CS445 Lab Report: SQL Injection Attack

Austin Decker

November 01 2024

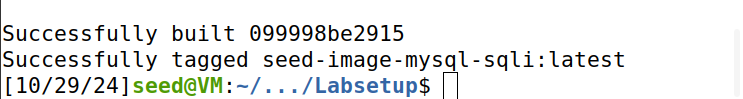
# Environment Setup:

## Address Mapping:

I added the following address `10.9.0.5 [www.seed-server.com](http://www.seed-server.com/)` mapping to the *//etc/hosts/*

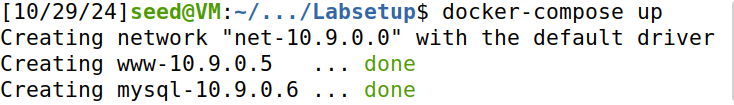
## Container Setup:

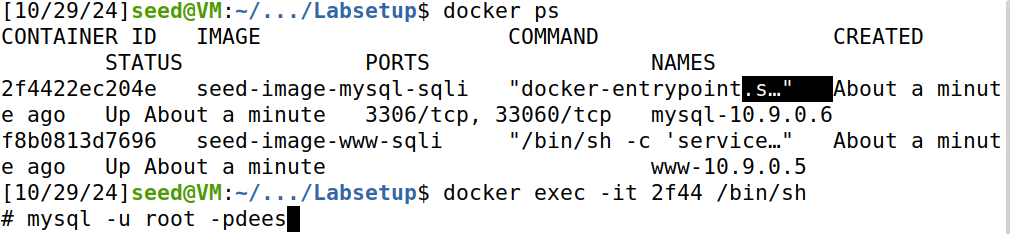
I ran docker-compose build in the terminal:



# Task 1: Getting Familiar with SQL Statements:

Getting inside mysql docker container shell:



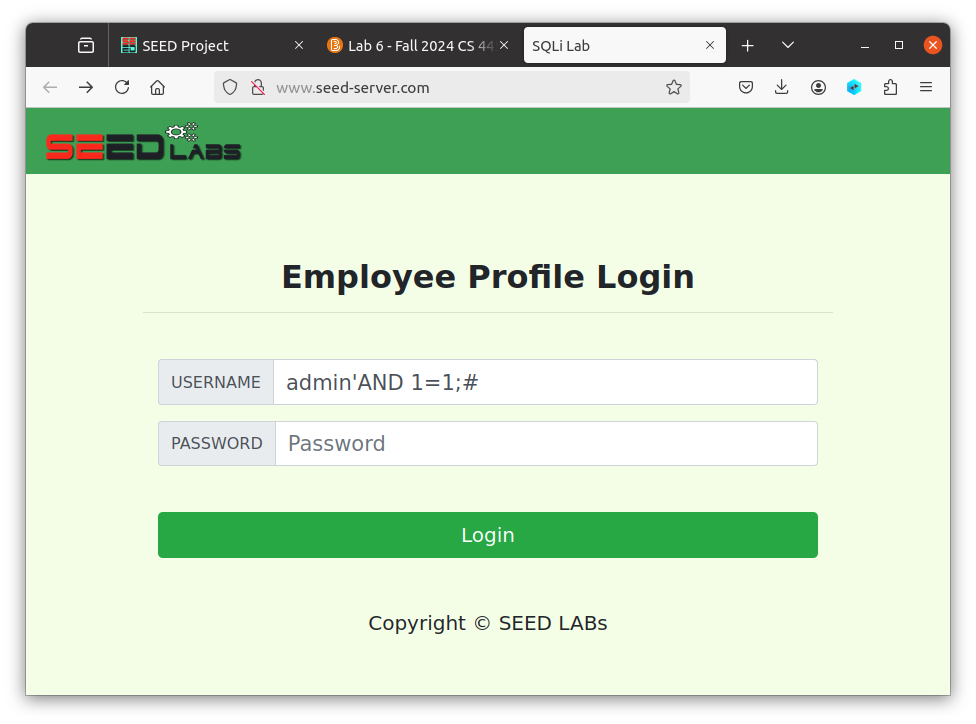


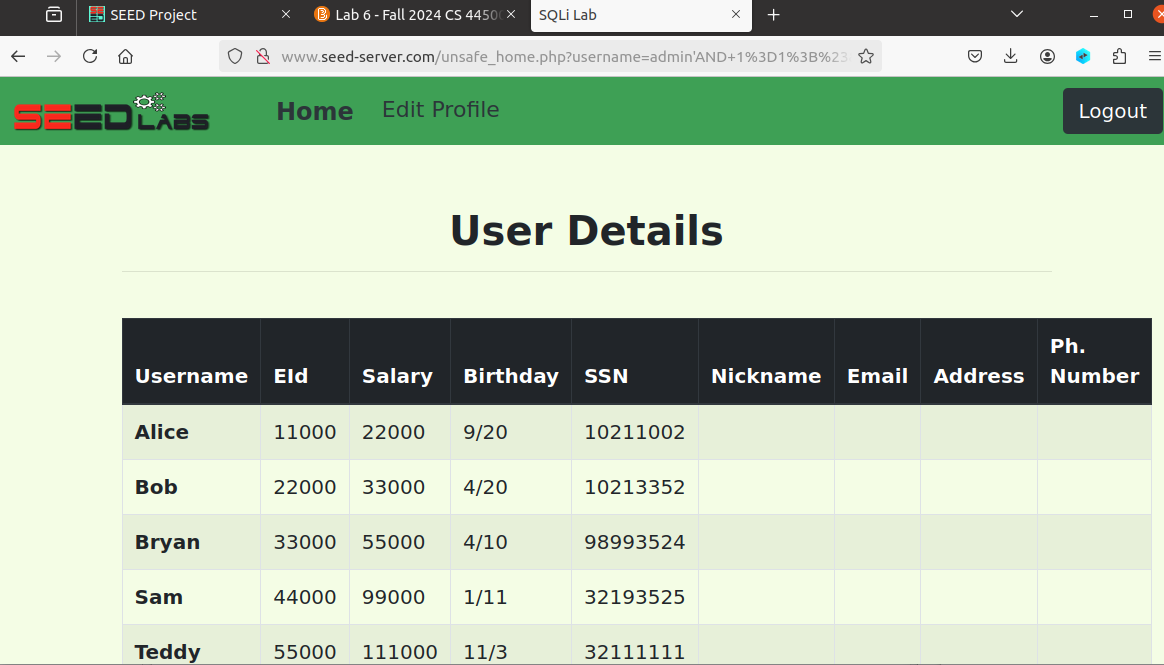
Switching to sqllab\_users db and printing out the rows from credential:

# Task 02: SQL Injection Attack on SELECT statement

## Task 2.1: SQL Injection Attack From Webpage

To get the admin panel, I put in the string [admin’AND 1=1;#]

Result of SQL Injection:

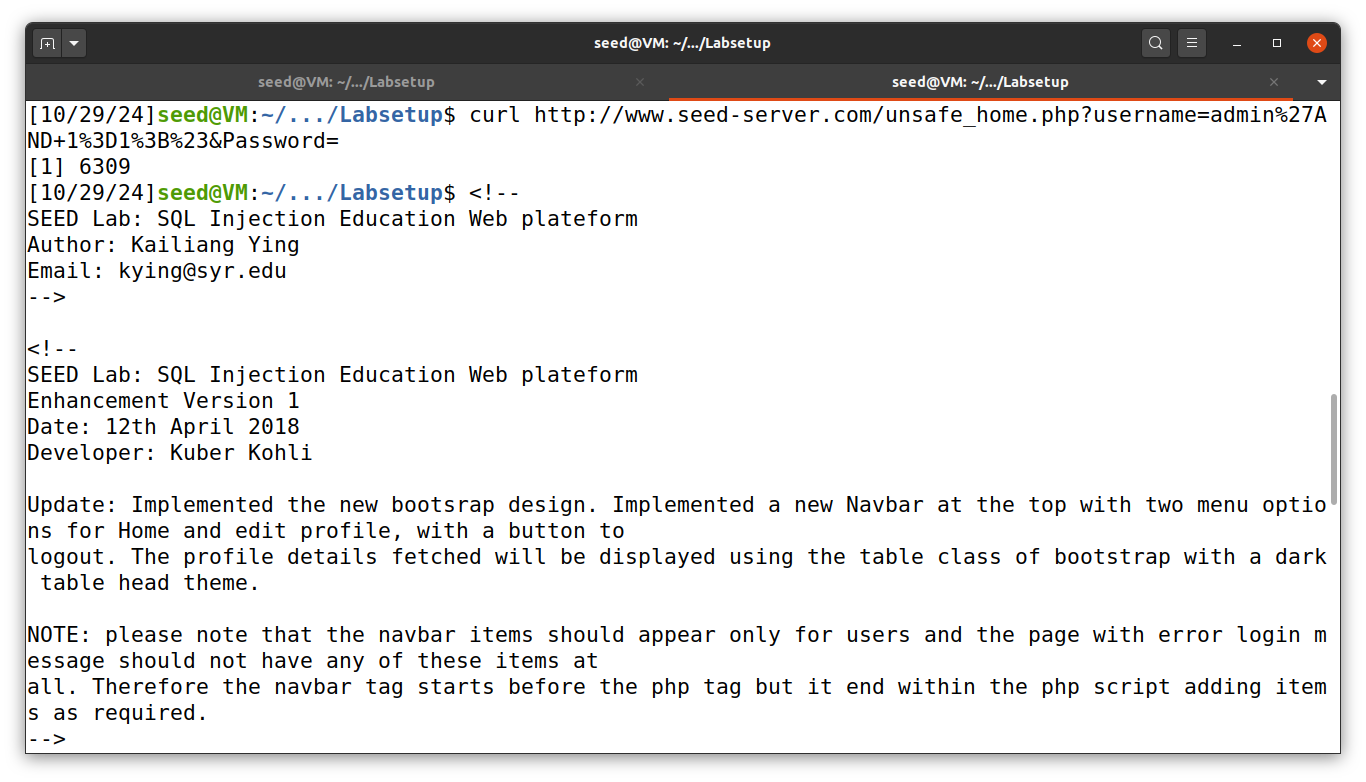


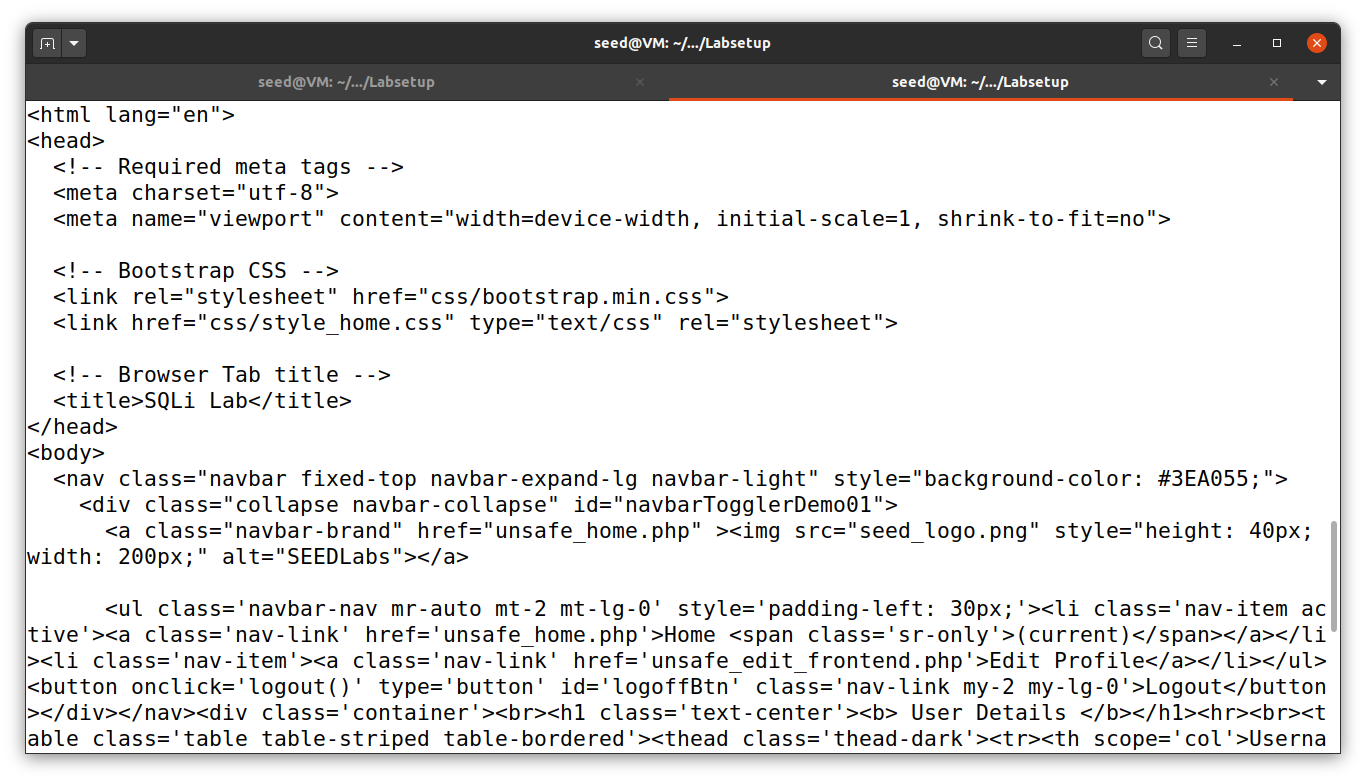
## Task 2.2: SQL Injection Attack From Terminal

Using Curl Command in Terminal to gain access to the admin dashboard:



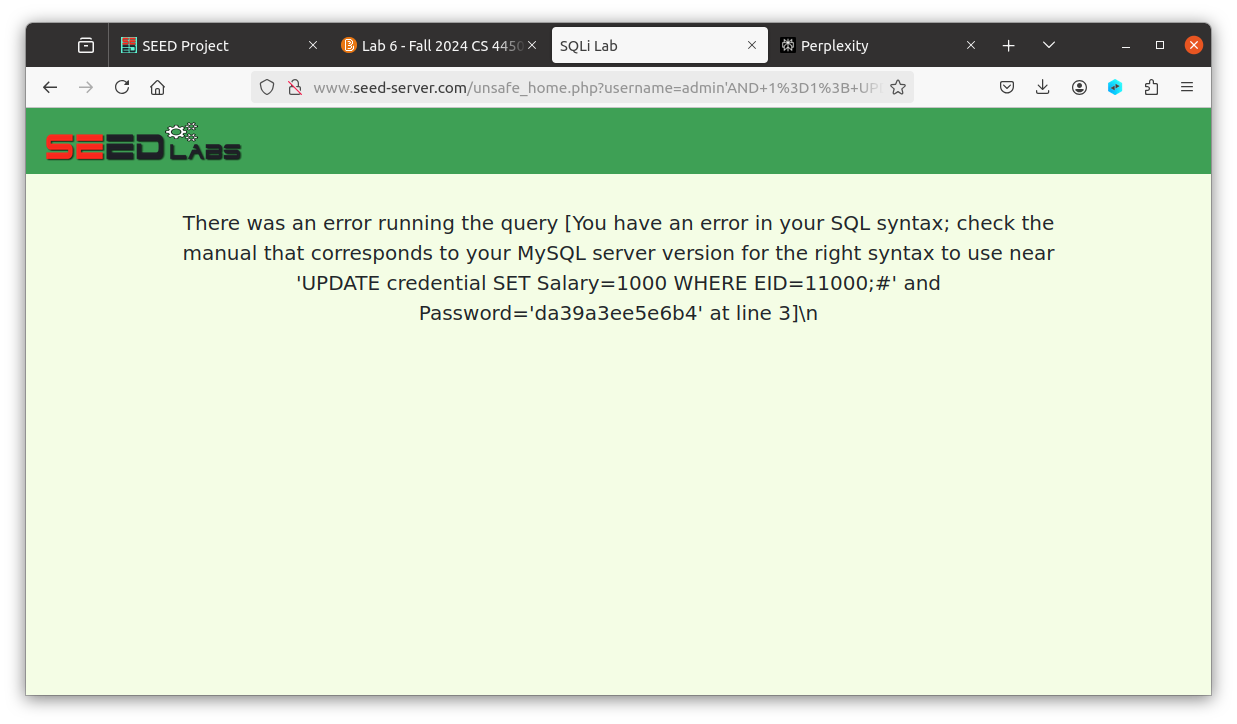
Results from Successful SQL Injection:



  
Output is too big to show in screenshots.

## Task 2.3: Append a new SQL statement

I tried appending another SQL statement after the first one but I got an error:

admin'; UPDATE credential SET Salary=1000 WHERE EID=11000;#

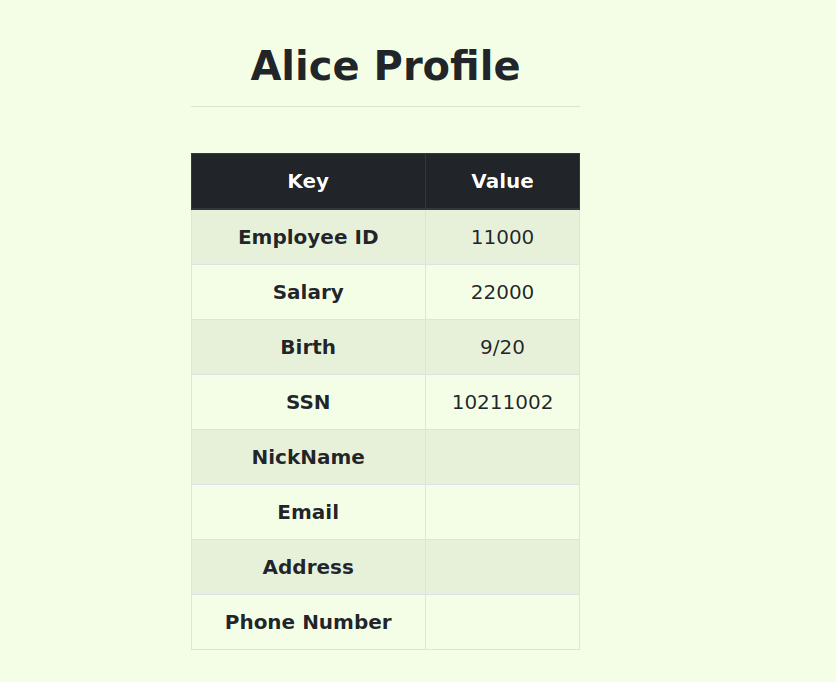
**Observations:**

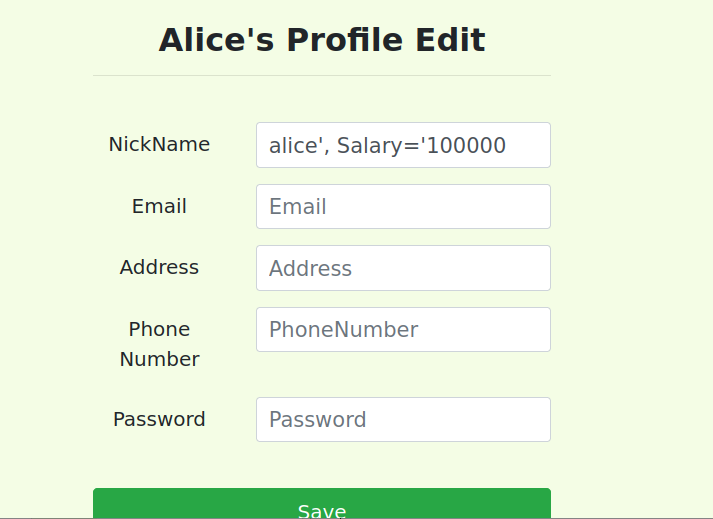
The reason this did not work was because of the php mysqli query api. It does not allow multiple queries by default. If we want to be able to do multiple queries, the code must use multi\_query().

# Task 3: SQL Injection Attack on UPDATE Statements

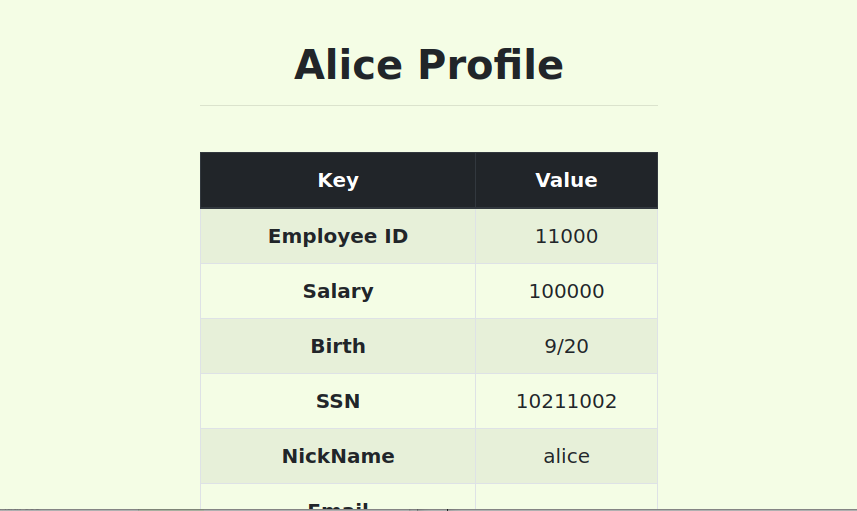
## Task 3.1: Modify Own Salary:

Alice’s Profile before SQL injection:

SQL Injection String:



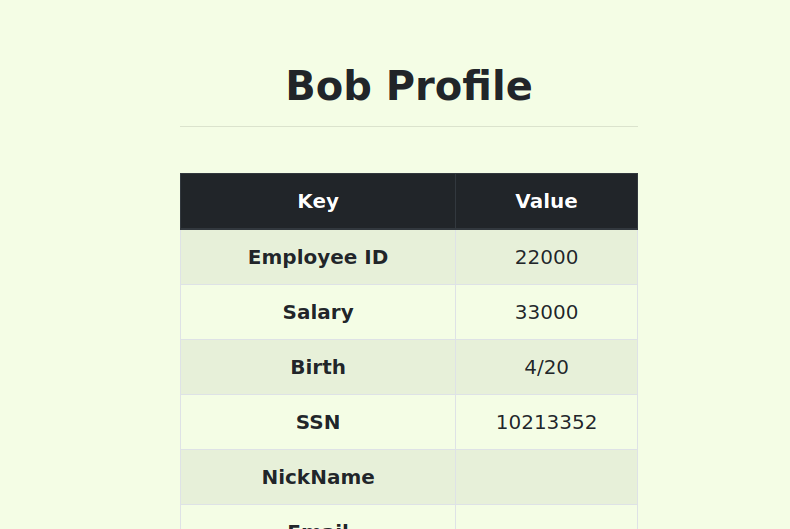
Alice’s Profile after SQL injection:



As you can see from the screenshots above, I successfully managed to update Alice’s Salary.

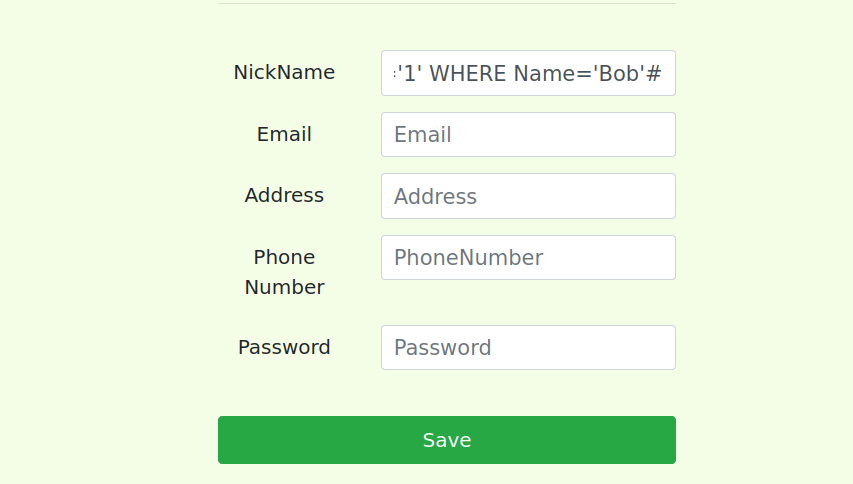
## Task 3.2: Modify other people’s Salary

For this task I will use Alice to change Bob’s Salary. Here is Bob’s Profile before the SQL Injection:

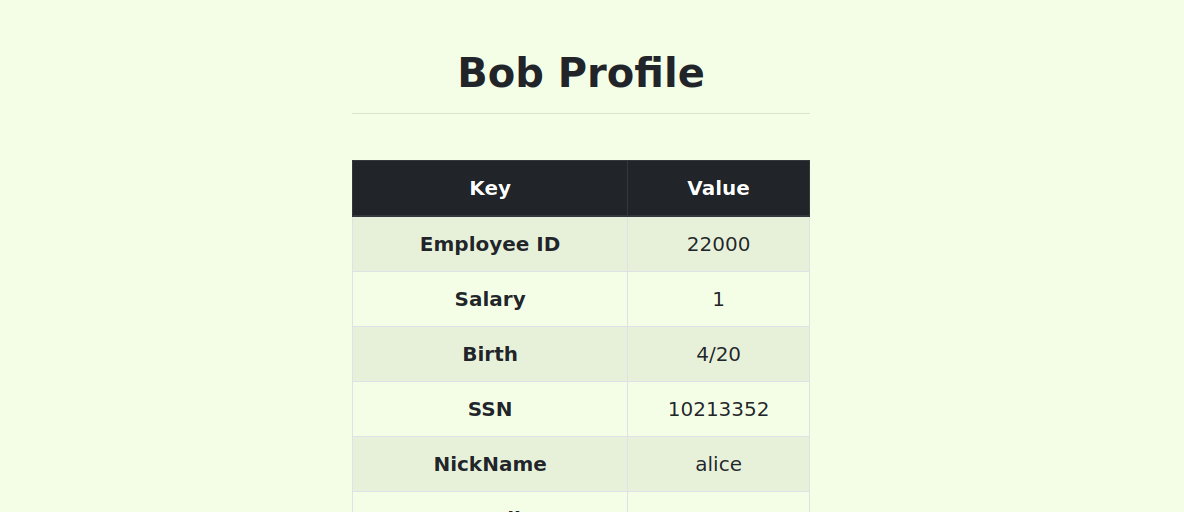


Alice SQL Injection:

**alice', Salary='1' WHERE Name='Bob'#**



Bob’s Profile After SQL Attack:

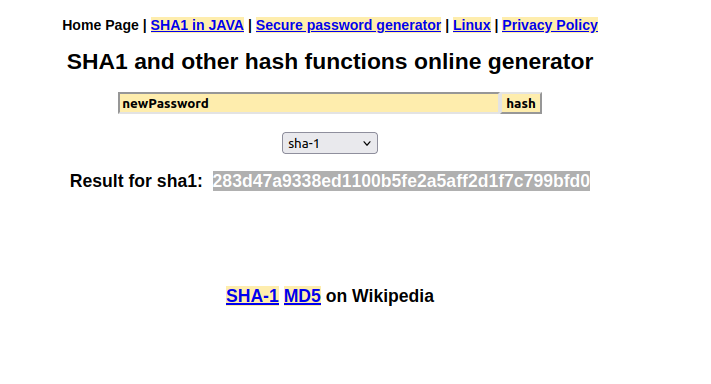


As you can see above, as Alice, I was able to change Bob’s Salary. I also accidentally changed his nickname, but to fix that, I would use the string **‘, Salary='1' WHERE Name='Bob'#** instead.

## Task 3.3: Changing Other peoples’ password

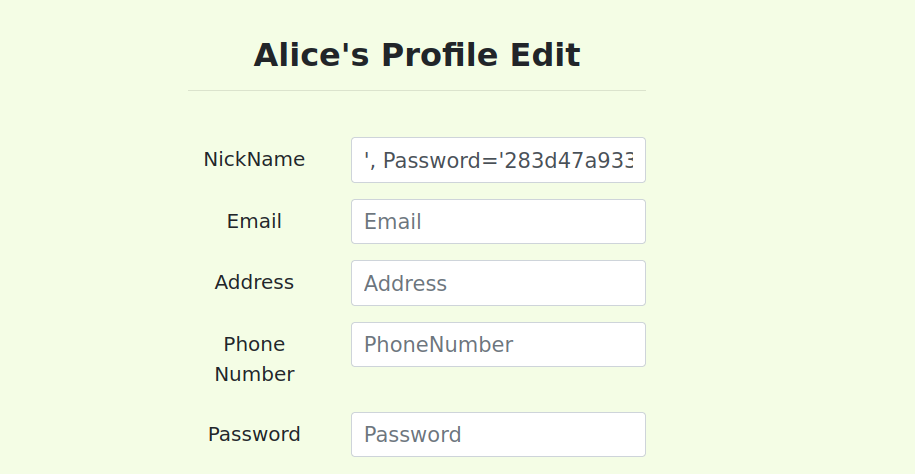
First Attempt:

I figured since the password is stored as a sha1 hash, I could use an online hashing site to hash the new password before changing it.

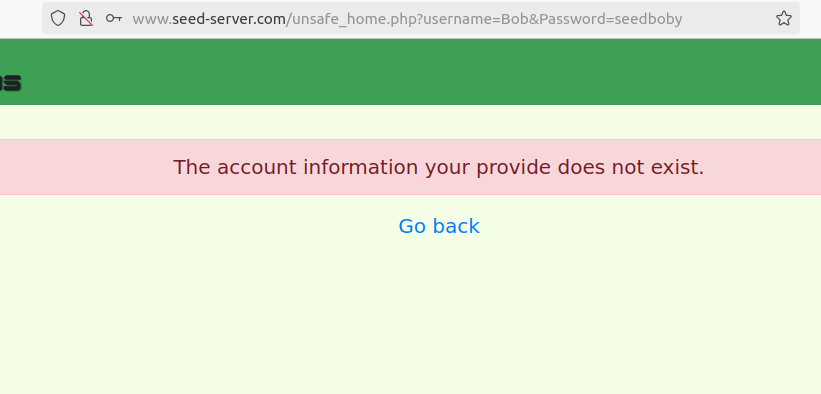


I then use the same strategy to change a field of Bob’s like in task 3.2:

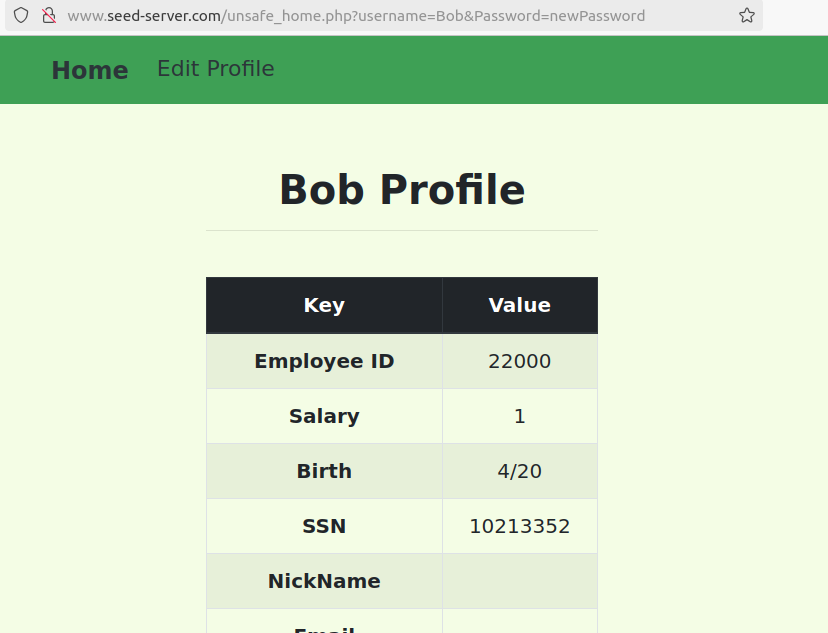
**', Password='283d47a9338ed1100b5fe2a5aff2d1f7c799bfd0' WHERE Name='Bob' #**



Now I try to login to Bob’s account using the default password “seedboby”. I get the error message:

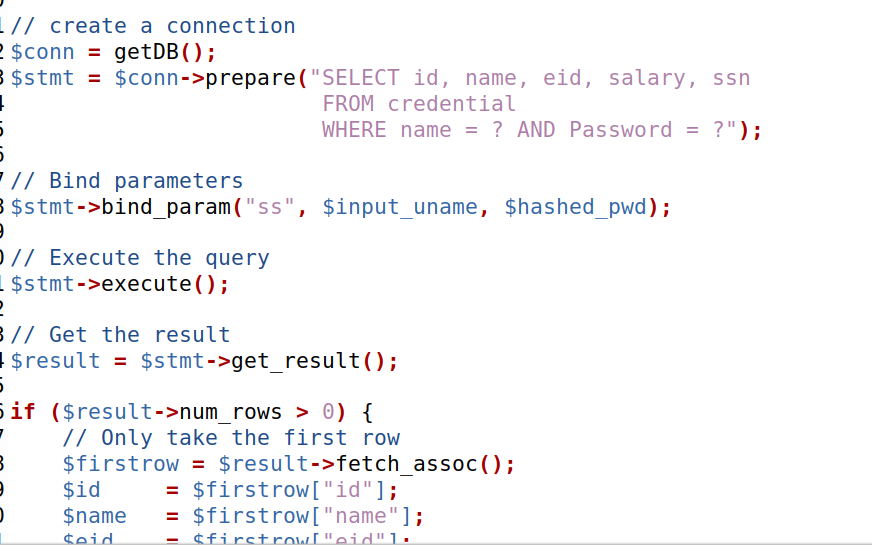


Now I login with the new password “newPassword”:

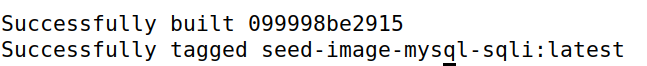
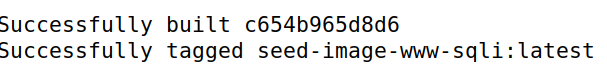
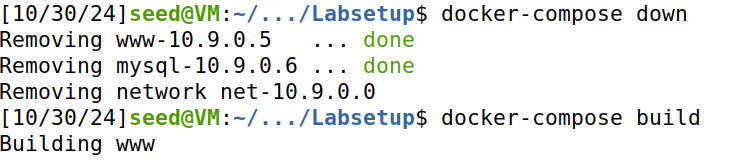
I was successfully able to switch Bob’s password with a new password. Now Alice can change any changeable information about Bob.

# Task 4: Countermeasure – Prepared Statements

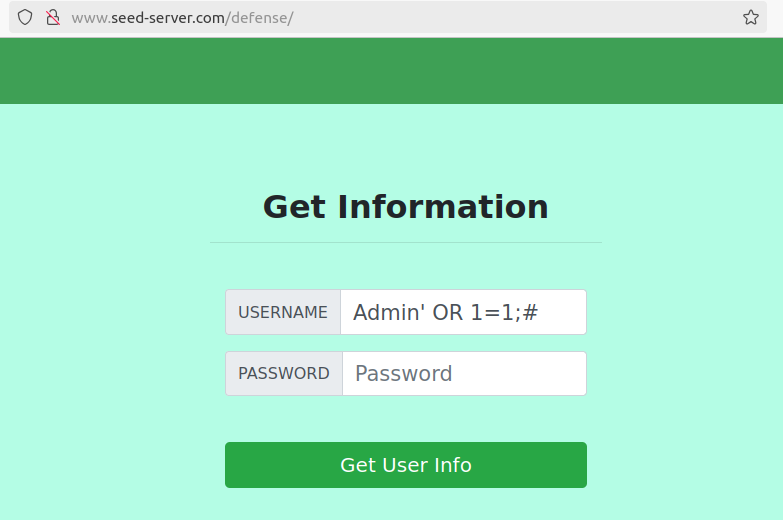
**I changed the unsafe query to a prepared statement in defense/unsafe.php:**



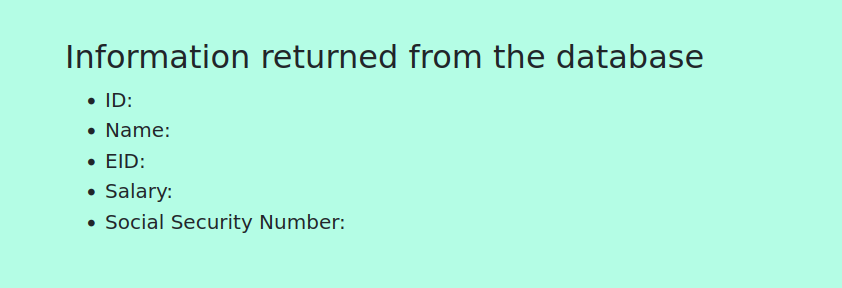
**rebuilding docker container with updated code:**



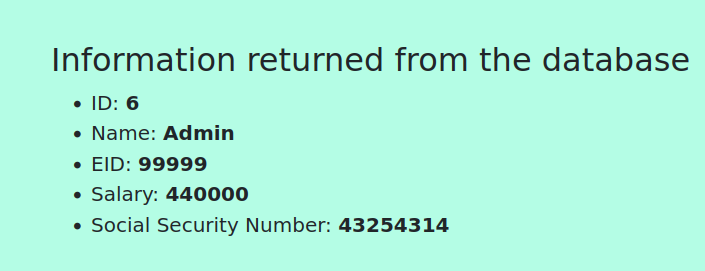
Attempting to log in as admin without password:



Result:



Properly logging into account with correct password:



As the screenshots above show, real information is shown only when the user provides a valid password and username. Trying to inject SQL to circumvent the password requirement was not successful.