SQL Injection Attack Lab Report

Mudit Vats
mpvats@syr.edu
11/16/2018

Table of Contents

Overview

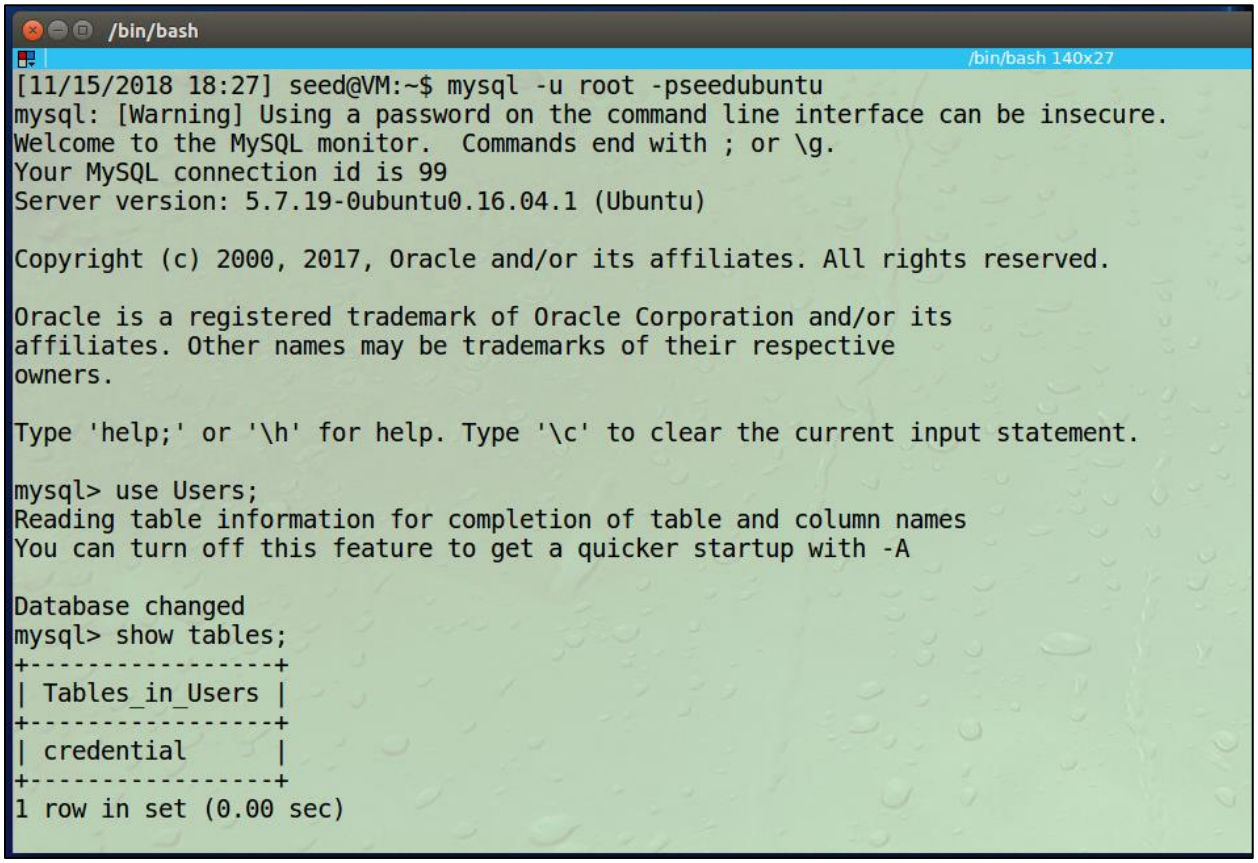
This lab report presents observations and explanations for the tasks described in the SQL Injection Attack Lab.

Task 1: Get Familiar with SQL Statements

Goal: Get Familiar with SQL Statements.

In the figure below we can see –

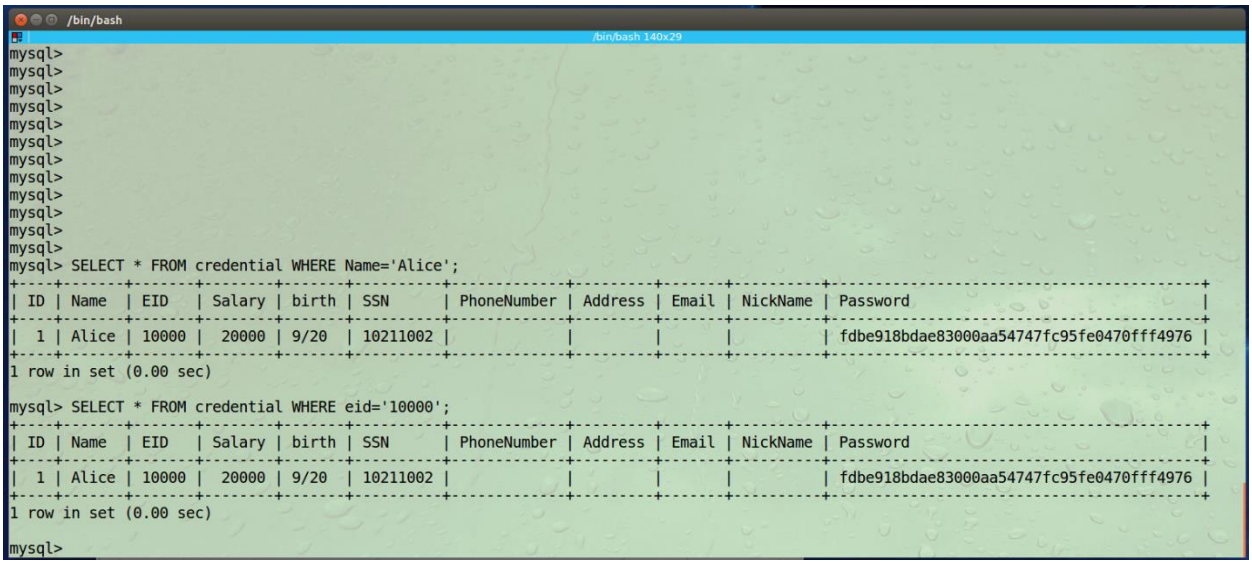
1. Logging into the MySQL client.
2. Using the command "use Users;". This command allows us to use the "Users" database.
3. Using the command "show tables;". This command shows the tables within the Users database. In this case, there is one table: "credential".



In the figure below, we see the query to get all the info for the employee "Alice". I present two ways. First we query the credential database where the name is "Alice". That is the first query. We can see the item data ID, Name, EID, Salary etc. below.

Next, we execute the same query but we change the where clause to look for employee ID (EID) that is equal to 10000, which is Alice's EID. In this case, as well, we get the same item data.

I present both way since the task specifically asked for item information on "Alice" the name. But, Alice may not be unique. In this lab, "Alice" is unique, but that is not always the case. In a large company, there may be many Alices. The EID, however, is unique, so if this is known, this would be the better way to get Alice's information.



Observations / Explanation

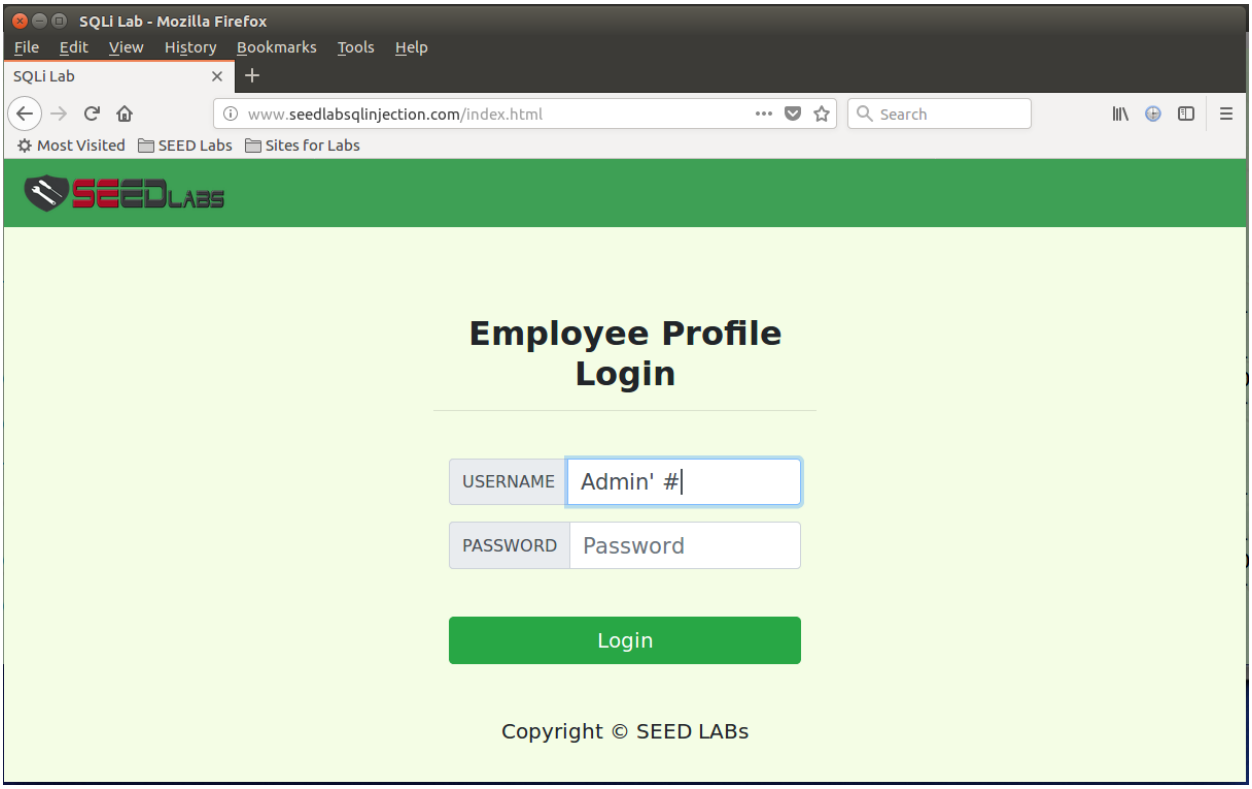
By logging into the MySQL client we can execute queries on the table data. In this case, we executed queries to get information on "Alice". We successfully executed queries using the name and EID as the criteria for which data to pull. We also used the Users database to find out which tables exists and then used the only table "credential" for our queries.

Task 2: SQL Injection Attack on SELECT Statement

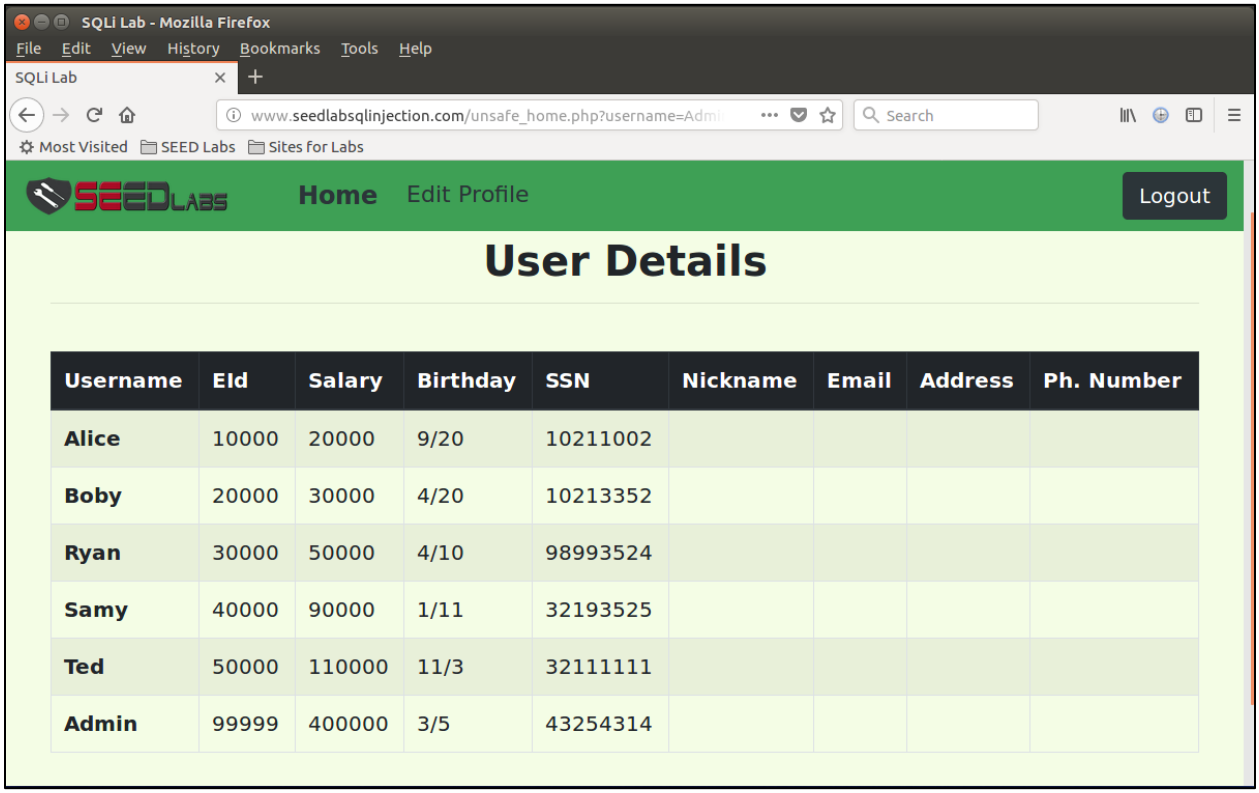
2.1 SQL Injection Attack from webpage

Goal: Login knowing the Username, but while not knowing the password.

In the figure below we can see a login attempt using the username "Admin; #" and no password specified.



In the figure below, we see the successful login.



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				

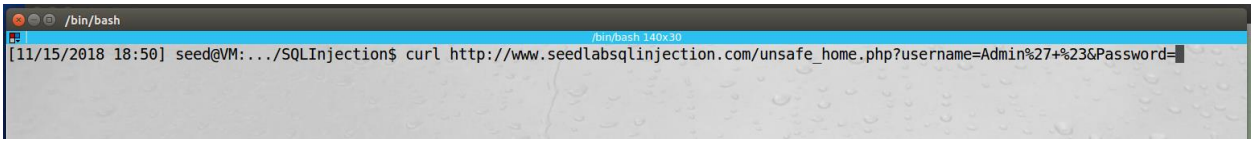
Observations / Explanation

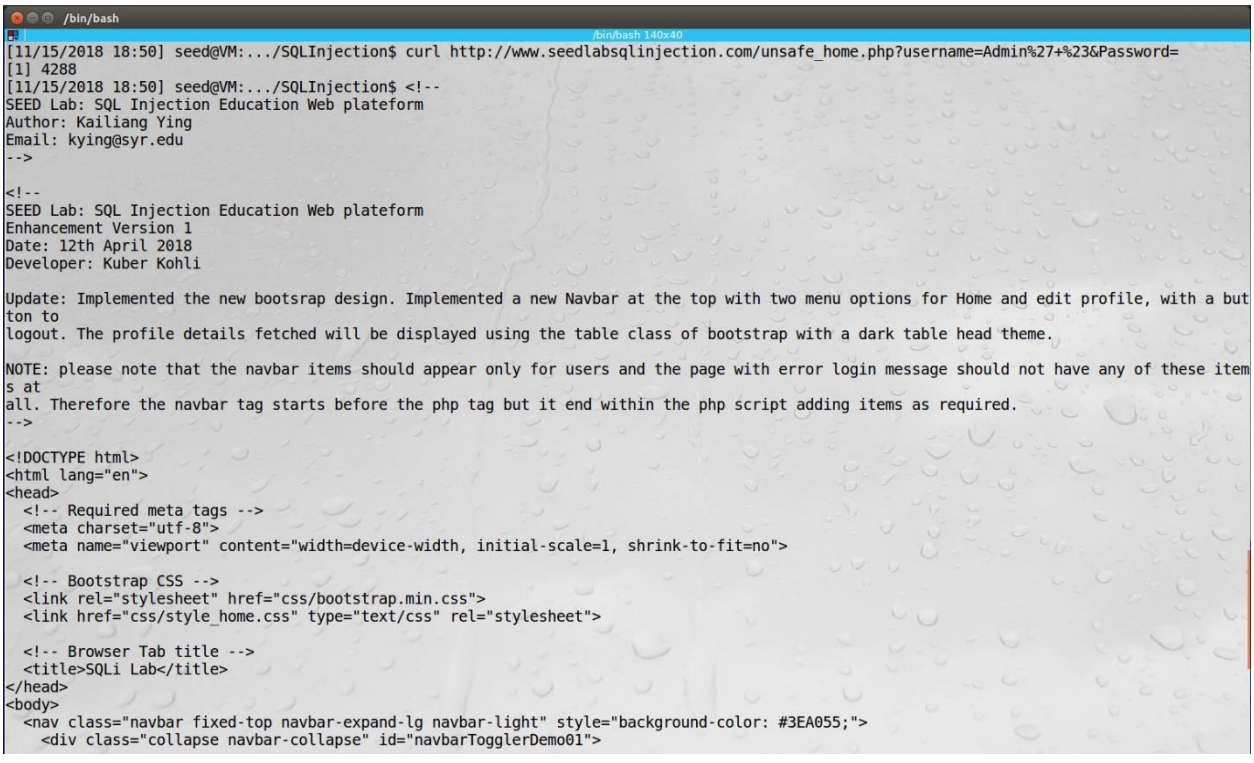
By using "Admin' #" we were able to successfully login to the web site. This works because "Admin'" completes the SQL query statement used in the WHERE part for the $input_username. The pound symbol "#" is then used to comment-out the rest of the query which includes the password part. By doing this, the query executes without a password constraint thus allowing only the username to be used for the query. This then allows the successful retrieval of the table's records.

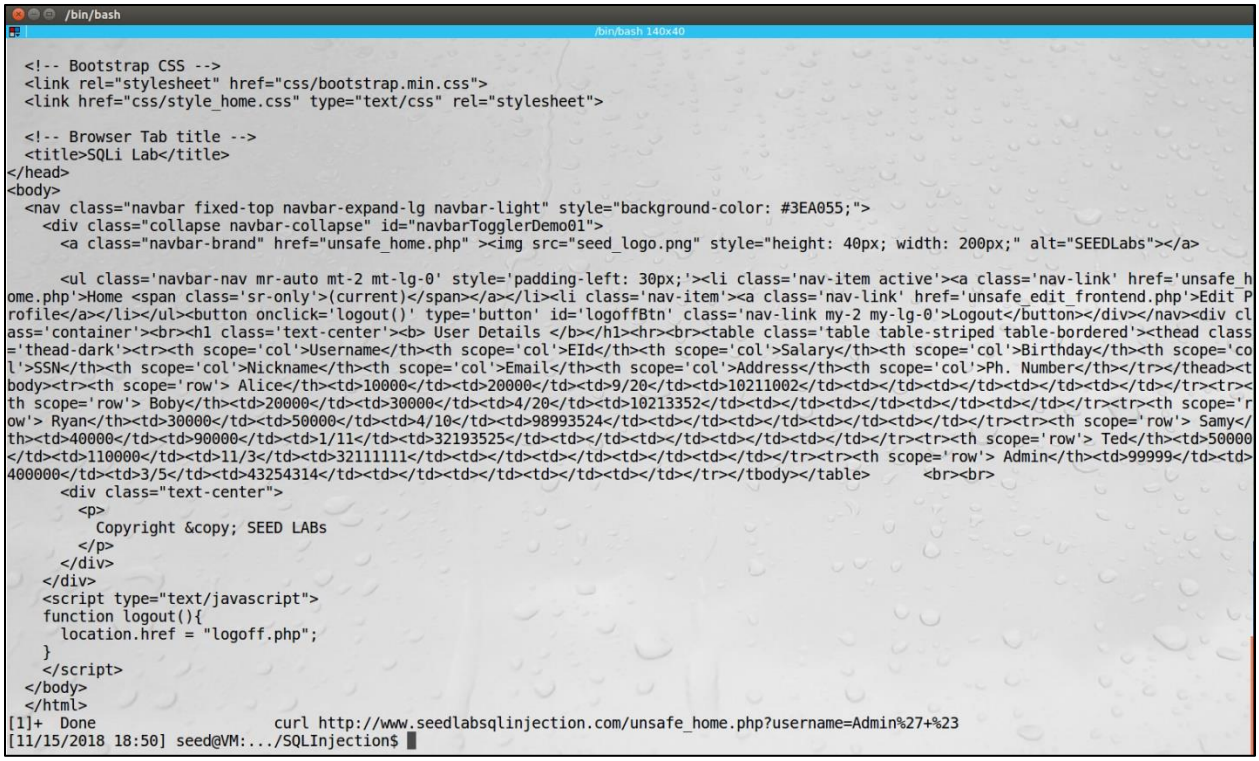
2.2 SQL Injection Attack from command-line

Goal: Goal: Login knowing the Username, but while not knowing the password. This time, use command-line.

In the figure below we can see –







Observations / Explanation

This is the same explanation as 2.1. The only addition is that since we are specifying the query via CURL we format characters such as the single quote and pound symbol with their hexadecimal representations using the percent sign so that the http URL will be interpreted correctly.

2.3 Append a new SQL statement

Goal: Get Familiar with SQL Statements.

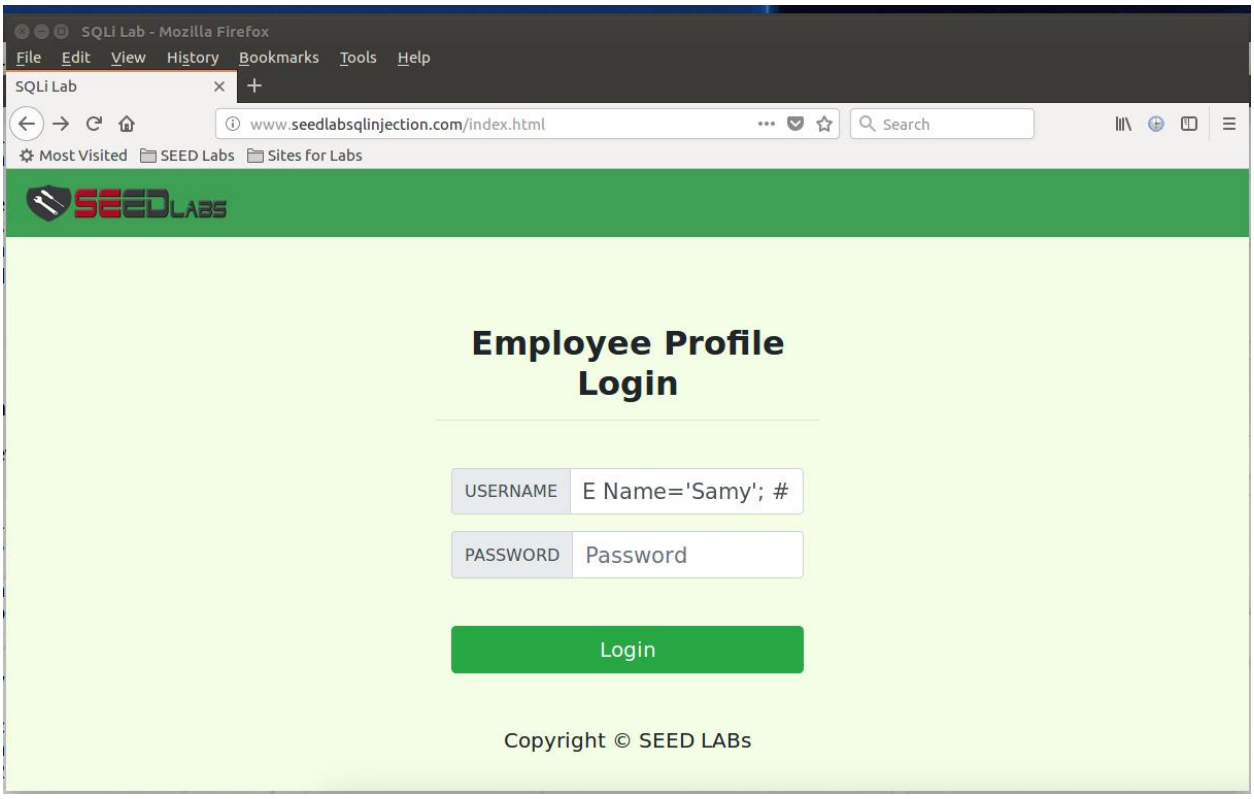
For this exercise, I'm using the multiple SQL command below:

- Admin'; DELETE FROM credential WHERE Name='Samy'; #

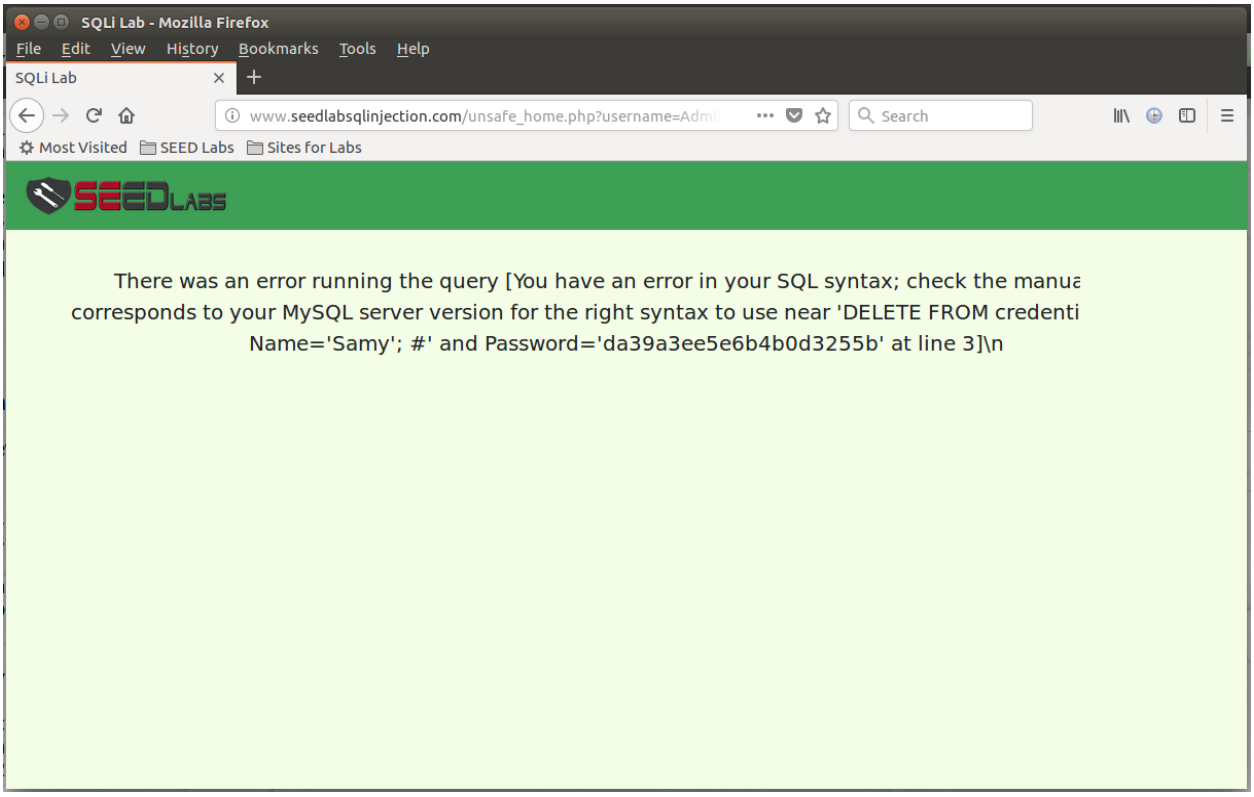
The parts are:

1. "Admin';" – This is the first part which finishes the '$input_uname' in the WHERE clause of the authentication query.
2. "DELETE FROM credential WHERE Name='Samy';" – This part is the DELETE SQL statement where we are deleting the row (single record) where "Samy" is the name.
3. "#" - This is the last part which lets the rest of what in the authentication code's SQL statement become a comment. This includes the password part which is now commented out.

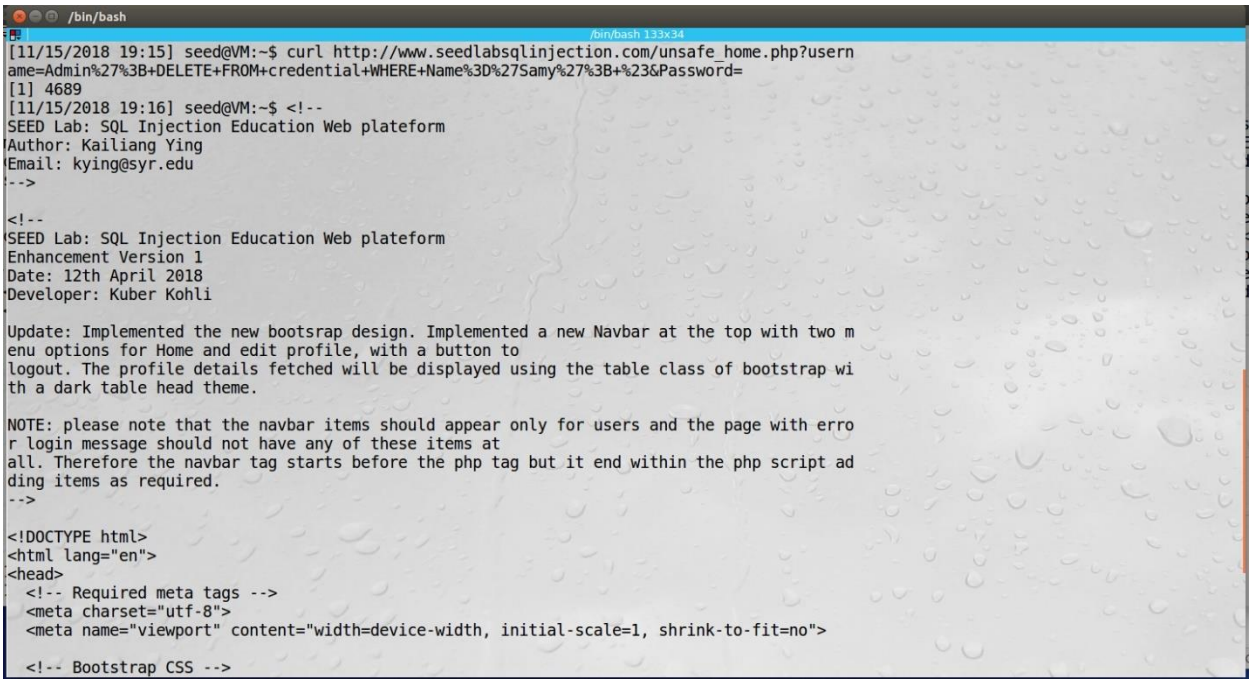
In the figure below we can see the query being placed into the USERNAME field. The screen show below this one show the results.

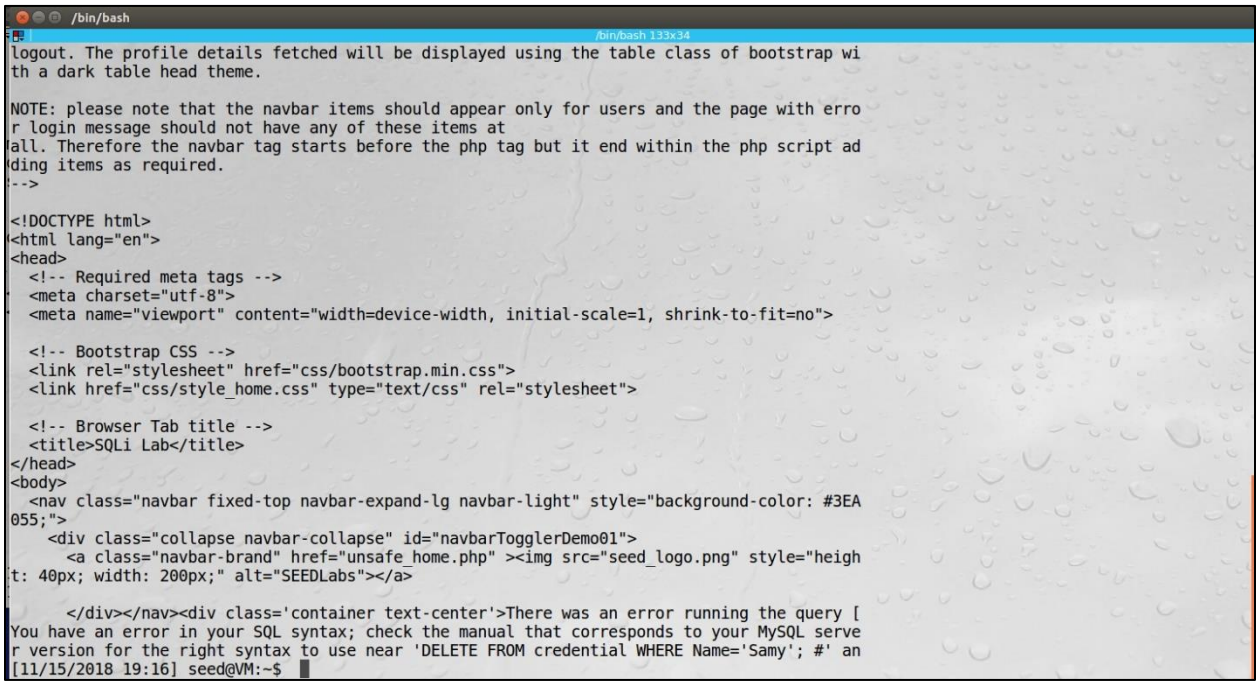


The figure below shows the results of the query operation.



The figures below attempt the same query as above, but in the form the CURL command can utilize. The figure below shows the CURL execution of the query on the website (first line). The rest of the figure and the follow-on figure show the results of the query.





Observations / Explanation

Whether running the multi statement query in the website or the command-line, we see the error message similar to: "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 'DELETE FROM credential WHERE Name='Samy'; #' and Password='da39a3ee5e6b4b0d3255b' at line 3]\n".

As the class textbook (Computer Security: A Hands-on Approach, Wenliang Du) states on page 194, chapter 11 = "Such an attack does not work against MySQL, because in PHP's mysqli extension, the mysqli::query() API does not allow multiple queries to run in the database server. This is due to the concern of SQL injection".

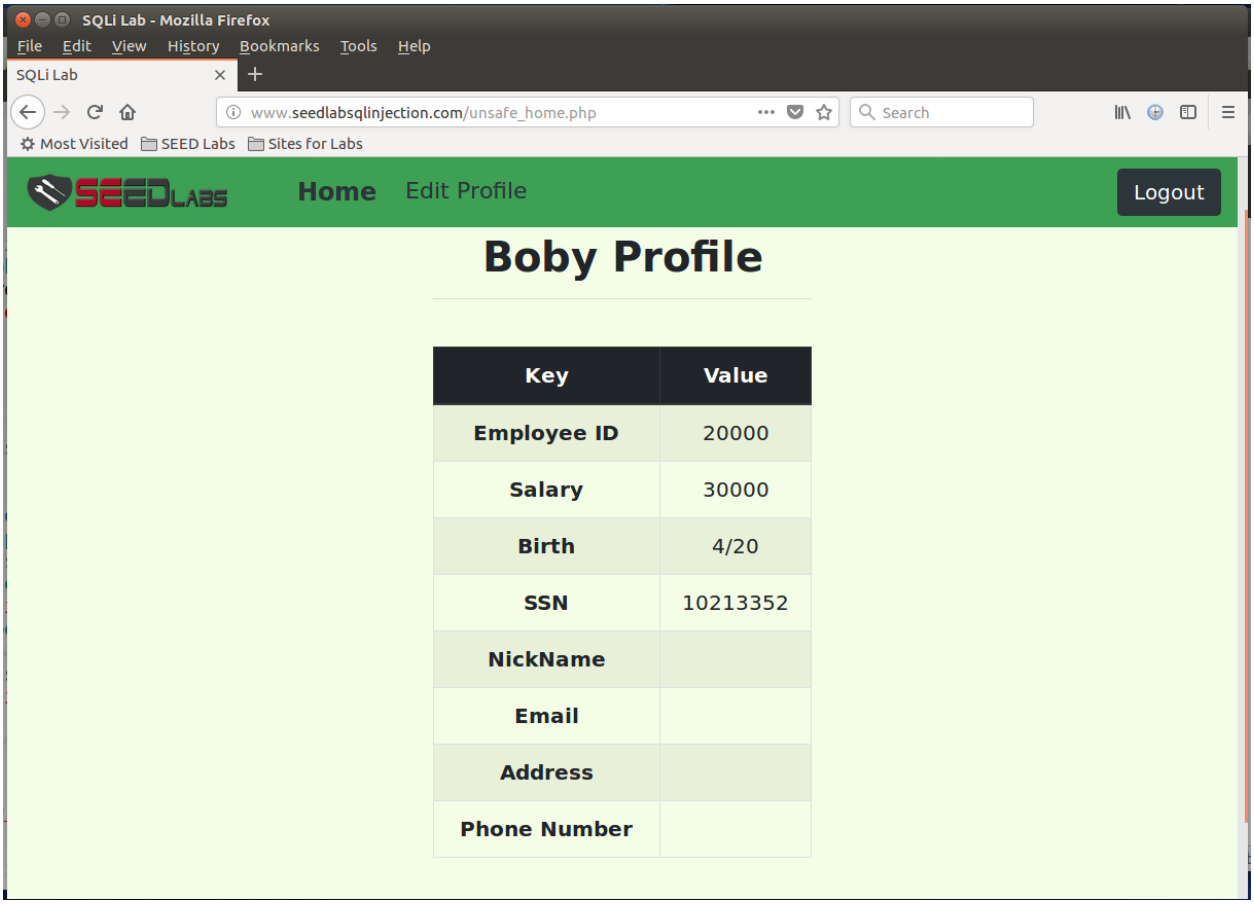
This definitely seems to be the case here since the multiple queries cannot be executed via the website or CURL command-line.

Task 3: SQL Injection Attack on UPDATE Statement

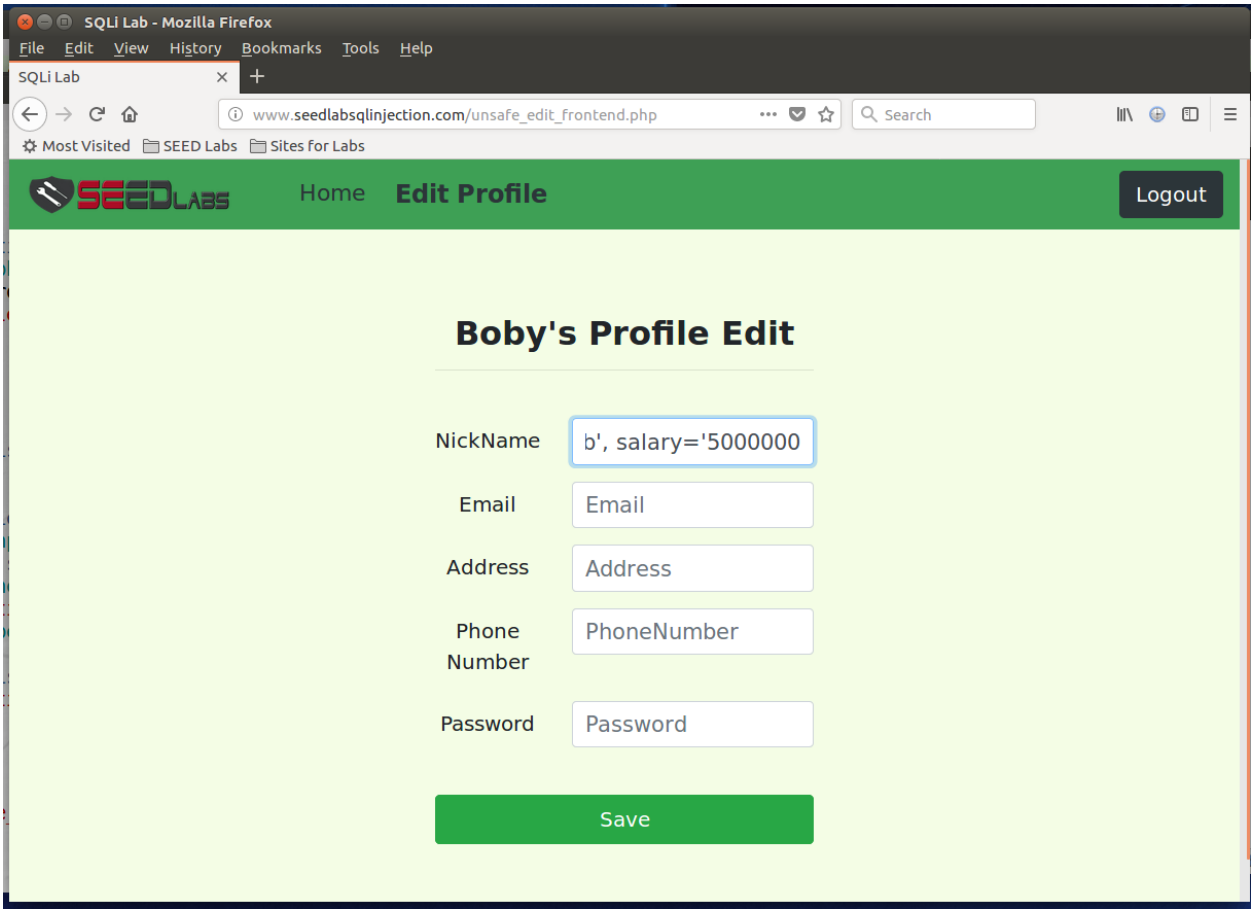
3.1 Modify your own salary

Goal: Using the Edit Profile page, update your salary! Please note, for this exercise, I used Boby as the disgruntled employee instead of Alice. The main reason is that I deleted Alice's record from the table when experimenting with Task 2.3.

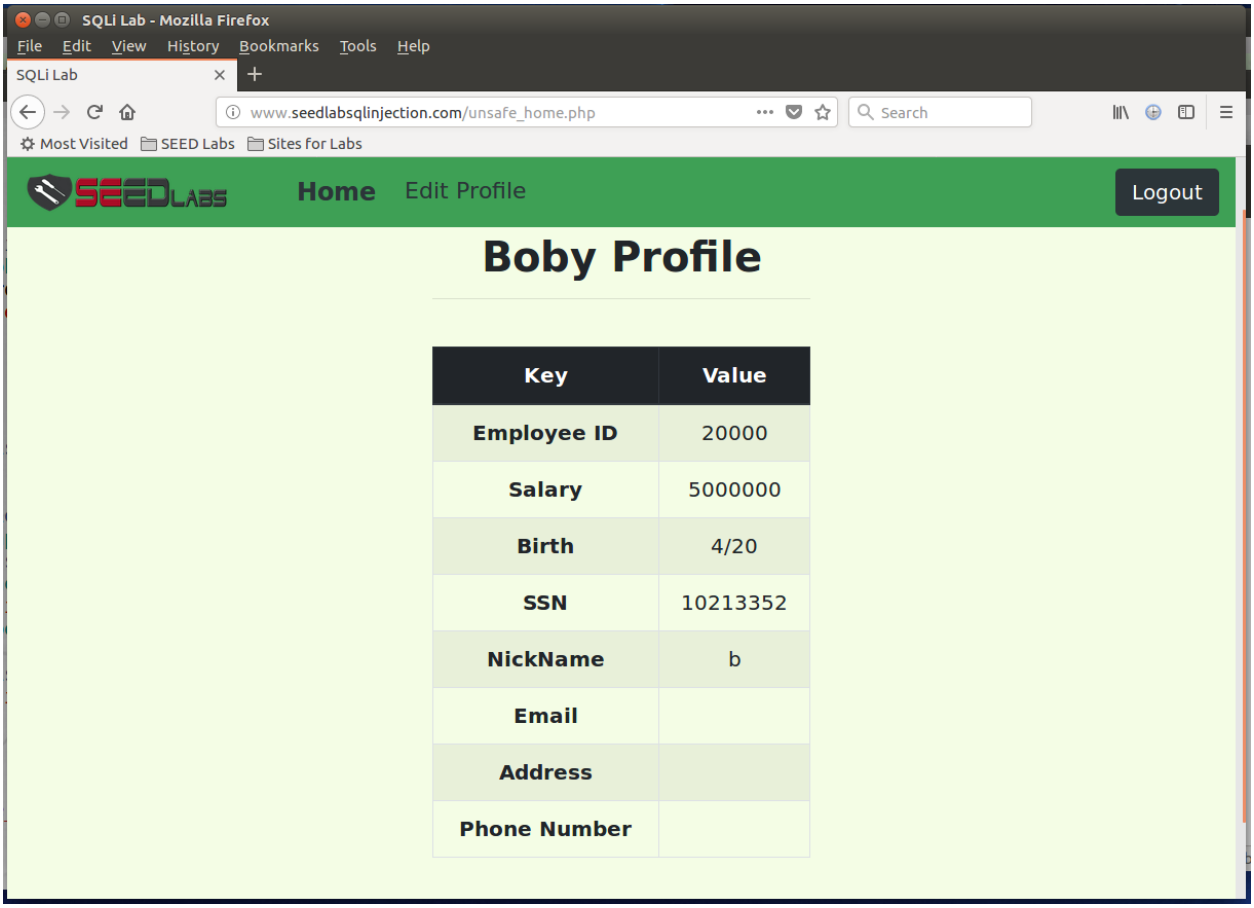
In the figure below we can see Boby's profile. Please note the 30000 salary.



In the figure below, we see the Nickname being specified as "b', salary='5000000".



In the figure below, we see that Boby's salary has been updated to $5000000! Yay! 😊



Observations / Explanation

By putting the text below into the Nickname field, we were able Boby's salary.

- "b', salary='5000000"

The parts of this text is explained below –

1. "b'" – This text completes the Nickname text. Boby's new nickname becomes "B". Like "Hi, B!".
2. "salary='5000000" – This text adds the salary field and value to the UPDATE parameter. Note, we do not need to close the single quote before the 5000000 because this text is inserted before the single quote at the end of the original Nickname field.

To visualize, the text input changes the SQL in the unsafe_edit_backend.php file (insertion in yellow highlight):

From this:

$sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";

To this:

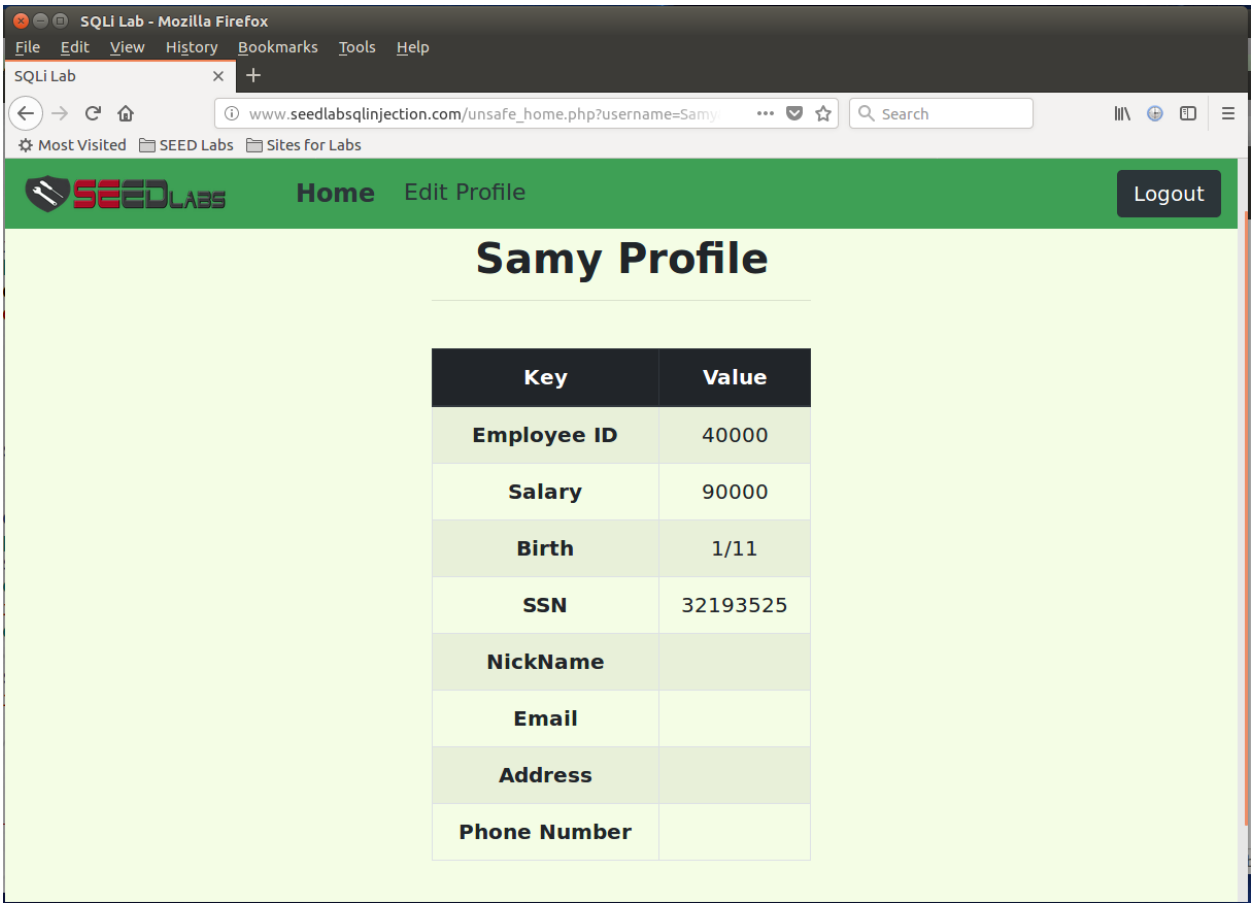
$sql = "UPDATE credential SET nickname='$input_nickname b', salary='5000000',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";

The end result is that we were able to successfully update Boby's salary, while logged in as Boby!

3.2 Modify other people' salary

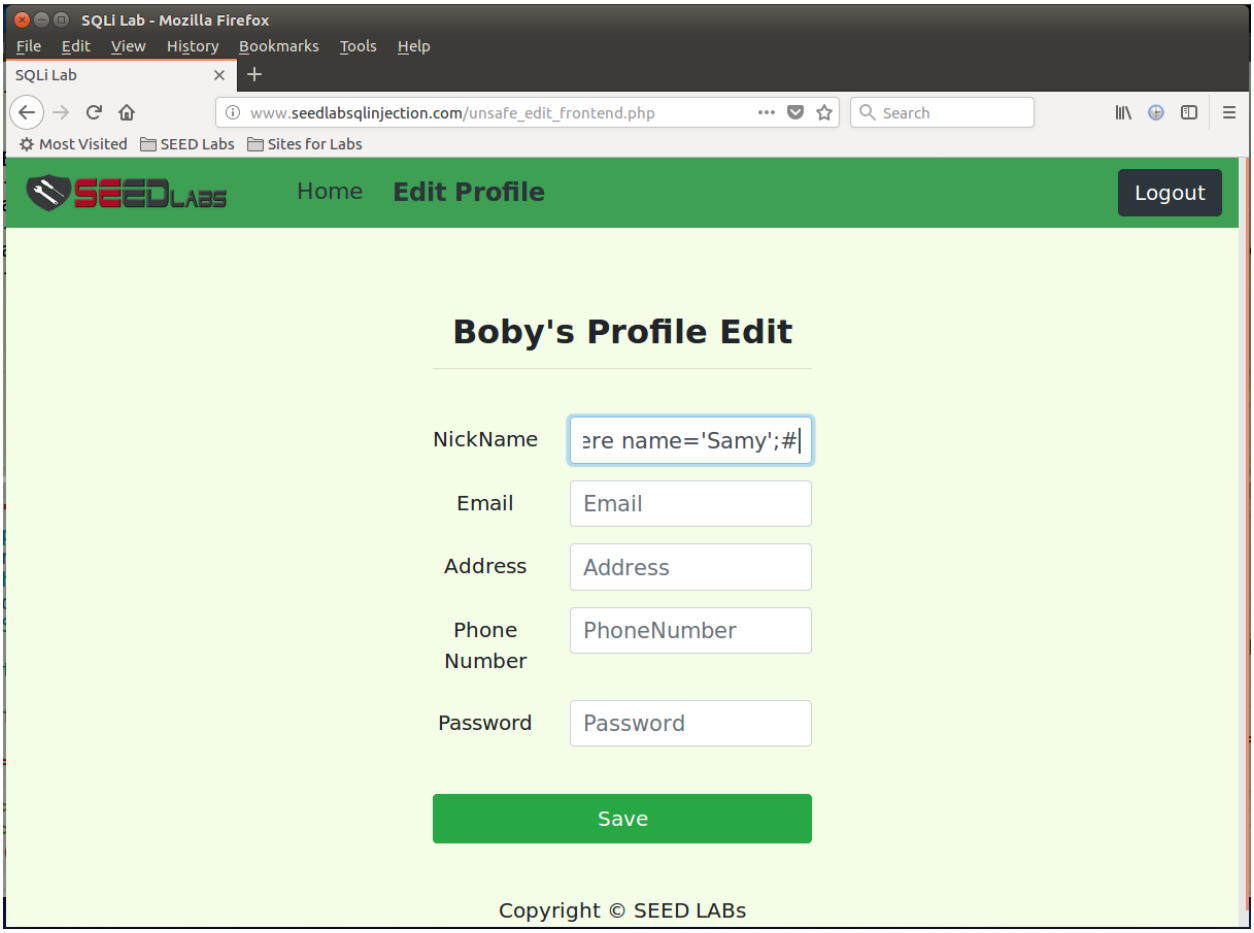
Goal: Using the Edit Profile page, update some else's salary. For this exercise, I am updating Samy's salary, not Boby; i.e. Samy is Boby's boss. Boby updated his salary in the previous exercise. For this one, Boby will update Samy's salary… because he is disgruntled. ☹

In the figure below, we see Samy's profile. Here, we can see that his salary is $90,000.

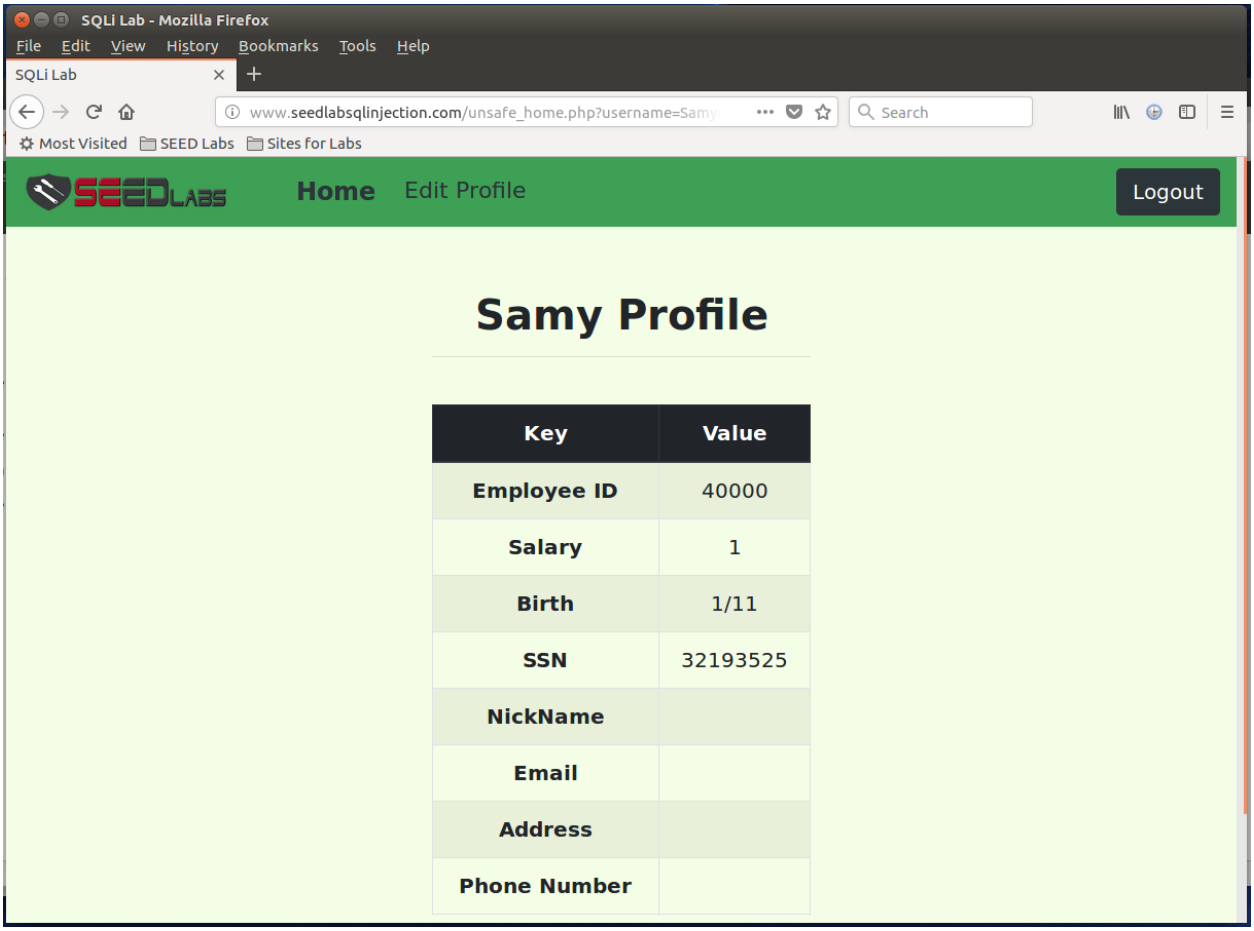


In the figure below, we are logged in as Boby. We are attempting an attack on Samy to change his salary to $1. In the NickName field, we insert the following text:

- "', salary='1' where name='Samy';#"



In the figure below, we are logged in as Samy again. We can see his salary is now $1.



Observations / Explanation

From Boby's Edit Profile page, we were able to modify Samy's salary. We did this by using the following text in the NickName field when editing Boby's profile.

- "', salary='1' where Name='Samy';#"

The parts of the text are explained below:

1. "'," – This text completes the Nickname text quote.
2. "salary='1' where name='Samy';" – This text sets only Samy's salary to $1.
3. "#" – This sets the remaining SQL query line in the PHP file to a comment.

To visualize, the text input changes SQL in the unsafe_edit_backend.php file (insertion in yellow highlight):

From this:

```
$sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
```

To this:

$sql = "UPDATE credential SET nickname='', salary='1' where name='Samy';#',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";

The end result is that we were able to successfully update Samy's salary, while logged in as Boby!

- Also, using this text also works "', salary='1' where EID='40000';#". This method uses Samy's EID instead of his name.

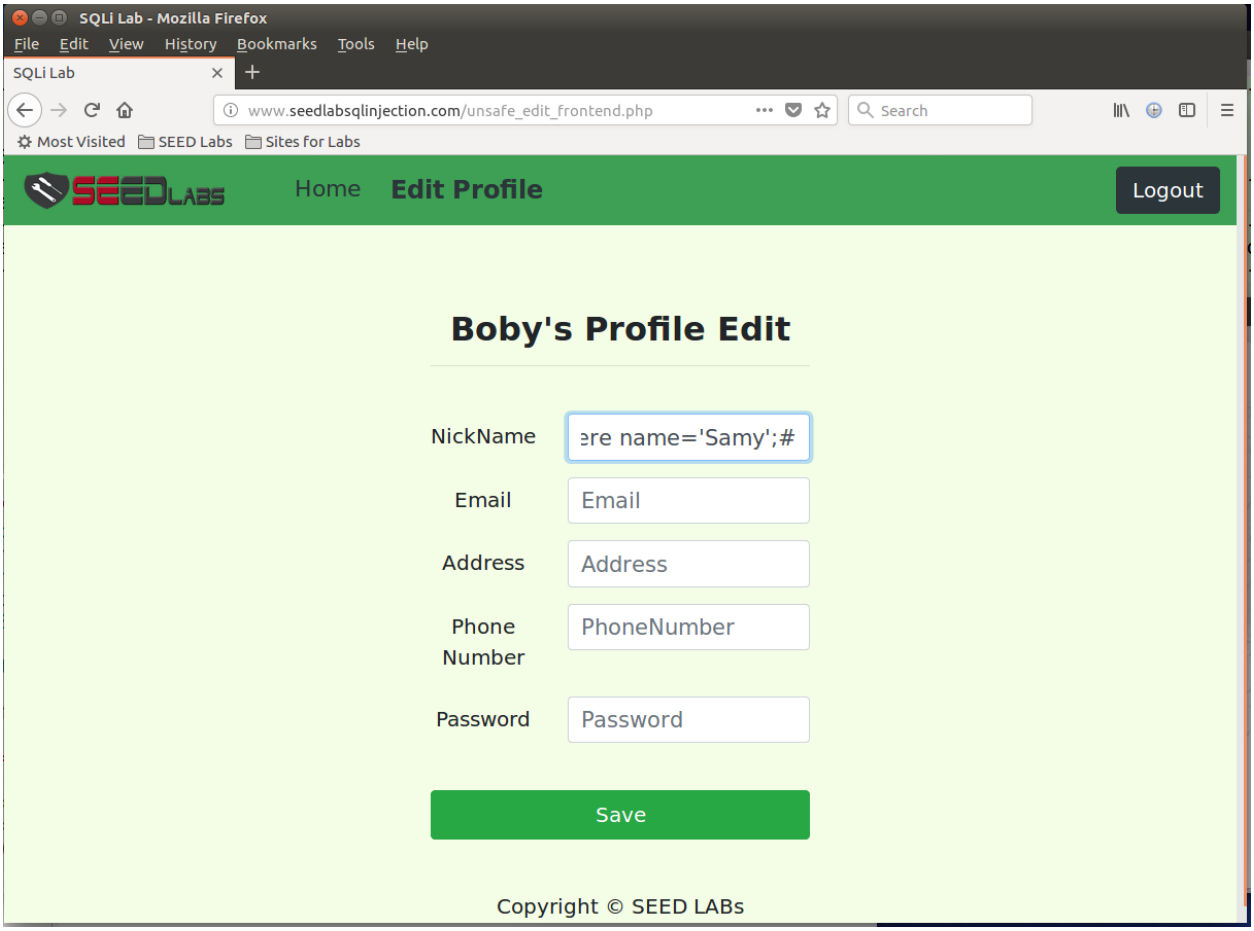
3.2 Modify other people's password.

Goal: Using the Edit Profile page, update some else's salary. For this exercise, I am updating Samy's password, not Boby; i.e. Samy is Boby's boss. Boby updated his salary in the first exercise. Then Boby update Samy's salary to a much lower salary. Now, Boby is going to take it one step further and change Samy's password.

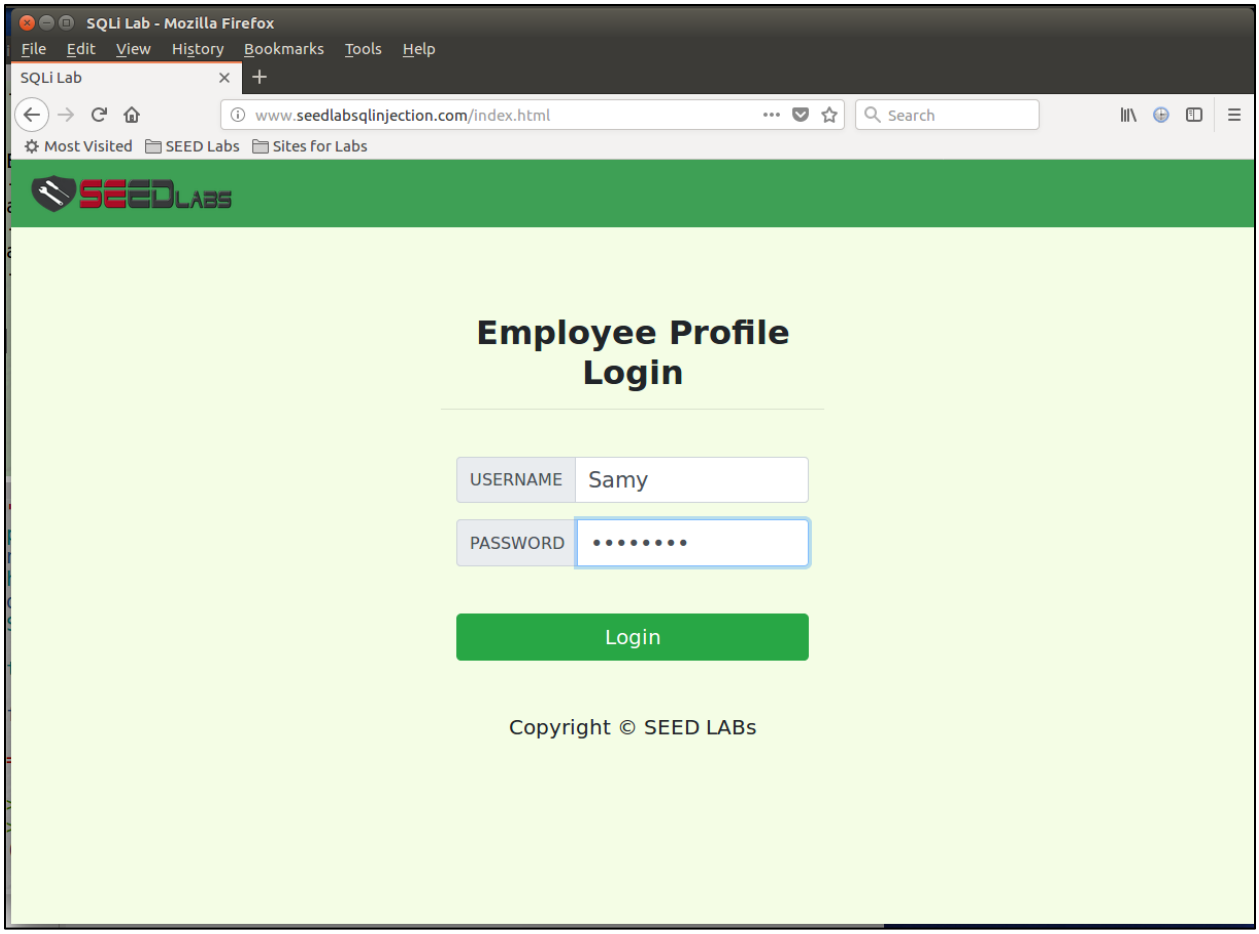
For this exercise, I'm using the following:

1. ', Password='52e51cf3f58377b8a687d49b960a58dfc677f0ad' where name='Samy';# - This is the text used in Boby's profile to change Samy's password hash.
2. "attacker" – This is the password that will be Samy's new password.
3. "52e51cf3f58377b8a687d49b960a58dfc677f0ad" – This is the hash value of the "attacker" password which will be stored in the database table per the update.

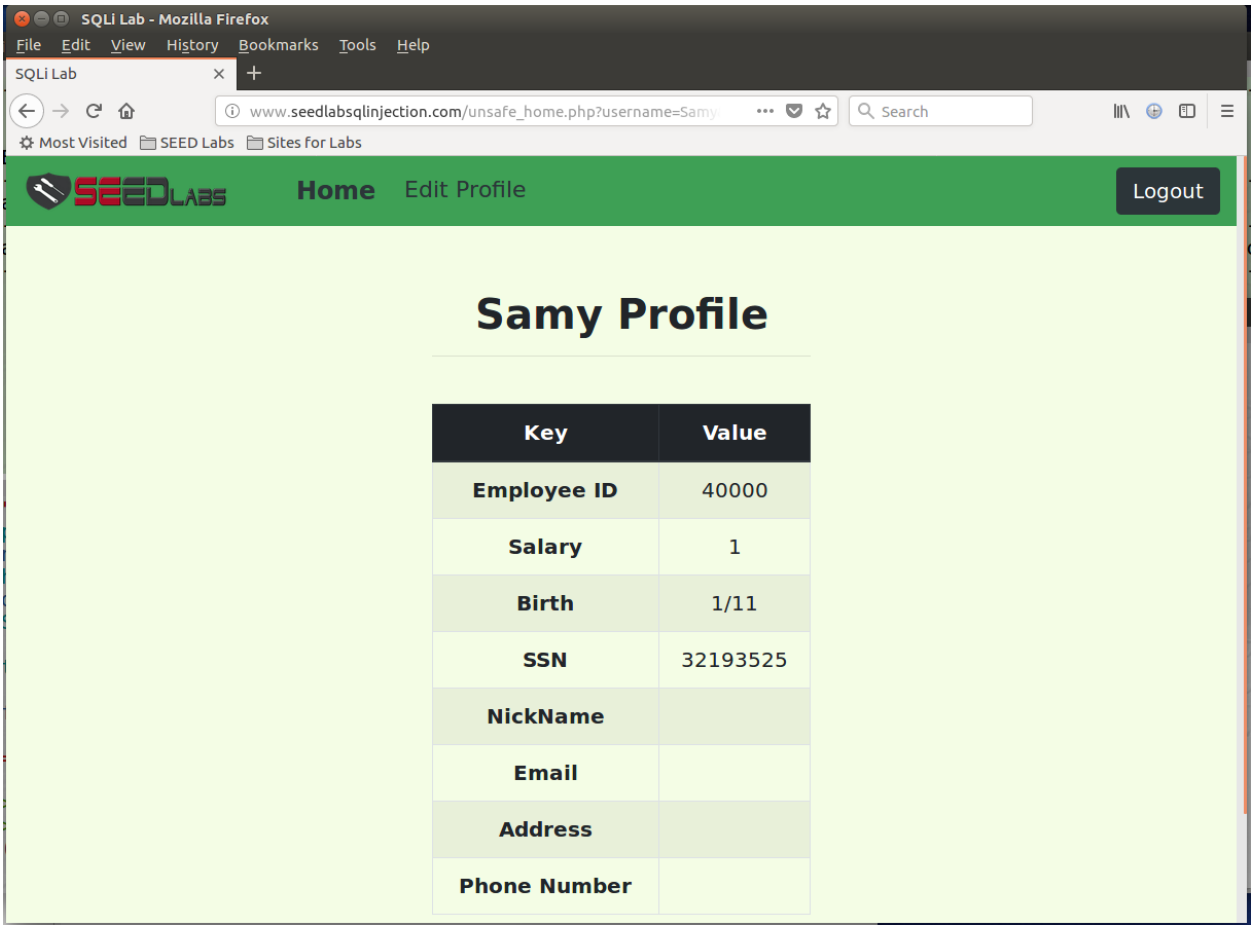
In the figure below, we are logged in as Boby and we've put in the attack text (in #1 above) into the NickName field.



We now attempt to login to Samy's account using the new "attacker" password.



In this figure, we successfully login to Samy's account with the "attacker" password.

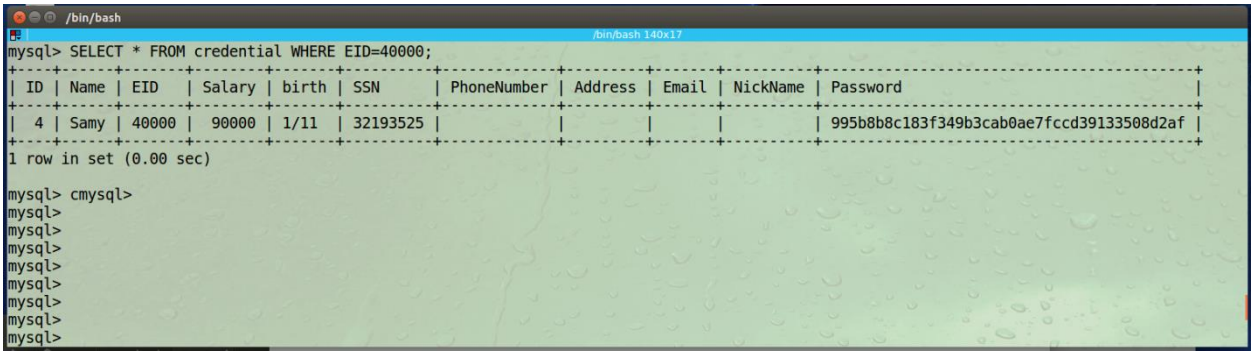


In the figure below, we show the code for the getpwd.php file. We use this file to generate the SHA1 hash of the "attacker" password. Since the table stores only the hash of the password, we must update the value of Samy's password to the hash value; i.e. not the actual password. This is done for security reasons so actual passwords aren't stored in the database which can be risky if exposed. SHA1 is a one way hash which makes reverse engineering a hash to a password very difficult*.

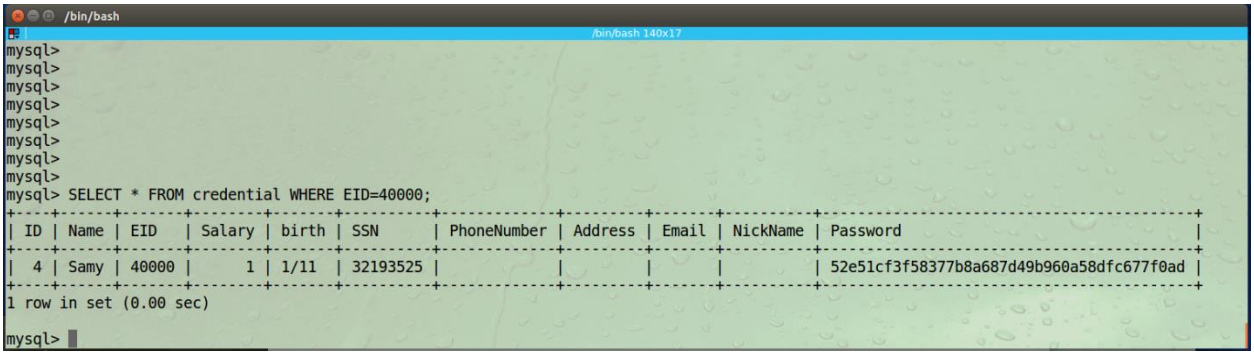
** In practical terms, there are methods like rainbow tables which can be used to lookup hash values to reverse engineer password values, BUT, at a high level, hashing is a better idea then storing the explicit password AND stronger hash functions and crypto/encryption methods can be used for better password management.*

In the figure below, we see the SHA1 hash value of "attacker".

It’s difficult to see that new password worked after changing it when showing the website login screen. In the figure below, we see the Password field of Samy before the attack. Not the hash value in the Password field.



In the figure below, we see the new hash value in the Password field for Samy. The value of the Password is the new SHA1 hash value computed for “attacker”.



Observations / Explanation

From Boby’s Edit Profile page, we were able to modify Samy’s passowrd. We did this by using the following text in the NickName field when editing Boby’s profile.

- ', Password='52e51cf3f58377b8a687d49b960a58dfc677f0ad' where name='Samy';#

The parts of the text are explained below:

1. “’,” – This text completes the first single quote for the NickName field.
2. Password='52e51cf3f58377b8a687d49b960a58dfc677f0ad' where name='Samy'; - This text updates Samy’s password to the “attacker” SHA1 hash value.
3. “#” – This text comments-out the rest of the query (update) string.

$sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";

To visualize, the text input changes the SQL in the unsafe_edit_backend.php file (insertion in yellow highlight):

From this:

$sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password ='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";

To this:

$sql = "UPDATE credential SET nickname='', Password='52e51cf3f58377b8a687d49b960a58dfc677f0ad' where name='Samy';#',email='$input_email',address='$input_address',Password='$hashed_ pwd',PhoneNumber='$input_phonenumber' where ID=$id;";

Task 4: Countermeasure — Prepared Statement

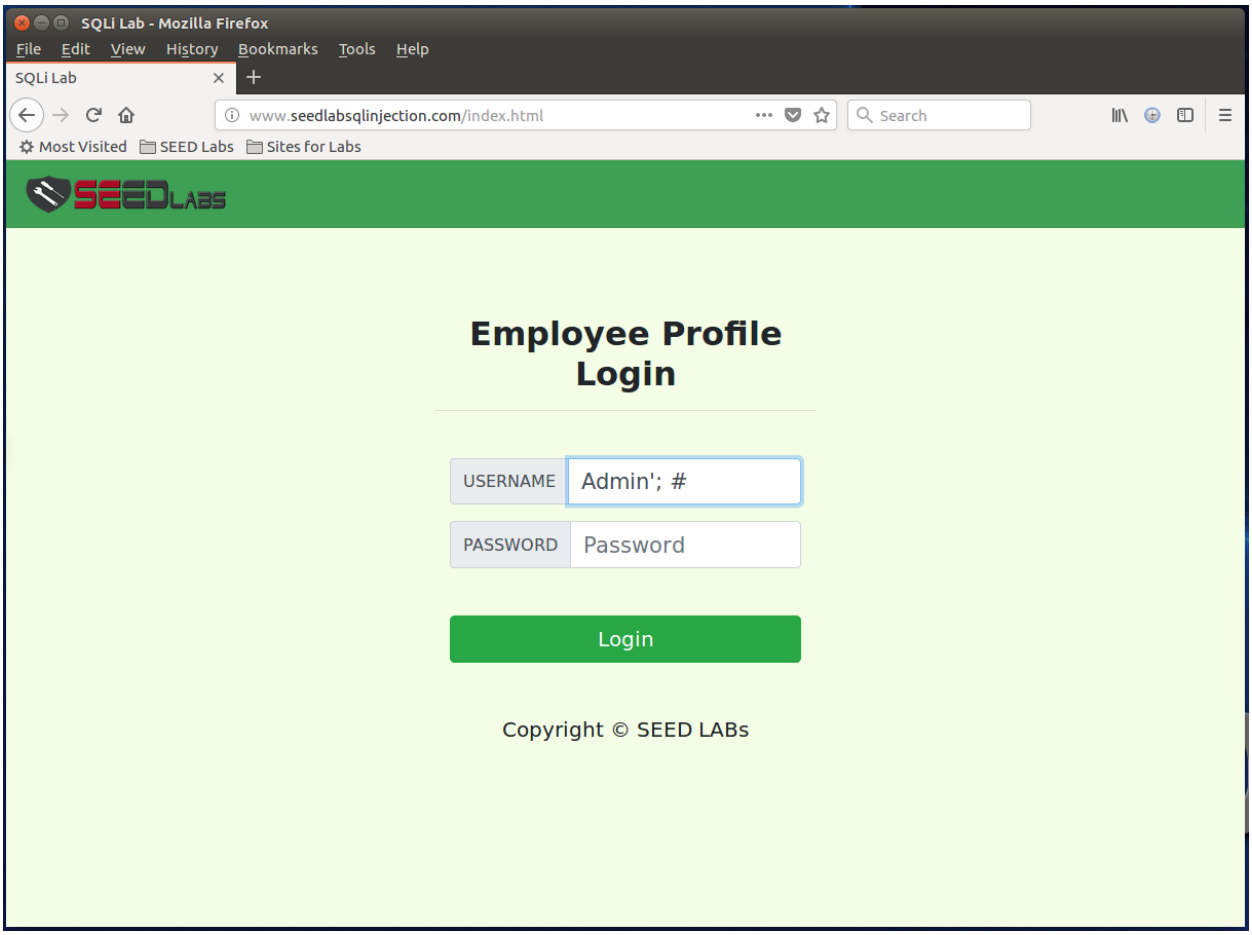
Goal: Use the prepared statement mechanism to fix the SQL injection vulnerabilities exploited by you in the previous tasks.

For this task, we reuse /var/www/SQLInjection:

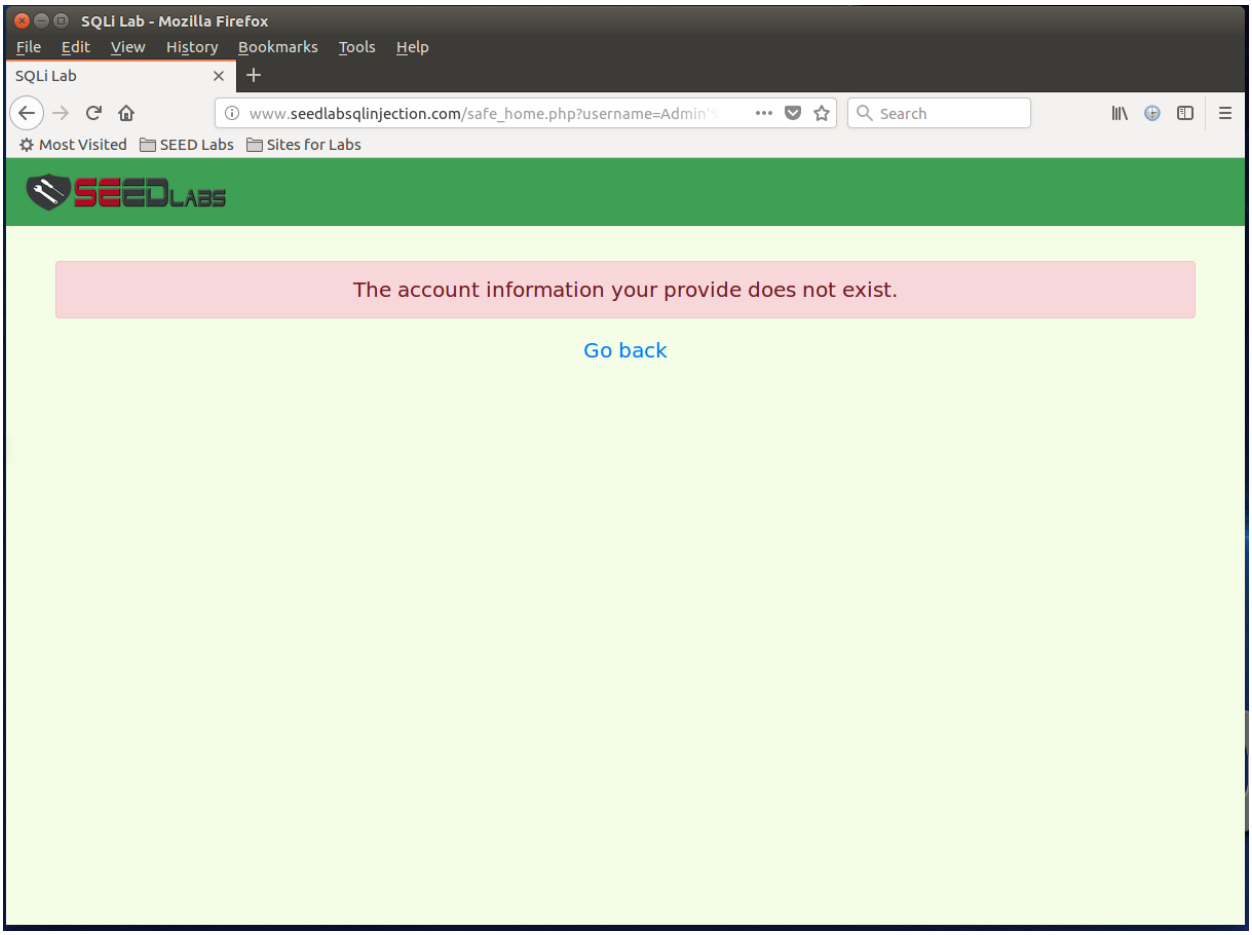
- safe_home.php – This file processes the login request in a "safe" manner; i.e. using prepared statements.
- safe_edit_backend.php – This file processes profile updates in a "safe" manner; i.e. using prepared statements.

In the figure below, we see that index.html is edited to now use safe_home.php to process (validate) the login username/password.

We attempt the same login from task 2.1 to see if we can login as Admin without specifying the Admin's password.



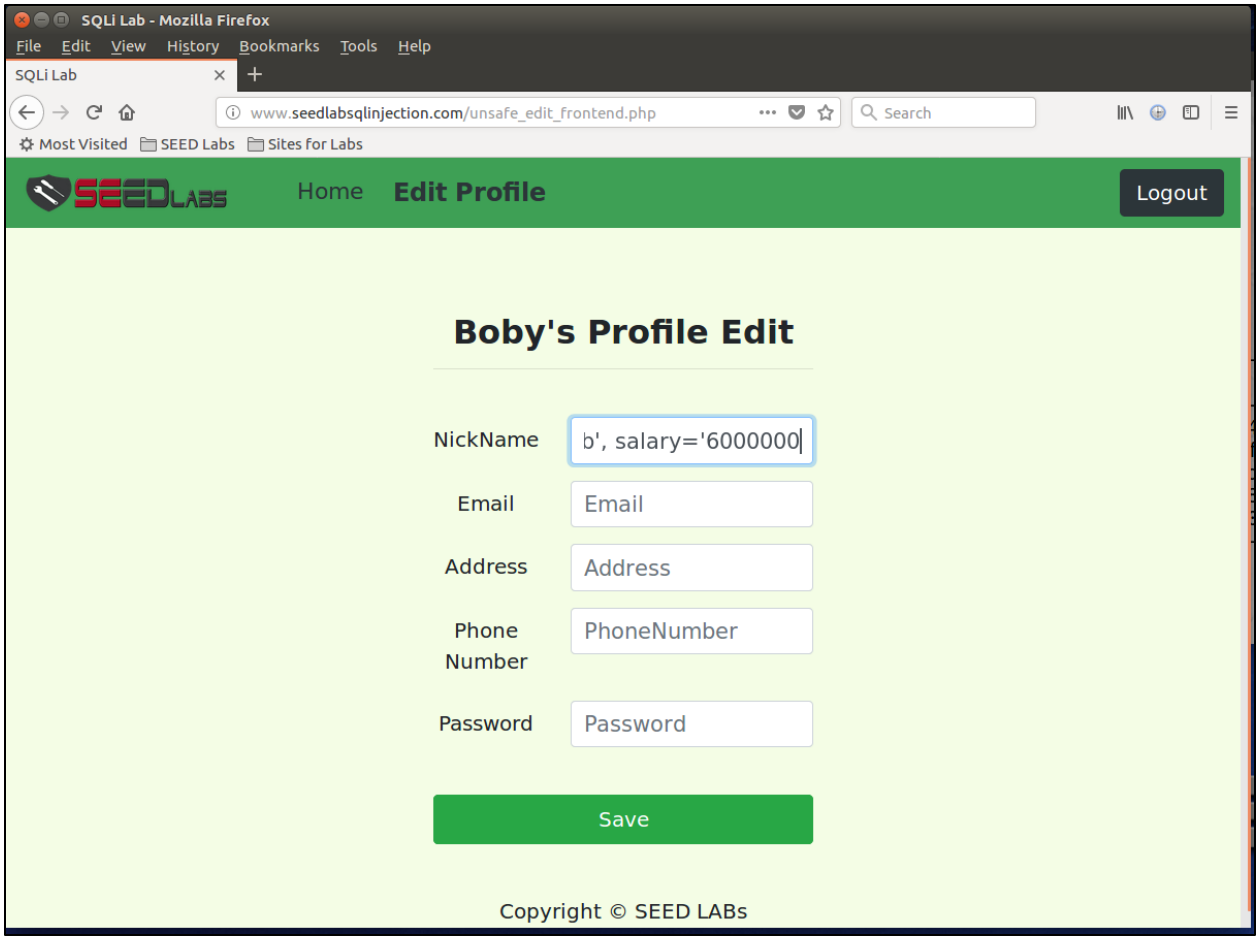
The figure below shows that the login attempt failed.



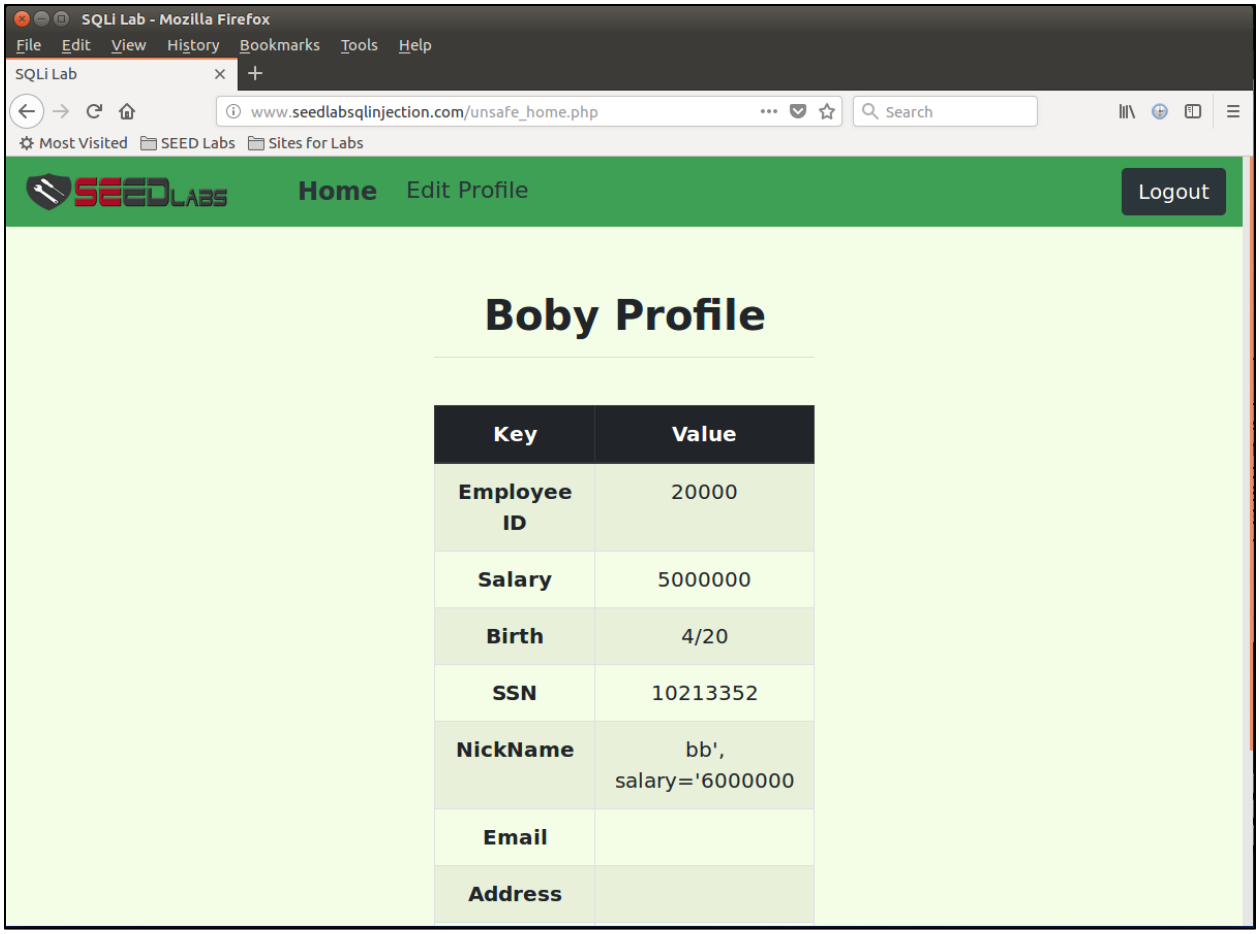
In the figure below, we change unsafe_edit_frontend.php to use safe_edit_backend.php so that prepared statements are used when editing profiles.

In the figure below, we see another attack which was in task 3.1. This attack allowed Boby to modify his own salary. The following SQL update command was used (changed NickName to “bb” and salary to 6000000 so I can see changes):

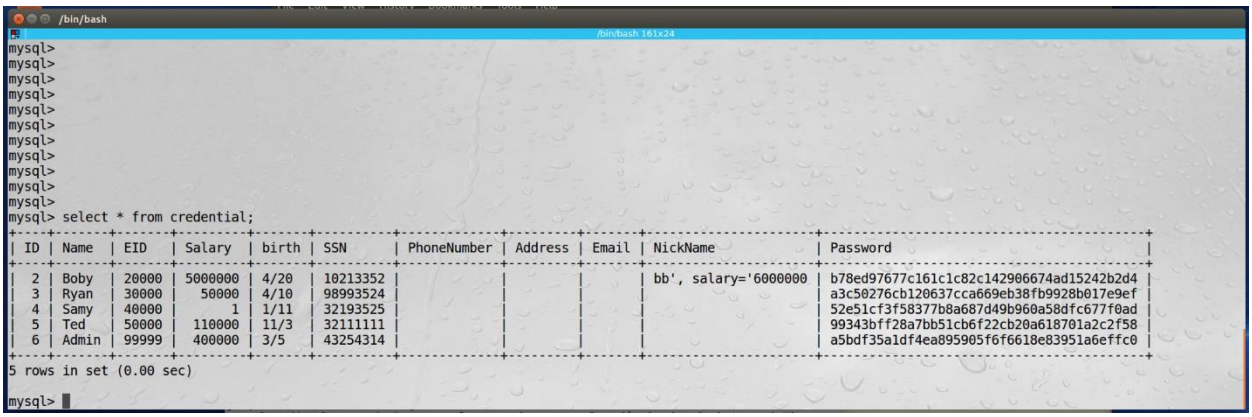
- “bb', salary='6000000”



In the figure below, we see the web pages after the Save. No data was changed.



Looking at the credential table after the attempted attack, we the figure below. Instead of executing the attack, the NickName field is changed to the attack string!



Observations / Explanation

Two examples were provided here:

1. We try to change the SQL query at the Login screen to login without a password.
2. We try to change the credential table via the Edit Profile to change salaries.

With both example and using safe prepared-statement versions of the PHP files, the attacks (i.e. input text) were treated as input text and not used to substitute within the query itself. This is accomplished by compiling the queries in advance and binding to the data separately. This is in contrast to the previous method of comingling the SQL statement and user input by inserting the user input directly into the SQL statement. This, obviously, proved to be very bad and attack prone.

In the first example, since "Admin'; #" is not a valid username, the login failed. The text is treated like data and not inserted into the query string as expected with the prepared statement.

In the second example, we see that the input text was indeed treated as data since that text did change the NickName, but did not change the query. Bottom-line, prepared statements are the safer way to go!