SQL-INJECTION VULNERABILITY AND DEFENSE

Database Assignment#3

OCTOBER 30, 2024

SUBMITTED TO: HINA BINT E HAQ

Submitted by: i22-1609, i22-1725

Contents

Introduction

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 the web applications 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. The SQL injection attack is one of the most common attacks on web applications.

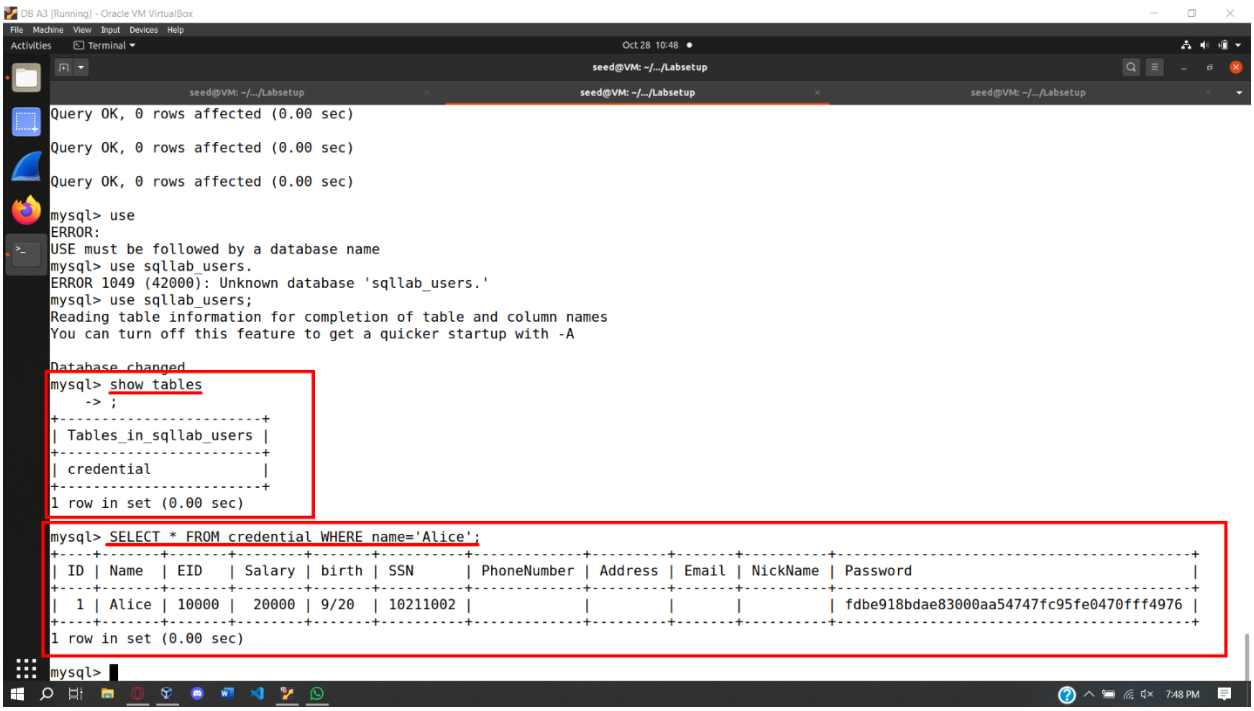
In this Seed Lab we were required to configure the seed lab setup that was given at provided link in the assignment document. Our goal was to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks.

Steps & Explanation

To start the lab modules, we first needed to run the docker to compile the modules of the given lab.

Task 1: Get Familiar with SQL Statements

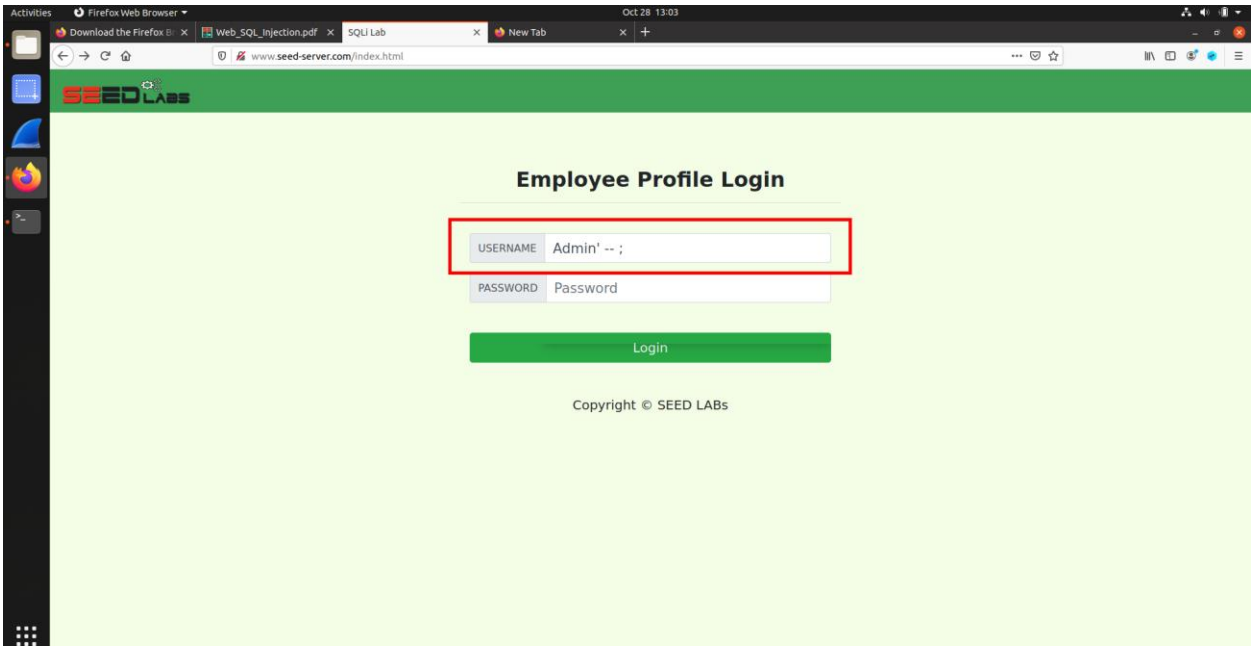
As given in the lab manual, we logged in MySQL using the credentials/commands 'mysql-u root-pdees ' and then executed the command *show tables* which show us all the database tables. The only entry was Alice and all their data in the database.



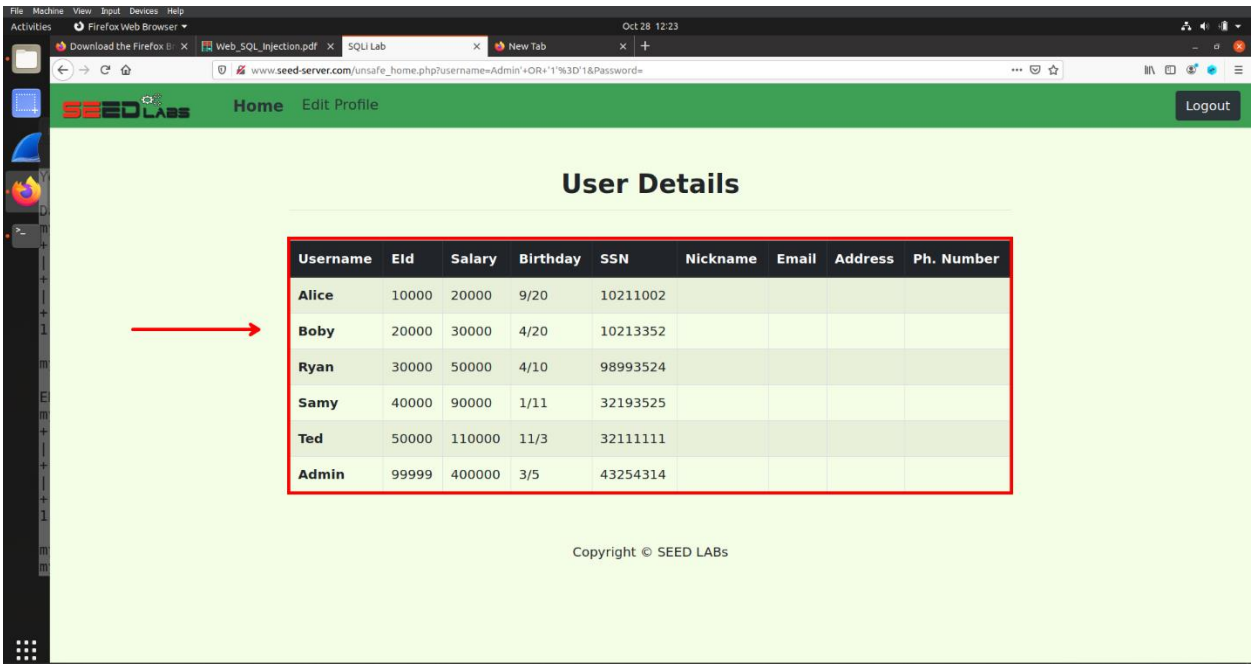
Tasks 2 & 3: SQL Injection Attack on SELECT Statement & Attack from webpage

In the second task, we did the most common and easiest SQLi query "*Admin' -- ;*" to attack the SELECT statement on the backend side of the webpage responsible for fetching the data from the database.

The apostrophe (') is used to end the string which will be used as username and the (- -) are used to comment whatever comes after the username. With this the query will always be true even if the password is empty or wrong input is given.



So, after giving this as username, the website returns all the user's data without any error because according to the backend server, the query contained no error and was able to be processed without any error.



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				

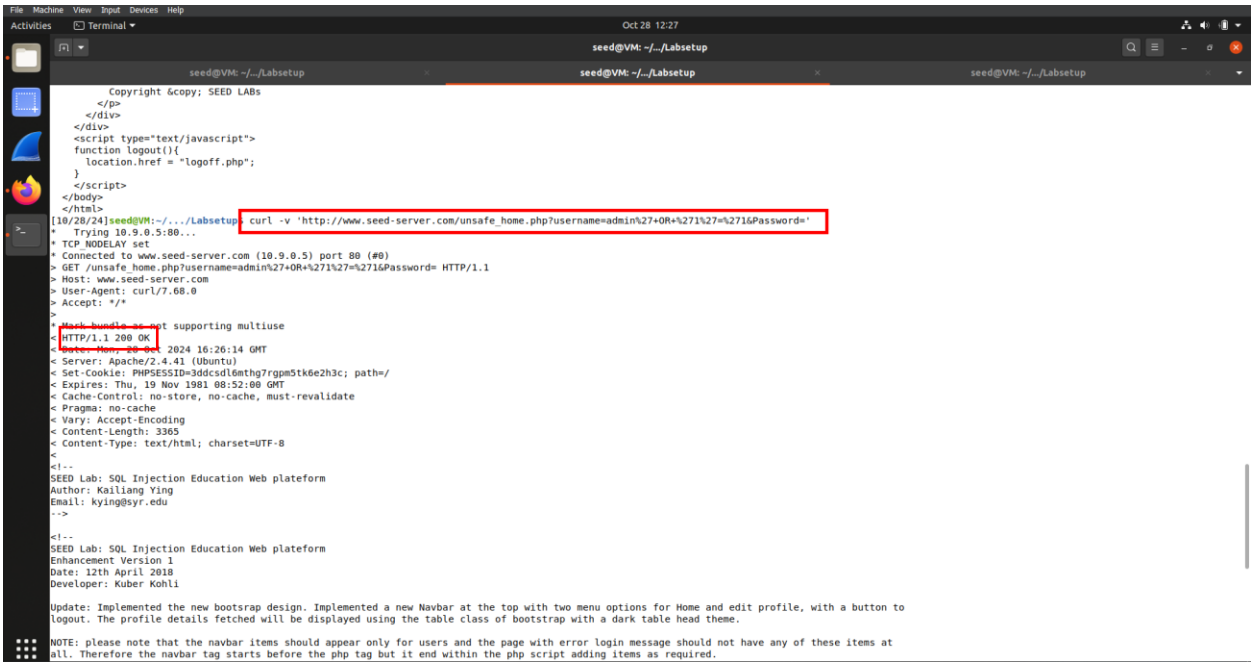
Task 4: SQL Injection Attack from the command line.

This task required us to generate the web request from the command line using curl or wget utilities.

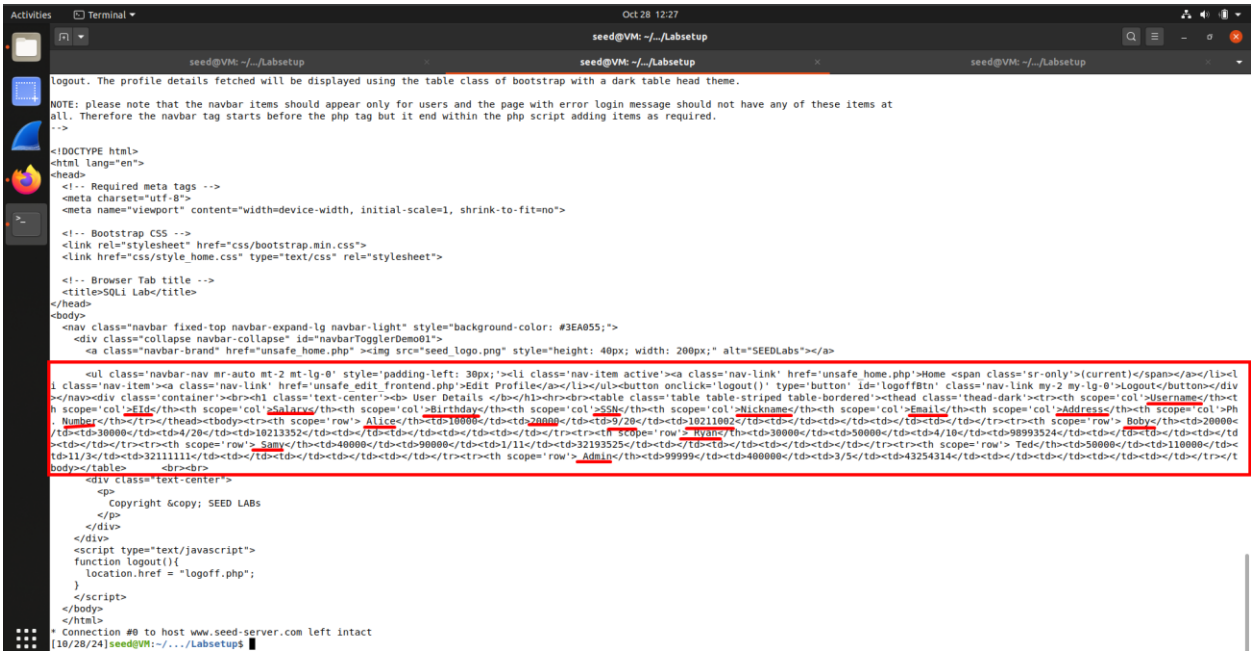
"*Curl -v 'http://www.seed-server.com/unsafe_home.php?username=admin%27+OR+%271%27=%271&Password=*'"

Using this command we invoked a request that attacked the database using URI instead of directly inputting the SQLi in the login form.

- *%27* is used for apostrophe(') in web requests.
- + is used for space.
- OR is used as OR operator in SQL queries which will make the right part of the query which will become 1=1 true even if we give the wrong username.
- & separates the username and password.



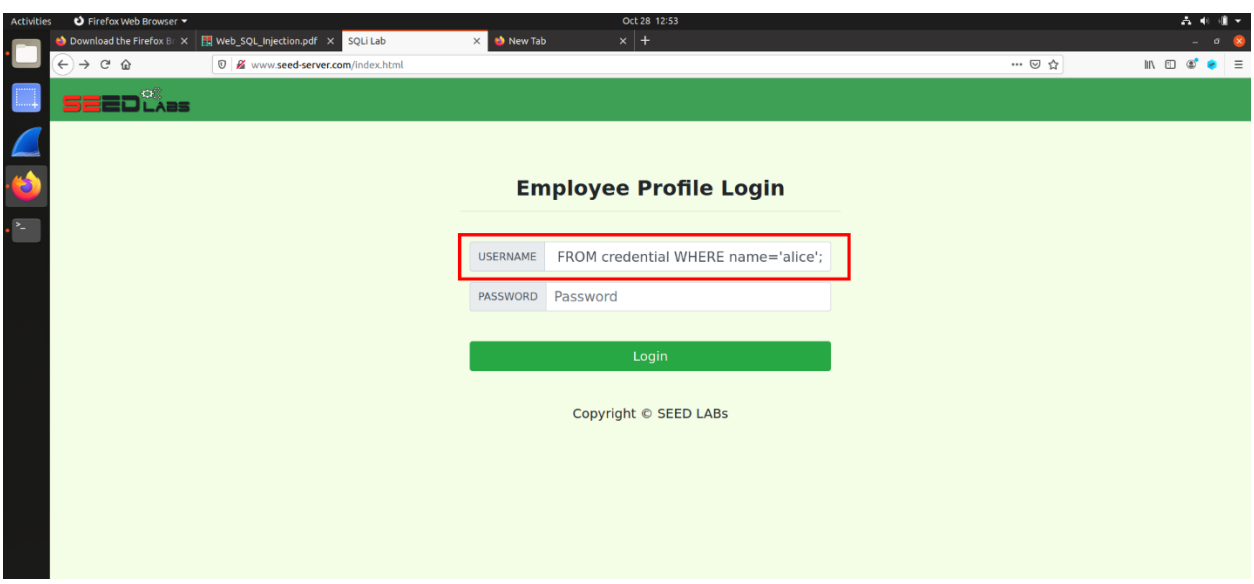
With this, we get a response '*200 ok'* from the telling us that our request was processed and we got a valid response which we can use to further carry out our attack.



Task 5: Append a new SQL statement

This task was rather simple. We needed to give two queries rather than one to see if they work and they did not work.

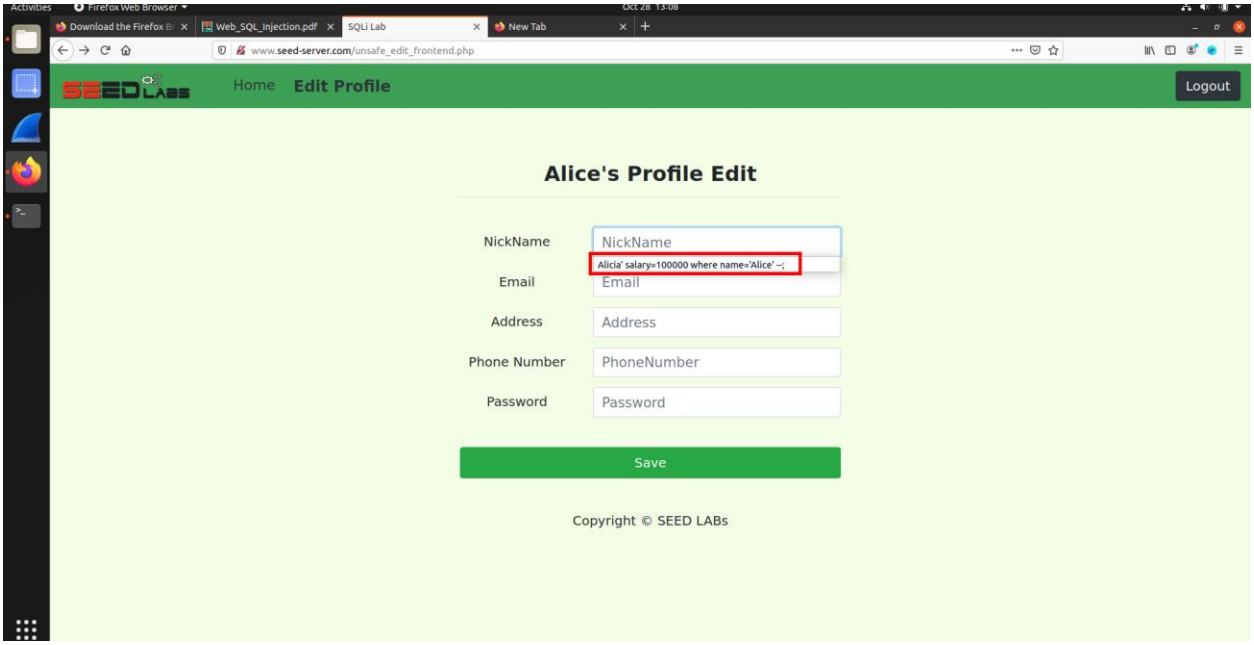
The query we inserted was " *Admin' OR '1'='1; DELETE FROM credential WHERE name='alice';"*. In this we try to login as admin as well as Delete Alice's credentials but this does not work. One way to run multiple queries is to use UNION keyword in our quries but that is a little different.

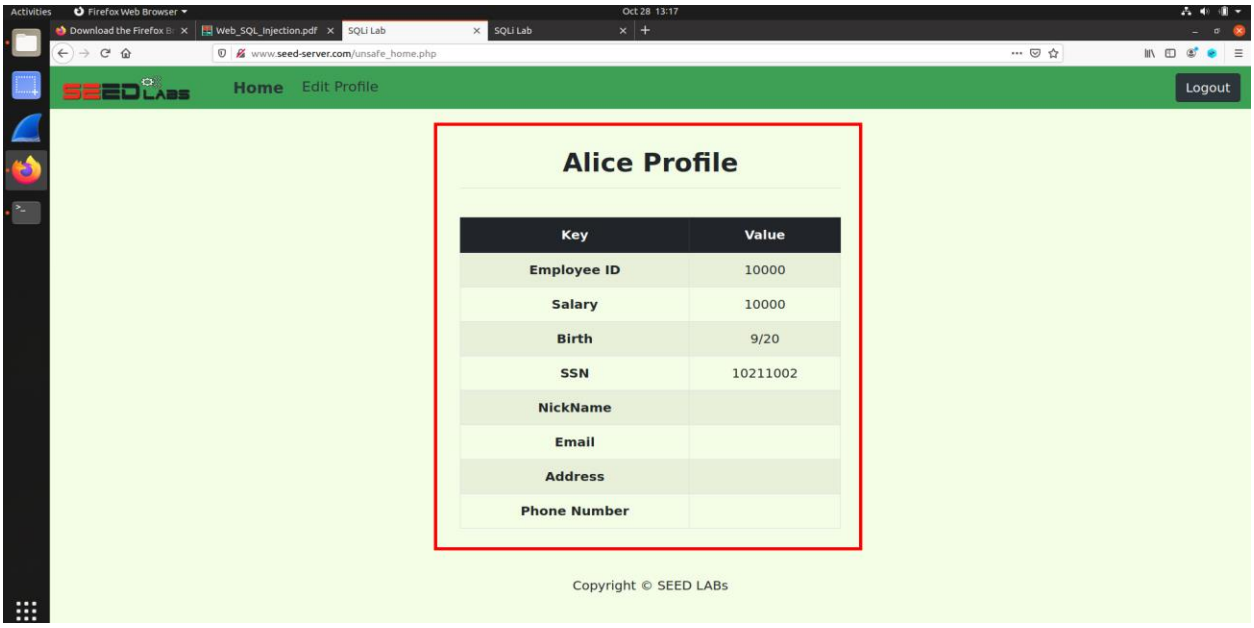


Task 6 & 7: SQL injection Attack on UPDATE Statement & Modify your own Salary

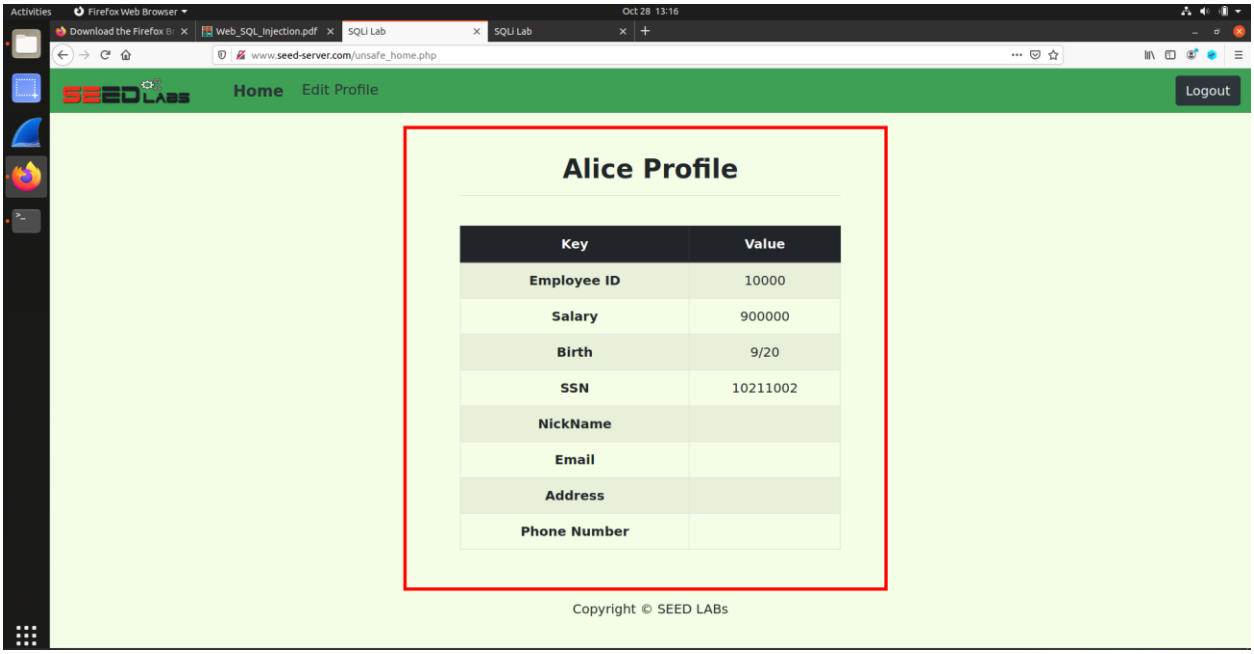
The task for this was to update the salary of Alice using SQLi. We logged into Alice's account from where we got access of the profile where we can edit certain fields.
We know that the website uses MySQL database, so we only needed to insert the a valid SQL injection query which will change the salary of Alice.

Alice', salary=100000 where name= ‘Alice’ -- ;





1 Before

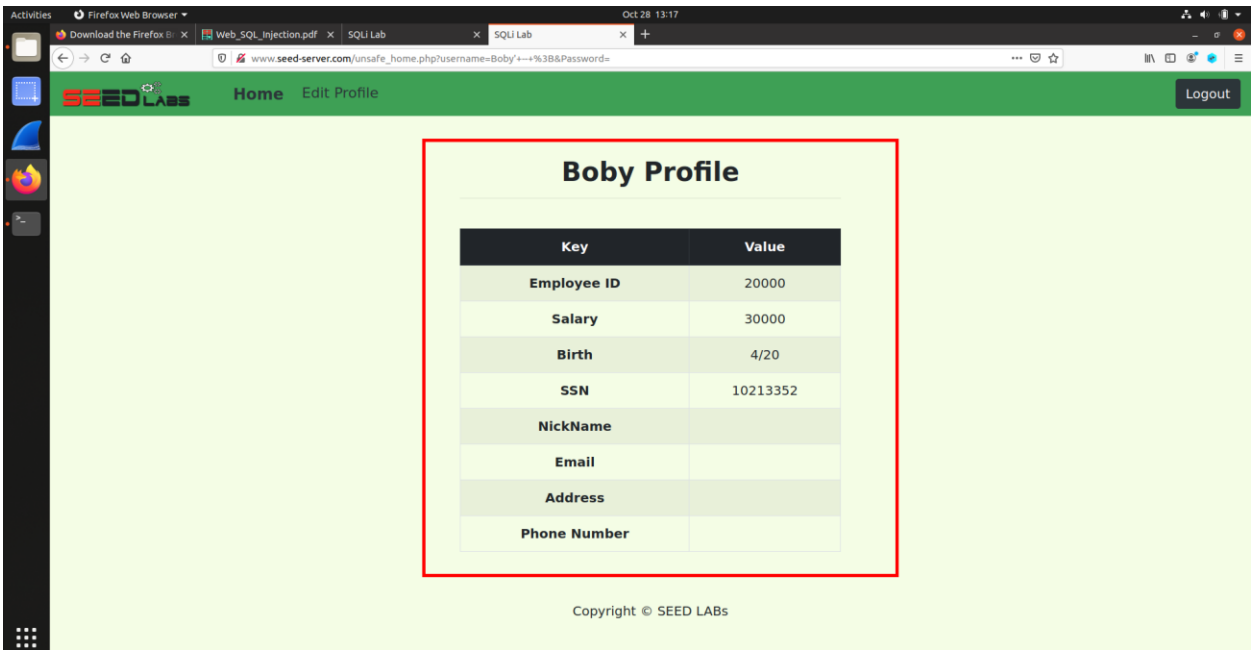


2 After

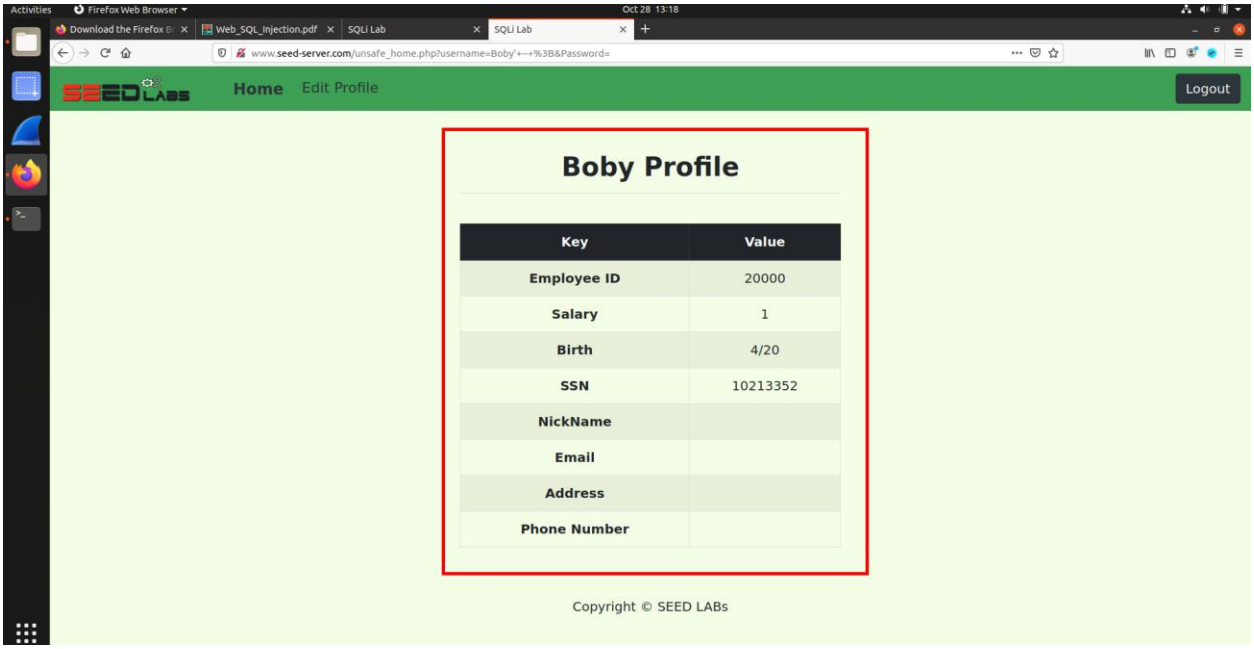
Tasks 8: Modify other people's salary

', Salary=1 where name='Boby' #

Using this query we modified Boby's Salary to 1.



3 Before

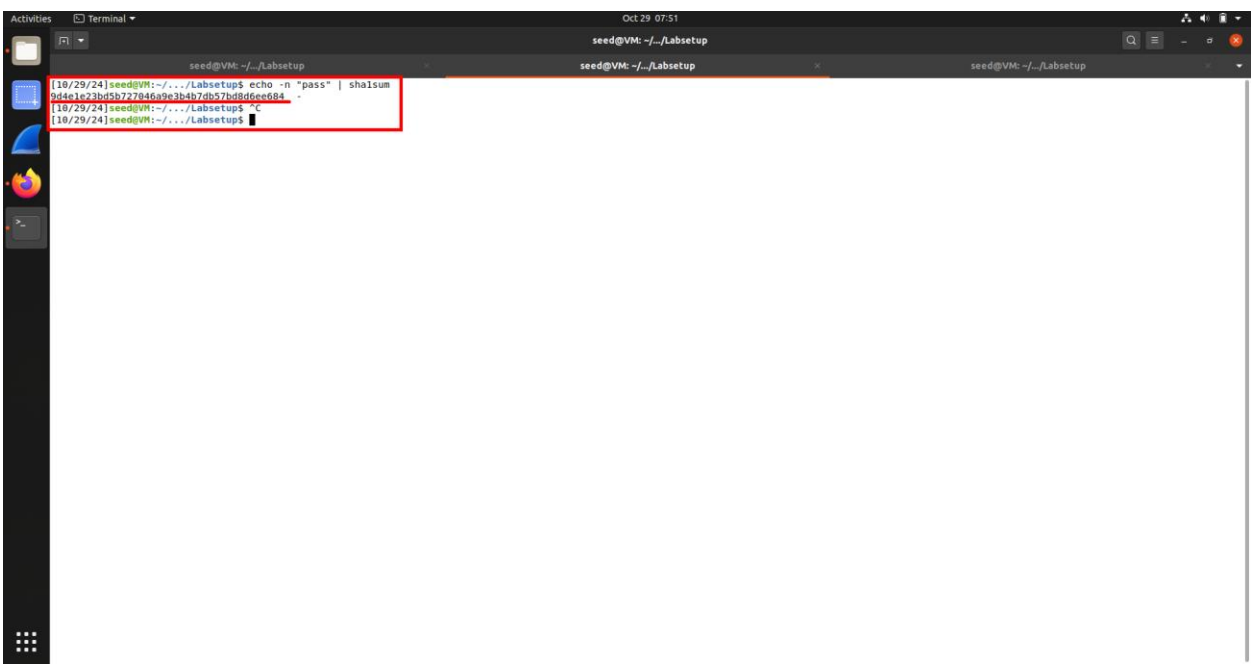


4 After

Task 9: Modify other people's password

In this task we started with generating with a password hash using sha1 hashing algorithm using the command *echo -n "pass"* | *sah1sum*.

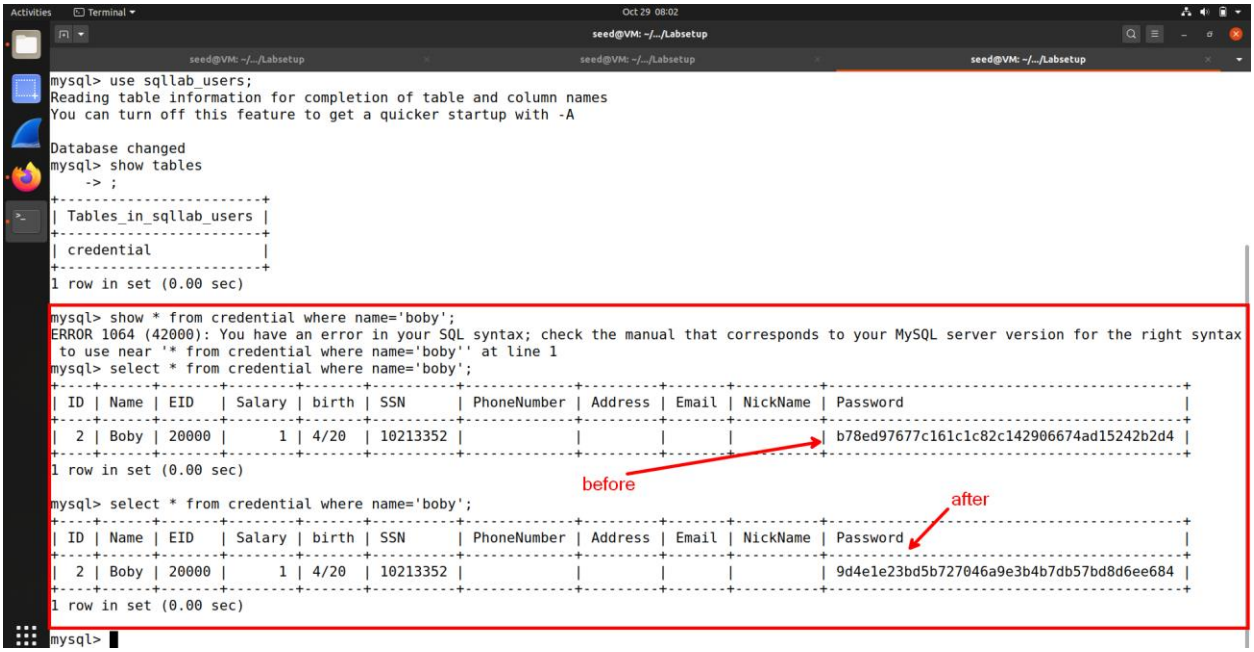
The sha1 hash we got for the text "pass" was **9d4e1e23bd5b727046a9e3b4b7db57bd8d6ee684.**



After generating the hash we need to use into SQLi to change the password in the database. To our ease we already knew that the database stores hashes and uses sha1 hashing to hash them. So the next thing was to make the query which is given below.

', Password='9d4e1e23bd5b727046a9e3b4b7db57bd8d6ee684' WHERE name='Boby' #

Using the Update statement we changed the password of Boby. You can see the before and after of the SQL request.



Task 10: Countermeasure — Prepared Statement

To countermeasure SQL Injections, we strictly need to change the way the server or in this case, the backend receives the requests. For that we changed the that were sent to the database.

$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn FROM credential WHERE name = {$name} AND Password = {$password}");

In the above statement, the server would send the user input as it is the user would input without any checks. So the user was able to Inject harmful queries. To mitigate this we changed how the server would send the queries to the database.

$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn FROM credential WHERE name = ? AND Password = ?");
$stmt->bind_param("ss", $input_uname, $hashed_pwd); // Bind parameters to the query
$stmt->execute(); // Execute the query

We changed the above command such that the user input isn't directly sent to the database, rather it is first stored in some variables on the server side and then added to the query when sent. What this does is whatever the user inputs in the username and password field are strictly perceived as the username and password and not as extended query.