Jon Franck

3/27/2021

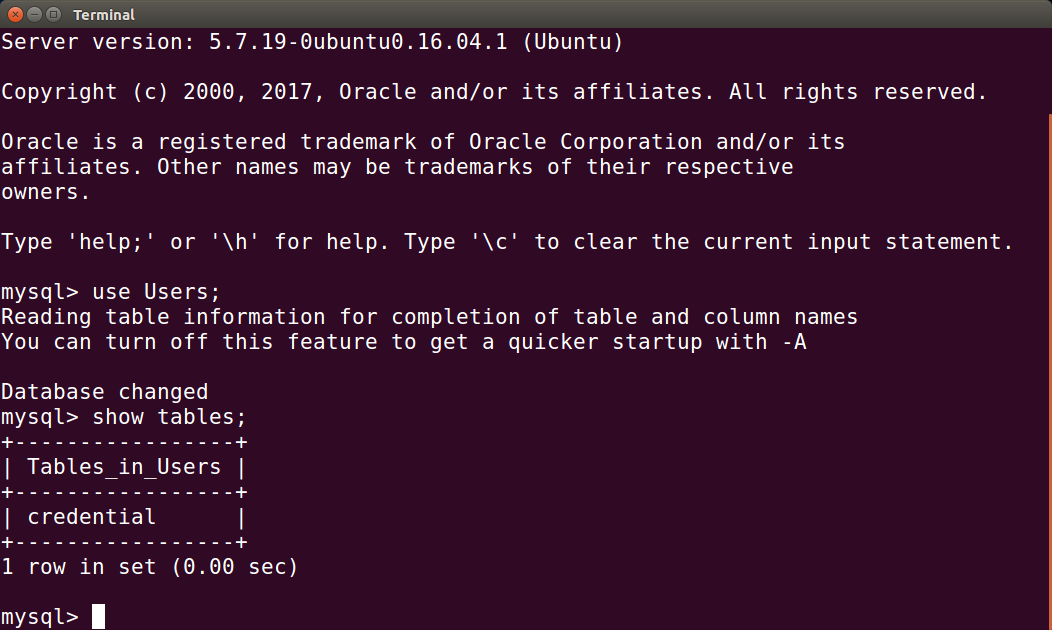
CSC 4413

# SQL Injection

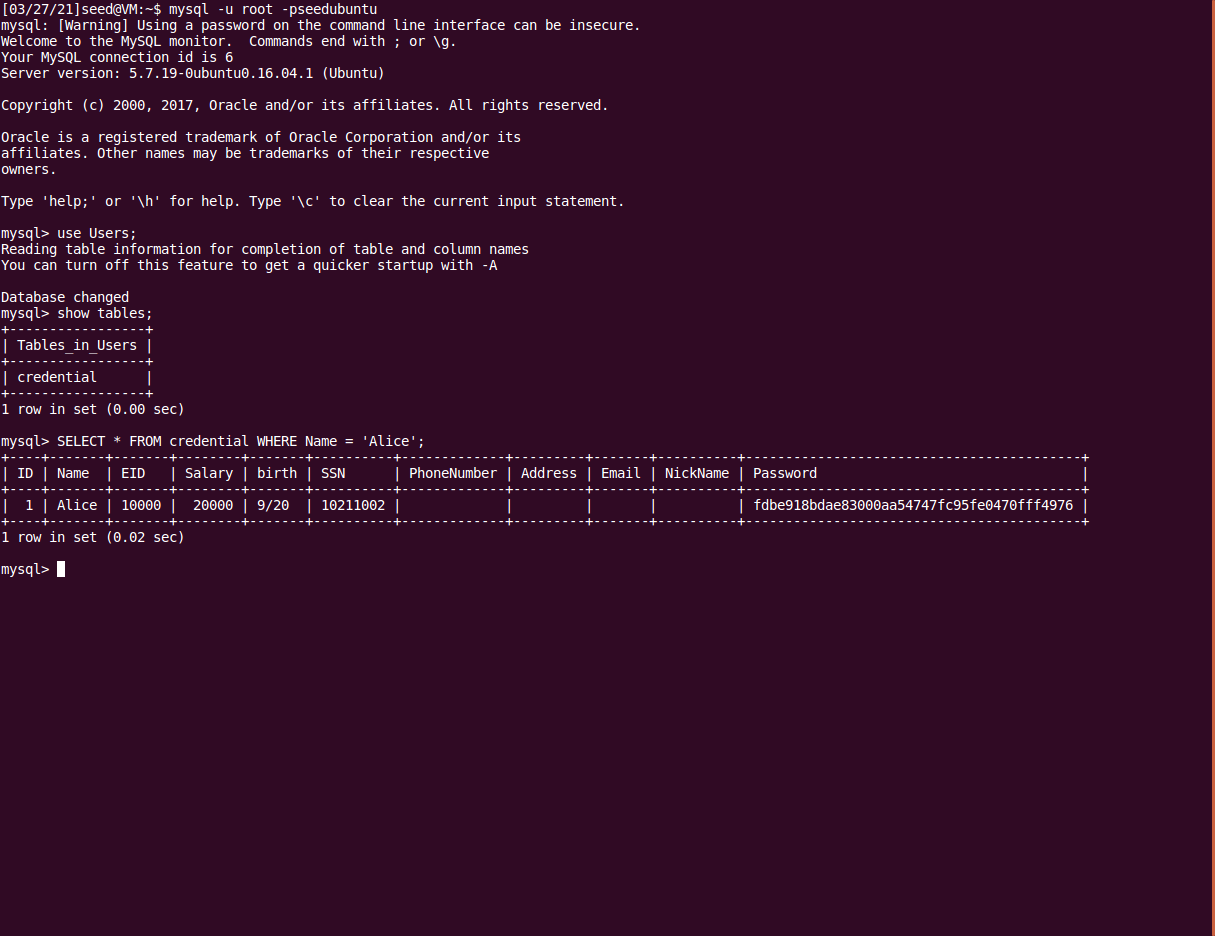
**Objective:** Students’ goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics:

• SQL statement: SELECT and UPDATE statements  
• SQL injection  
• Prepared statement

## Task 1 – Get Familiar with SQL Statements

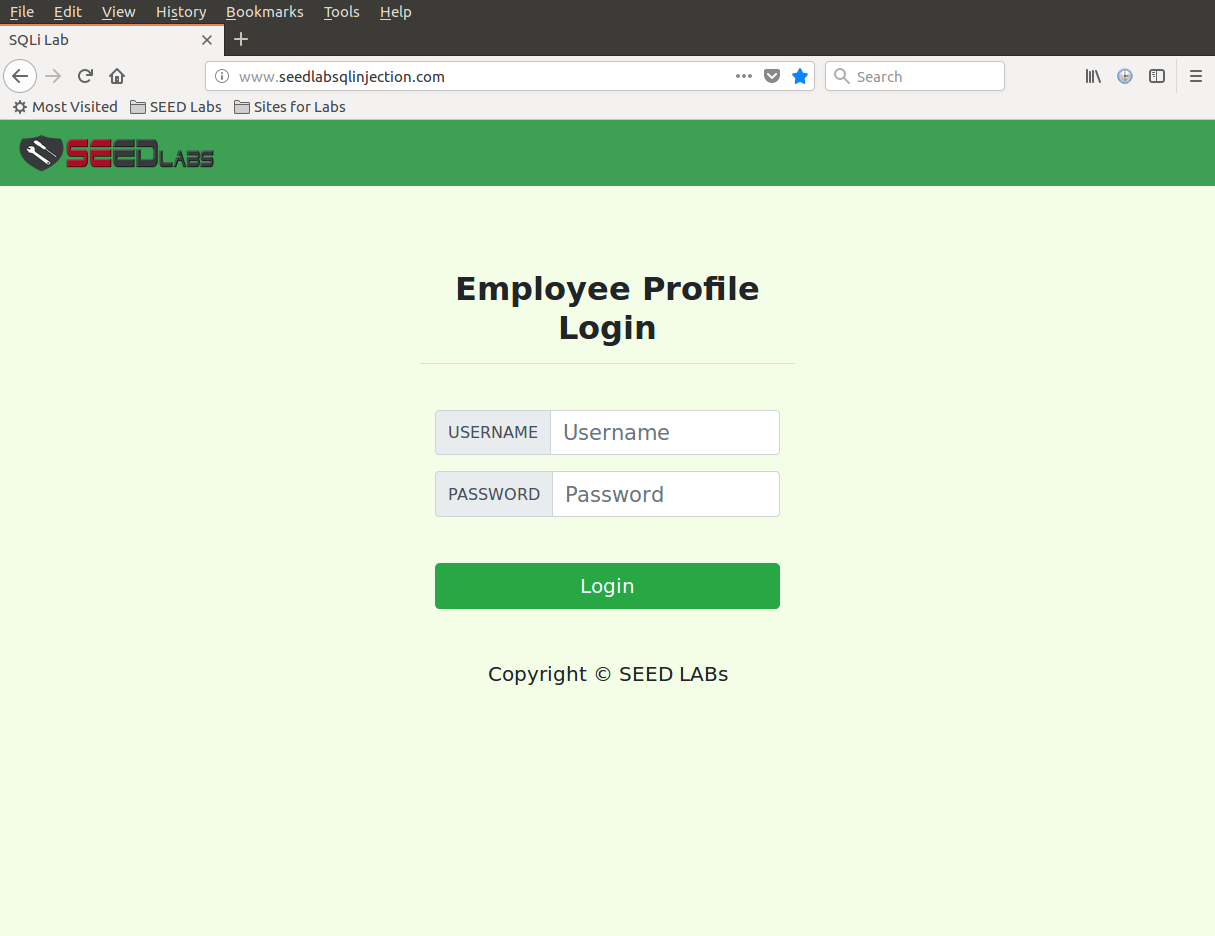


Use the above commands to login to the MySQL console, then load and display the Users database.



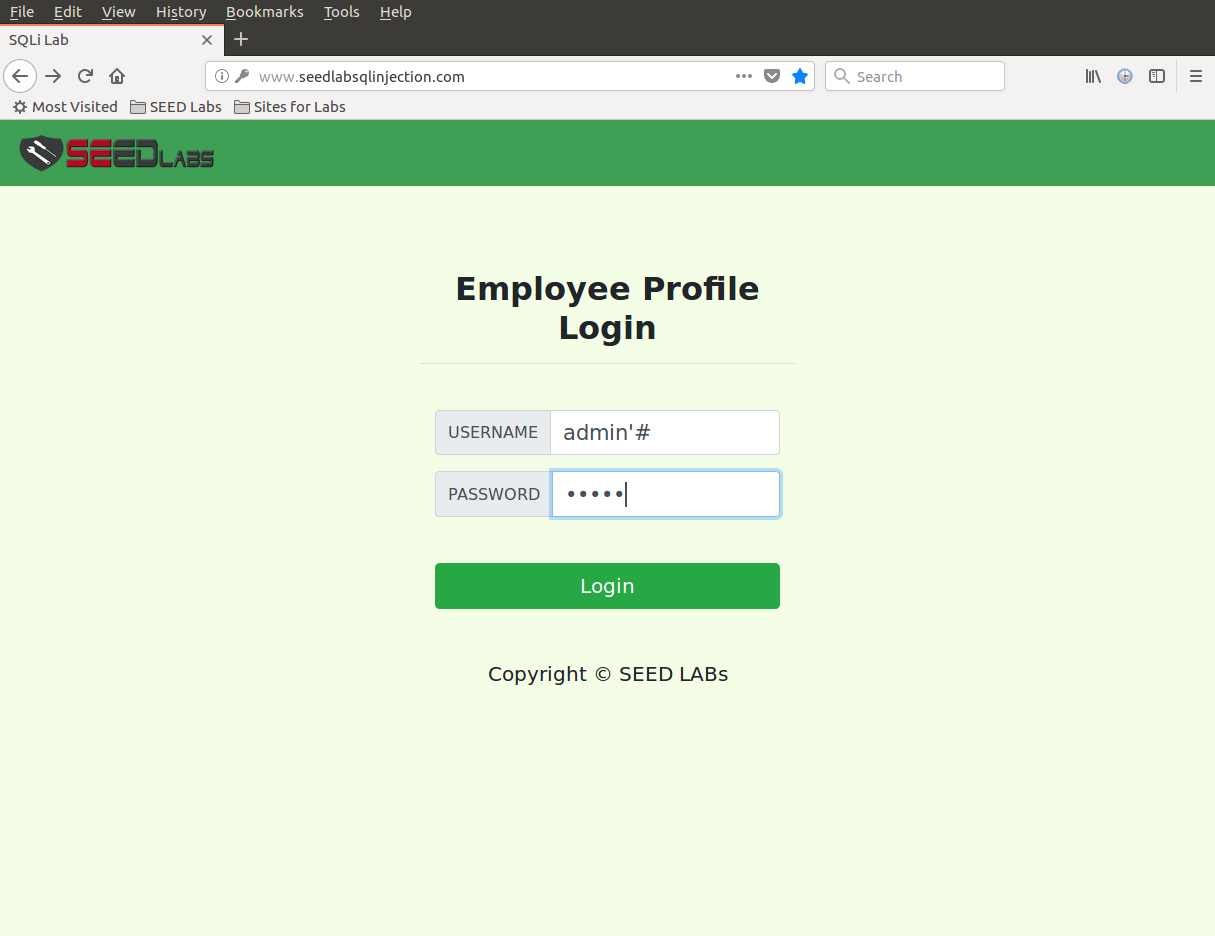
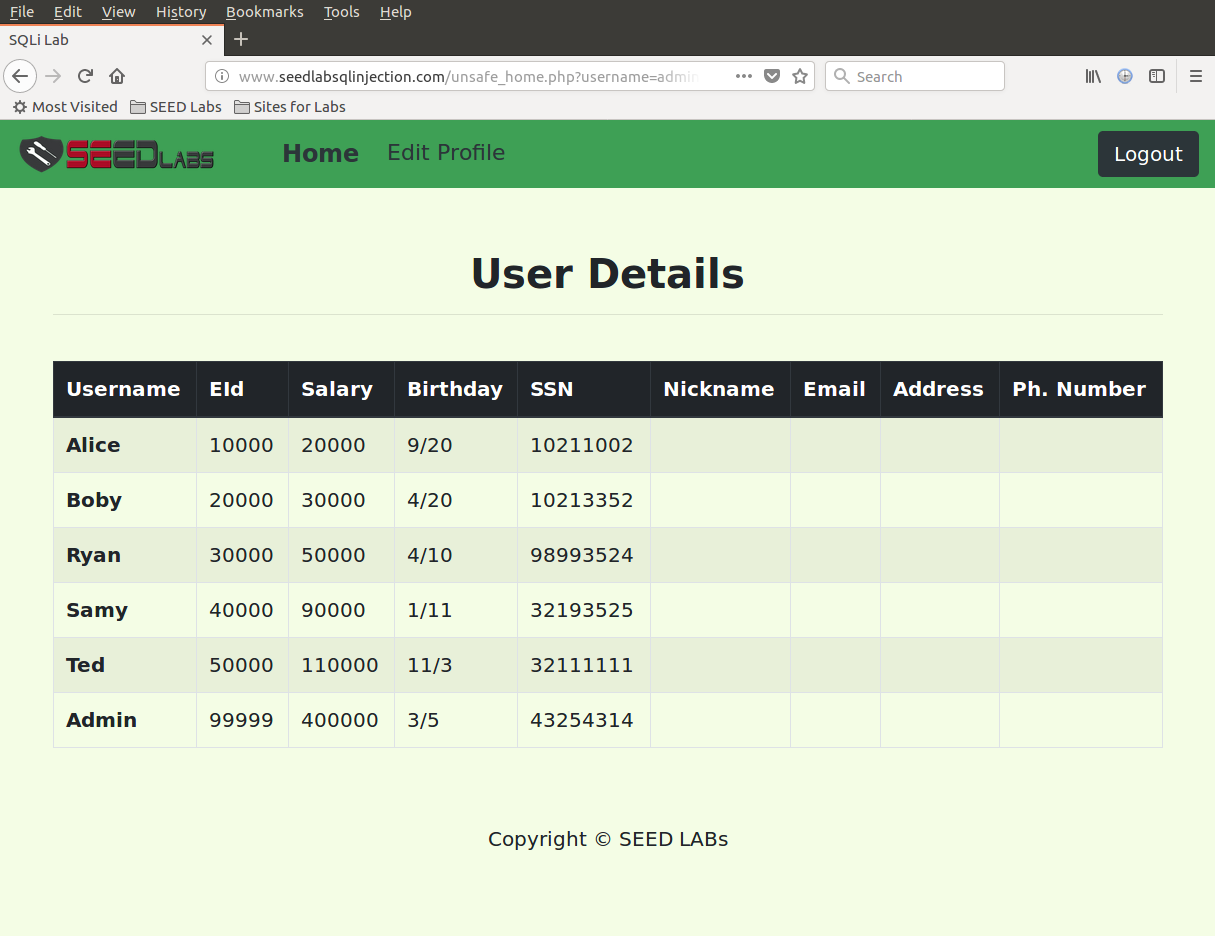
The SELECT statement above returns all fields (\*) FROM the ‘credentials’ table for employees with the name Alice (WHERE Name = ‘Alice’).

## Task 2 – SQL Injection Attack on SELECT Statement



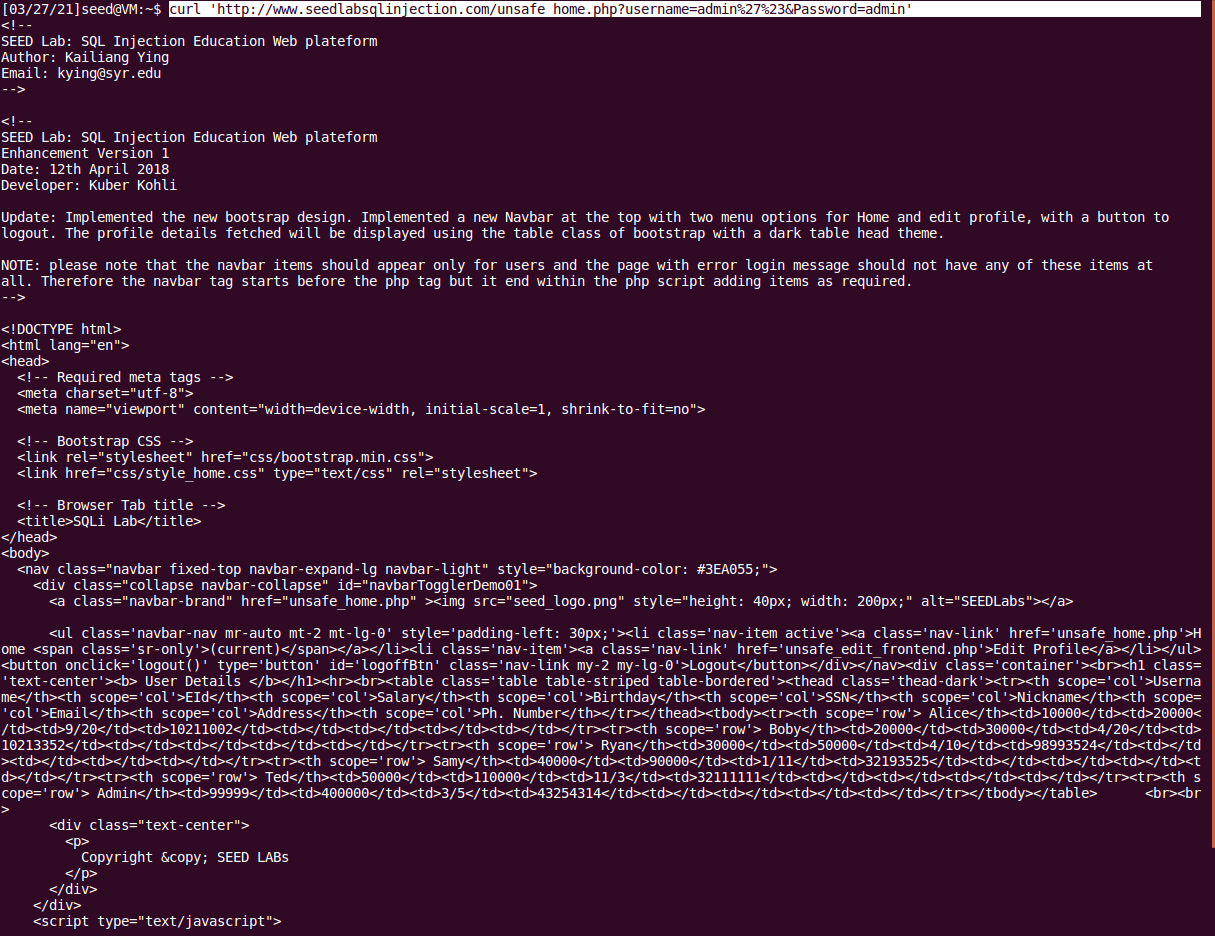
Our goal is to use the SELECT statement vulnerability to gain authentication without having a username/password.

**Task 2.1** – login as the Administrator to you can view all of the other user’s login credentials.

To exploit this vulnerability, we first assume that the Administrator username is ‘admin’. By including the endquote and # symbol, we cause everything after the ‘$input\_uname’ in the PHP file’s SELECT statement to be commented out – meaning we don’t need to know the password. Here I entered ‘admin’ as the password, as the webpage can still check to make sure a password is entered and give a JavaScript alert if there is not.

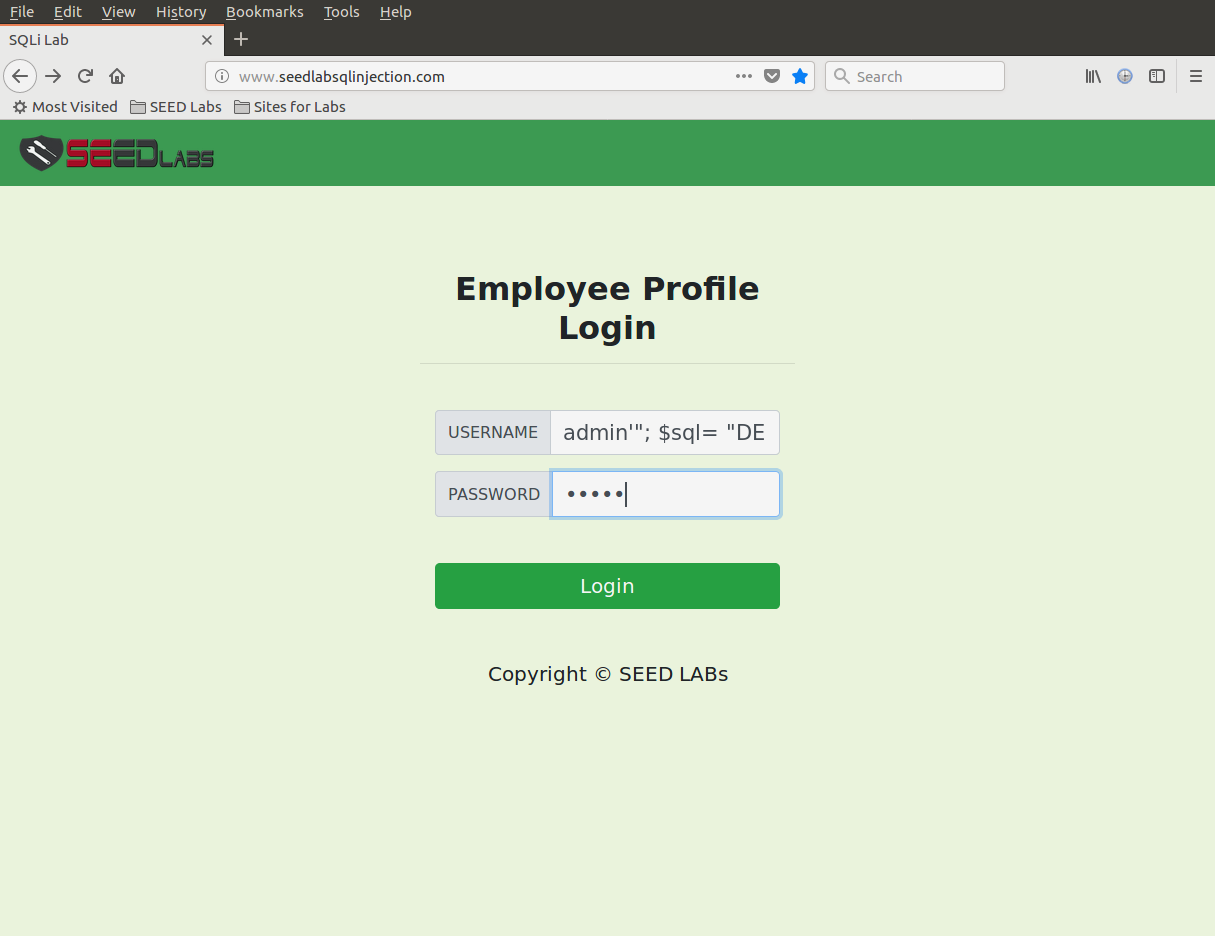
**Task 2.2** – Complete the same task but from command line, with the ‘curl’ command.



Basically, we just copy/paste the URL from the web browser to the command line after the ‘curl’ command. Note the single quote and # symbol are encoded as %27 and %23, respectively.

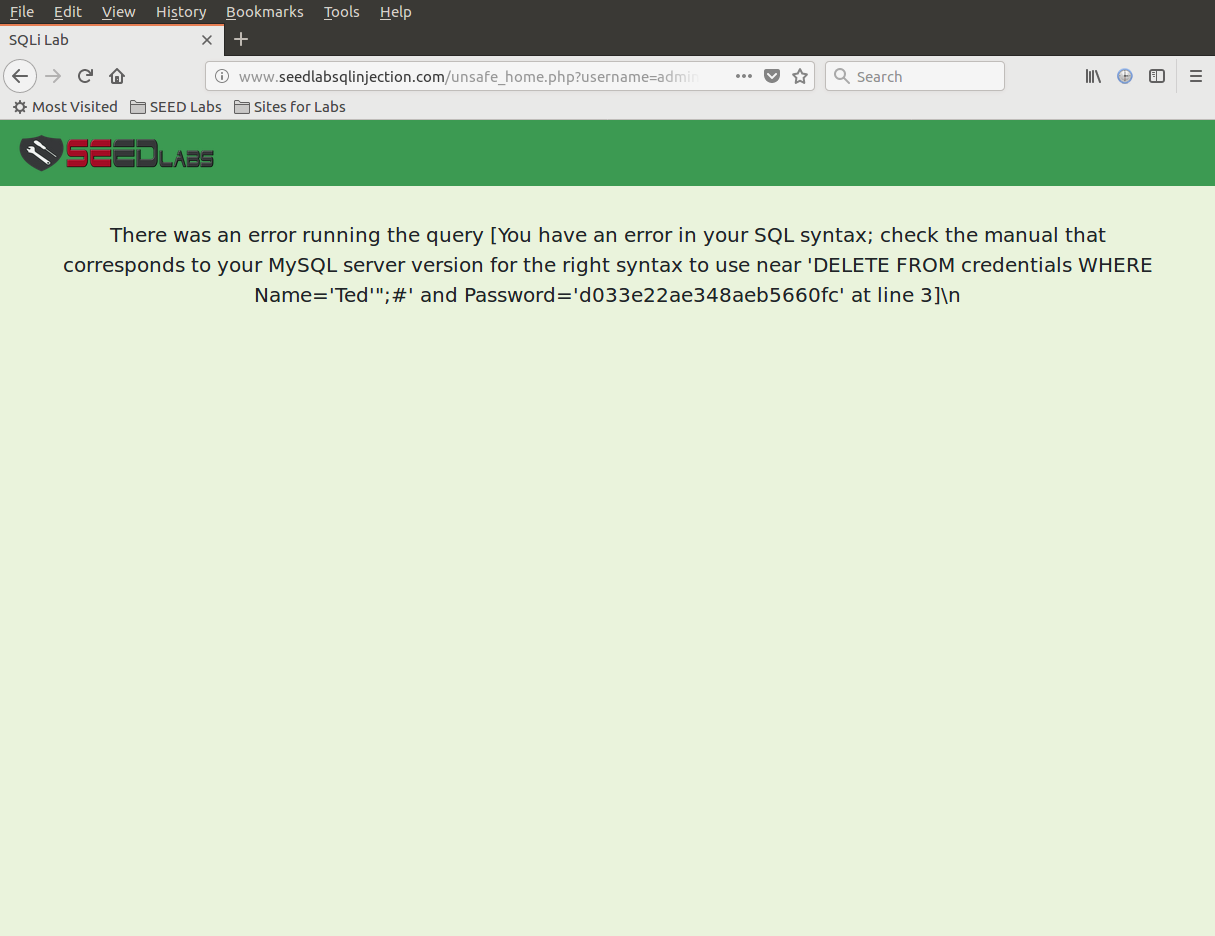
**Task 2.3** – Append a new SQL statement

By using a semicolon, we can end the SQL statement and write our own new statement. Our goal is to delete a record from the database.



Here, I used the following username input to try to append a new SQL statement:

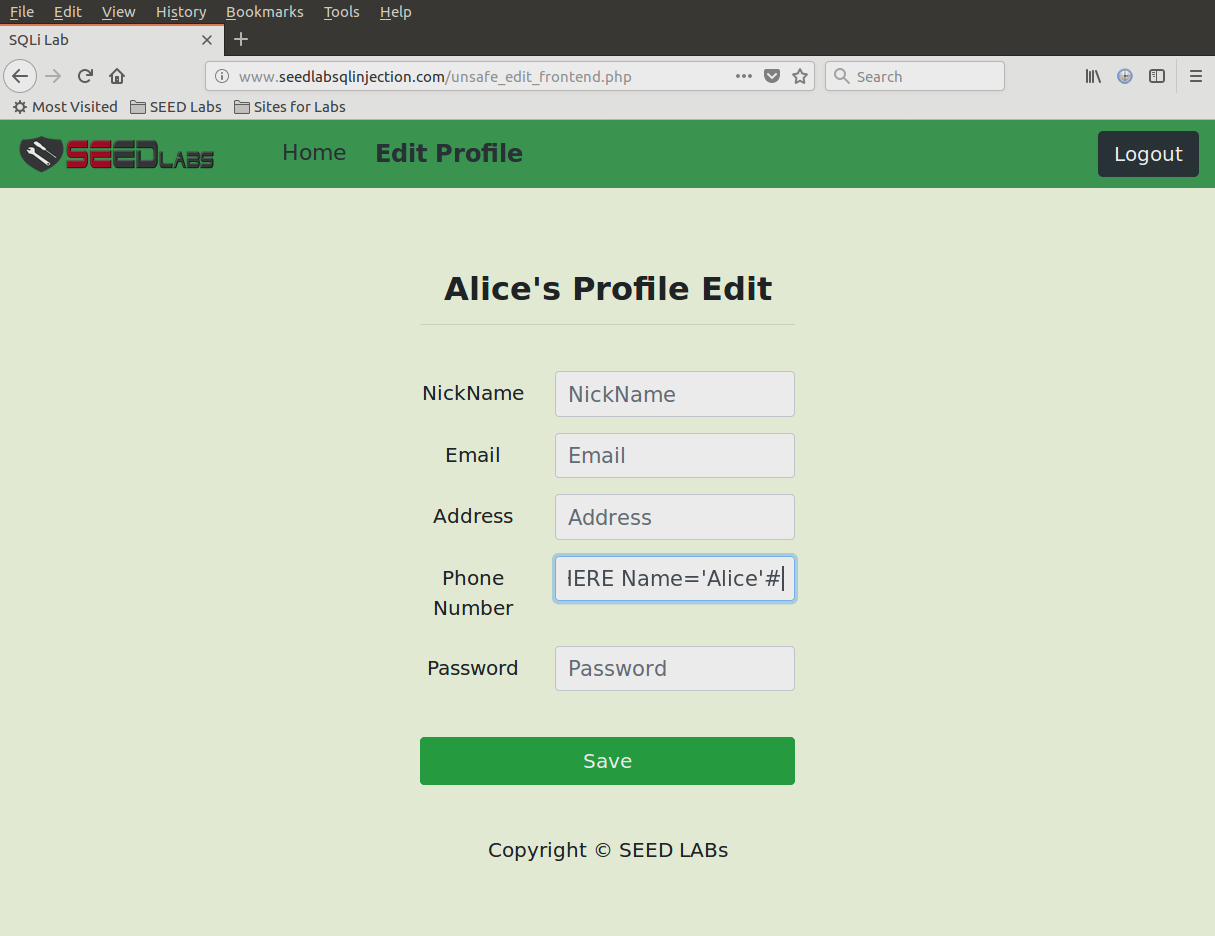
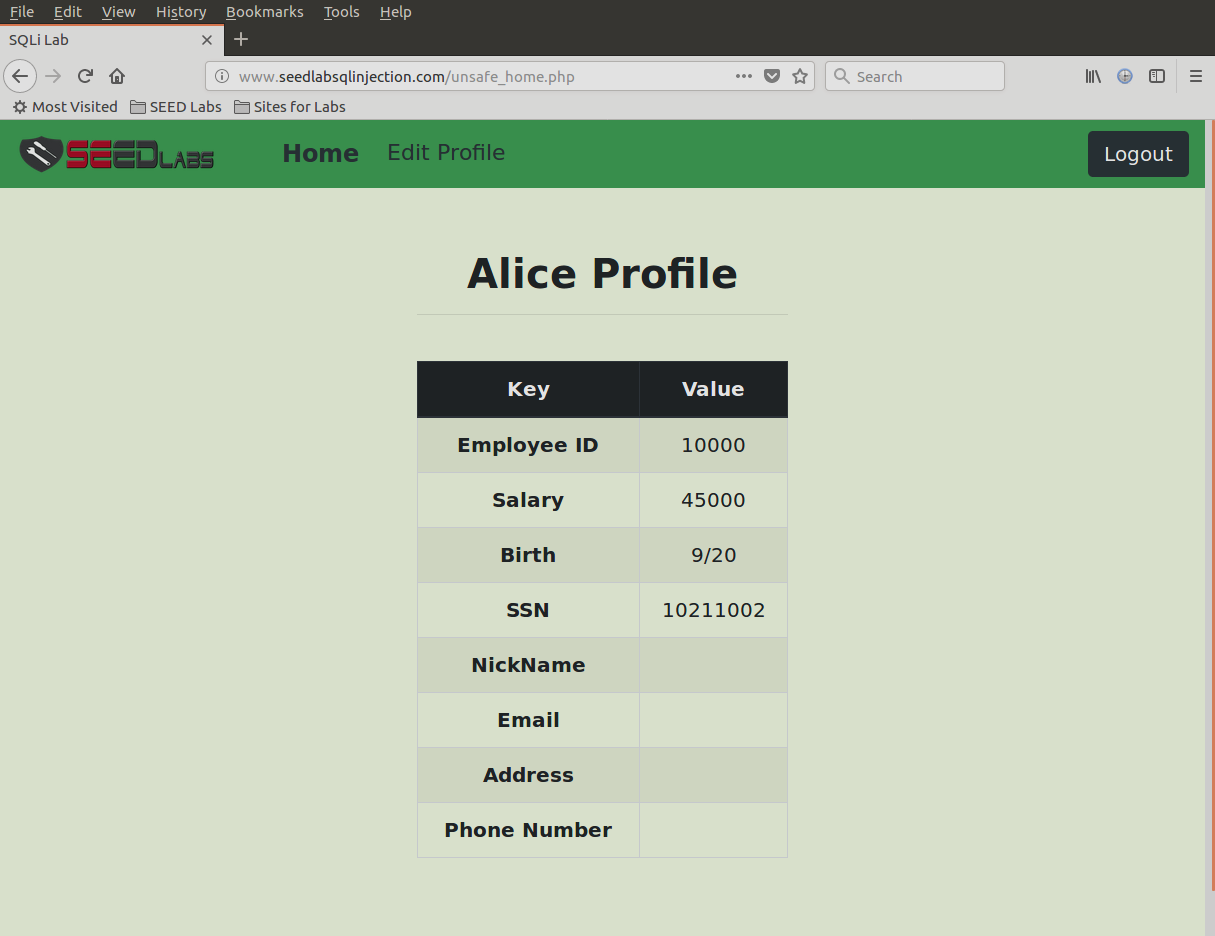
admin'"; $sql= "DELETE FROM credentials WHERE Name='Ted'";#



I don’t see where the syntax error is, so I don’t believe that this is the actual issue. I don’t think it is possible to use more then one SQL statement in one string.

## Task 3 – SQL Injection Attack on the UPDATE Statement

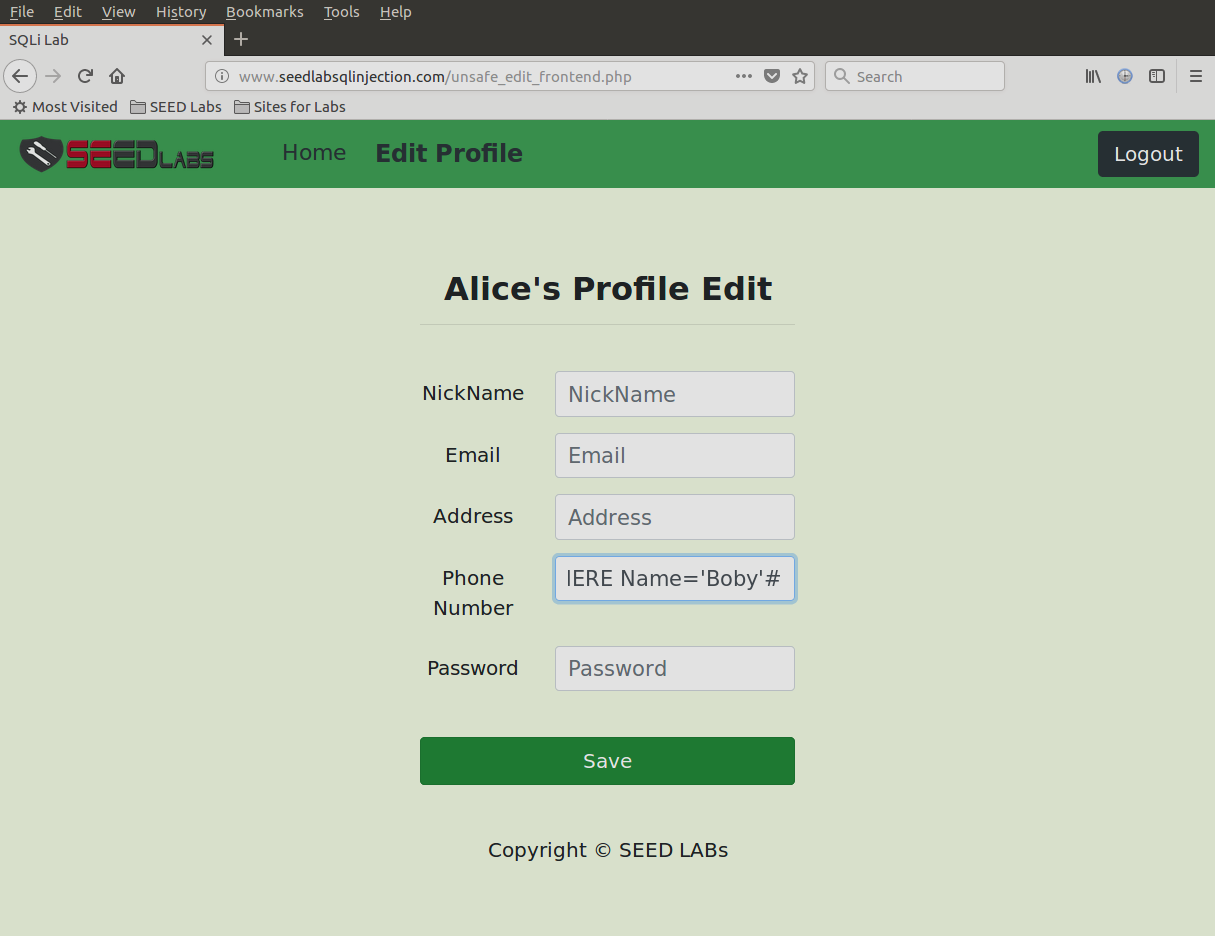
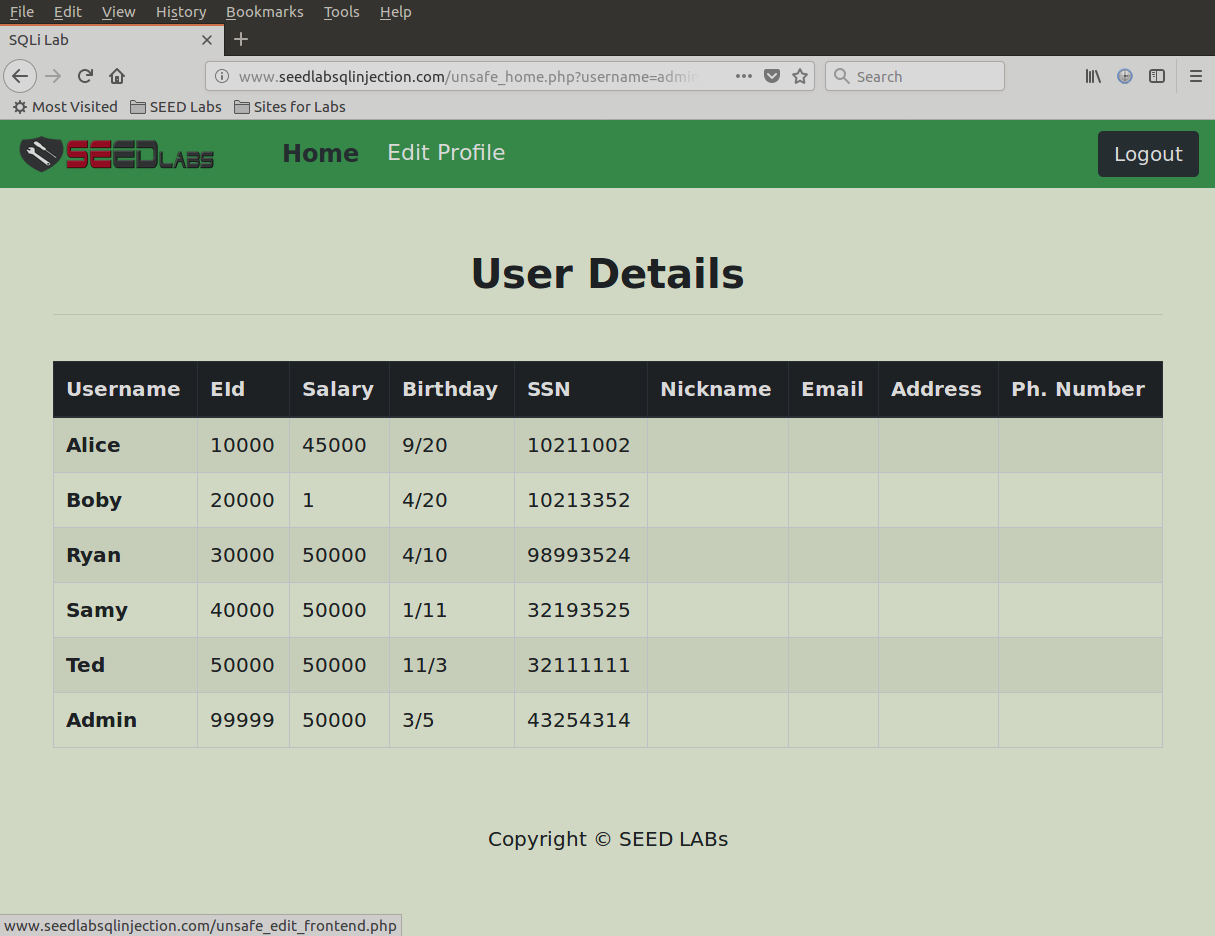
**Task 3.1** – Modify Your Own Salary

I logged in to my account (Alice) and used the following input in the Phone number field to edit the UPDATE statement in order to change my salary:

‘, salary=45000 WHERE Name=’Alice’#

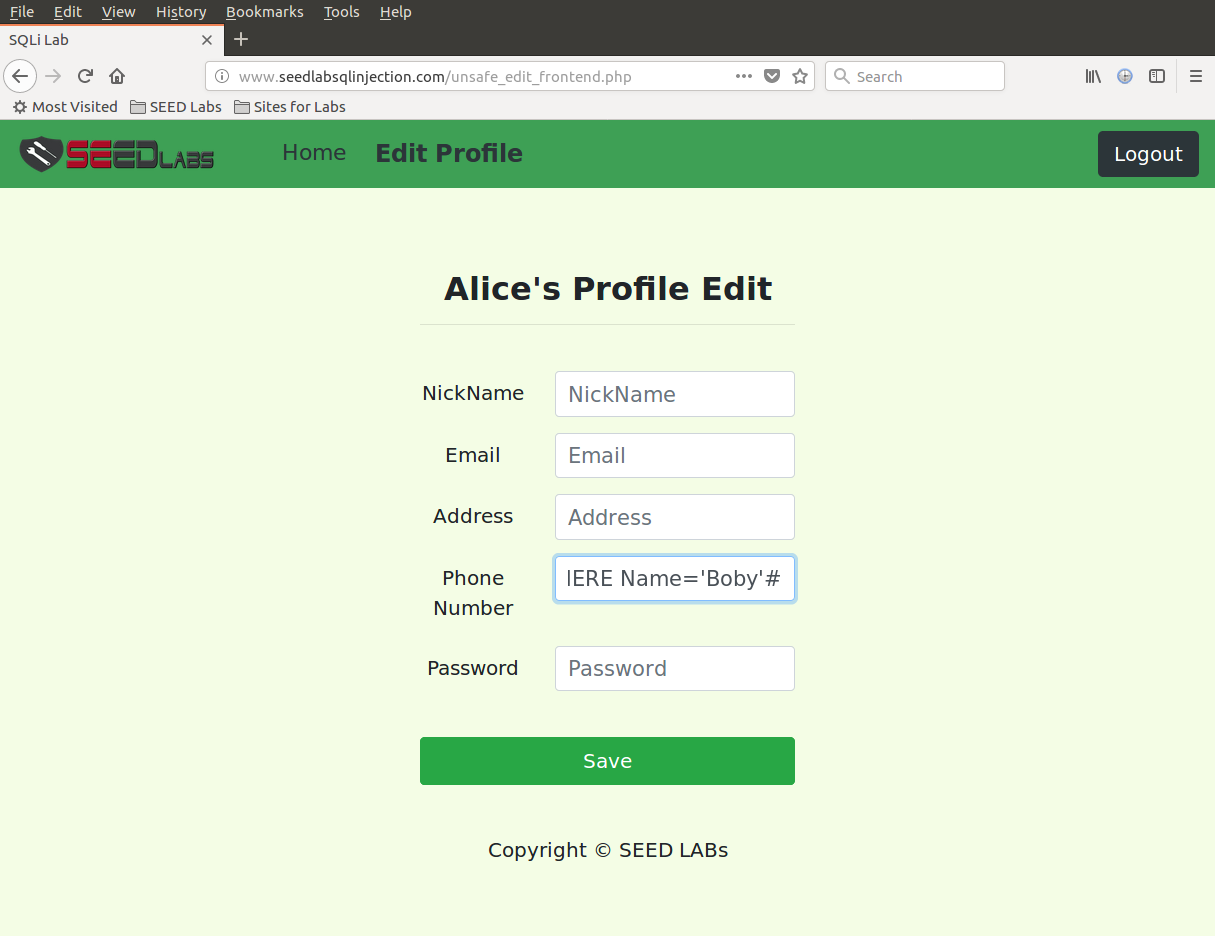
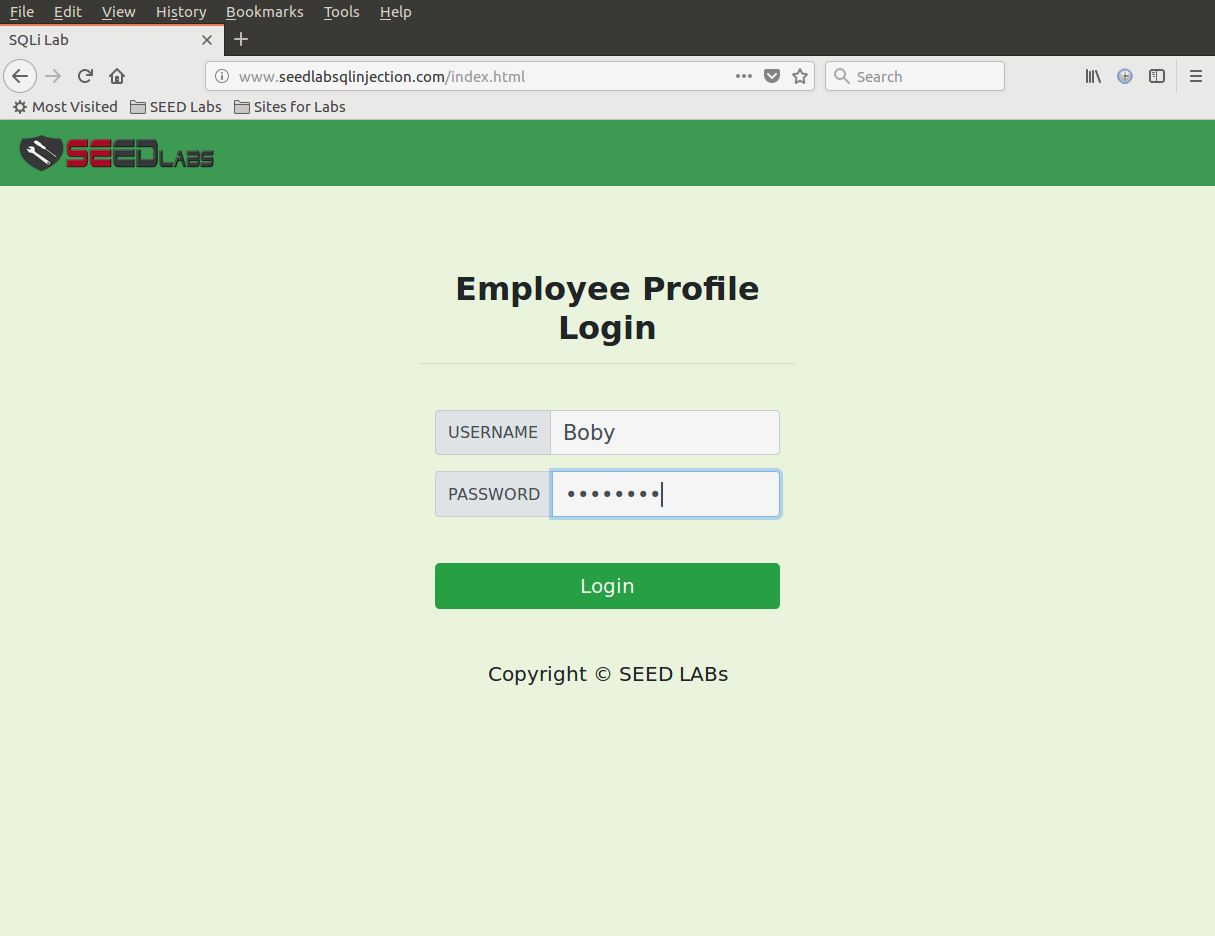
**Task 3.2** – Modify Another Person’s Salary

I used the same input but changed the salary value to 1 and the name to ‘Boby’. After logging in as the admin, we can see that the attack worked.

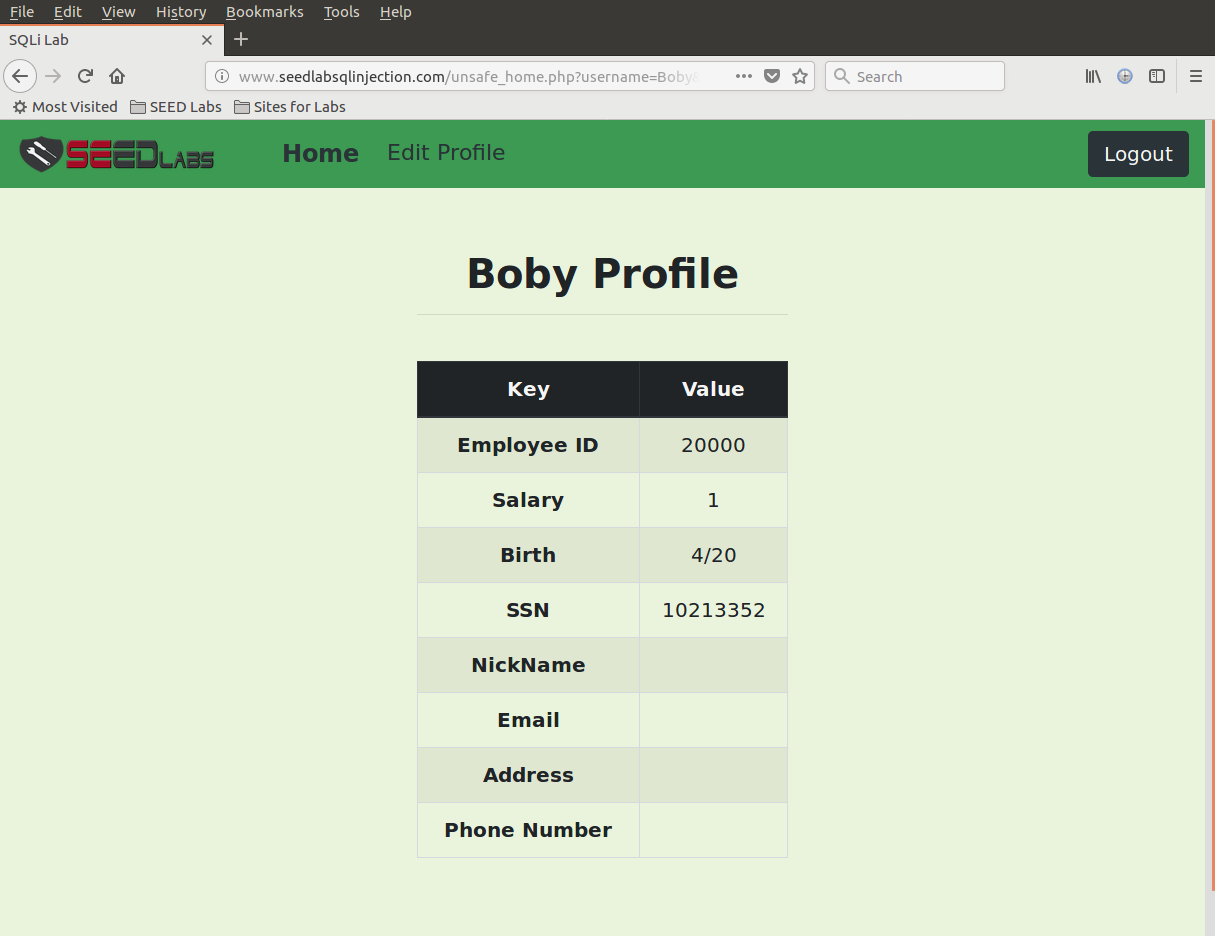
‘, salary=1 WHERE Name=’Boby’#

**Task 3.3** – Modify Another Person’s Password

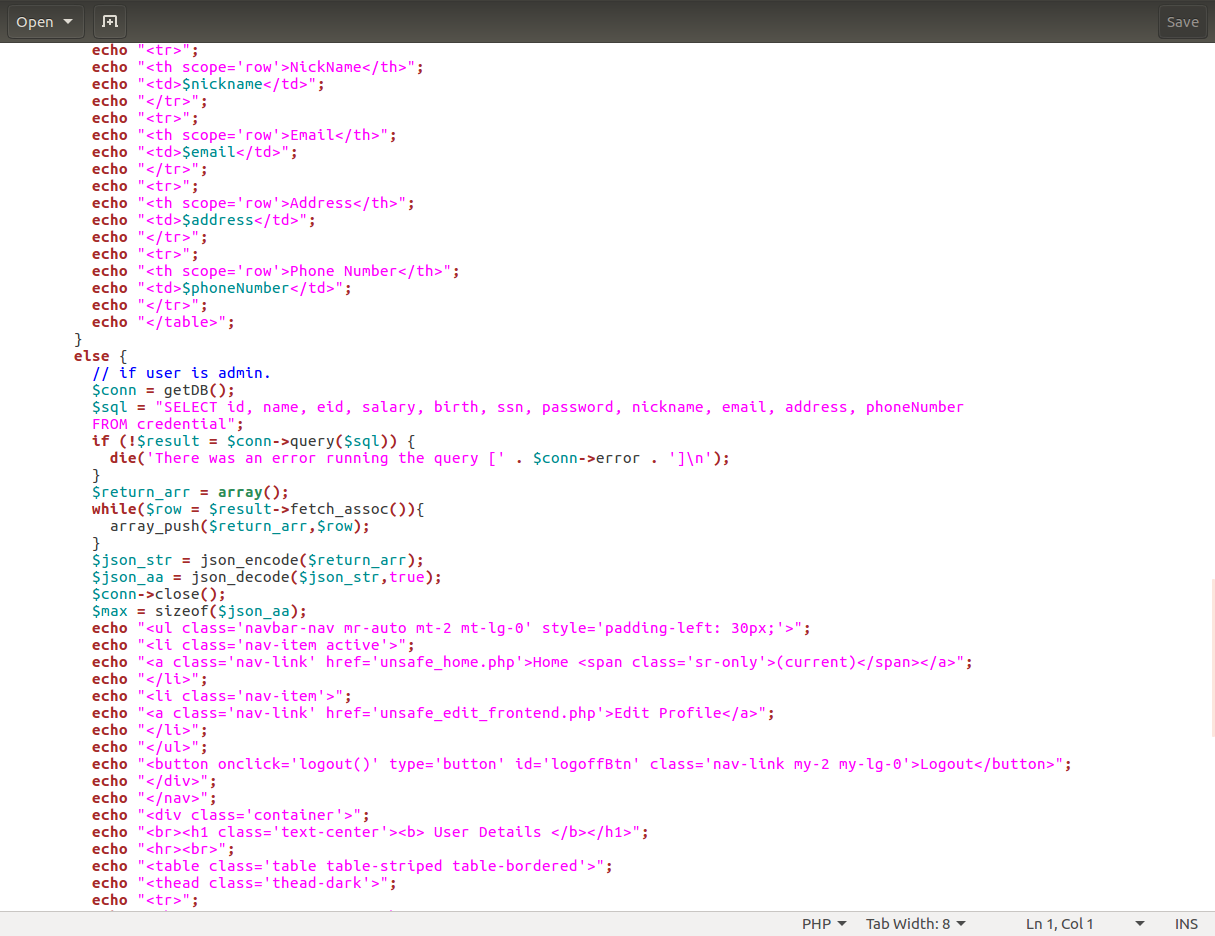
Since we know that the database stores SHA1 encoded passwords, we just have to encode the password in our SQL statement:

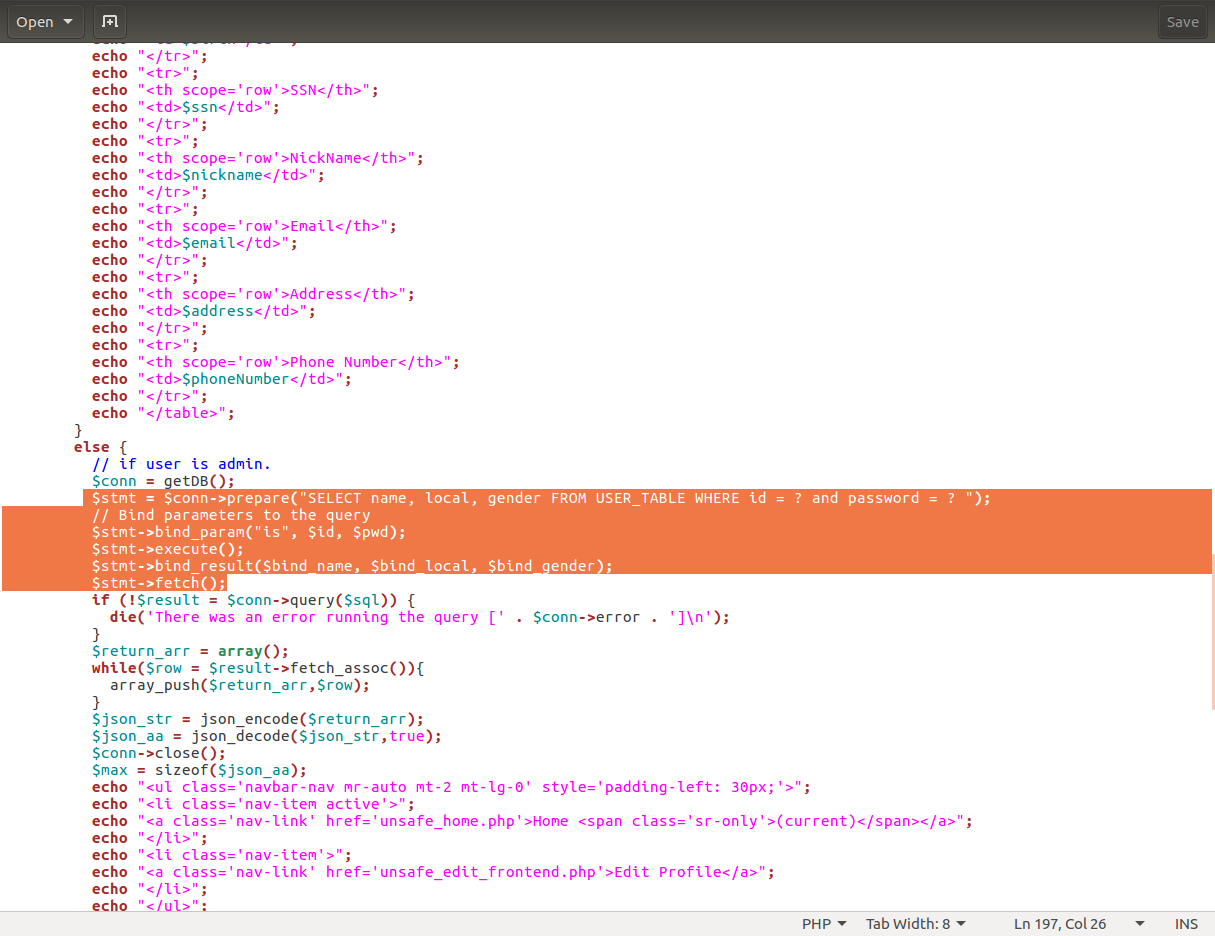
', Password=sha1('password') WHERE Name='Boby'#

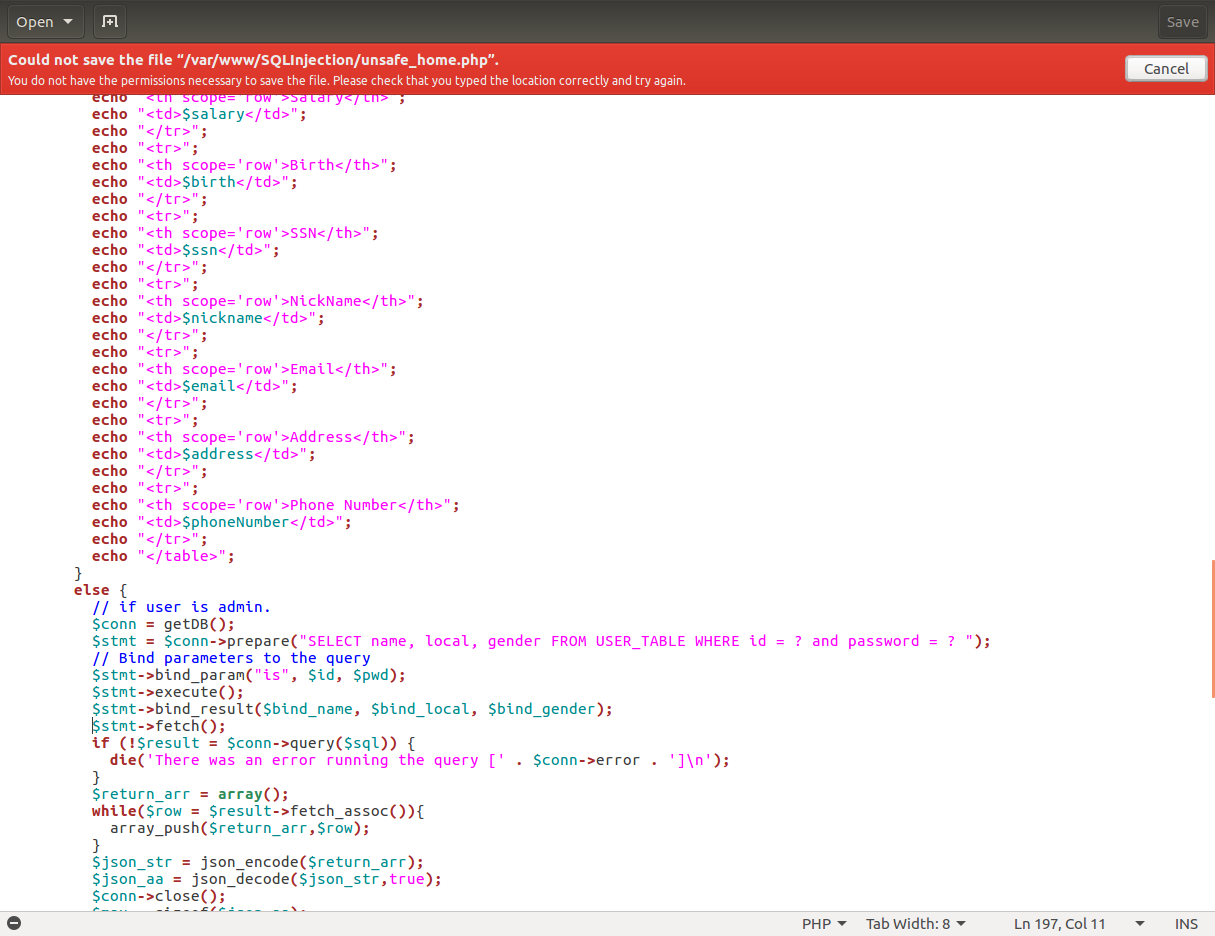


We can see that we were able to login to Boby’s profile using his new password.

## Task 4 – Countermeasure (Prepared Statement)







I changed the unsafe\_home.php to use the prepared SQL statement, however it gave me an error when I tried to save it saying that I was not authorized to overwrite the existing file, and was therefore unable to test if the vulnerability was fixed.