Lab 3

Computer Security 460

Web Security - SQL Injection

80pts Total (20 Theory + 60 Lab)

Alexander Crespo

(20pts) Theoretical Questions - Answer briefly but succinctly

5pts Describe briefly how SQL Injection works.

A user uses a part of a website where input is used to inject malicous SQL code. They take advantage of the system not using the principle of isolation and modify an input to change how a certain piece of code works. For example they change the password field to modify the SQL query and get access to data from the website.

5pts Which Principle does SQL Injection violates?

The principle that SQL Injection violates is the principle of isolation. You are not supposed to mix code and data and this attack takes advantage of it.

10pts Describe how Prepared Statements protect against SQL Injection

Prepared statements protect against SQL injection because the query is precompiled and the input is only used as data and will not execute any different code. This makes sure to seperate code and data so that an attacker cant execute an SQL injection attack.

Submission Guidlines for the Lab:
You need to submit a succint lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will receive partial credits. There are three main tasks, each task is 20 Credits. For tasks that have more than one questions credits are distributed evenly.

SQL Injection Attack Lab

1 Overview

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before being sent to the back-end database servers.

Many web applications take inputs from users, and then use these inputs to construct SQL queries, so they can get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. SQL injection is one of the most common attacks on web applications.

In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics:

- SQL statements: `SELECT` and `UPDATE` statements
- SQL injection
- Prepared statement

Readings. Detailed coverage of the SQL injection can be found in the following:

- Chapter 12 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.

Lab Environment. This lab has been tested on our pre-built Ubuntu 20.04 VM, which can be downloaded from the SEED website. Since we use containers to set up the lab environment, this lab does not depend much on the SEED VM. You can do this lab using other VMs, physical machines, or VMs on the cloud.

2 Lab Environment

We have developed a web application for this lab, and we use containers to set up this web application. There are two containers in the lab setup, one for hosting the web application, and the other for hosting the database for the web application. The IP address for the web application container is `10.9.0.5`, and The URL for the web application is the following:

```
http://www.seed-server.com
```

We need to map this hostname to the container's IP address. Please add the following entry to the `/etc/hosts` file. You need to use the root privilege to change this file (using `sudo`). It should be noted that this name might have already been added to the file due to some other labs. If it is mapped to a different IP address, the old entry must be removed.

```
10.9.0.5        www.seed-server.com
```

2.1 Container Setup and Commands

Please download the `Labsetup.zip` file to your VM from the lab's website, unzip it, enter the `Labsetup` folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved `Dockerfile` can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "`docker ps`" command to find out the ID of the container, and then use "`docker exec`" to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

MySQL database. Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. For this lab, we do want to keep the data in the MySQL database, so we do not lose our work when we shutdown our container. To achieve this, we have mounted the `mysql_data` folder on the host machine (inside `Labsetup`, it will be created after the MySQL container runs once) to the `/var/lib/mysql` folder inside the MySQL container. This folder is where MySQL stores its database.

Therefore, even if the container is destroyed, data in the database are still kept. If you do want to start from a clean database, you can remove this folder:

```
$ sudo rm -rf mysql_data
```

2.2 About the Web Application

We have created a web application, which is a simple employee management application. Employees can view and update their personal information in the database through this web application. There are mainly two roles in this web application: `Administrator` is a privilege role and can manage each individual employees' profile information; `Employee` is a normal role and can view or update his/her own profile information. All employee information is described in Table 1.

Table 1: Database

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

3 Lab Tasks

3.1 Task 1: Get Familiar with SQL Statements

The objective of this task is to get familiar with SQL commands by playing with the provided database. The data used by our web application is stored in a MySQL database, which is hosted on our MySQL container. We have created a database called `sqllab_users`, which contains a table called `credential`. The table stores the personal information (e.g. eid, password, salary, ssn, etc.) of every employee. In this task, you need to play with the database to get familiar with SQL queries.

Please get a shell on the MySQL container (see the container manual for instruction; the manual is linked to the lab's website). Then use the `mysql` client program to interact with the database. The user name is `root` and password is `dees`.

```
// Inside the MySQL container
# mysql -u root -pdees
```

After login, you can create new database or load an existing one. As we have already created the `sqllab_users` database for you, you just need to load this existing database using the `use` command. To show what tables are there in the `sqllab_users` database, you can use the `show tables` command to print out all the tables of the selected database.

```
mysql> use sqllab_users;
Database changed
mysql> show tables;
+-----------------------+
| Tables_in_sqllab_users |
+-----------------------+
```

```
| credential             |
+------------------------+
```

After running the commands above, you need to use a SQL command to print all the profile information of the employee Alice. Please provide the screenshot of your results.

3.2 Task 2: SQL Injection Attack on SELECT Statement

SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database. Our employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers.

We will use the login page from `www.seed-server.com` for this task. The login page is shown in Figure 1. It asks users to provide a user name and a password. The web application authenticate users based on these two pieces of data, so only employees who know their passwords are allowed to log in. Your job, as an attacker, is to log into the web application without knowing any employee's credential.

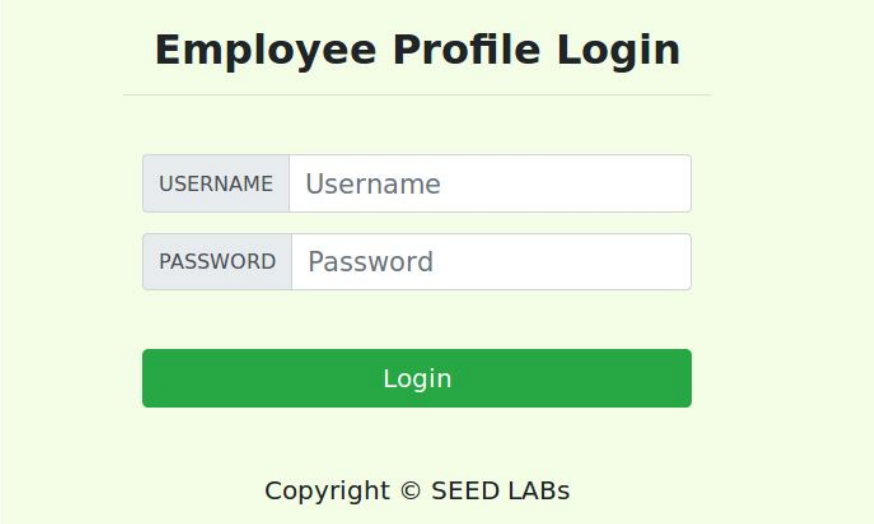


Figure 1: The Login page

To help you started with this task, we explain how authentication is implemented in the web application. The PHP code `unsafe_home.php`, located in the `/var/www/SQL_Injection` directory, is used to conduct user authentication. The following code snippet show how users are authenticated.

```
$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) {
  if(name=='admin') {
     return All employees information;
  } else if (name !=NULL){
    return employee information;
  }
} else {
  Authentication Fails;
}
```

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the `credential` table. The SQL statement uses two variables `input_uname` and `hashed_pwd`, where `input_uname` holds the string typed by users in the username field of the login page, while `hashed_pwd` holds the `sha1` hash of the password typed by the user. The program checks whether any record matches with the provided username and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

Task 2.1: SQL Injection Attack from webpage. Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is `admin`, but you do not the password. You need to decide what to type in the `Username` and `Password` fields to succeed in the attack.

Task 2.2: SQL Injection Attack from command line. Your task is to repeat Task 2.1, but you need to do it without using the webpage. You can use command line tools, such as `curl`, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as `&`) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (`username` and `Password`) attached:

```
$ curl 'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
```

If you need to include special characters in the `username` or `Password` fields, you need to encode them properly, or they can change the meaning of your requests. If you want to include single quote in those fields, you should use `%27` instead; if you want to include white space, you should use `%20`. In this task, you do need to handle HTTP encoding while sending requests using `curl`.

Task 2.3: Append a new SQL statement. In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Please try to run two SQL statements via the login page.

There is a countermeasure preventing you from running two SQL statements in this attack. Please use the SEED book or resources from the Internet to figure out what this countermeasure is, and describe your discovery in the lab report.

3.3 Task 3: SQL Injection Attack on UPDATE Statement

If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify databases. In our Employee Management application, there is an Edit Profile page (Figure 2) that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to log in first.

When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed. The PHP code implemented in `unsafe_edit_backend.php` file is used to update employee's profile information. The PHP file is located in the `/var/www/SQLInjection` directory.

```
$hashed_pwd = sha1($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);
```

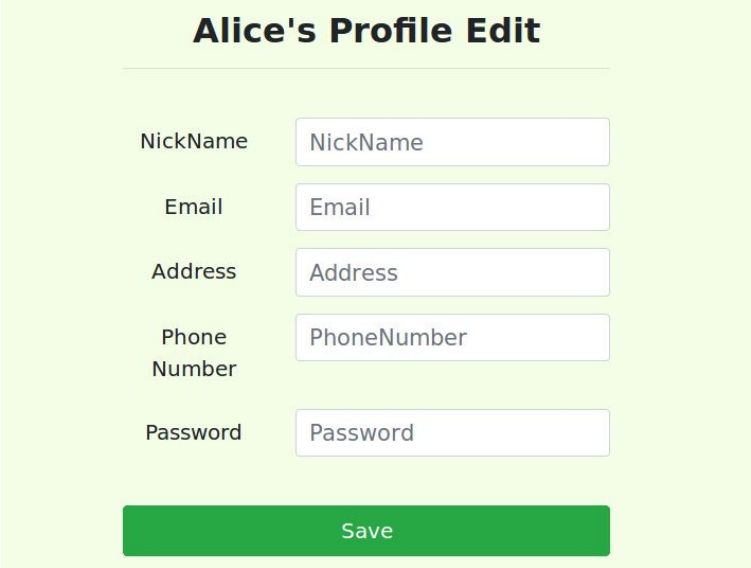


Figure 2: The Edit-Profile page

Task 3.1: Modify your own salary. As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Assume that you (Alice) are a disgruntled employee, and your boss Boby did not increase your salary this year. You want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page. Please demonstrate how you can achieve that. We assume that you do know that salaries are stored in a column called `salary`.

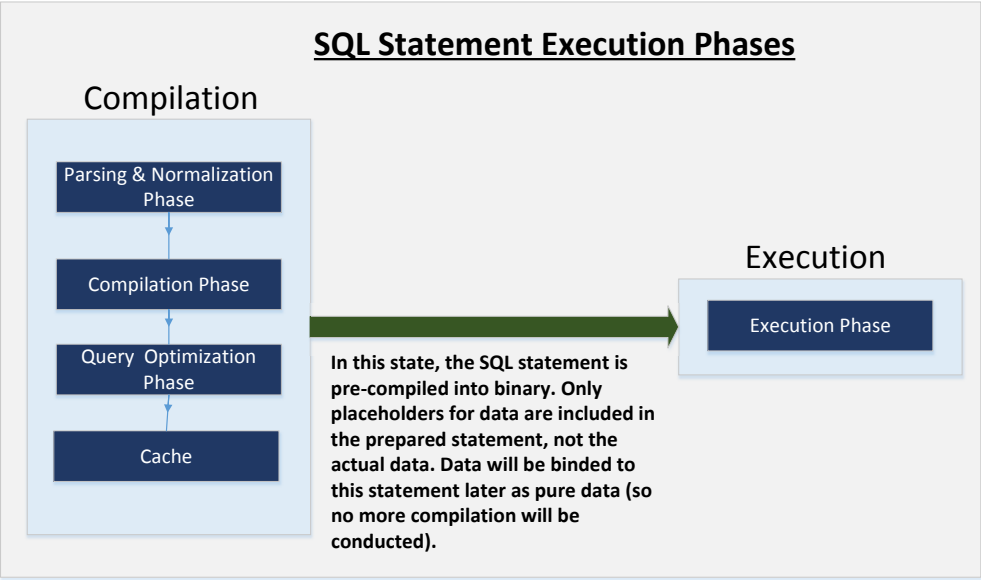


Figure 3: Prepared Statement Workflow

Task 3.2: Modify other people' salary. After increasing your own salary, you decide to punish your boss Boby. You want to reduce his salary to 1 dollar. Please demonstrate how you can achieve that.

Task 3.3: Modify other people' password. After changing Boby's salary, you are still disgruntled, so you want to change Boby's password to something that you know, and then you can log into his account and do further damage. Please demonstrate how you can achieve that. You need to demonstrate that you can successfully log into Boby's account using the new password. One thing worth mentioning here is that the database stores the hash value of passwords instead of the plaintext password string. You can again look at the `unsafe_edit_backend.php` code to see how password is being stored. It uses SHA1 hash function to generate the hash value of password.

SQL Injection Attack Lab

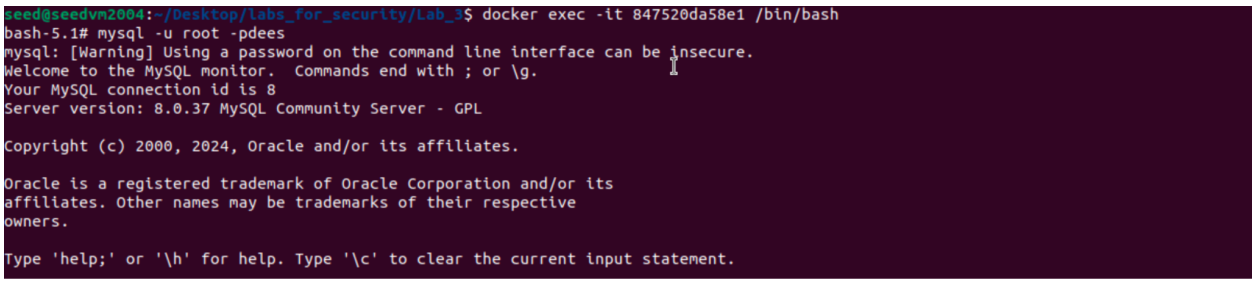
2.1

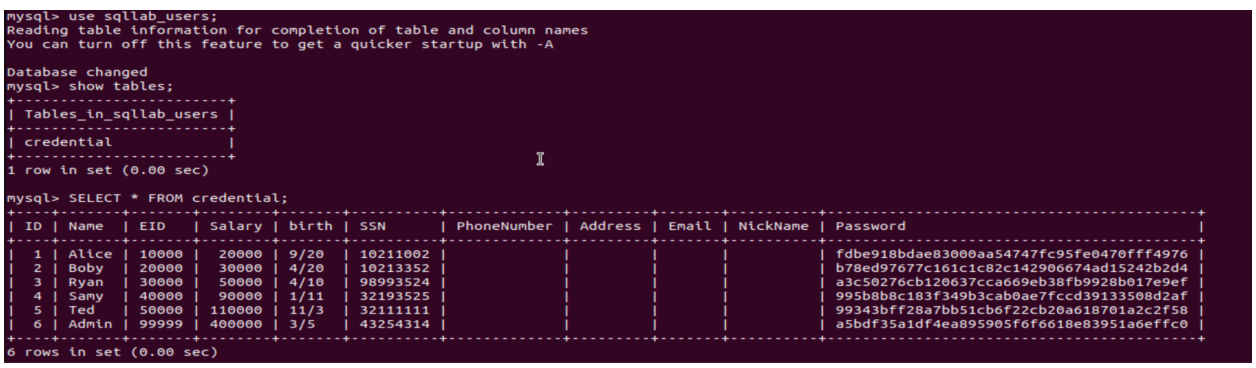
Worked as expected. I was able to run and set up new containers for this lab. Server is on stand by.

2.2

This task was just talking about the web application. I read it and got an understanding on the web application itself.

3.1





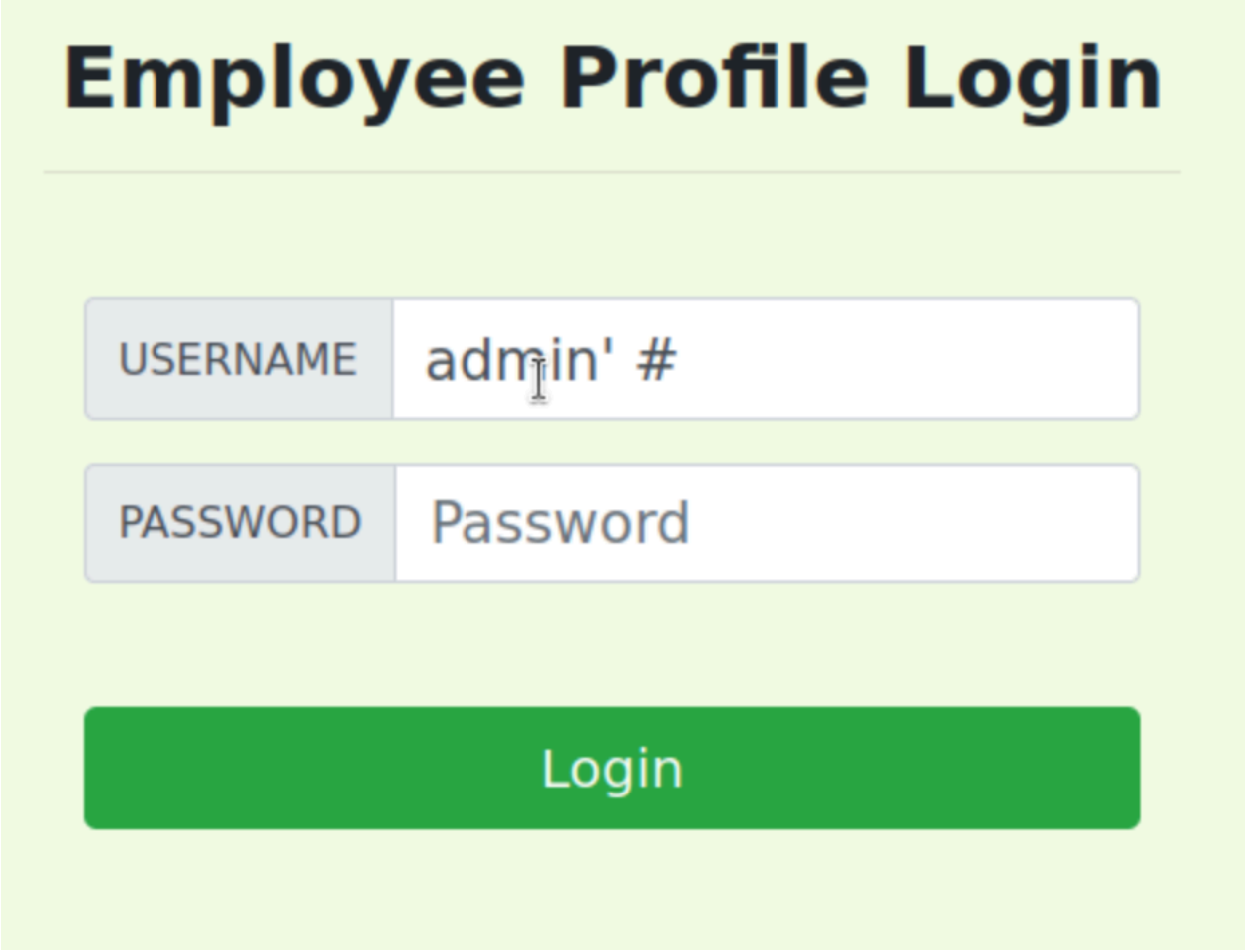
```
mysql> SELECT * FROM credential WHERE Name = "ALice";
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
```

Everything worked as expected. I ran the commands and had no issues. I was able to get the information I needed from the database and show all records using the select statement.

3.2

2.1

I wasn't sure what to expect for this. I only expected it to login once I was able to figure out what to put. I decided to try and comment out the password part which worked as expected due to the lack of input sanitization.

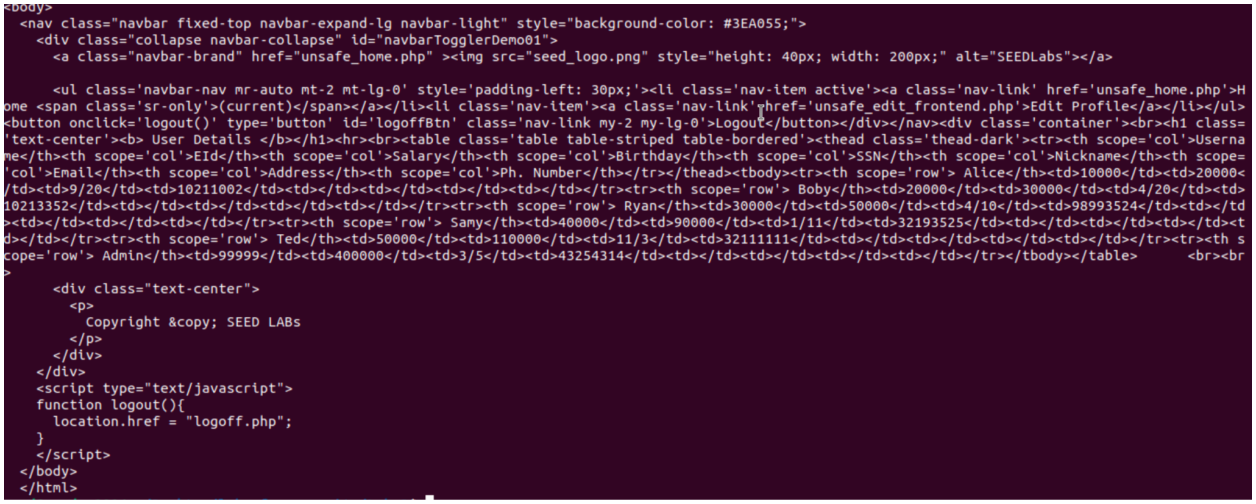


User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
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				

2.2

```
seed@seedvm2004:~/Desktop/Labs_for_security/Lab_3$ curl 'www.seed-server.com/unsafe_home.php?username=Admin%27%20%23'
<!--
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 bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- 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>
```



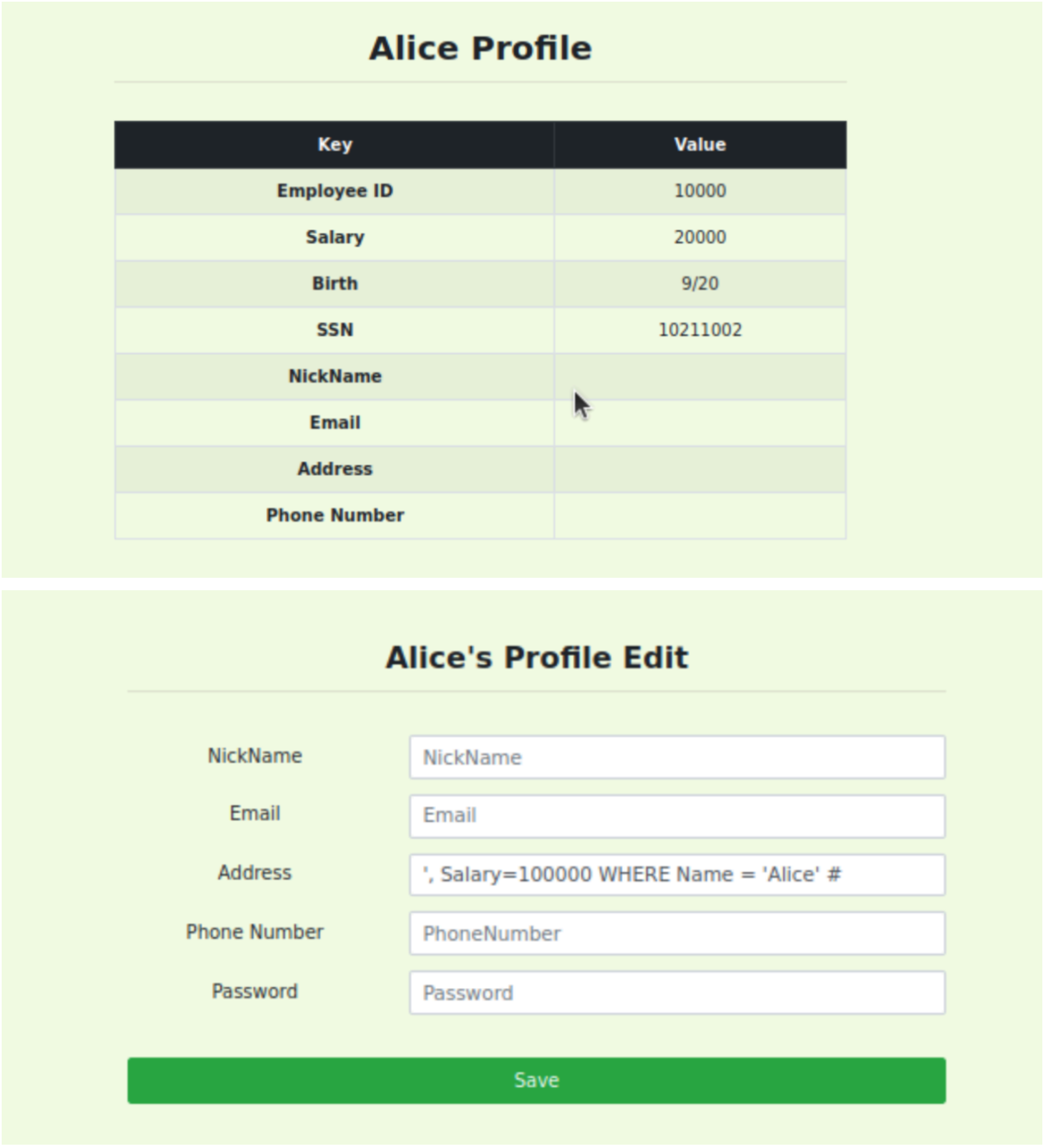
It worked as expected using the command line. Once I entered the curl command it executed and brought it up on my terminal. The only thing I had to figure out was that when copy and pasting the ‘ was different. So I entered it manually and it worked perfectly.

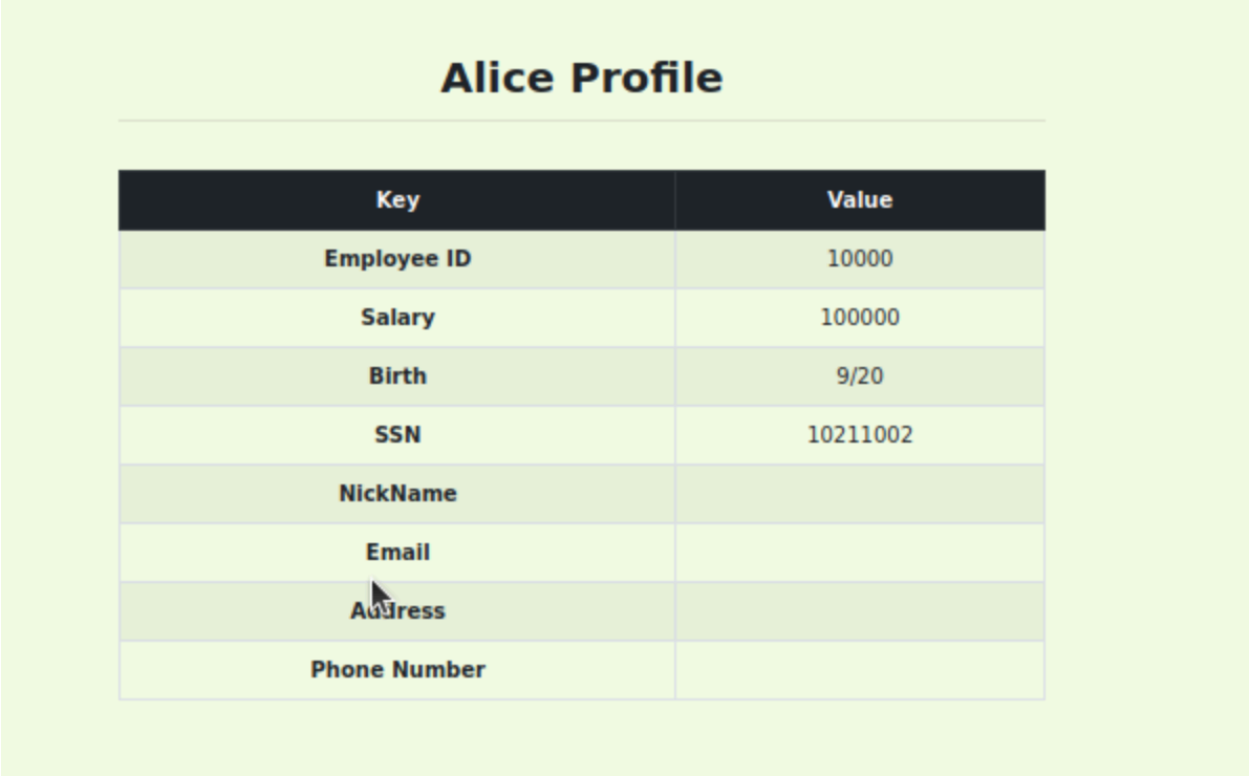
2.3

As discussed in this part it did not work. I tried dropping a table and it was stopped due to the multi statement prevention. This means that the security feature was working as intended.

3.3

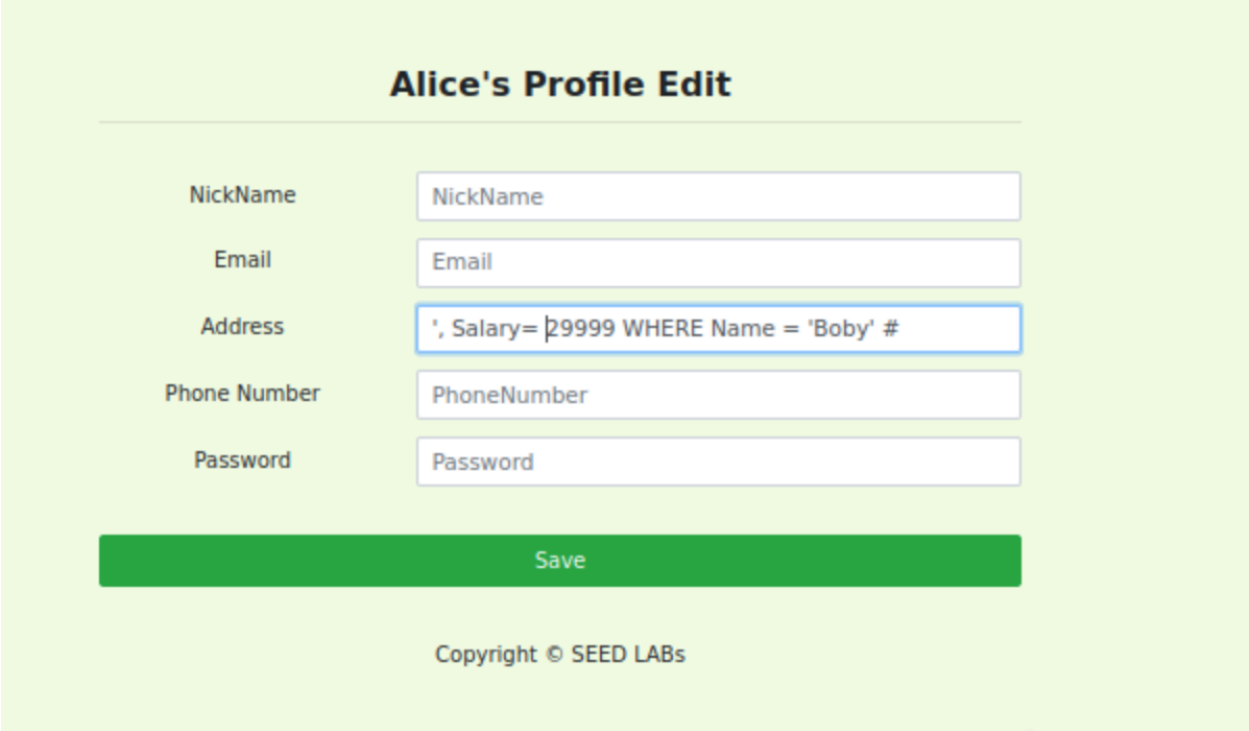
3.1

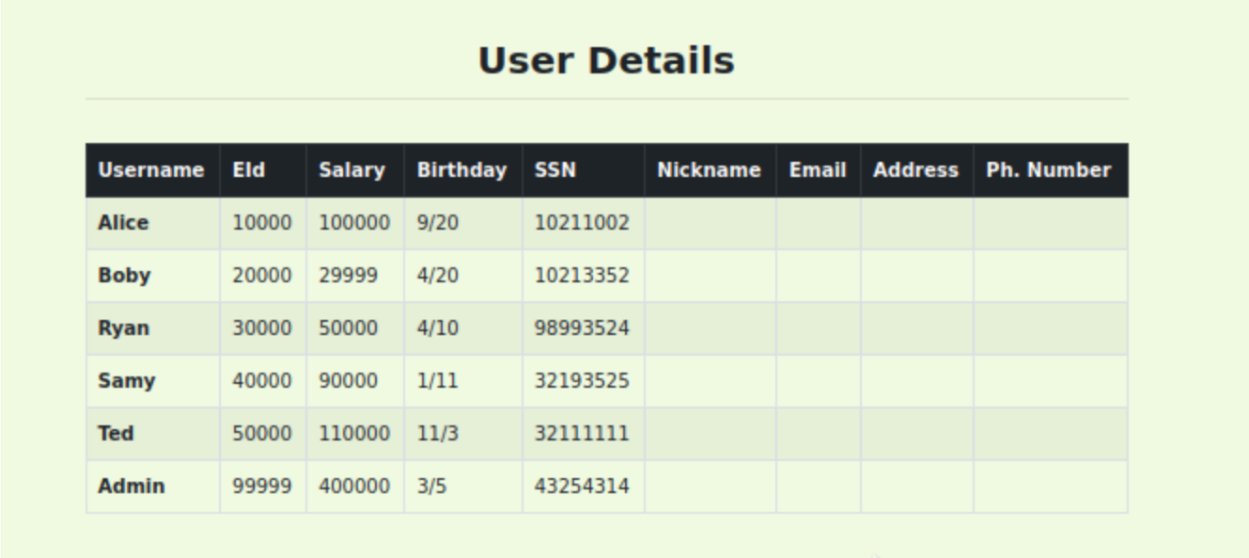

Alice Profile
Key
Value
Employee ID
10000
Salary
20000
Birth
9/20
SSN
10211002
NickName
Email
Address
Phone Number
Alice's Profile Edit
NickName
NickName
Email
Email
Address
', Salary=100000 WHERE Name = 'Alice' #
Phone Number
PhoneNumber
Password
Password
Save



I was able to figure out that using different parts in the edit profile worked. I entered ‘,Salary = 10000 WHERE Name = ‘Alice’ # in the update address and it worked. I tried it with nick name and email as well which worked.

3.2





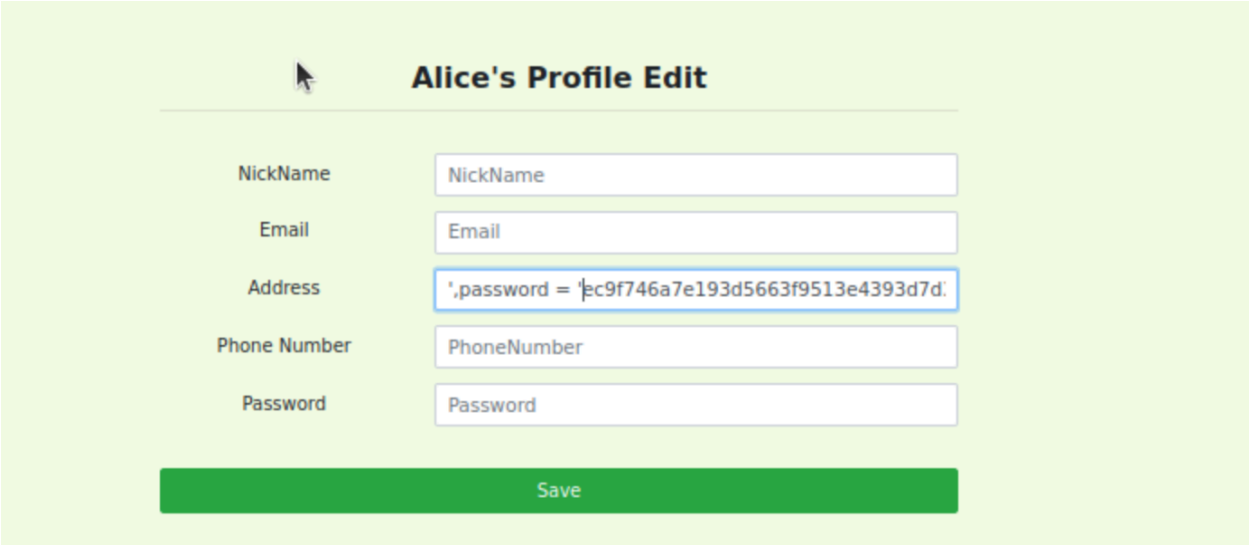
User Details

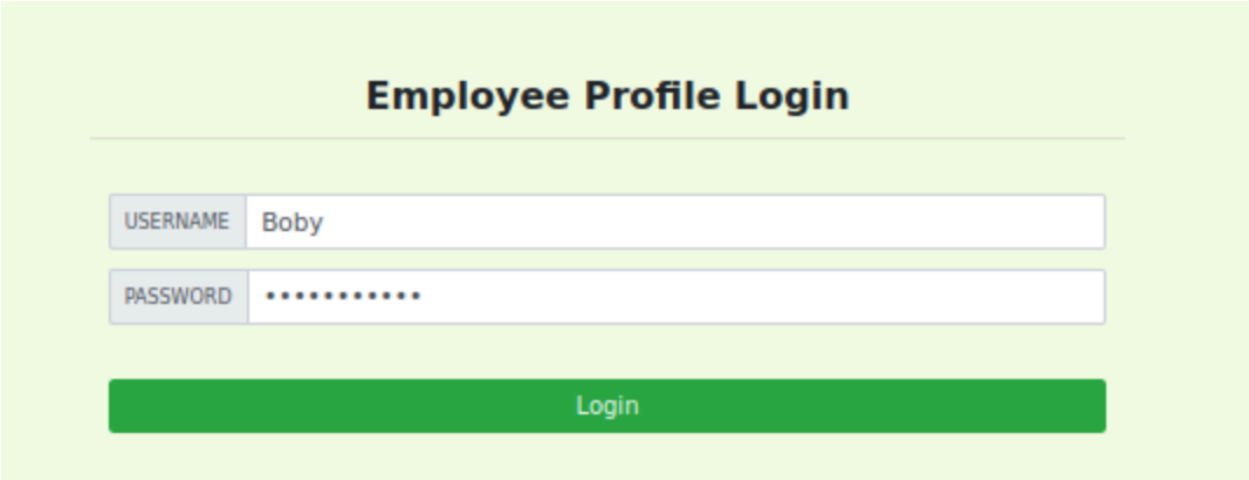
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	100000	9/20	10211002				
Boby	20000	29999	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

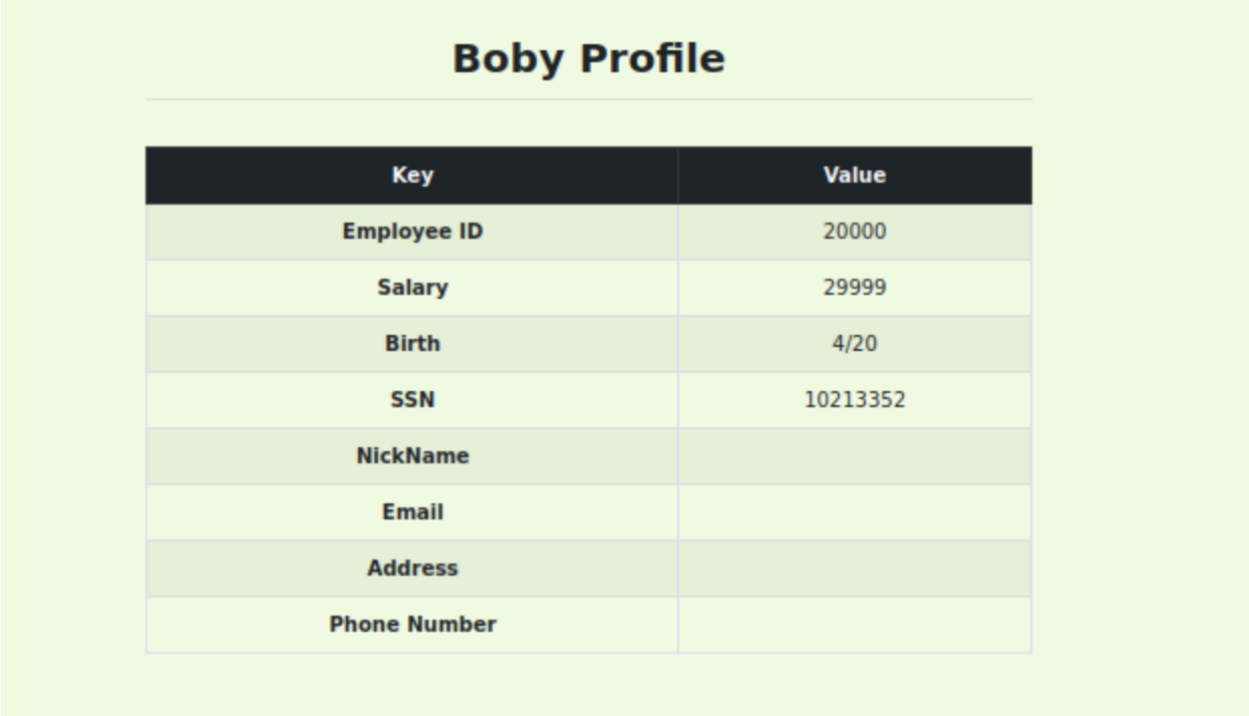
In order to change my boss's its easy. I just need to figure out what his salary is which is in the look up table for admin that we already got earlier and then run the same command in one of the parts in my edit profile picture just changing the name to Boby. It worked as expected and decreased my boss's salary by 1 :).

3.3

```
seed@seedvm2004:~/Desktop/labs_for_security/Lab_3$ echo -n "alicegotyou" | sha1sum
ec9f746a7e193d5663f9513e4393d7d3bb13720c  -
```







So the passwords are saved as hashes when editing them. So in order to do this I needed to find out what the password would be first after hashing. I used the command line for this and it worked as expected. I got the password hashed, and then used this hashed password to update the profile. I used the command ‘,password = ‘(hashed password)’ WHERE Name = ‘Boby’ #. Everything worked as expect and “Alice” was able to get into the bosses profile to do some extra damage.