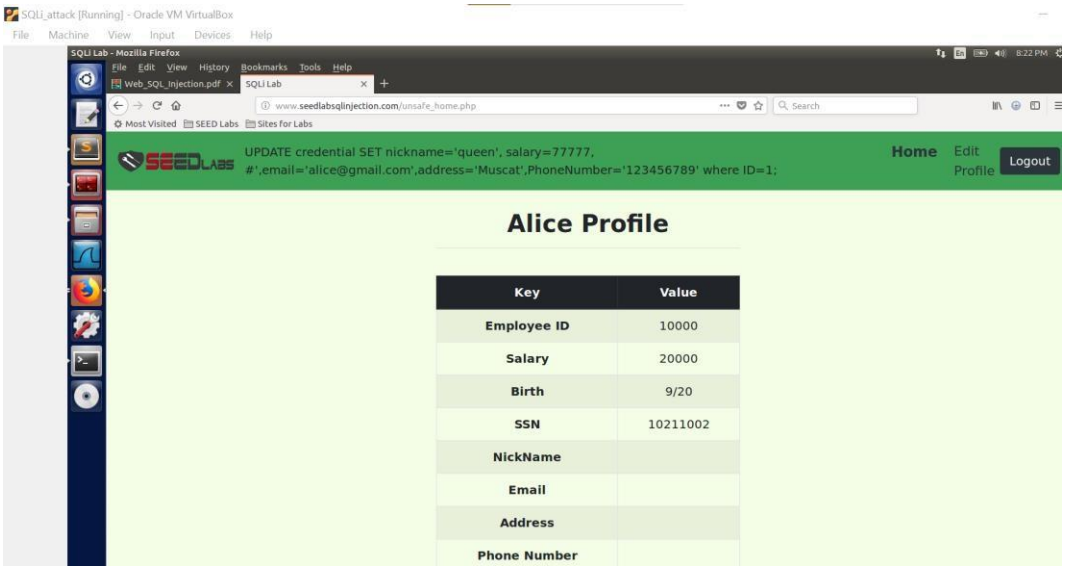
Now I retain to unsafe_home.php to command this statement



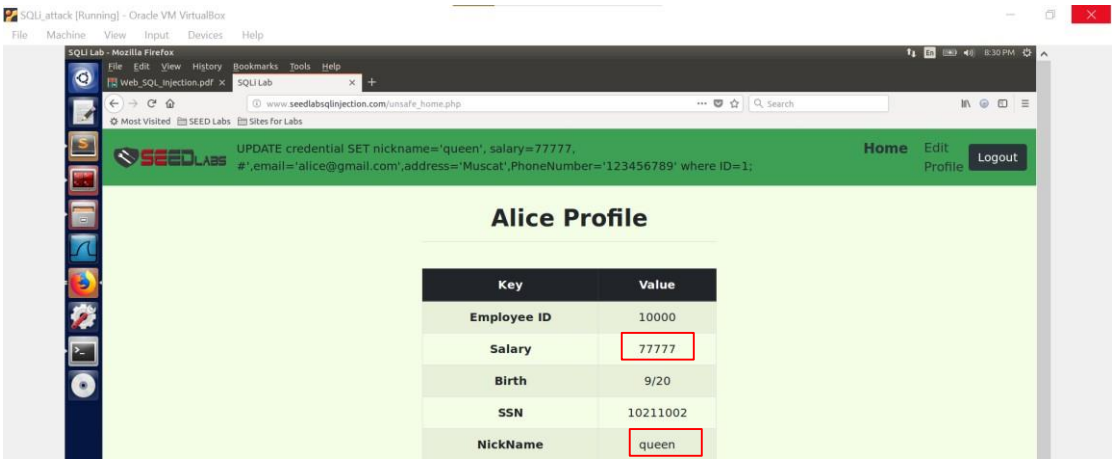
lats try it again



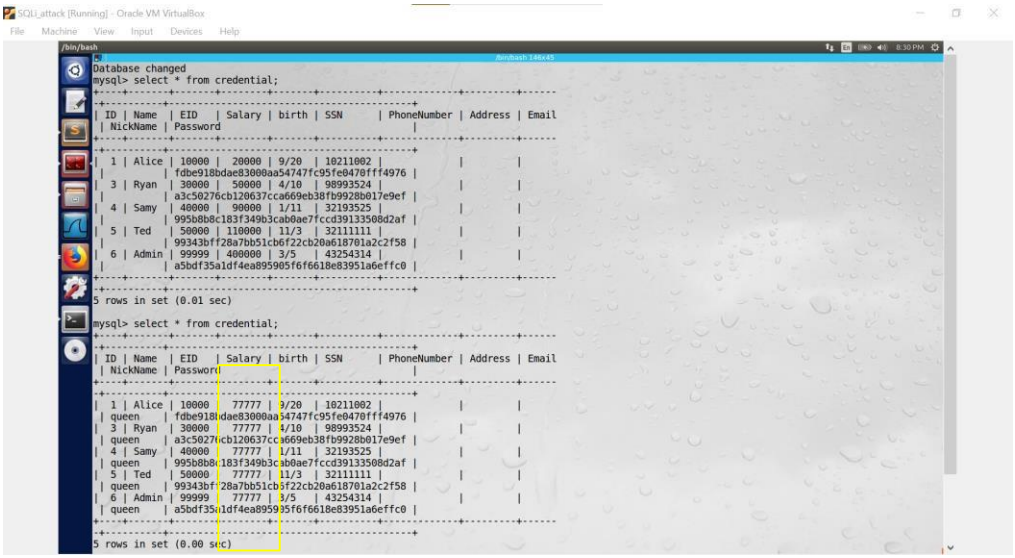
Hear, I make a mistake I forget put ' ' in the first one so I don't git change.



When I fix my mistake, I get the changes



Now I will re-check in terminal



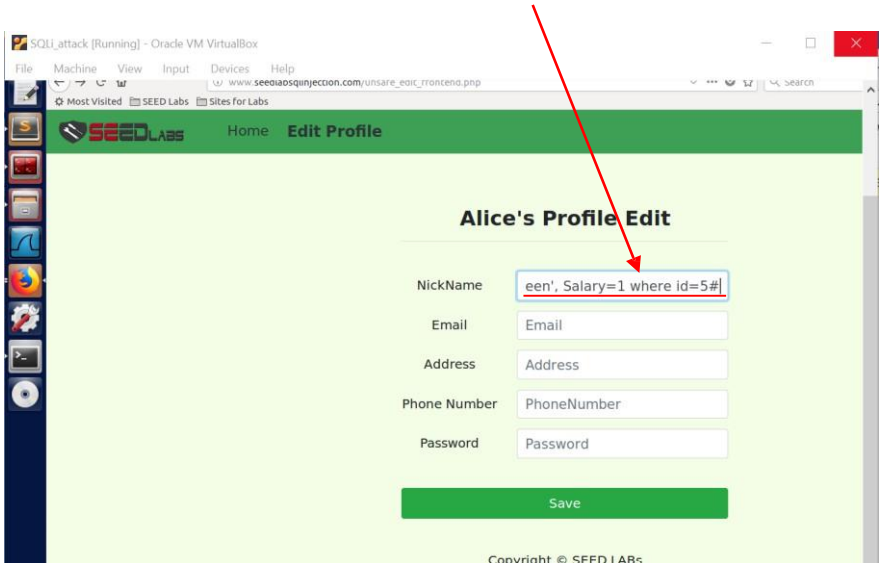
every employee's salary changes.

Note: that in this screenshot, Alice's salary has been altered from 1000 to 77777.

Description: I attempting to attack a SQL injection vulnerability by introducing code into the edit profile page in order to alter the pay of the current employee. I place a # at the end to comment out all the other values that follow so that we don't have difficulties with null or invalid input values from other input fields. I carry out this attack and change the salary field, even though it is not visible since the employee is not permitted to alter it. It can only be edited by the administrator. Since the attack was successful, Alice's pay has been adjusted.

Also, I can change the salary of other employees by use Alice profile

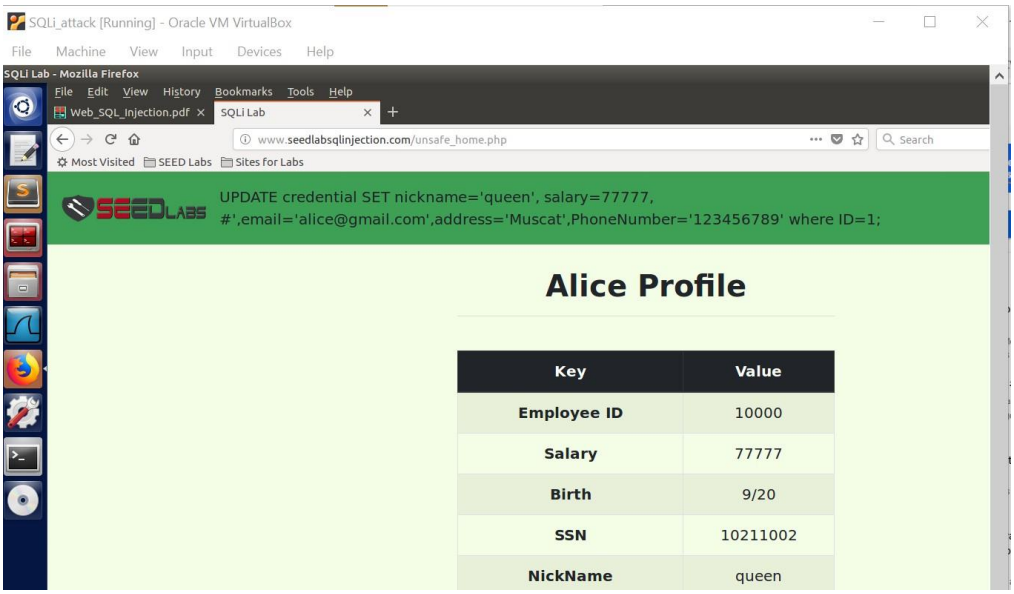
As I write here:



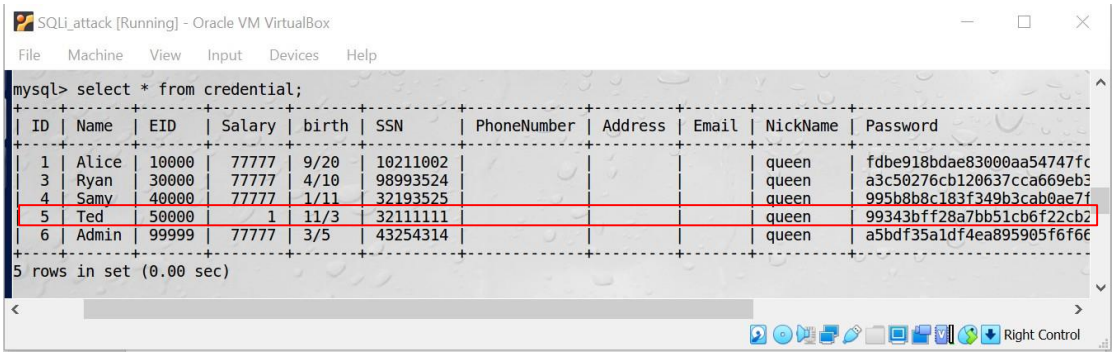
So, employee number 5 (who have ID=5) in the list he/she will get new salary 1doler only.

And that change – by attacker- just by using Alice profile.

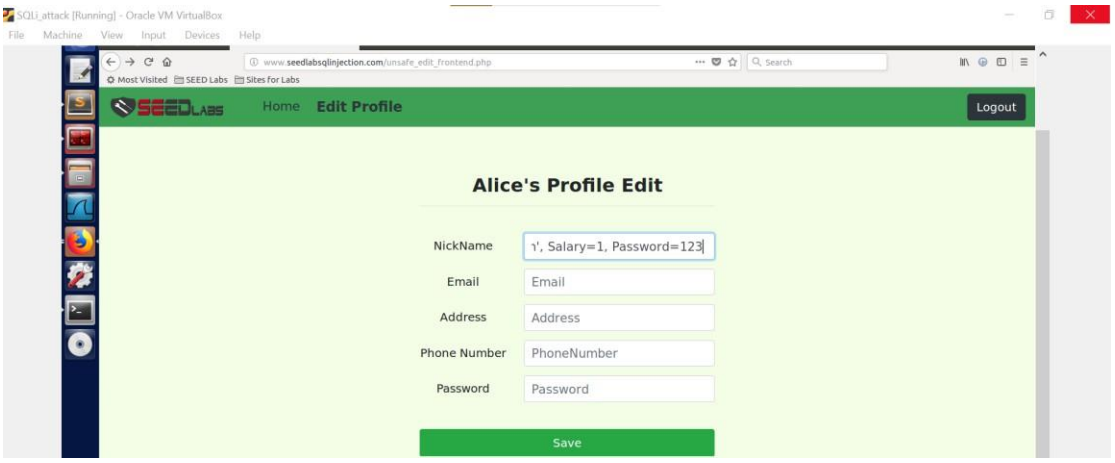
Save the change.



Than check by termainal

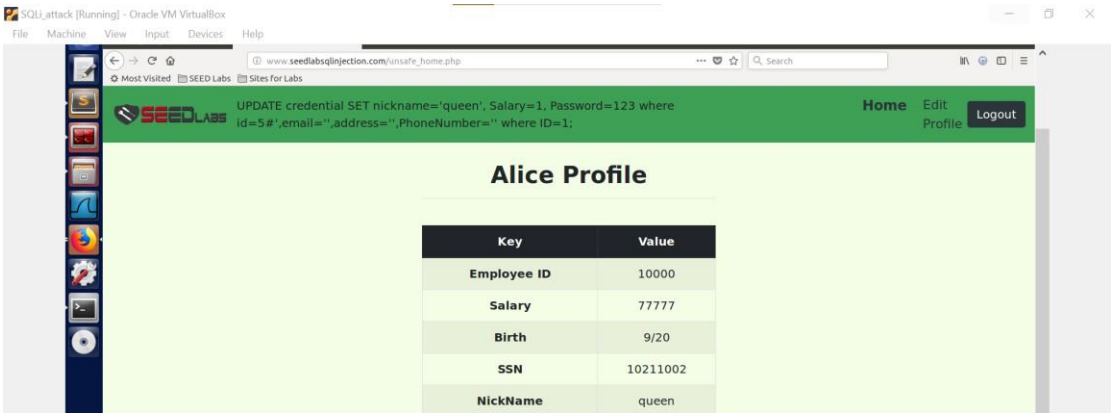


3.2: SQL Injection Attack on UPDATE Statement — modify other people' password.

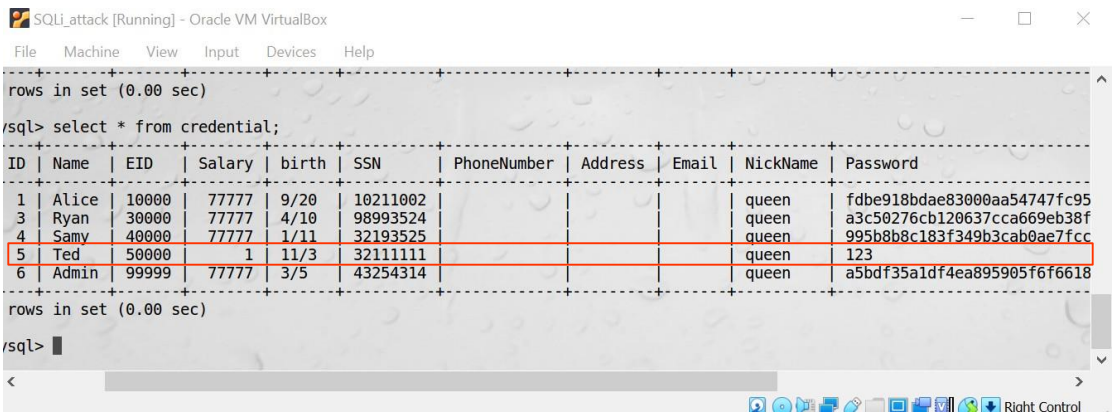


To modify Ted's password, I take a same procedure as before and insert the following in :Alice's profile field 'Phone number' by modifying the prior statement I wrote

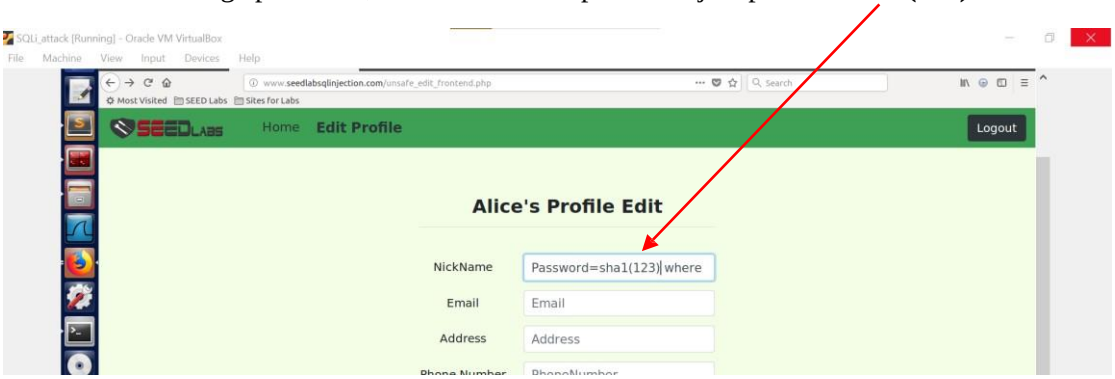
as the password #123 ,'



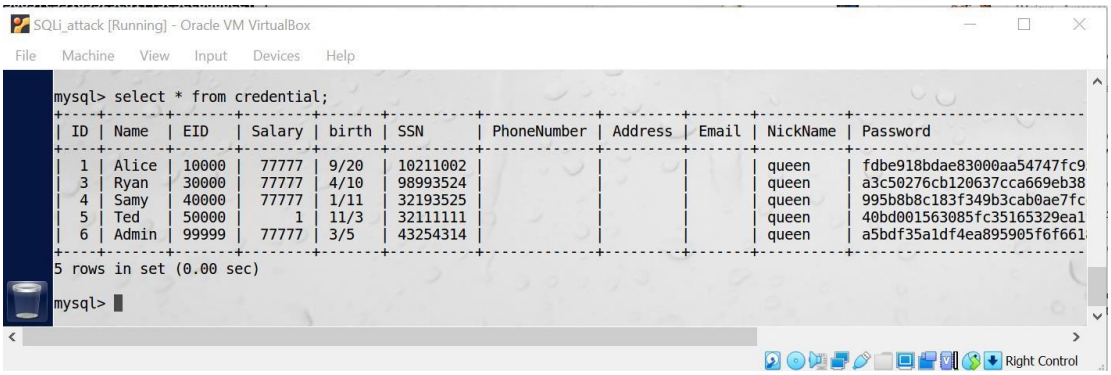
saving the new editing, than check in trmainal.



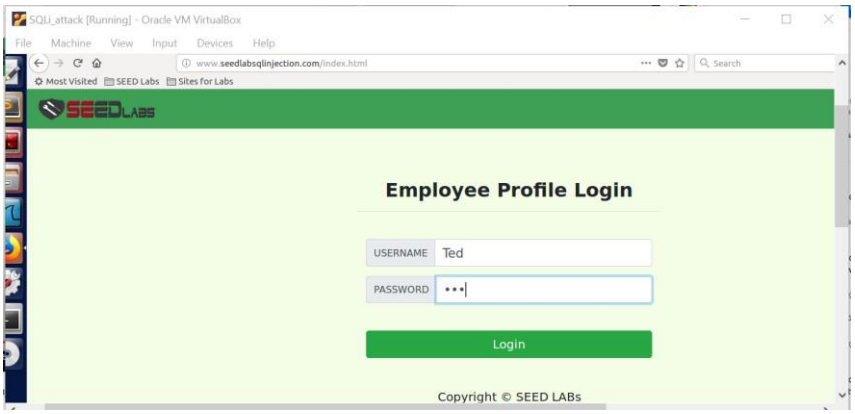
I change password, To retain the old password just put this: **sha1(123)**

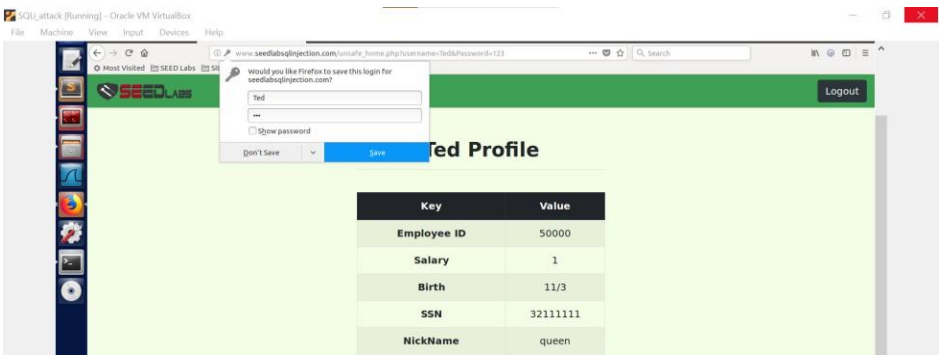


Now I will re-check if it works.



I will try again by using new password: 123





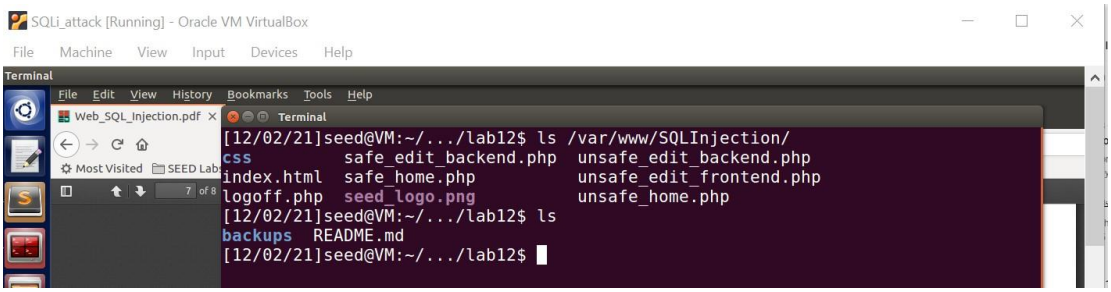
Now, when I log in with the new password, I notice that I am able to log in successfully. By utilizing the sha1 function in our input, I am essentially completing the identical steps as the

program. This demonstrates that I was successful in carrying out our SQL injection attack to alter the password.

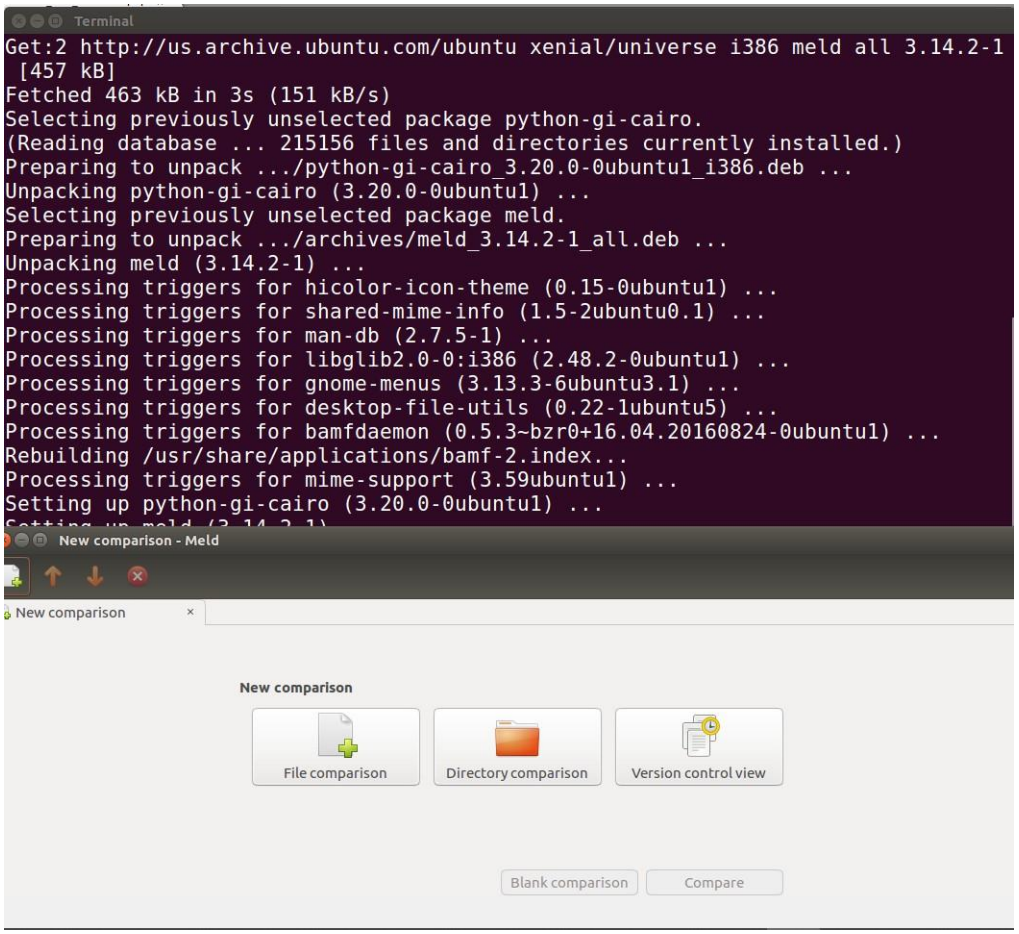
4.4 Task 4: Countermeasure.

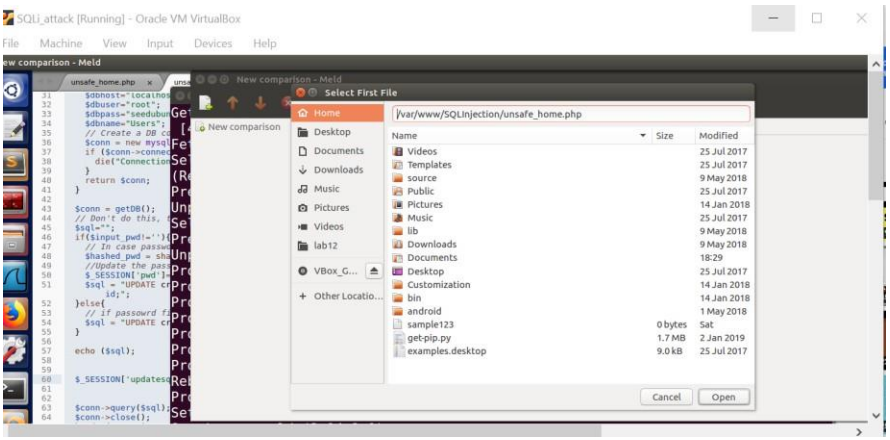
In this task4, I must modify the source code of both unsafe_home.php and unsafe_edit_backend.php in order for them to use the prepared statement technique. This will act as a deterrent to SQL injection attacks, keeping PHP code secure.

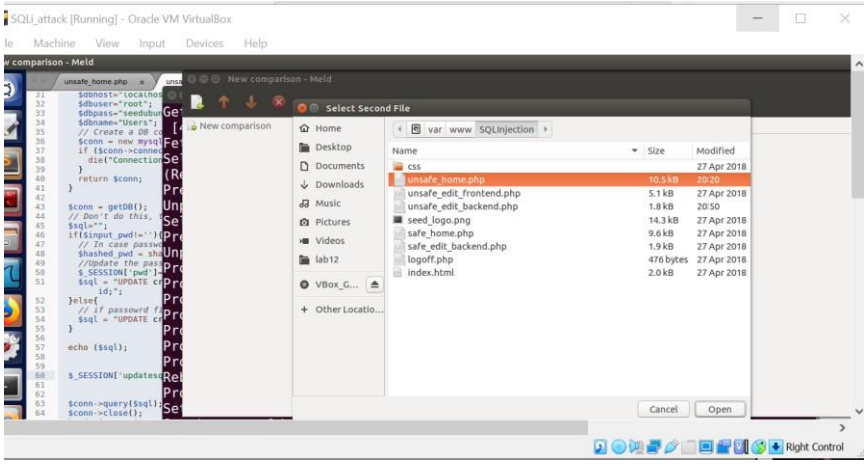
I will install meld it was useful in this task



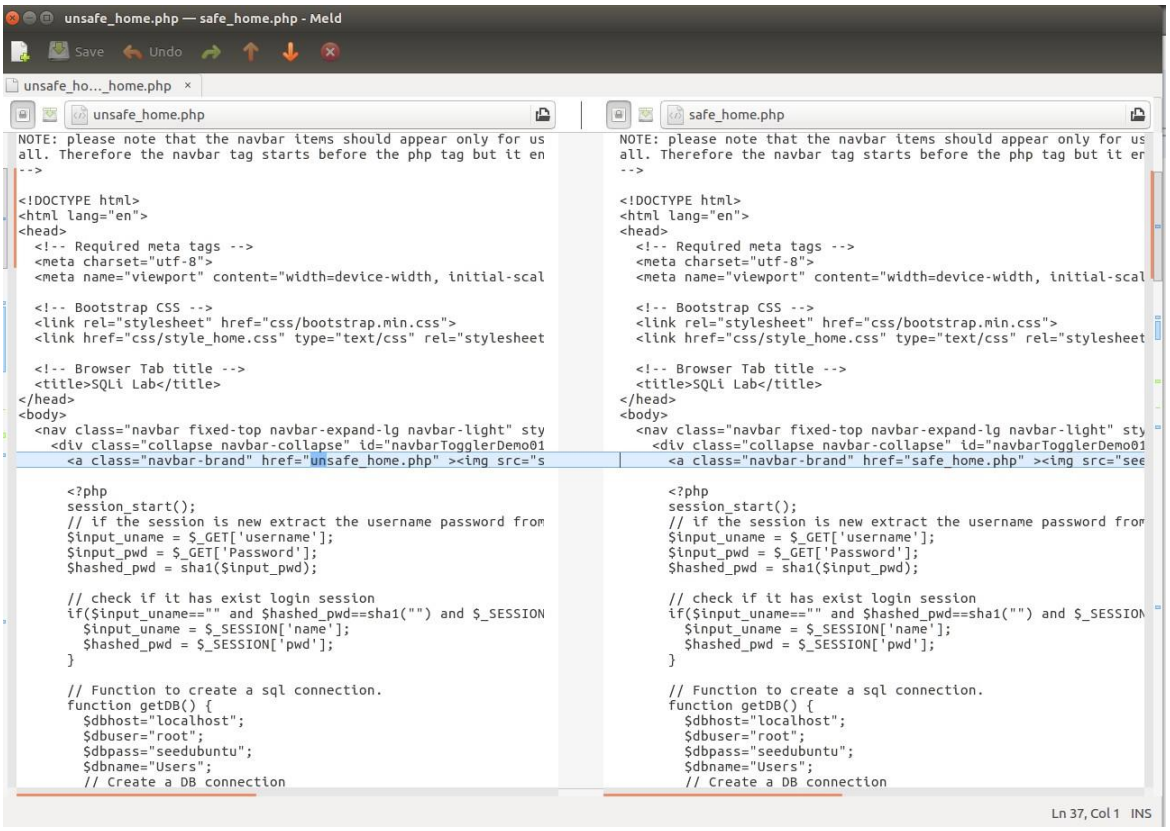
After that install meld

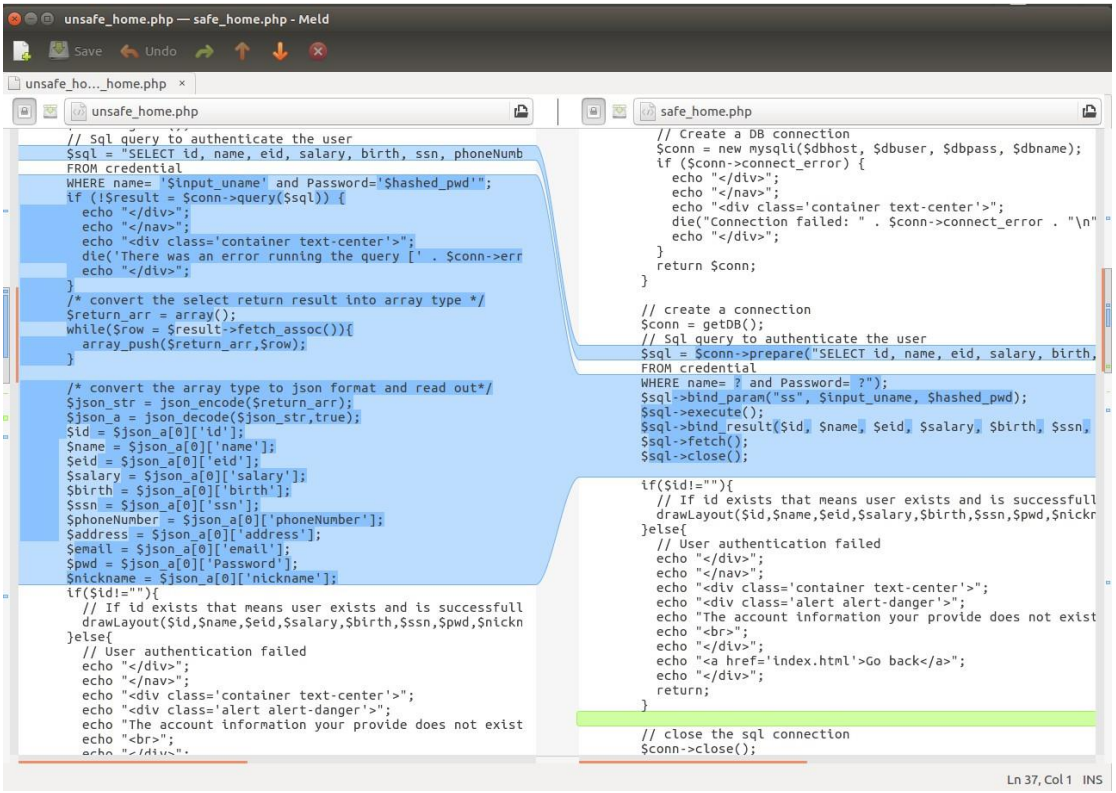


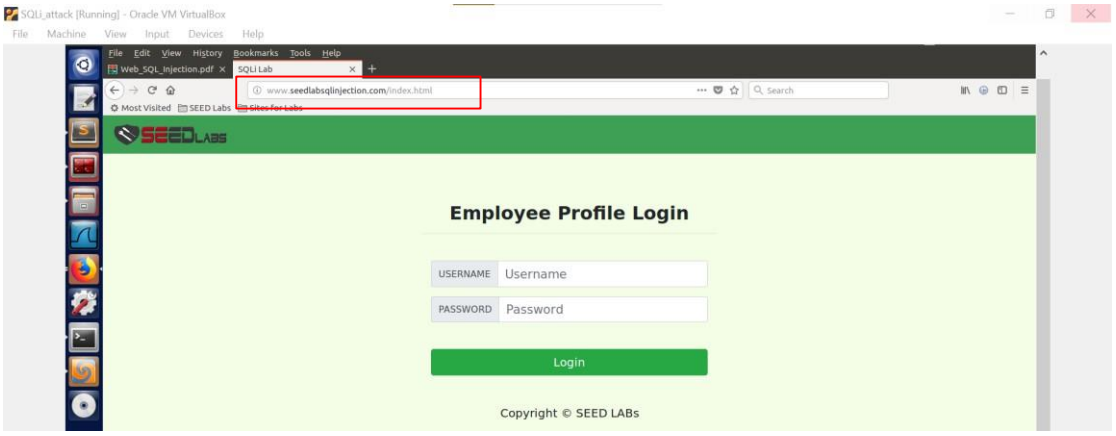




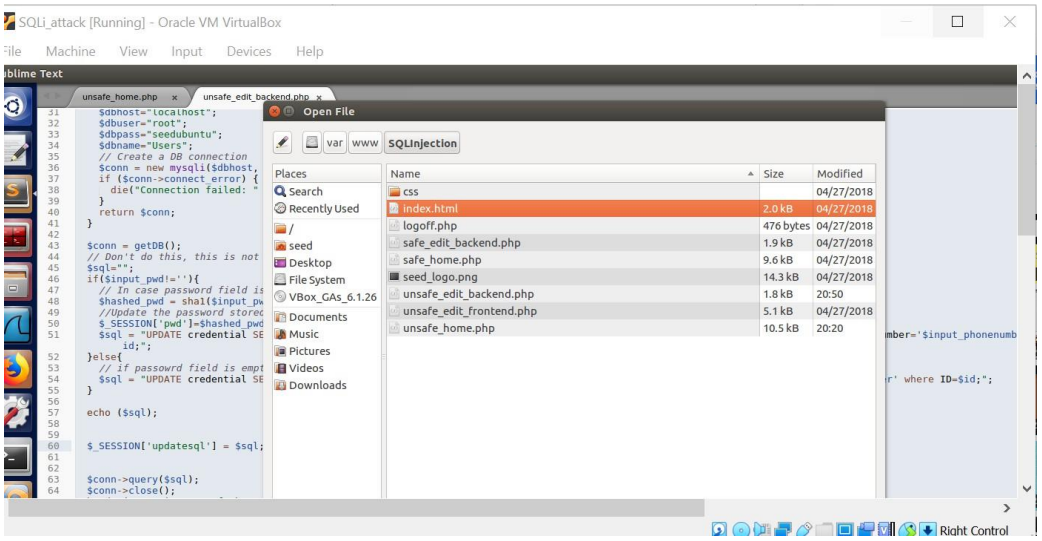
I will Compere between unsafe_home.php & safe_home.php

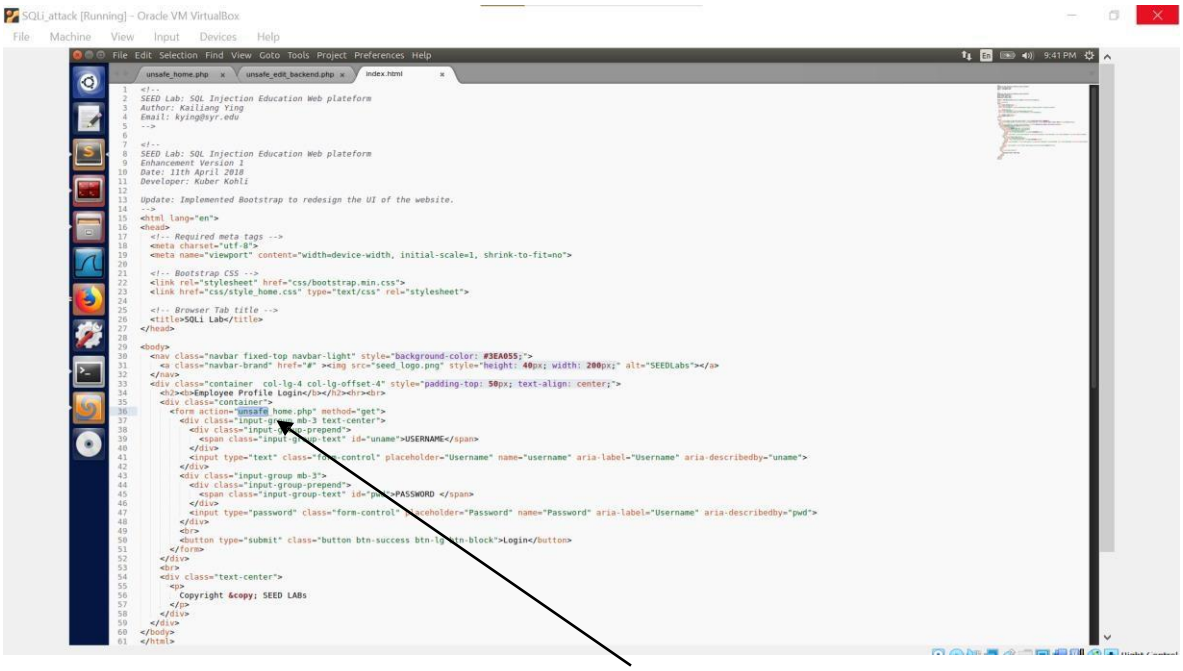




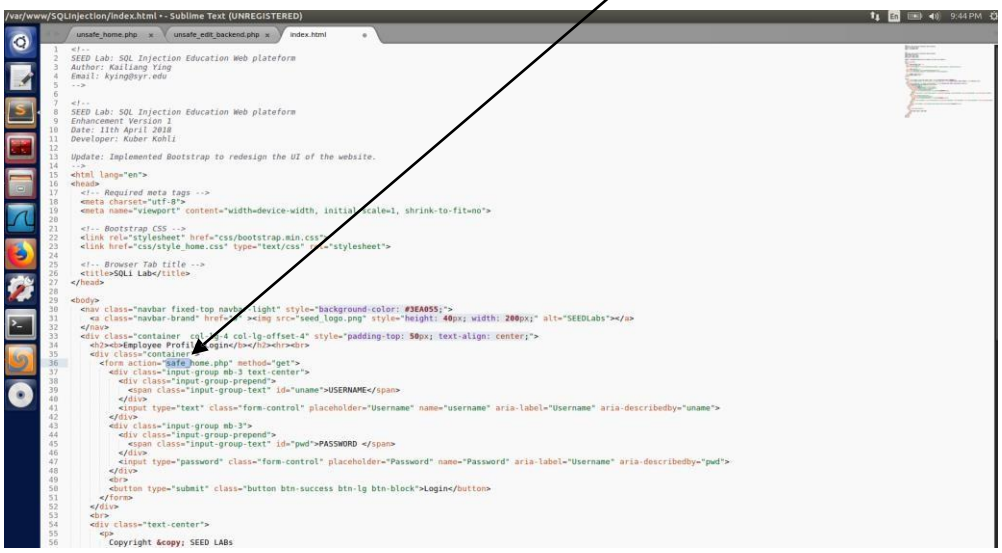


I will open index.html

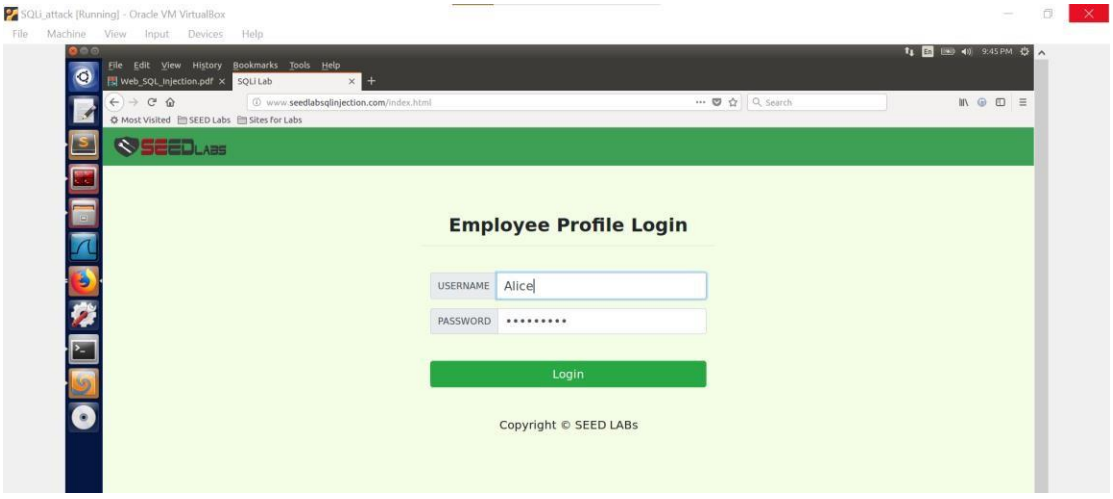


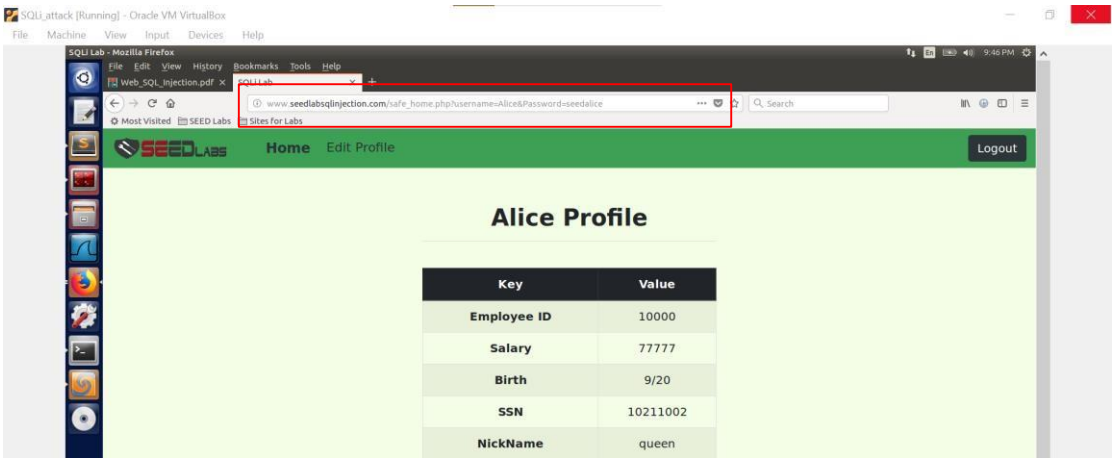


Change unsafe t to safe.



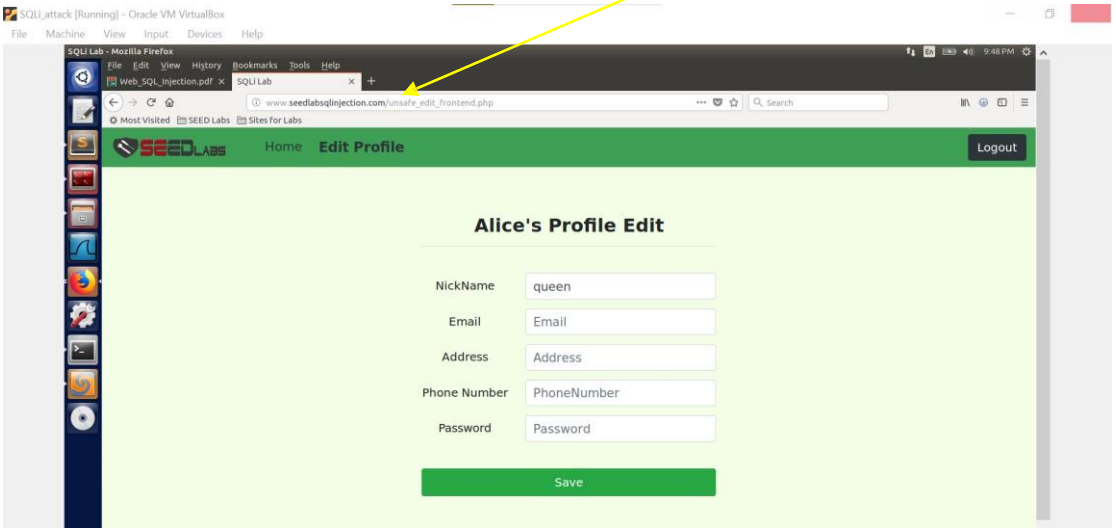
After refresh the website, I will log in to Alice profile.



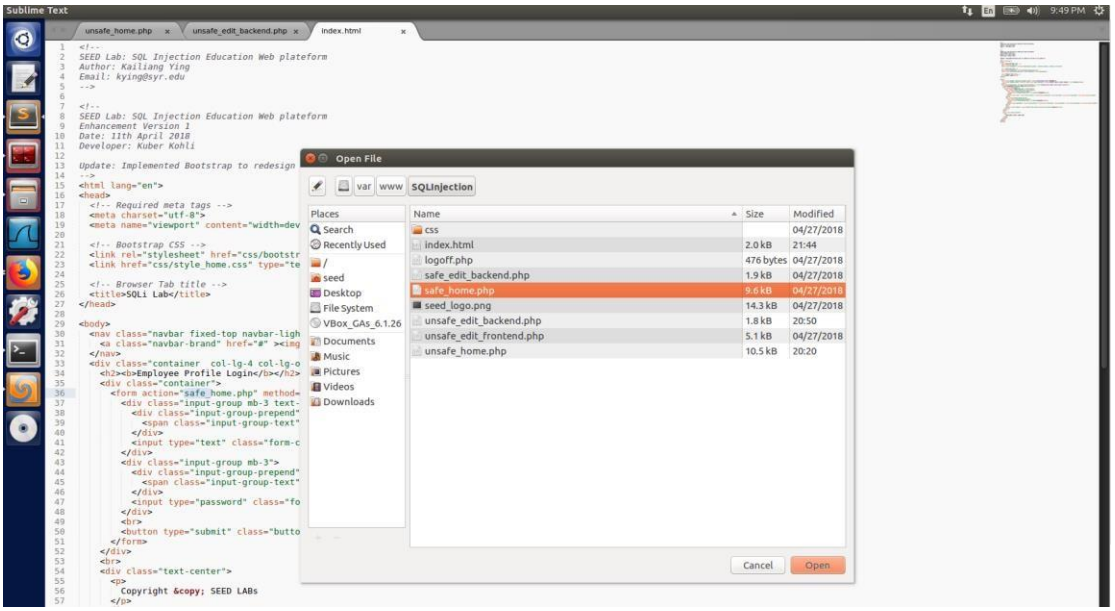


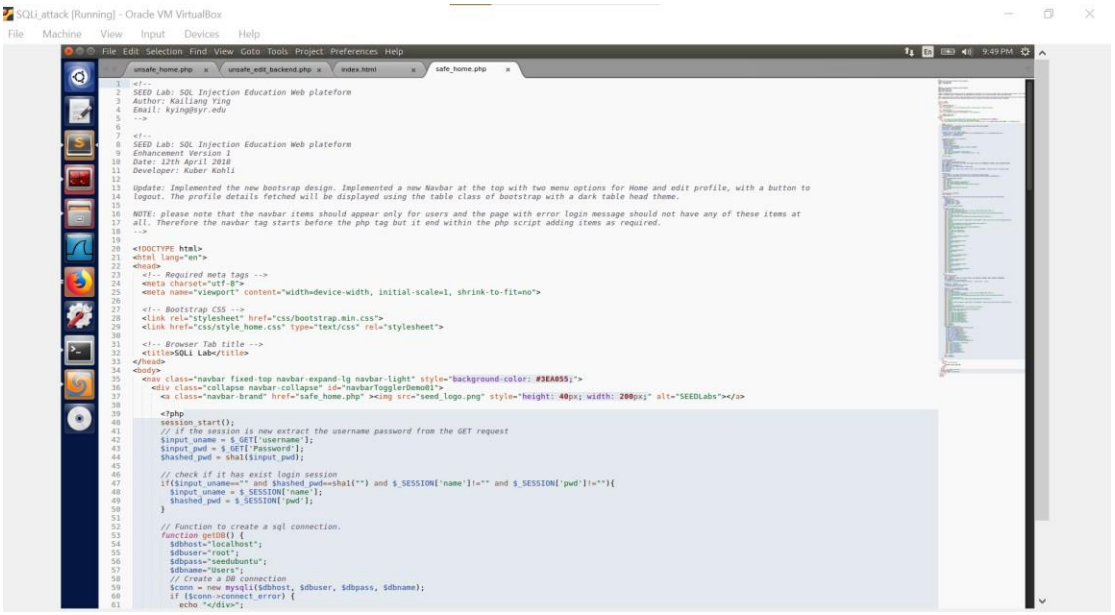
As we see to the link it changes to safe

But when we make an edit profile it shows unsafe.

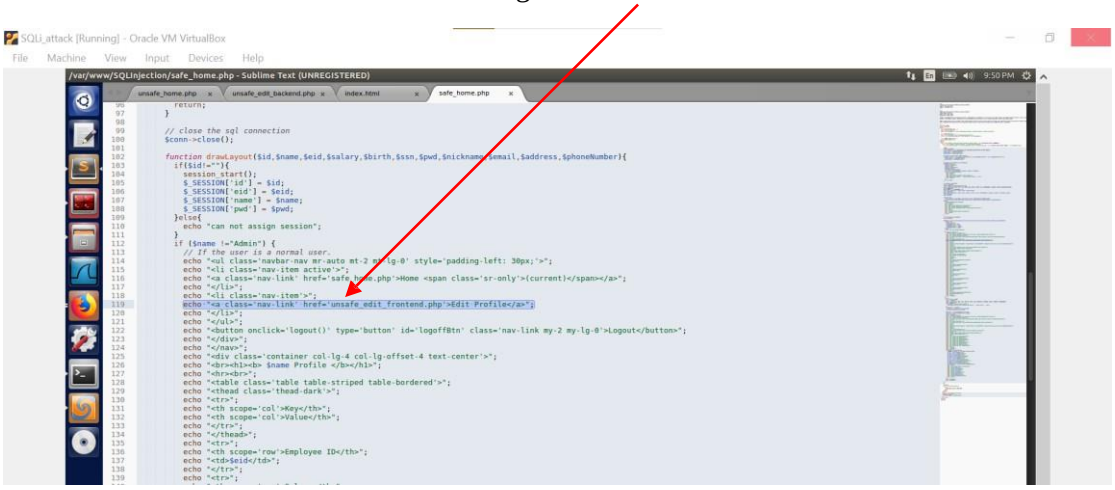


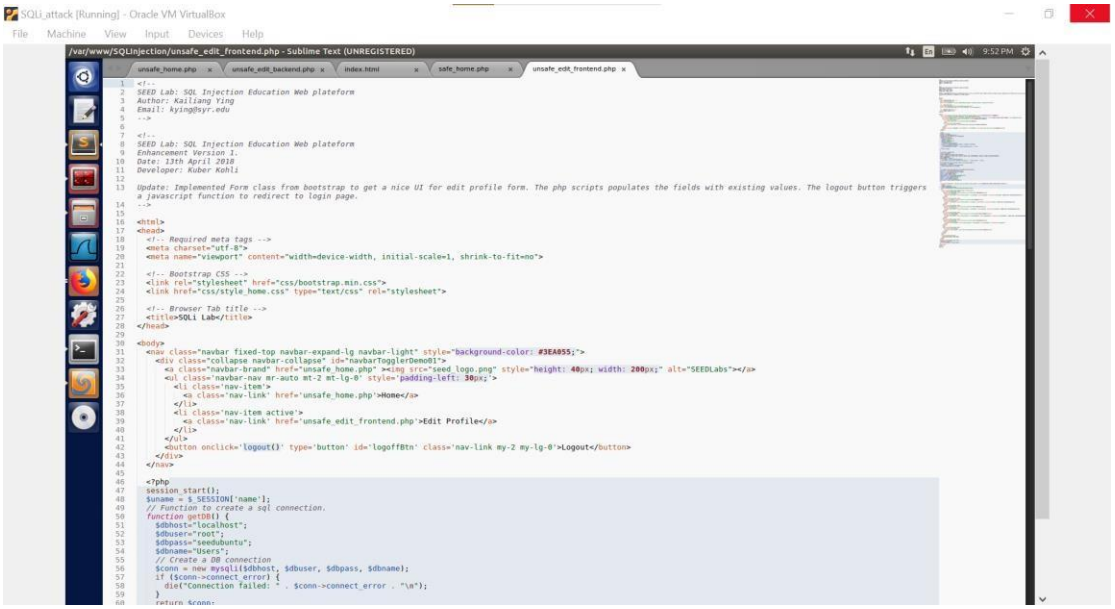
So I will open safe_home.php to check & editing.



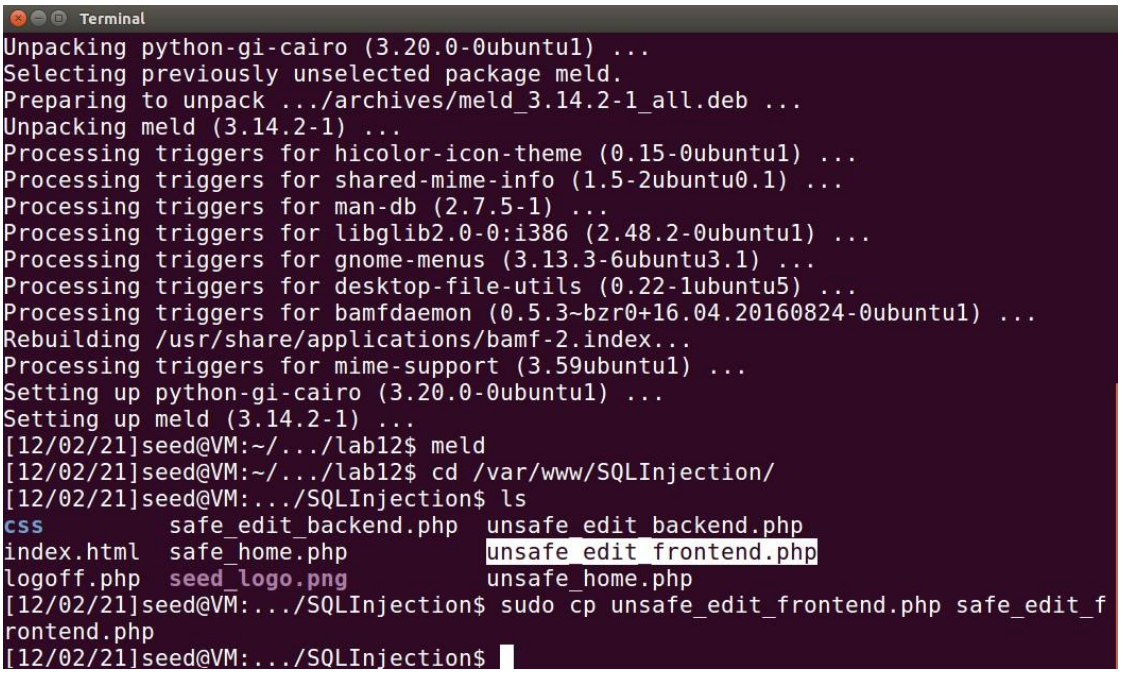


Change it to safe.





Close meld, and I will find that page of editing profile is change to safe_home.php.



5 Summary:

Task 1:

all about being acquainted with SQL commands. already has MySQL installed and a database set up for this experiment. The database is named Users, and it has a credential table.

Task 2:

Task 2.1 I log in as the administrator to the online application so that I can access all of the employees' information. I know the account's username is admin, but I don't know the password. To succeed in the assault, I must determine what to enter in the Username and Password areas.

Task 2.2, I must repeat the SQL injection, but this time I must use the curl command from the terminal rather than the browser.

Task 2.3 I utilize the same SQL injection vulnerability on the login page, but I also need to add another SQL statement that deletes a record from the table.

Task 3:

Task 3.1, Alice is dissatisfied with her pay. She wishes to alter it by taking advantage of the SQL injection attack vulnerability on the Edit Profile page. She is aware that salaries are saved in a column called 'salary.'

Task 3.2 Alice is still dissatisfied, so she uses SQL injection to reduce Ted's (her boss's) pay to $1.

Task 3.3, Alice realizes she isn't finished with Ted yet. She wants to alter his password to something she recognizes so she can access Ted's account.

Tasks 4:

In this task, to use the prepared statement method, modify the source code of both unsafe_home.php and unsafe_edit_backend.php. This will act as a deterrent to SQL injection attacks, keeping PHP code secure.

6 Conclusion:

In conclusion, from my notes I can discretion this lab the SQL injection attacks are among the most common and deadly online application flaws.

Now that I've finished this lab, I understand how SQL injection attacks are used by attackers to read, change, and remove data from an application's database.

I also learnt how to respond against these attacks by using prepared statements.

7 References:

- Secure Computer Networks course lectures / Moodle
- Lecture 27: Web Security: PHP Exploits, SQL
- Injection, and the Slowloris Attack , by Avi Kak / https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture27.pdf
- SQL injection/ https://portswigger.net/web-security/sql-injection
- Top 5 Most Dangerous Injection Attacks/Zbigniew Banach - Fri, 15 May 2020 -/ https://www.netsparker.com/blog/web-security/top-dangerous-injection-attacks/