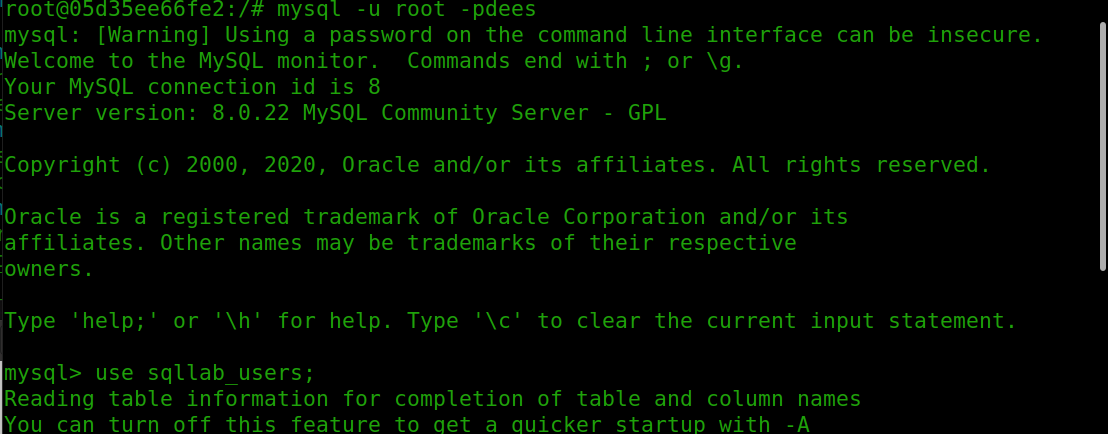
Val Robichaux

Homework 6

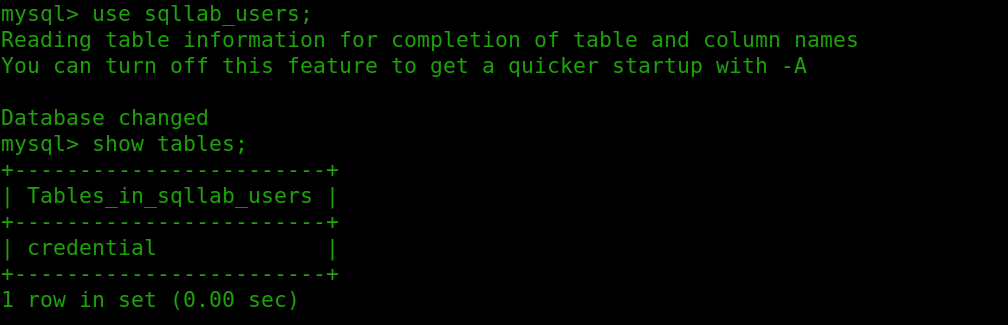
CSCE 465

# Deliverables

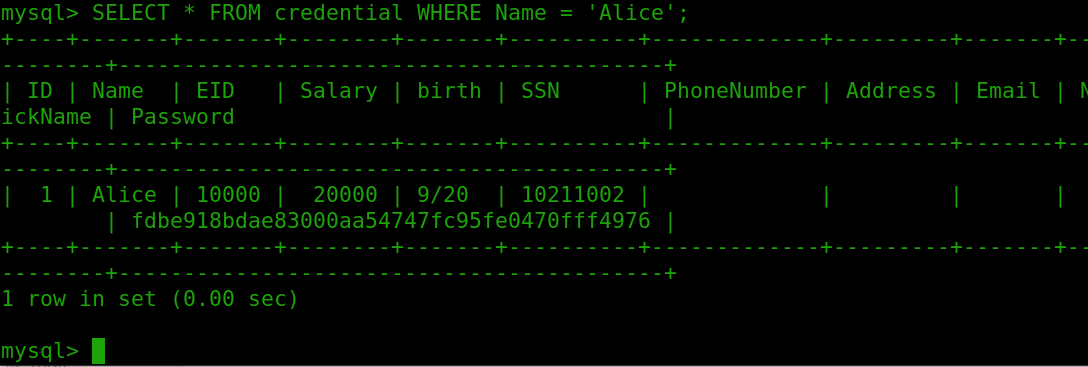
1. Get Familiar with SQL Statements



* Here I am logging into the container that contains our sql data

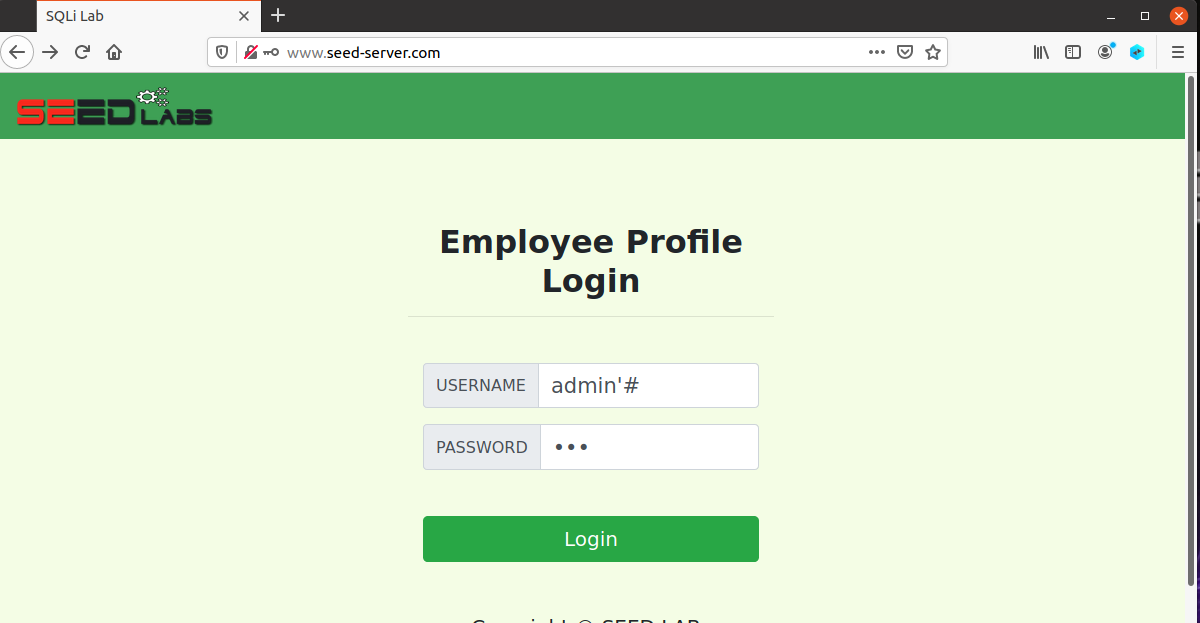


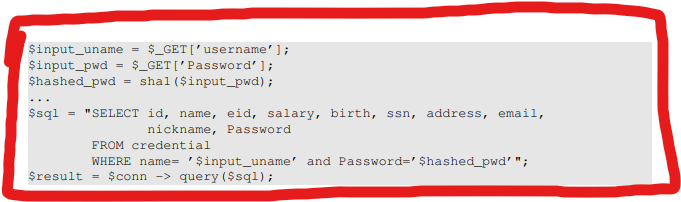
* Listing the tables and we can see that there is one table in the sqllab user’s database called credential



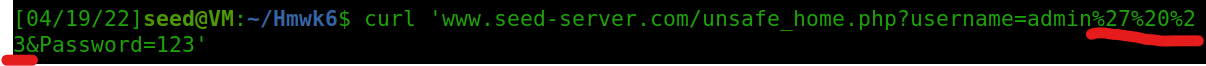
* Here we run the command to select ALL (\*) entries from the credential table where the name is Alice.

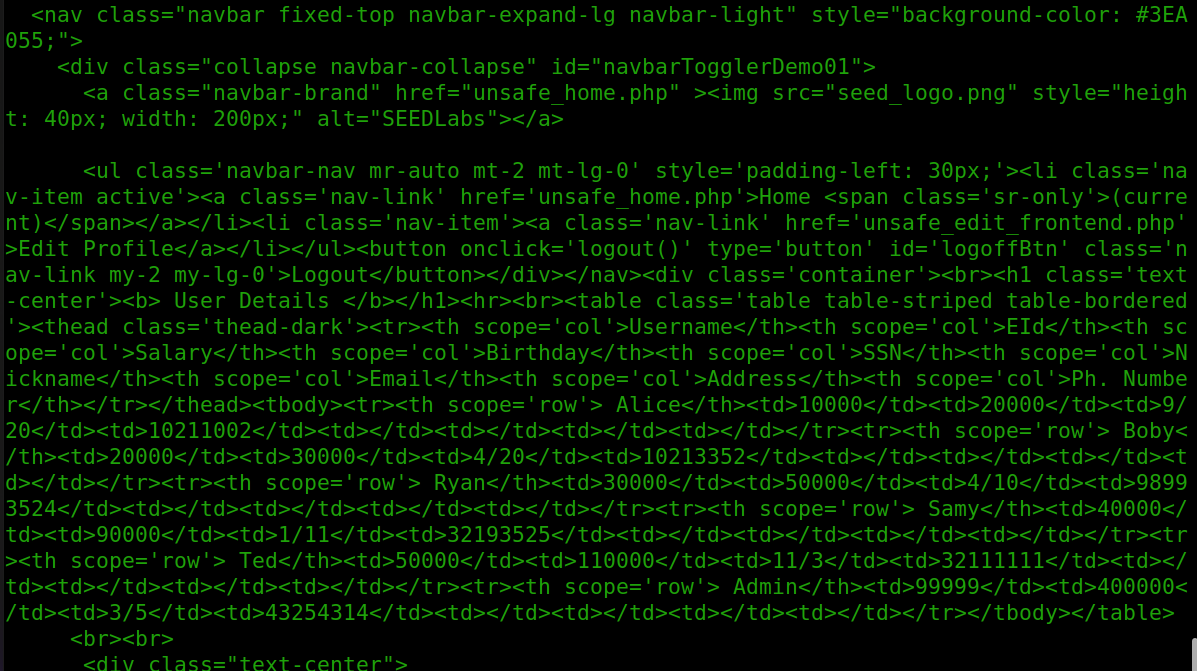
1. SQL Injection Attack on SELECT Statement
   1. SQL Injection Attack from webpage



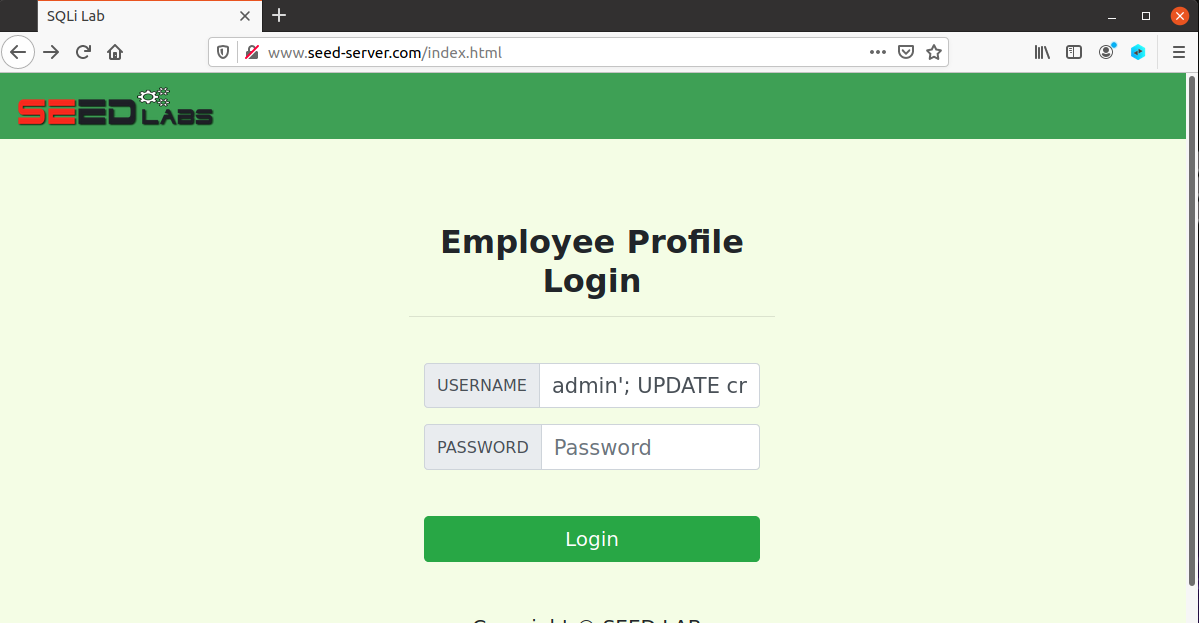


* Entering the username as admin’#
* The password is insignificant in this case and I just entered ‘123’ for the password
  + The reason this works is because the ‘# symbols acts as a comment identifier, this causes everything that is being requested from the SQL command after the identifier to become a comment
  + So the request thinks that you are the administrator because you provided the correct password, when in hindsight, I am just commenting out the rest of the statement leaving it to be WHERE name = ‘$input\_uname’ followed by one large comment.
  1. SQL Injection Attack from the command line

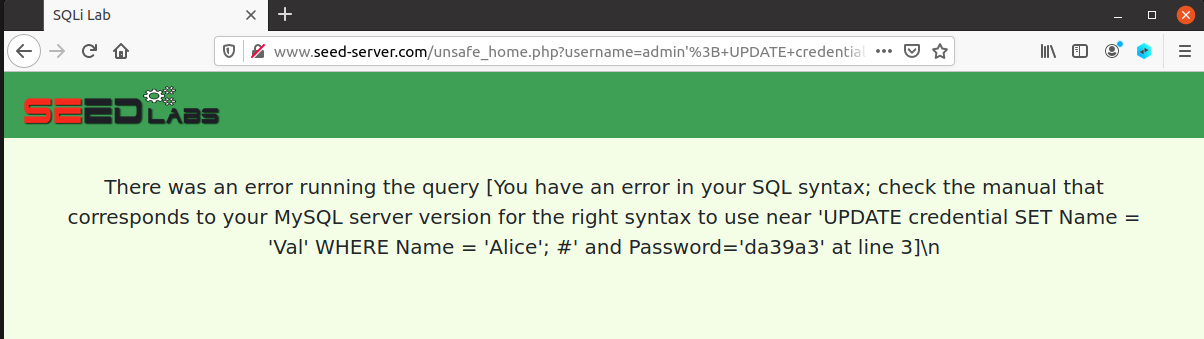




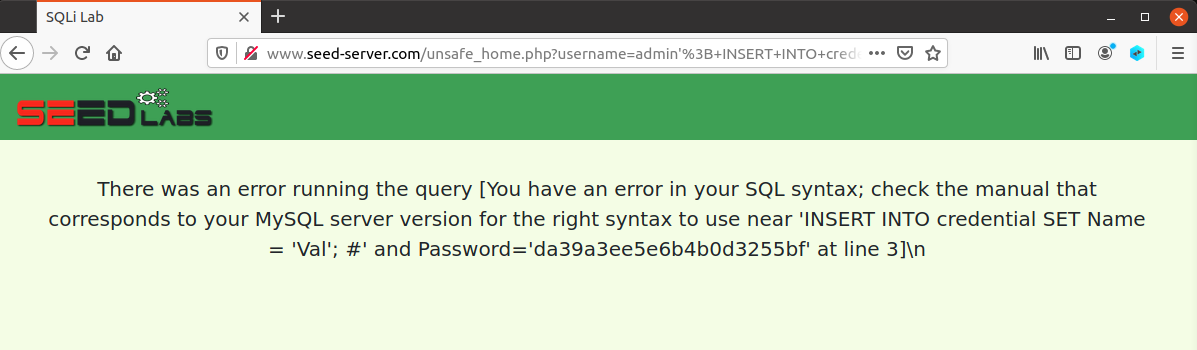
* I am using the given curl statement that was provided for us in the lab manual to successfully launch this attack
* The important part I have underlined in red in the first screenshot
  + The %27 is a single quotation mark
  + The %20 is a single white space
  + The %23 is a single pound symbol
  + The final result is a username that now looks like admin’ #
* Now we have completely launched the attack like in 2.1 but instead we have used the curl request.
* This username accomplishes the same goal as the username in 2.1 did, allowing us to comment out the second half of the authentication statement.
  1. Append a new SQL statement



* The SQL statement that I am using in this task is
  + admin'; UPDATE credential SET Name = 'Val' WHERE Name = 'Alice'; #

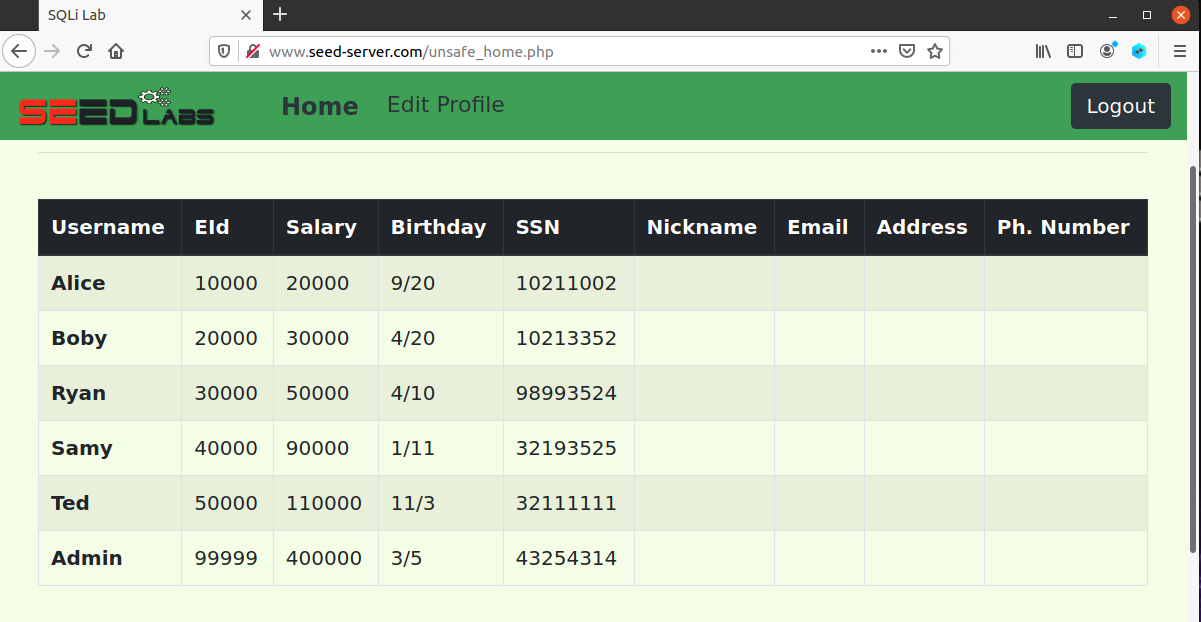


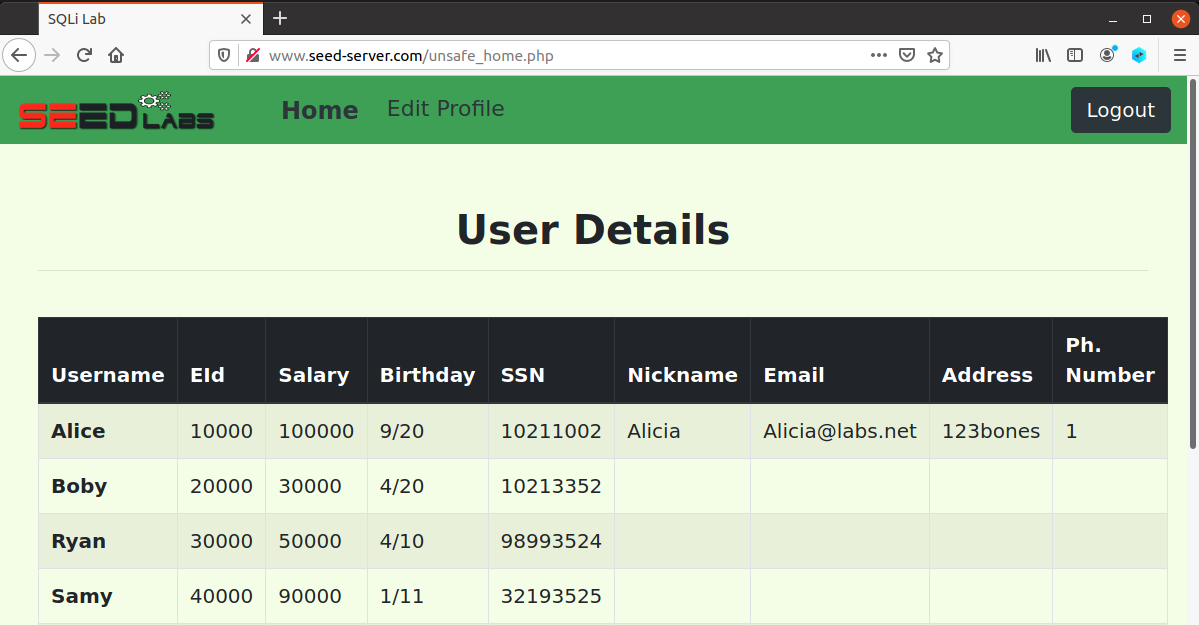
* As the lab manual states, the semicolon separates my two SQL statements. I am trying to update the Name entry from Alice to the name Val. When I click log in and try to run the query, we can see that the attempt is unsuccessful.

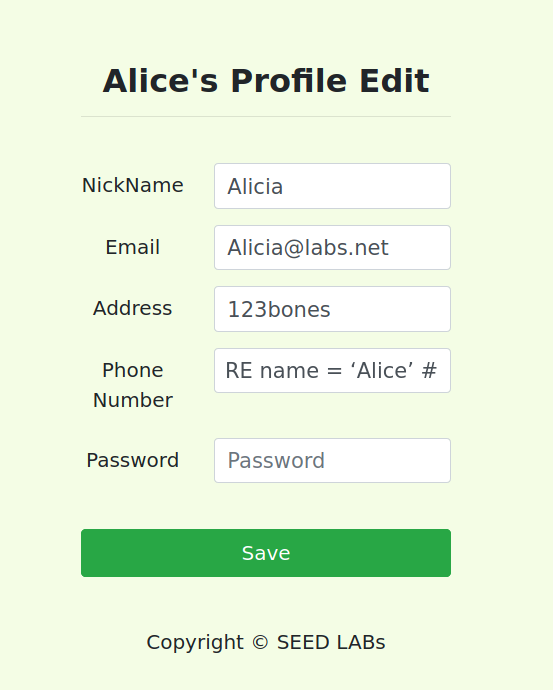


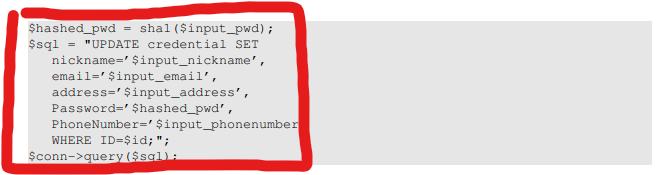
* I tried another query to insert my own name into the credentials table and it seemed like that did not work as well
* After some research, I found that it is using the PHP’s mysqli extension and the mysqli::query() API does not support multiple queries within the same call to the API.
* I think the reasonable thing to do here would be to not rely on the extensions’s shortcomings and avoid having multiple statements run in an SQL injection.

1. SQL Injection Attack on UPDATE Statement
   1. Modify your own salary

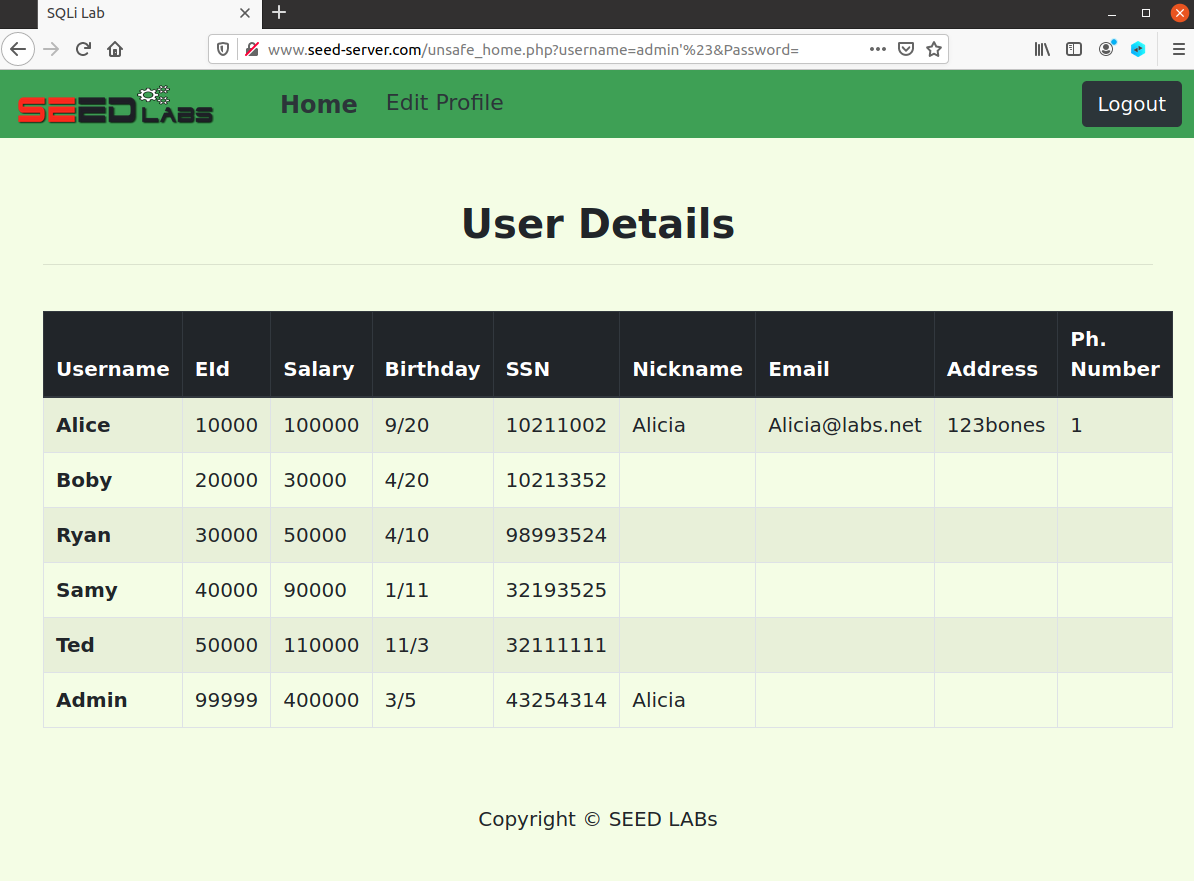


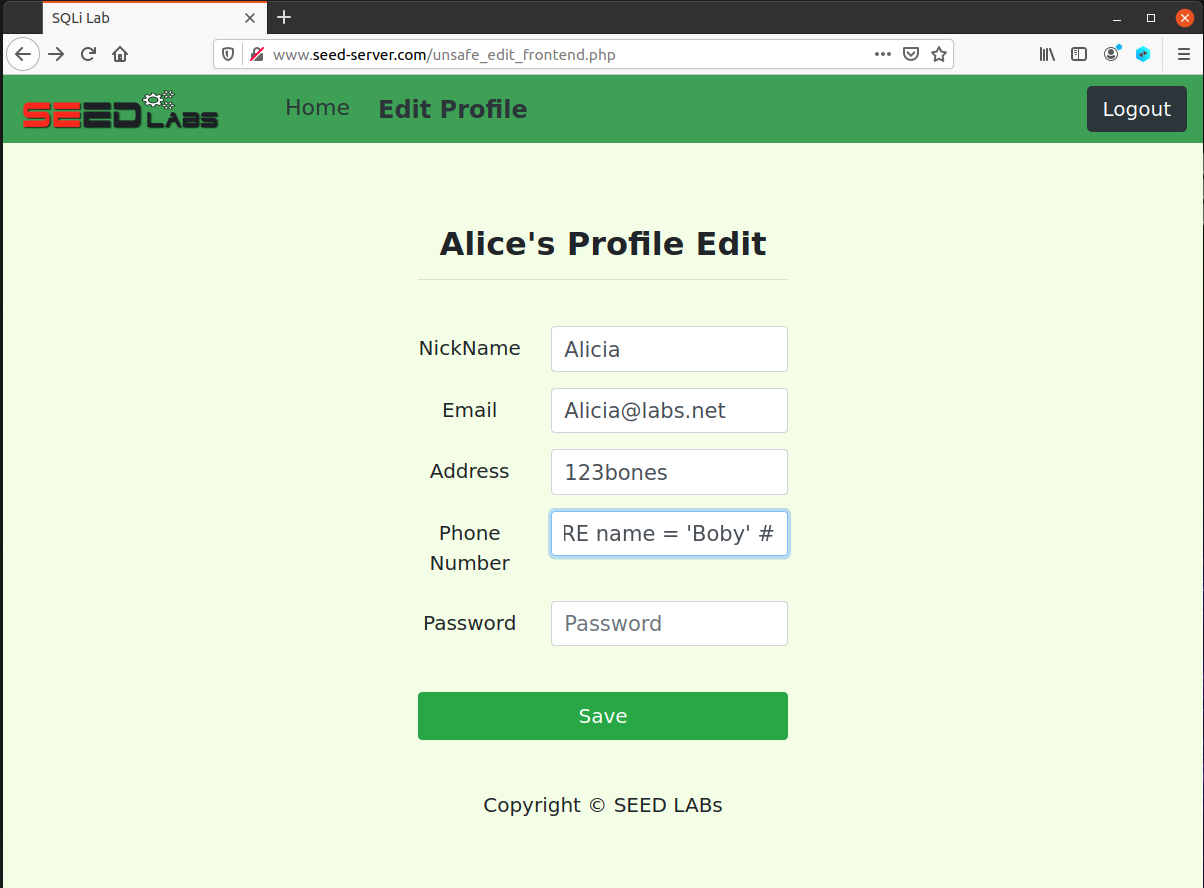


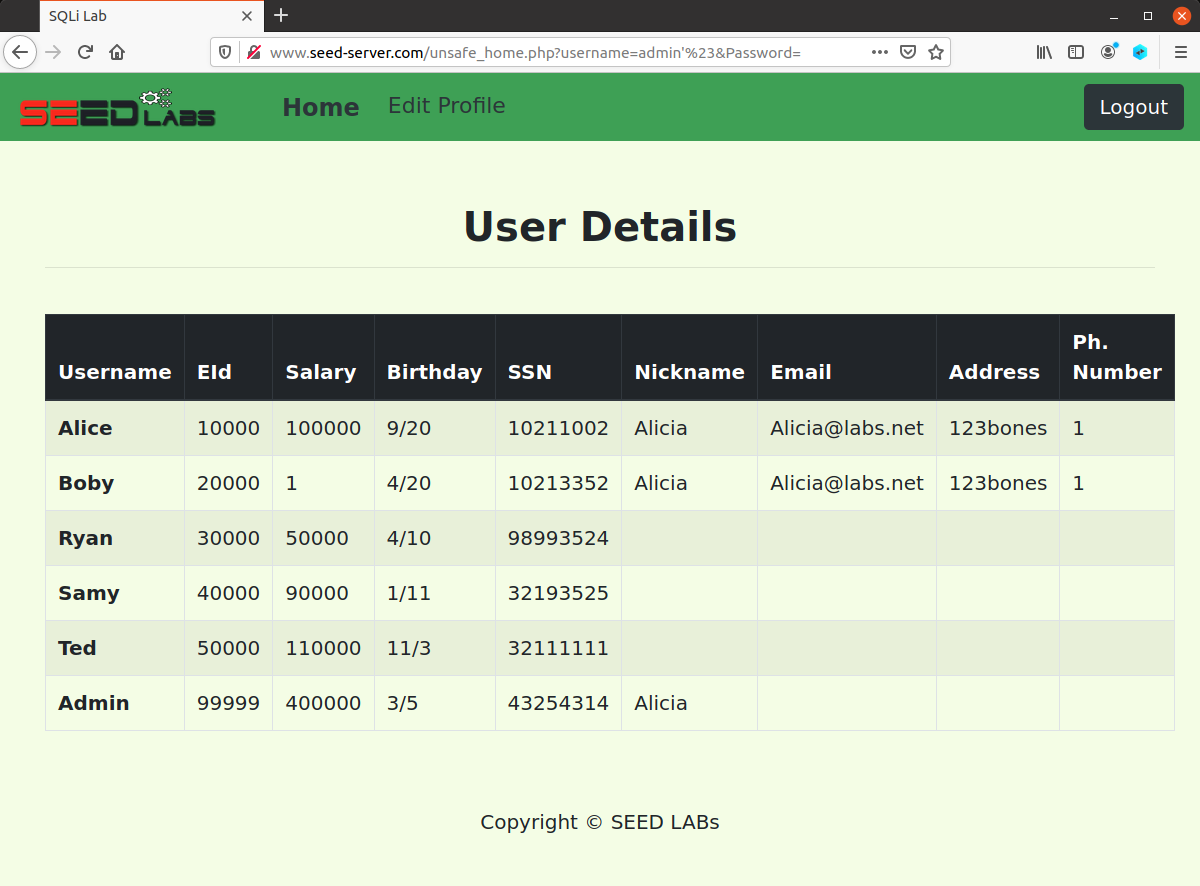


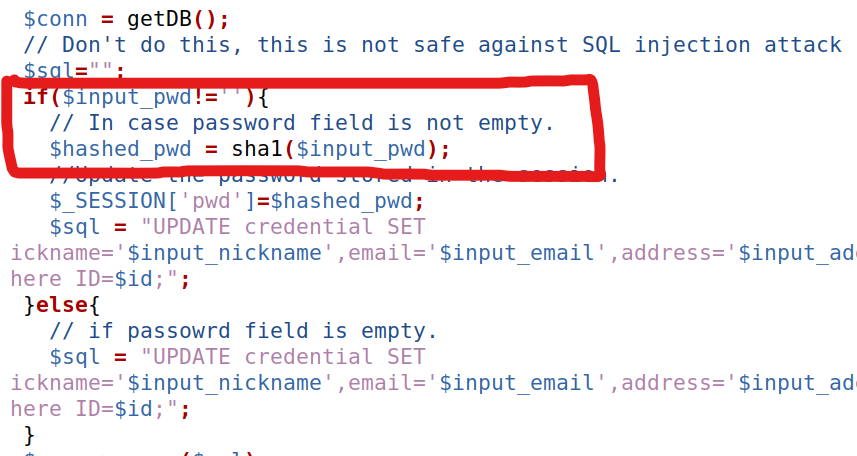


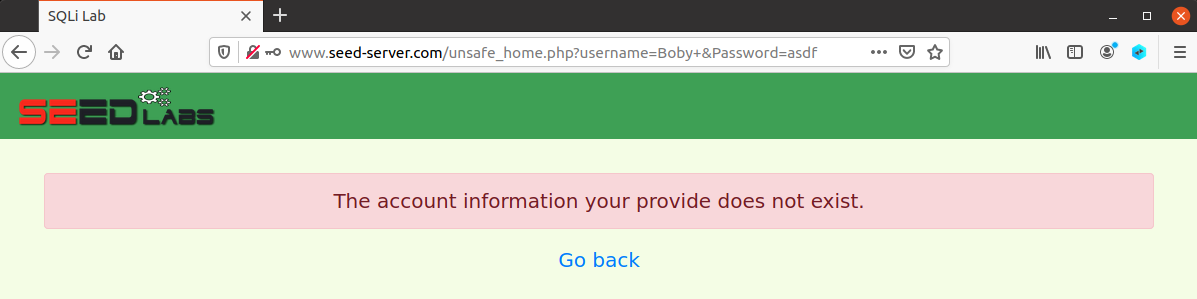
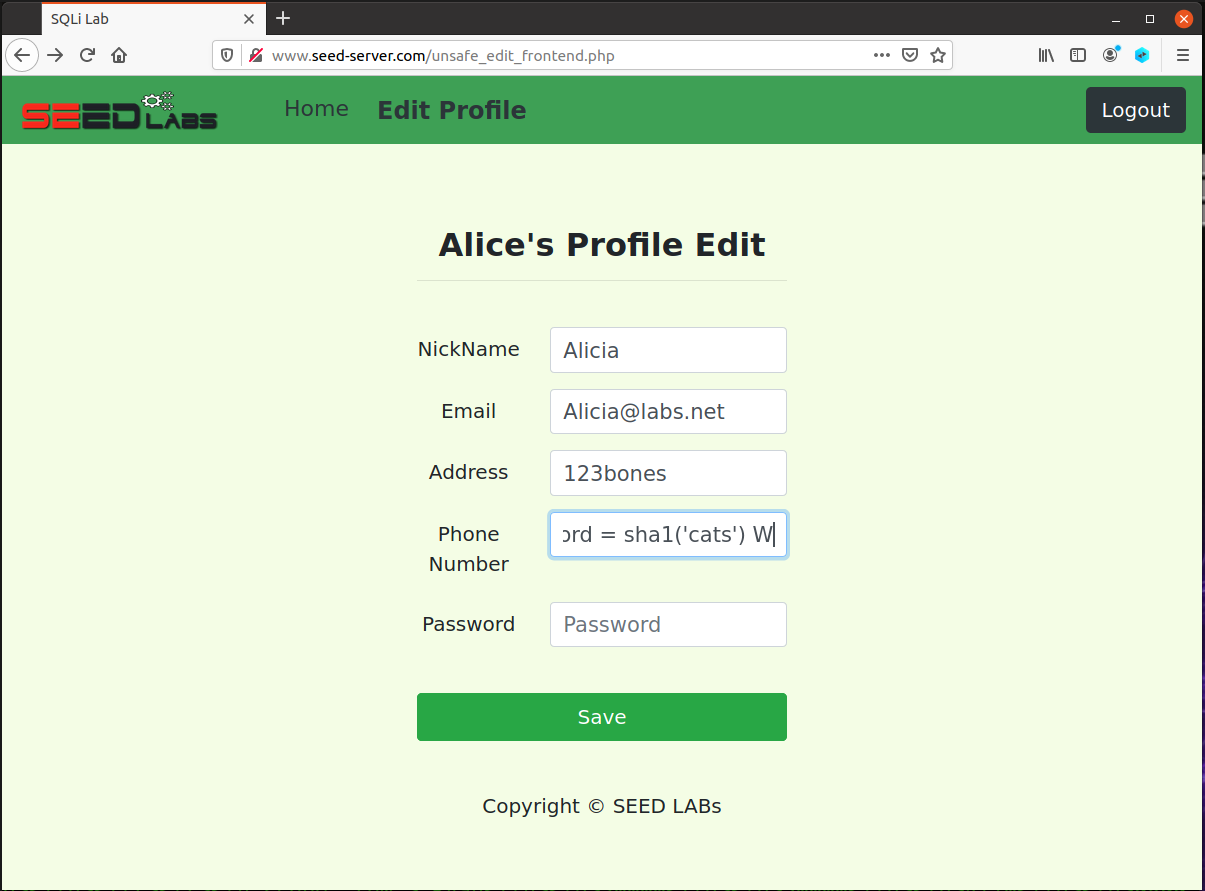
* The SQL statement I used for this attack was
  + 1’, salary = 100000 WHERE name = ‘Alice’ #
* On saving this changed to the profile you can see the updated table that now has Alice making a cool 6 figure salary instead of 20000
* This is only possible because by using the statement I entered into the phone number field it changes the query that is being passed to the MySQL server
* The new query that I am passing should look like the following now
  + UPDATE credential SET
    - Nickname = “Alicia”
    - Email = “[Alicia@labs.net](mailto:Alicia@labs.net)”
    - Address =”123bones”
    - Phone Number = ‘1’,
    - Salary = 100000 WHERE name = ‘Alice’
* It is easy now to see how I can use the phone number field in the end to append another statement to the field and change our own data!
  1. Modify other people’s salary

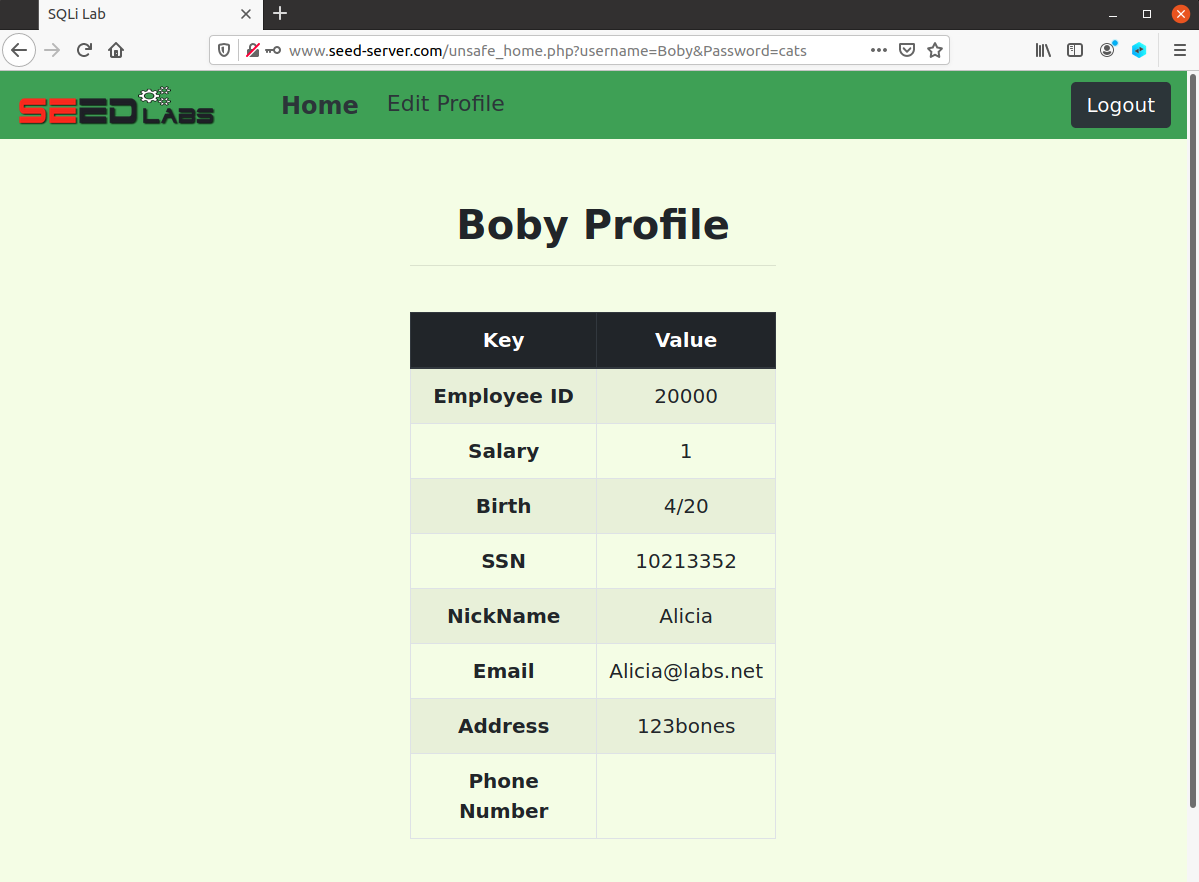
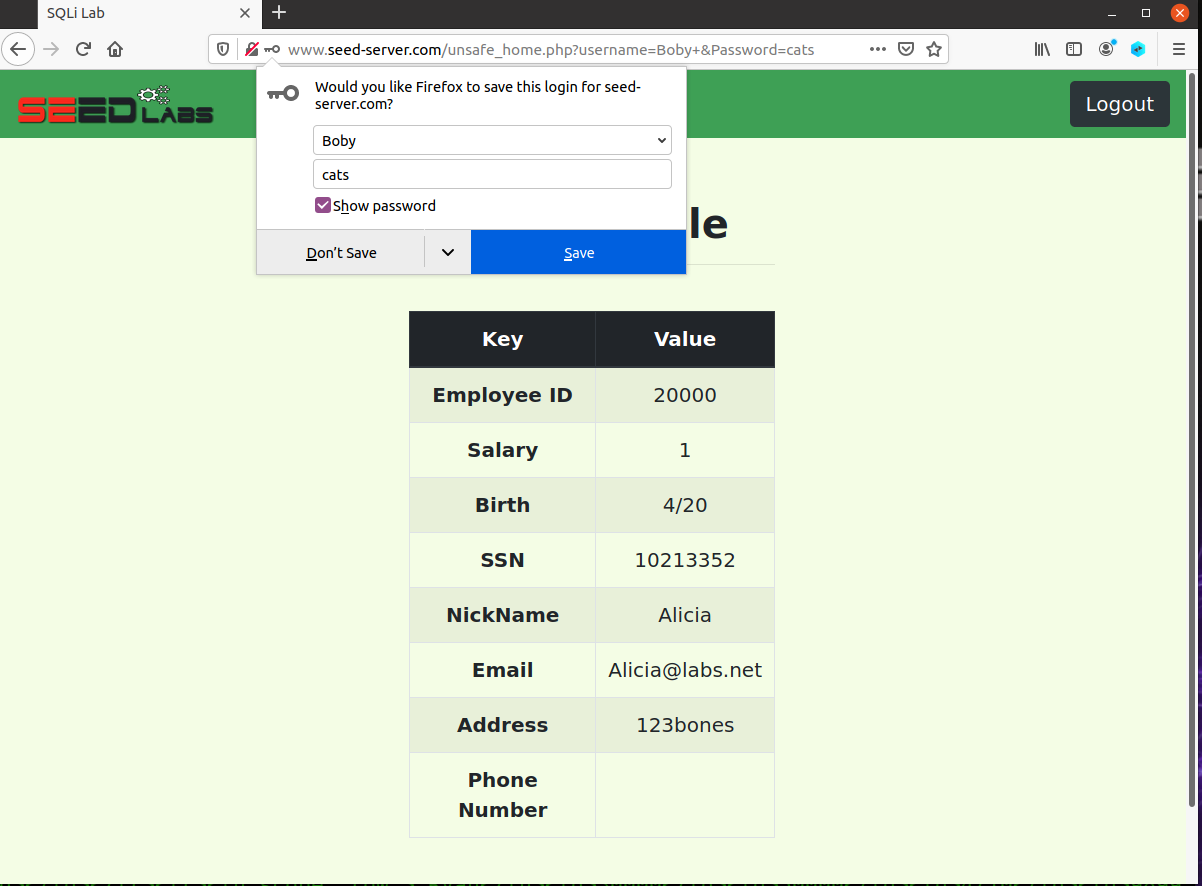






* I show the first original table where boby had his original salary
* After we see that boby is normal we use the following statement to change his salary from his normal salary to a singular dollar
  + 1', salary = 1 WHERE name = 'Boby' #
* We can see now that we have changed boby’s salary to one dollar
* This is the same methodology as the last task but instead of changing our own salary we can simply change the ID the statement looks at to ‘boby’ in order to change his salary now.
  1. Modify other people’s password

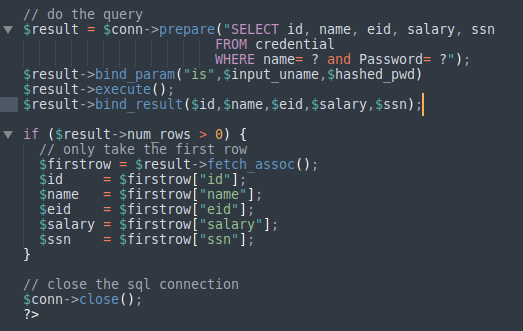




* In order to modify boby’s password we need to know how the password is being stored.
* We can see that in unsafe\_edit\_backend.php the password is being stored as a hash
* So simply to change this we want to edit the query that is being passed to the MySQL server with the password parameter for boby to something that we know.
* I used the following SQL command
  + ‘, Password = sha1(‘cats’) WHERE name = ‘Boby’ #
* Now I attempt to log in with an old password and can see that I can not access the page
* But when we enter the new password we can see through the webpage that my new password of ‘cats’ works and gets us into boby’s profile

1. Countermeasures

unsafe.php



* We fix the prepared statements that were written initially in the code so that the parameters are being bound and then we are binding those results to specific variables when the username and password areas are being filled in.
  + This ensures that the previous attack that is made using the admin user name can no longer work because the SQL statement can no longer be directly exploitable
* \*\***NOTE\*\***
  + I could not get this to work, as I think I am missing some important plugins but I do explain why theoretically my code should work.