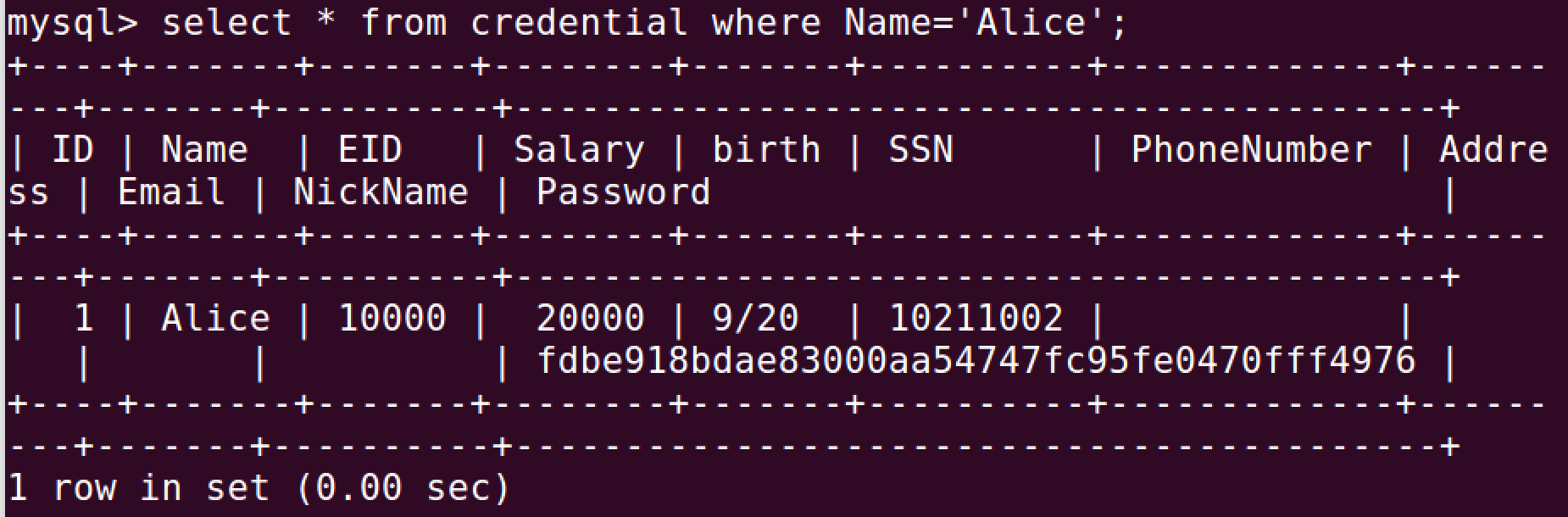
Lab 7

Task 1: Get Familiar with SQL Statements

Commands Used and Results:



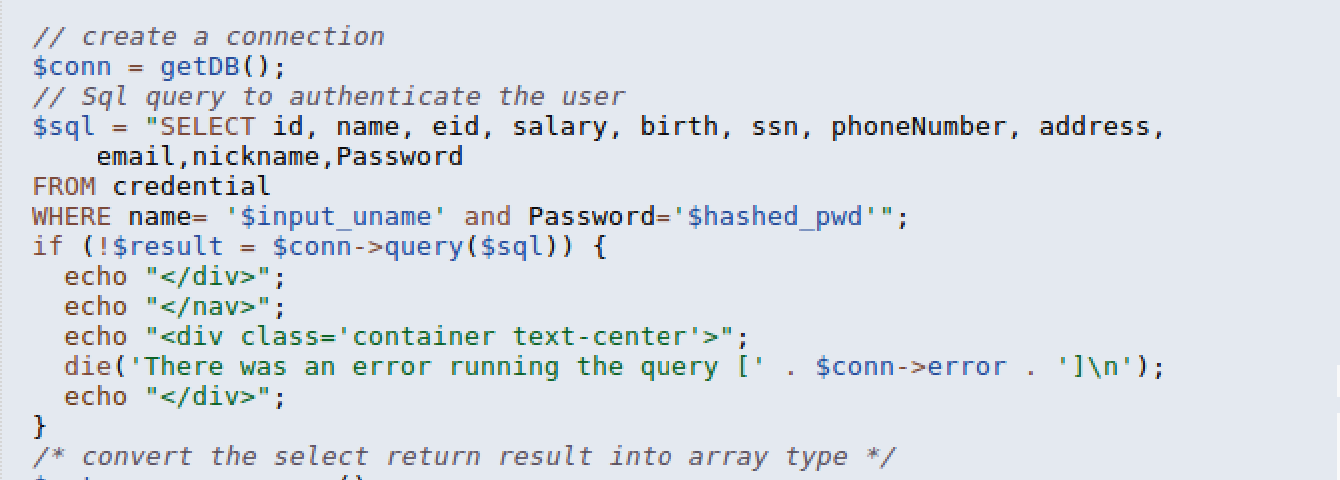
Comments:

It asked me to display the information for Alice so I had it grab and show me every record on the table where the Name was Alice.

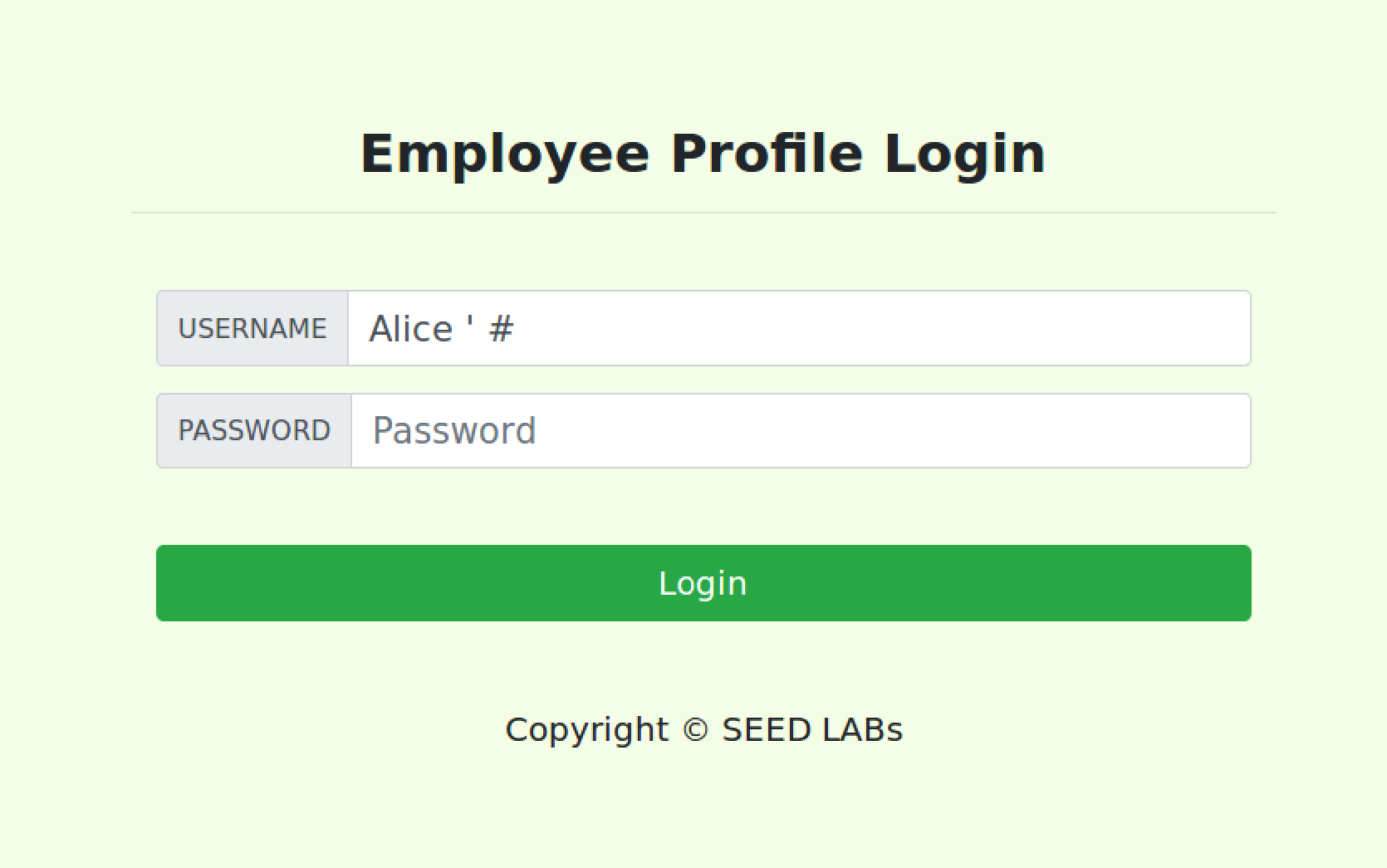
Task 2: SQL Injection Attack on SELECT Statement

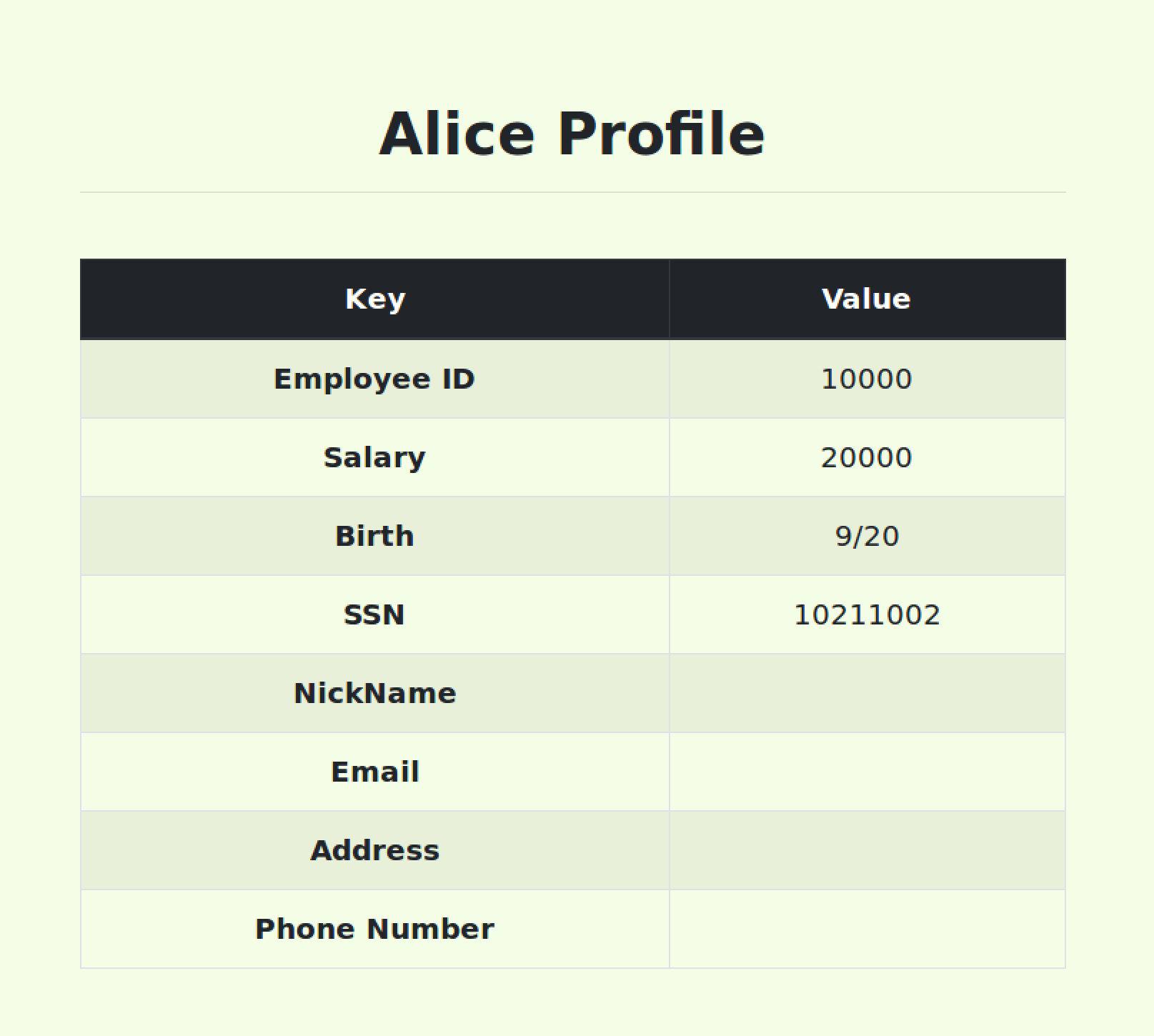
2.1: SQL Injection Attack from webpage

Code Used:



Commands Used and Results:



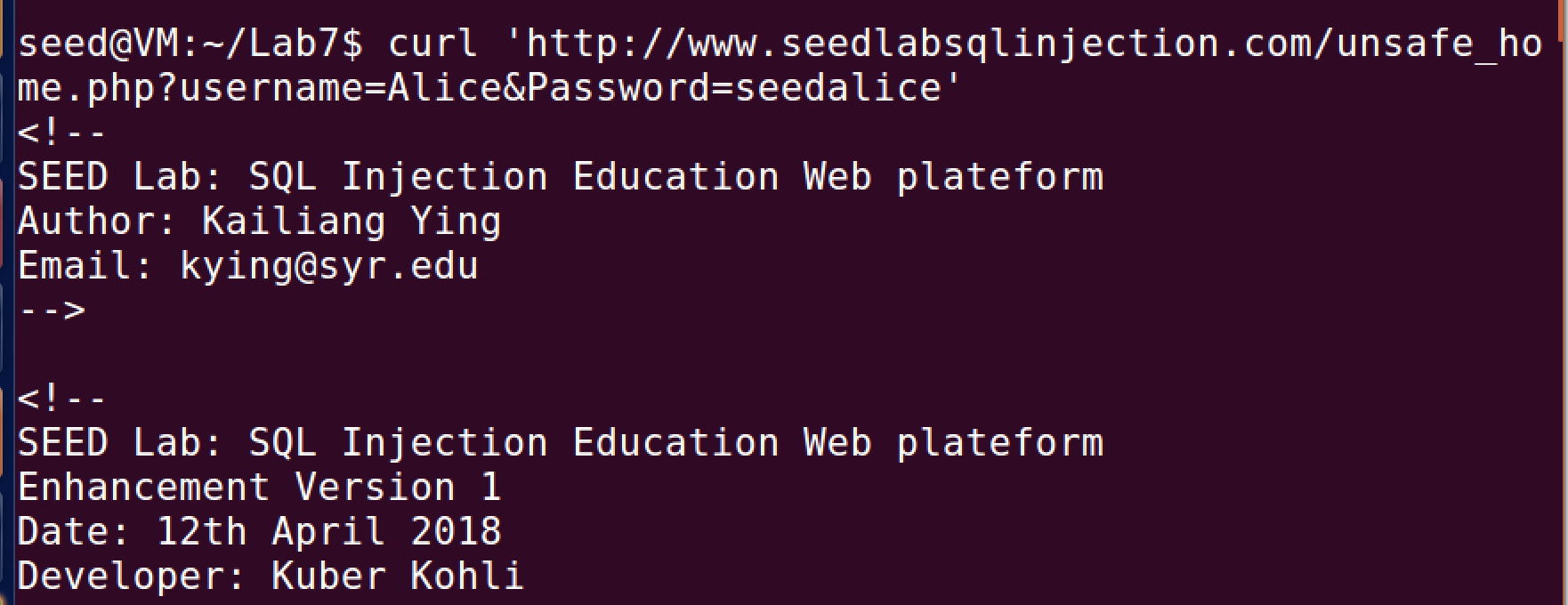


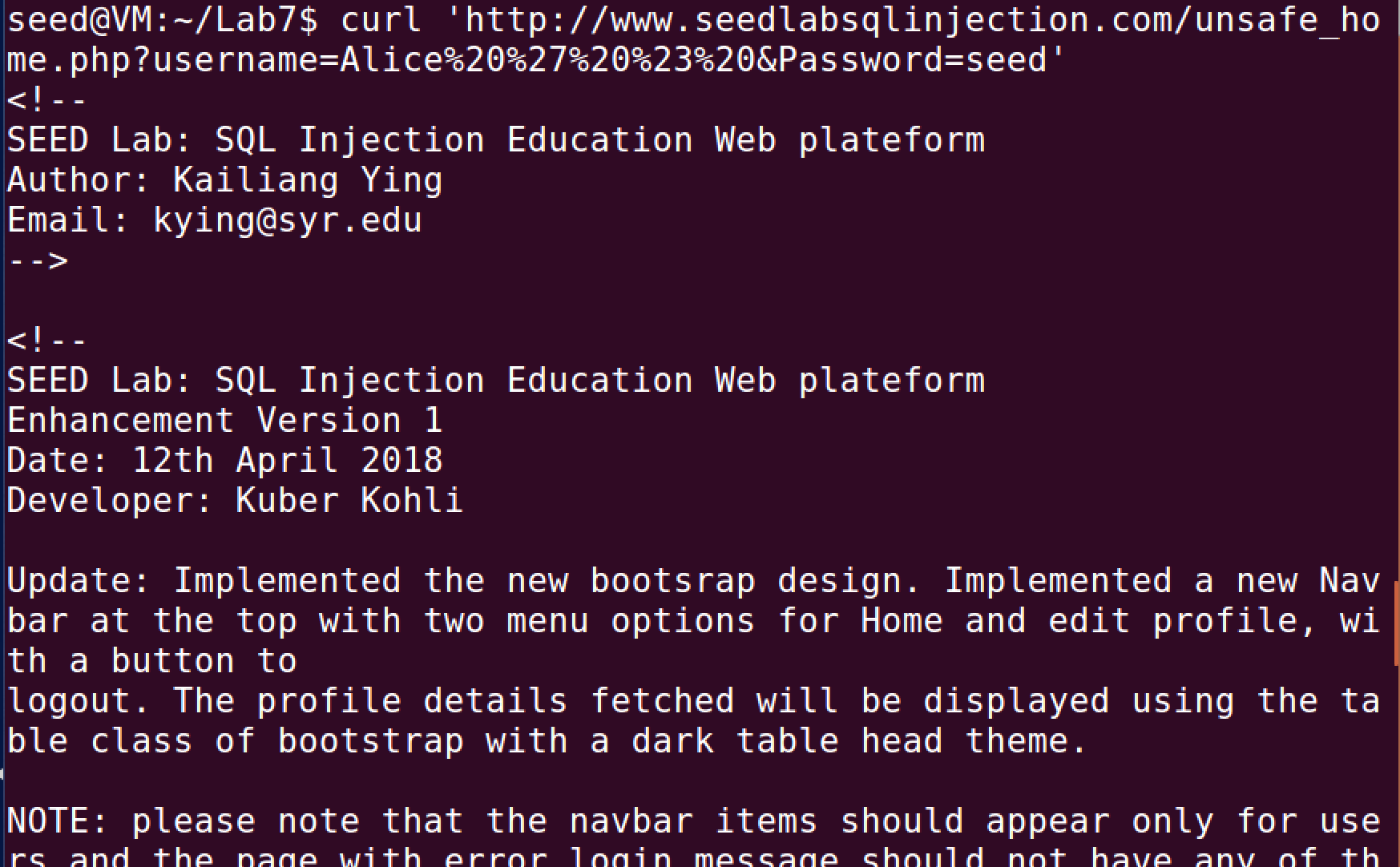
Comments:

The code is flawed in where you can make it to where if you put a ‘ followed by a # it actually cuts the code at the ‘ and comments the rest after #. Allowing you to only need the username to gain access.

Task 2.2: Injection Attack from command line

Commands and Results:



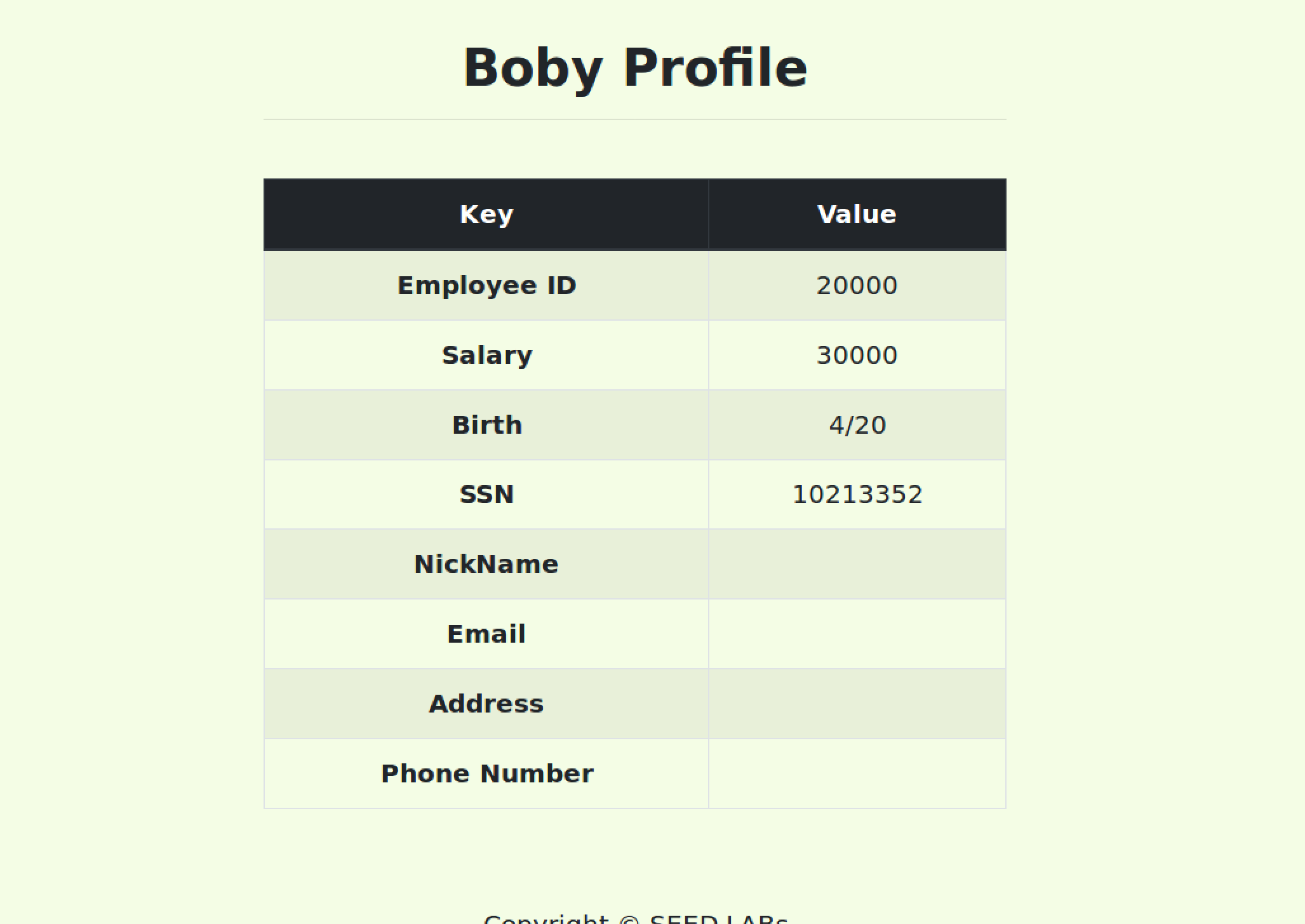


Comments:

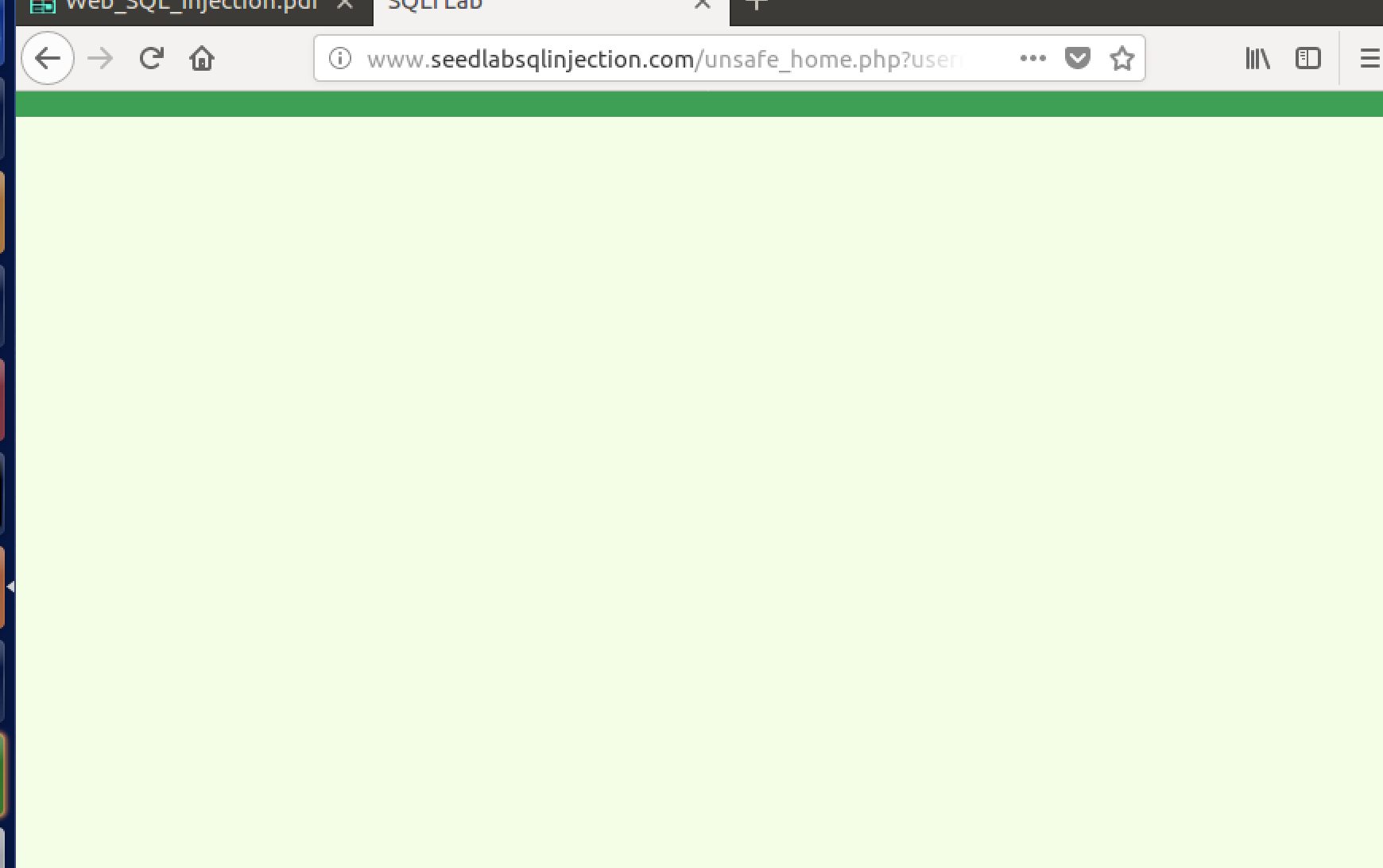
By encoding the URL and making sure it matches what the URL would be when the user would log in, you can run the injection from the terminal. Just like the last one, if it was a normal log in then you would need the username and the password, but because you add the ‘ # then you don’t need the password anymore as show when the password in the url of the second one which is encoded has the wrong password displayed but it still allowed me entry.

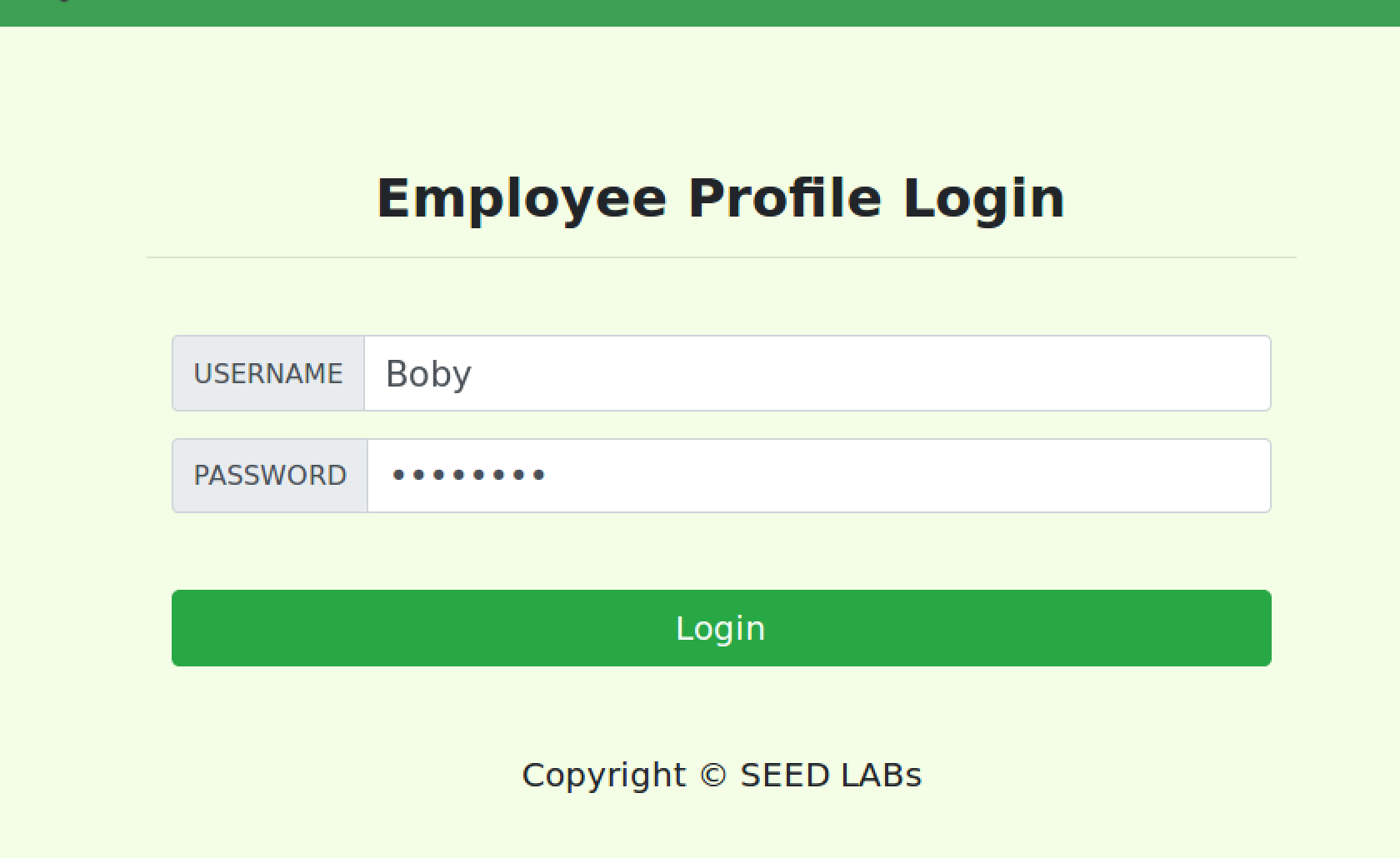
Task 2.3: Append a new SQL statement

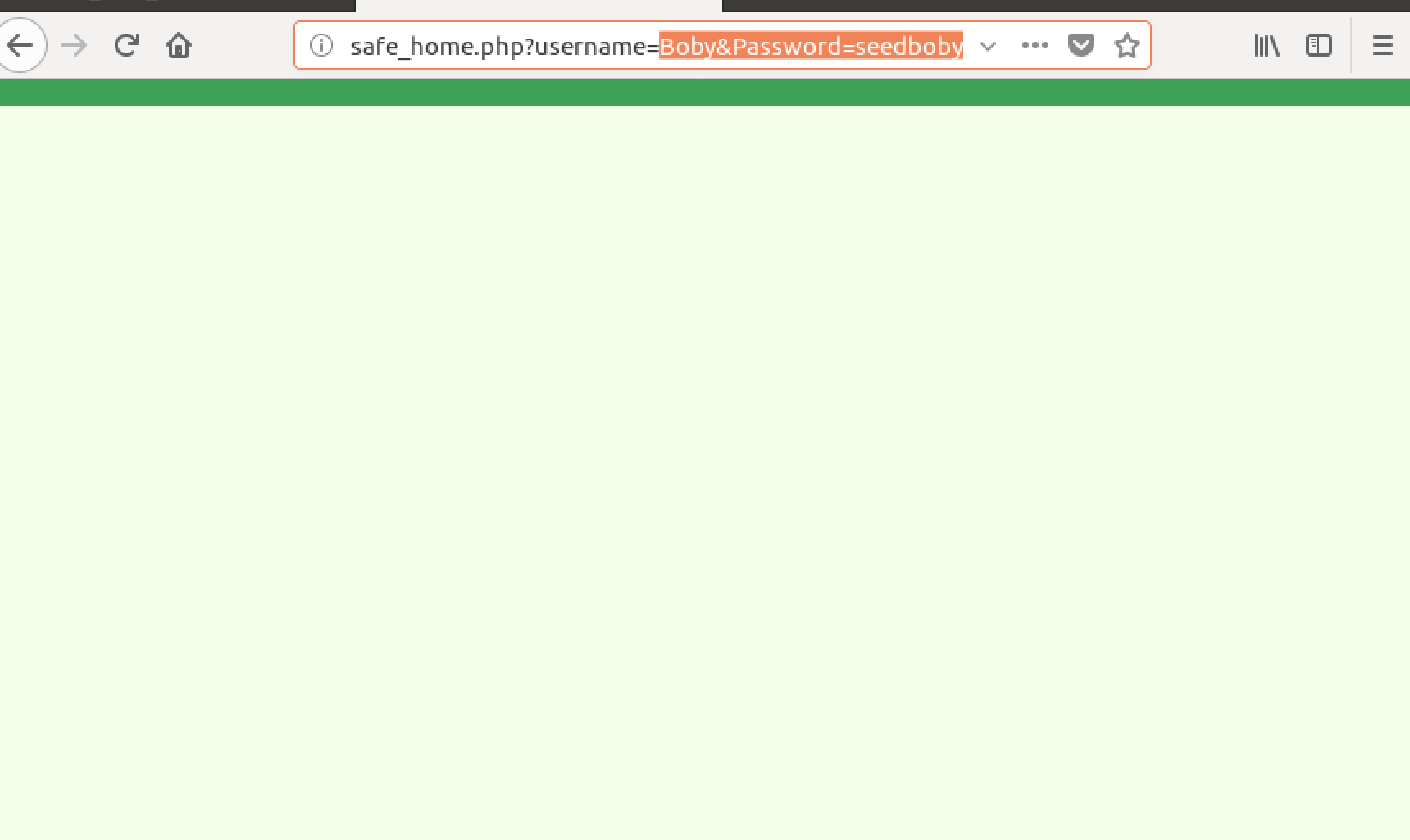
Commands and Results:

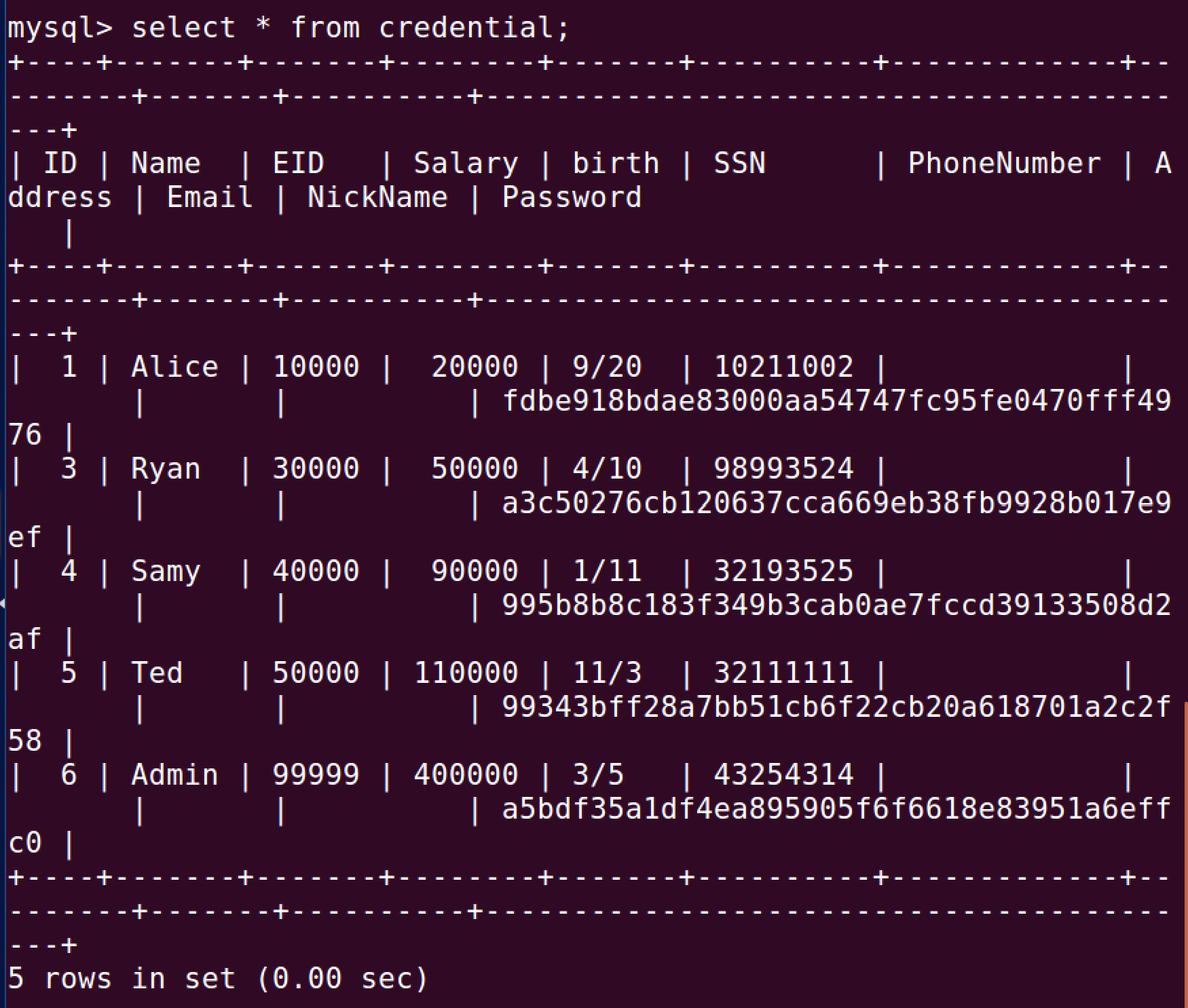


**Alice'; delete from credential where name='Boby'; #**

****

****

****

****

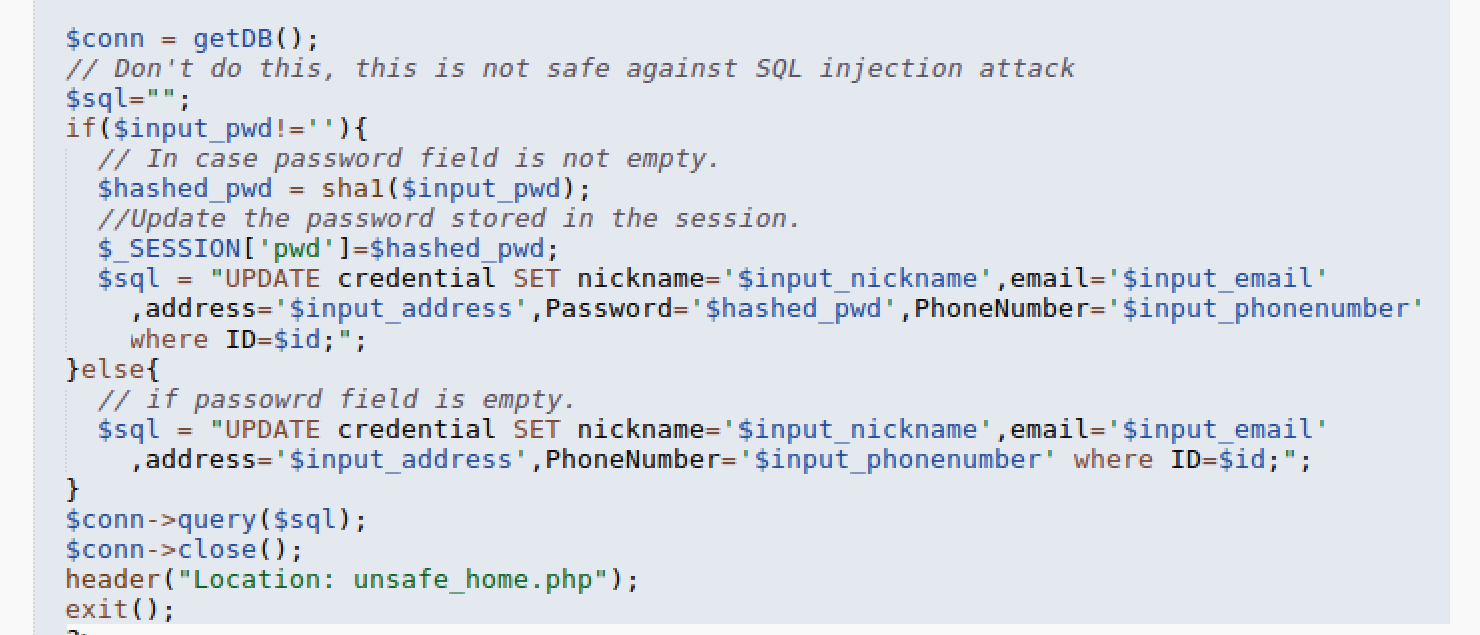
Comments:

To do this you have to split the code into two separate statements. Just like last time, we add the ‘ but we end the statement with a ; and then begin a new SQL statement which allows us to delete Boby from the table. After the statement, we have to end the statement with another ; and then comment out the rest of the original statement with a #.

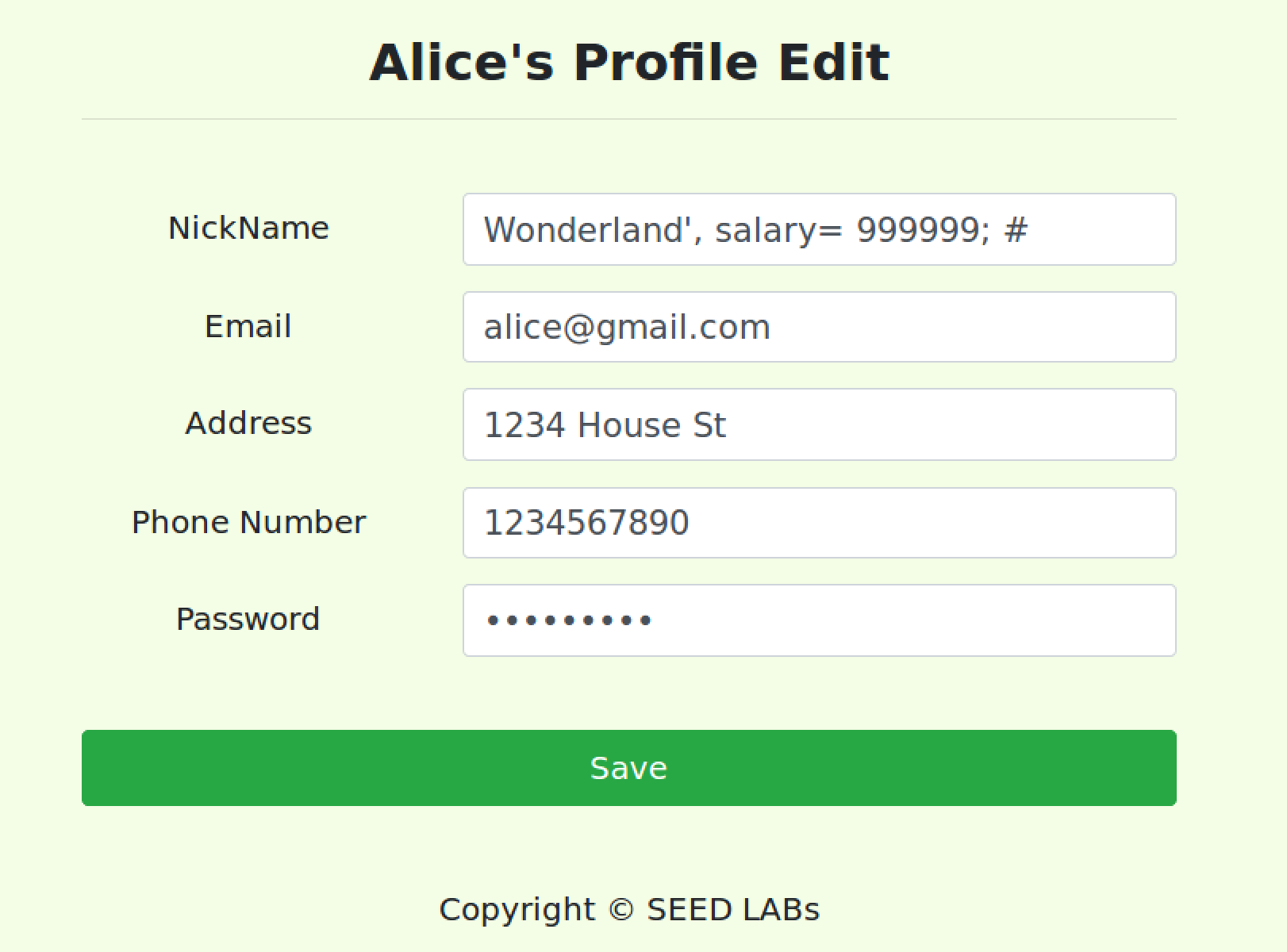
Task 3: SQL Injection Attack on UPDATE Statement

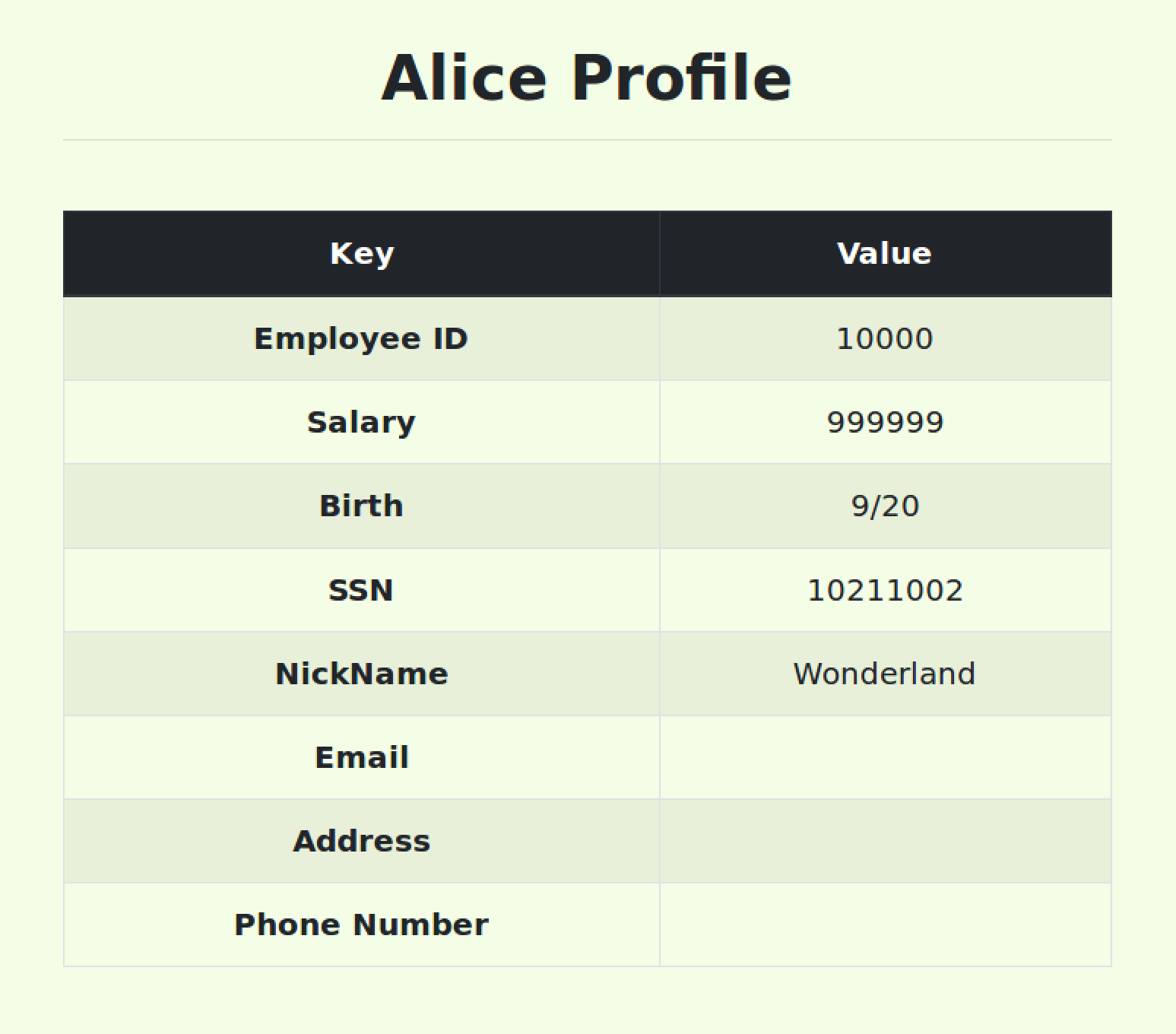
Task 3.1: Modify your own salary

Code Used:



Commands and Results:



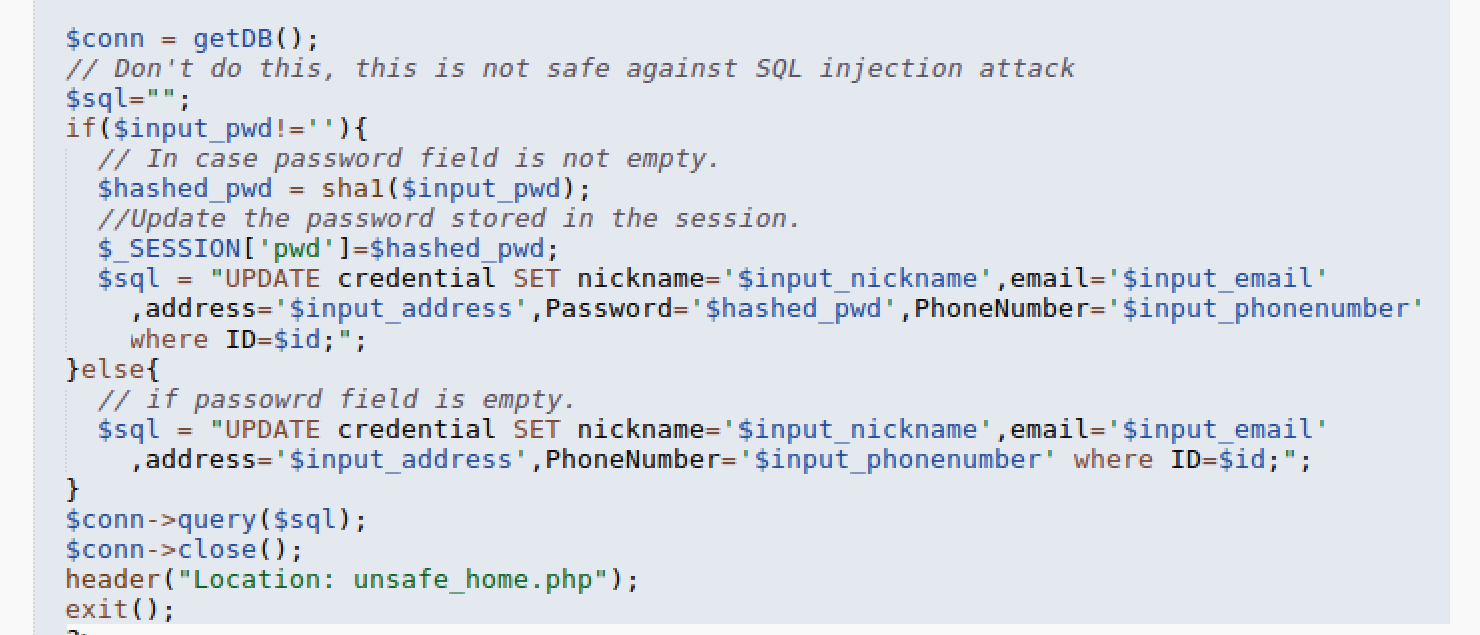


Comments:

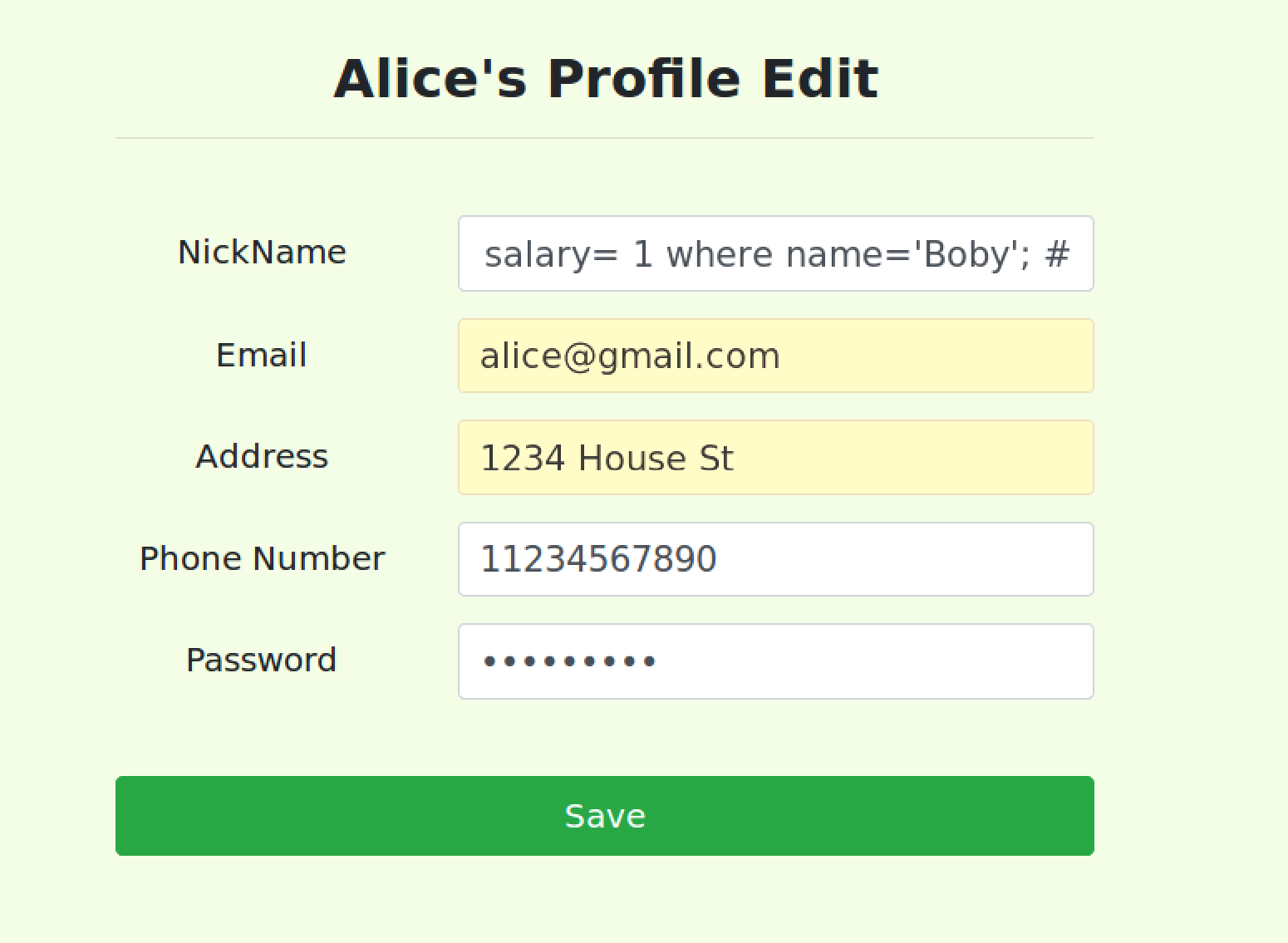
I want to change the salary of my own profile but there is now way to do it normally so we have to inject the statement to change it. By doing the same thing as before, we turn one statement into two. Using the ‘ ; and # characters, this allows us to change our salary into whatever we want.

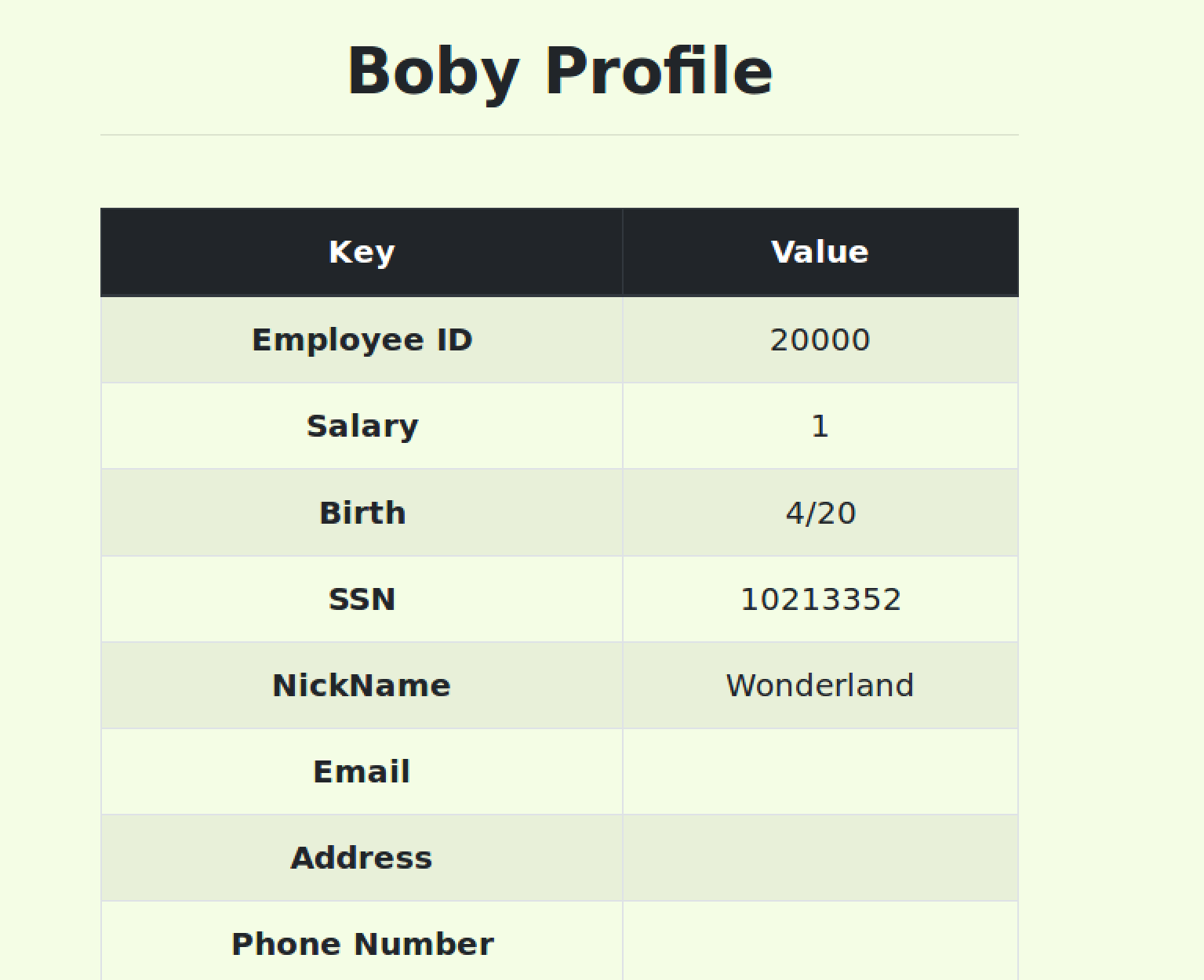
Task 3.2: Modify other peoples’ salary

Code Used:



Commands and Results:



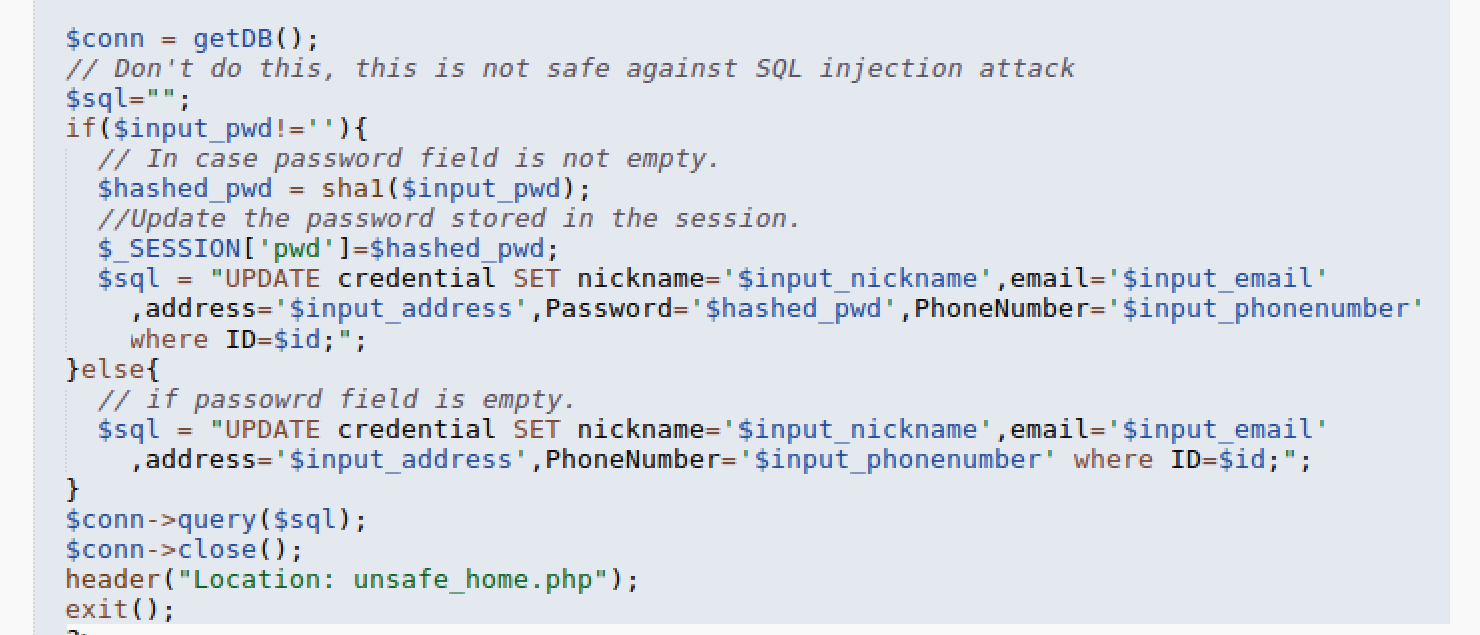


Comments:

Similarly we can also change the salary of our greedy boss and make him go broke. All I did was modify the salary statement to change it to 1 if the user was named Boby.

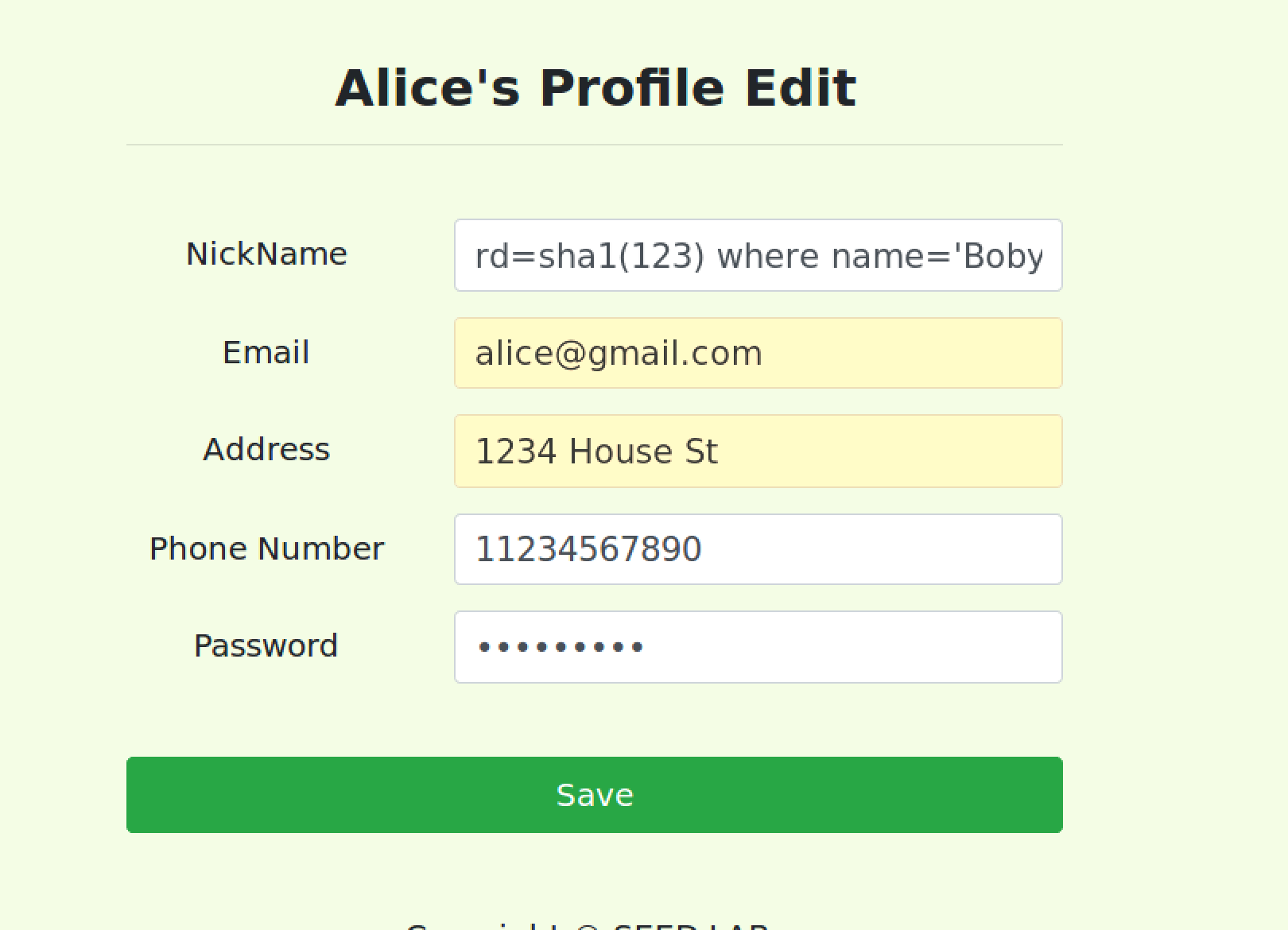
Task 3.3: Modify other peoples’ password

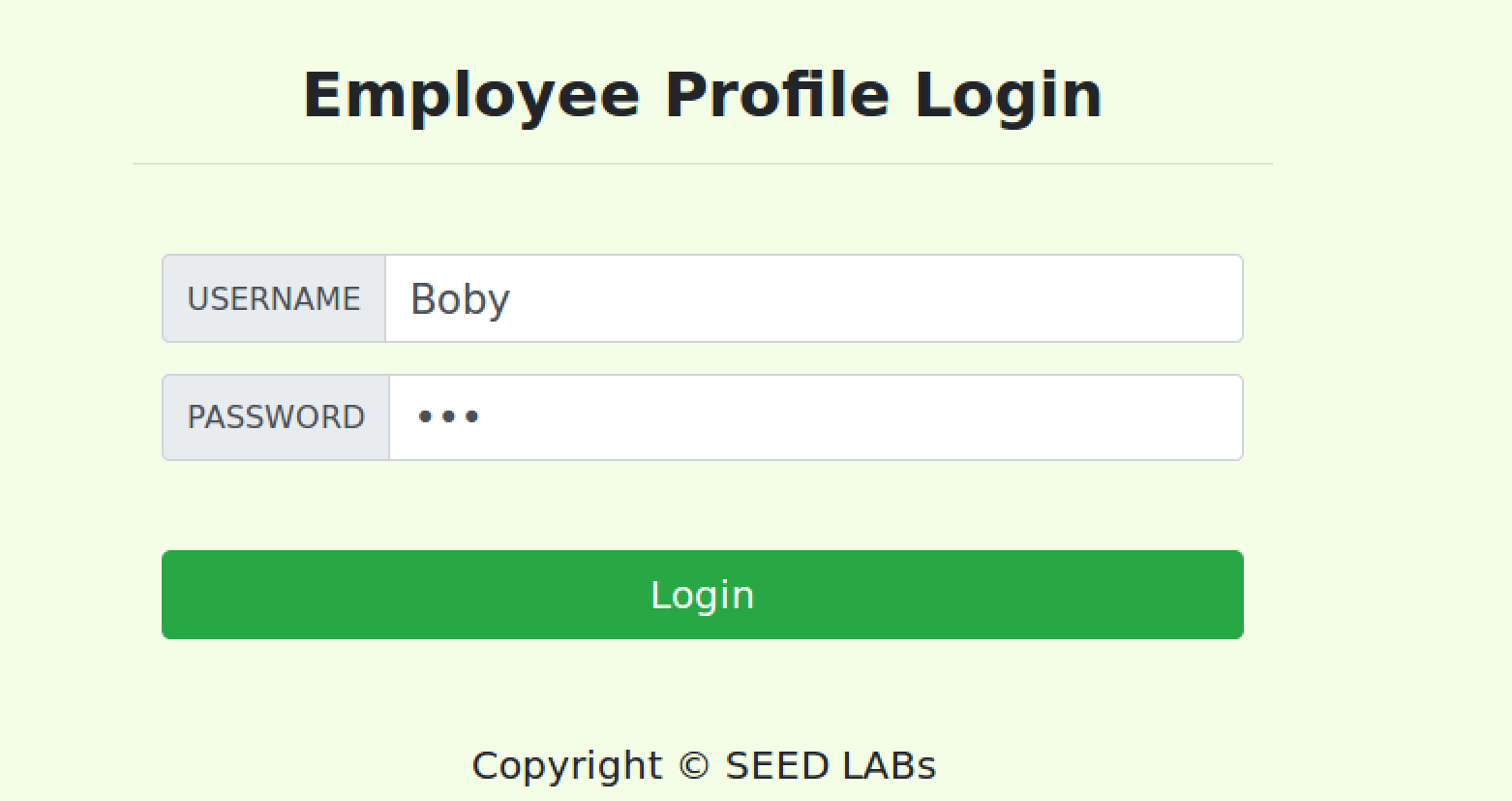
Code Used:

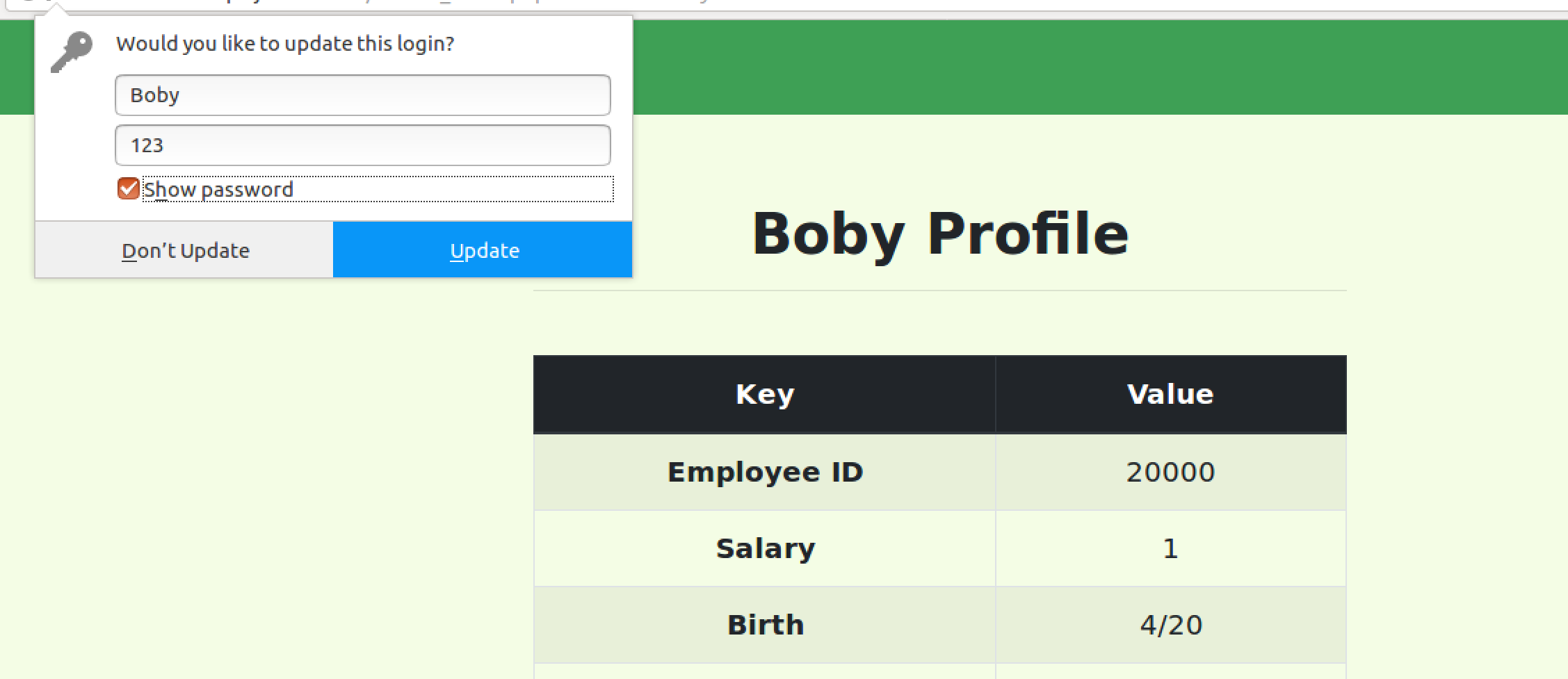


Commands and Results:

**Wonderland', Password=sha1(123) where name='Boby'; #**

****

****

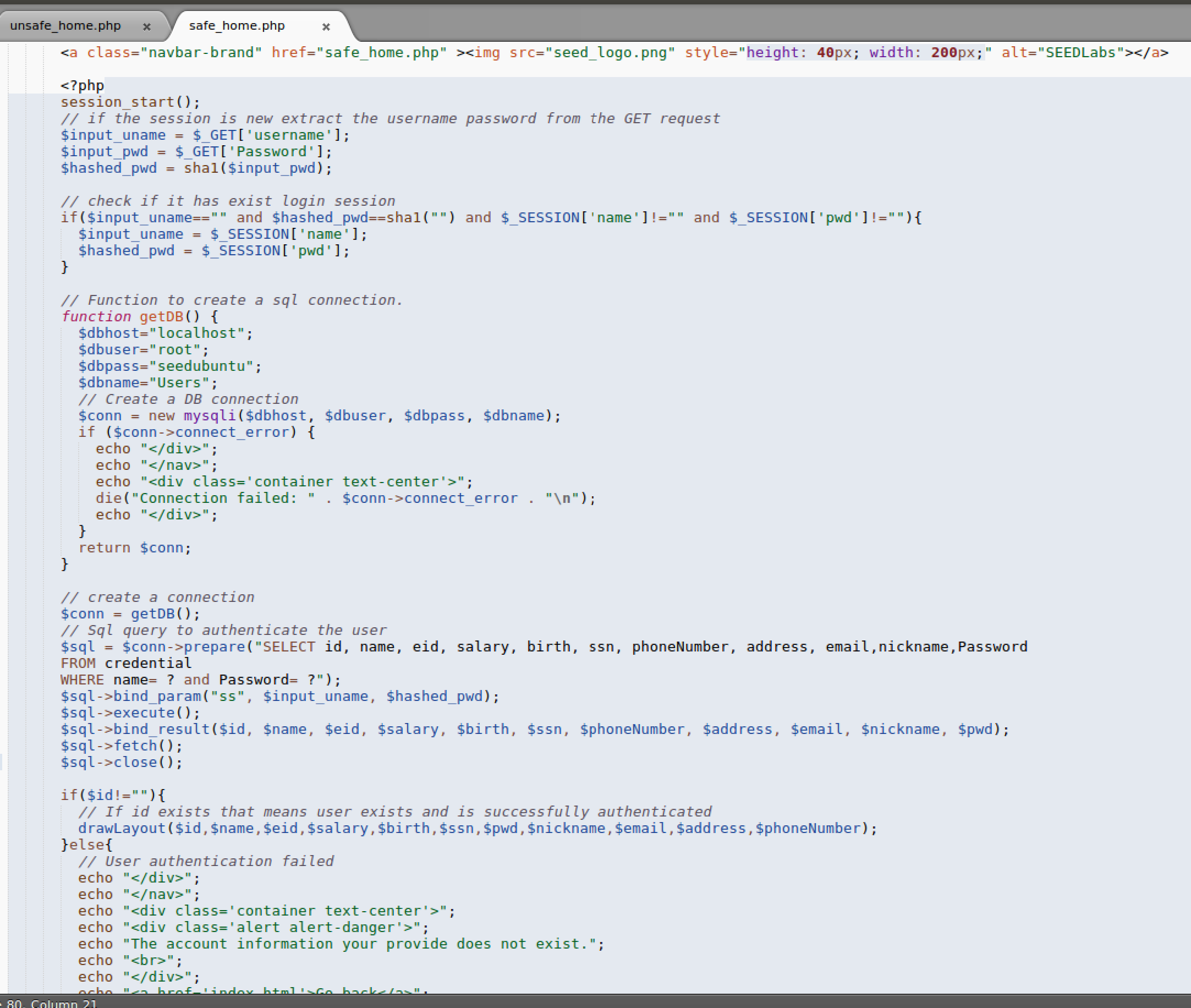
****

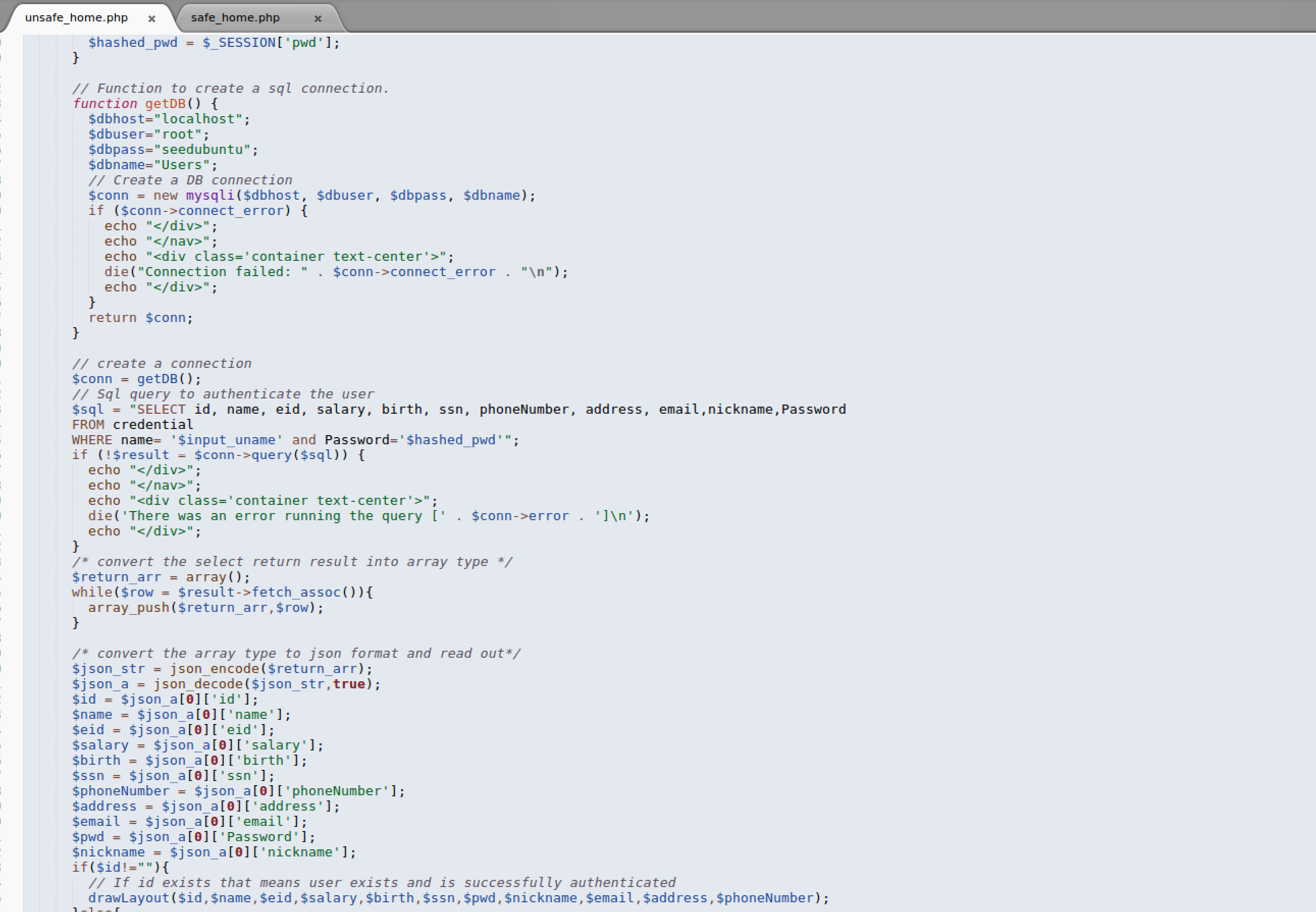
Comments:

The same thing as before but this time we have to change the user’s password and the tricky part about this is that the passwords are hashed in sha1() and so what I had to do was just incorporate that into the statement and I was able to change Boby’s password from seedboby to 123.

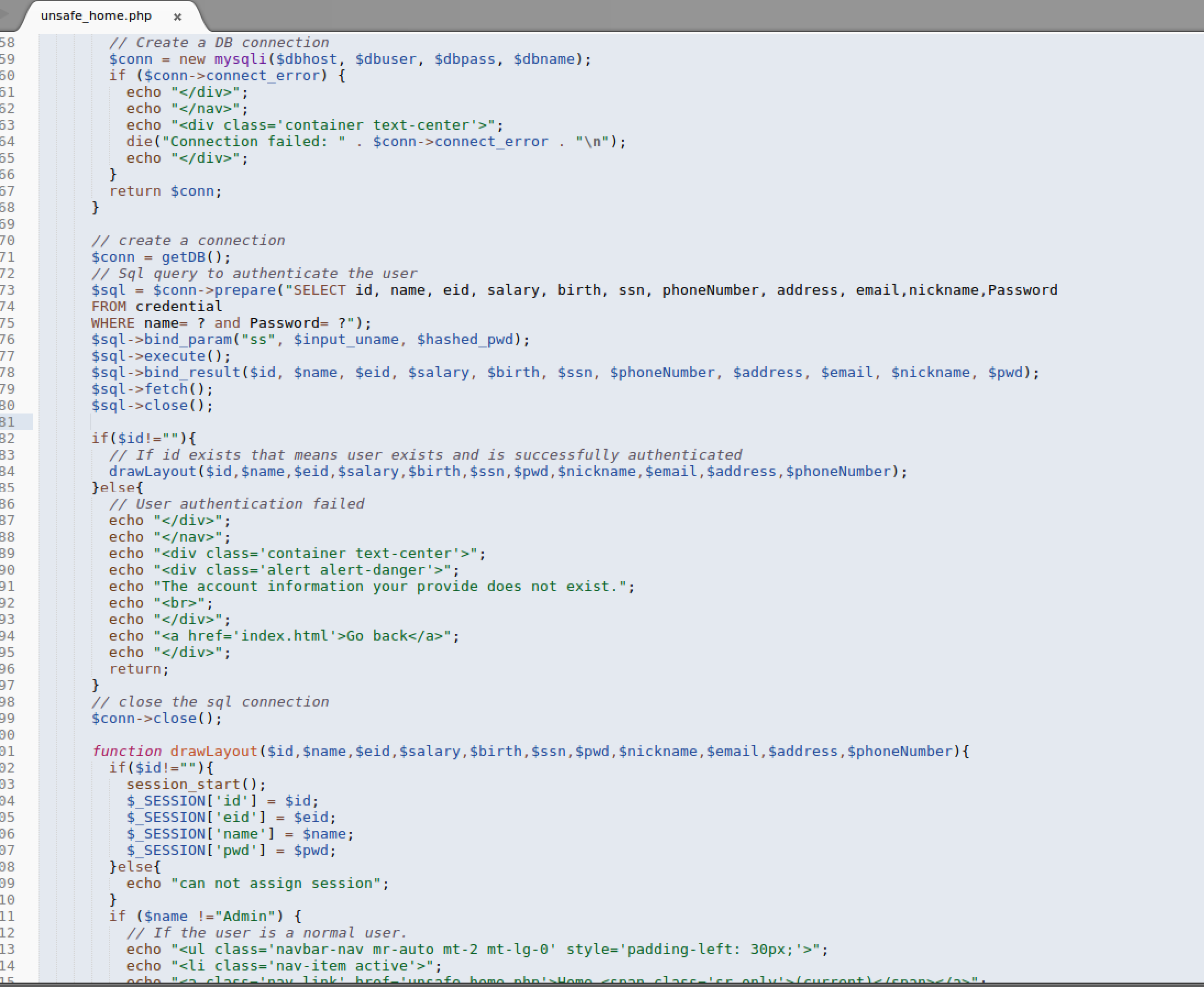
Task 4: Countermeasure – Prepared Statement

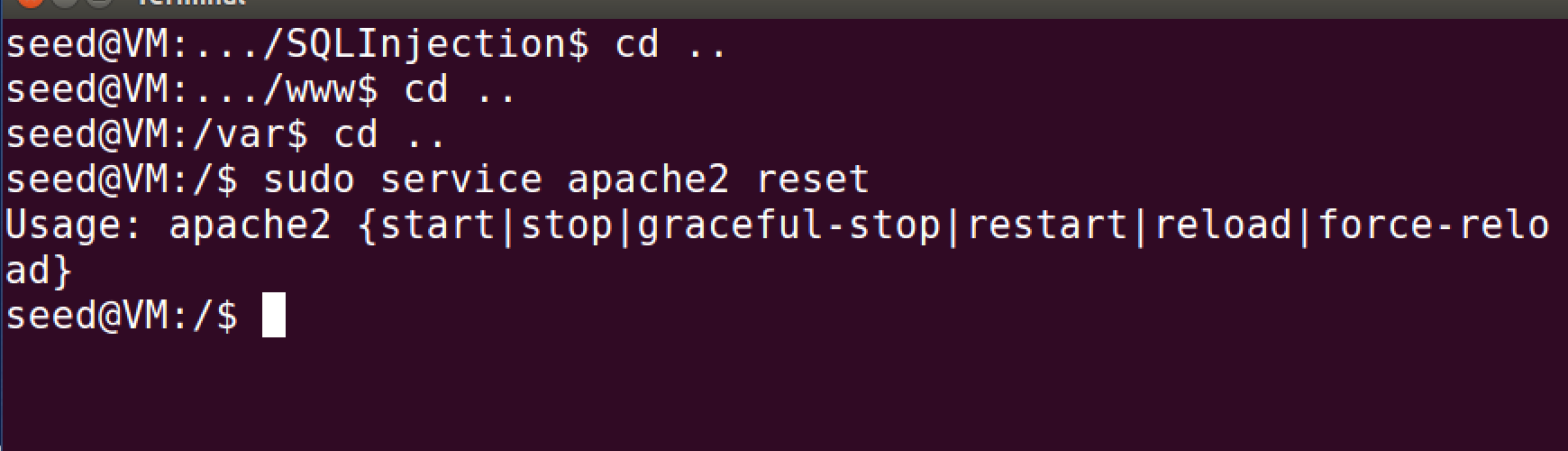
Code Used:

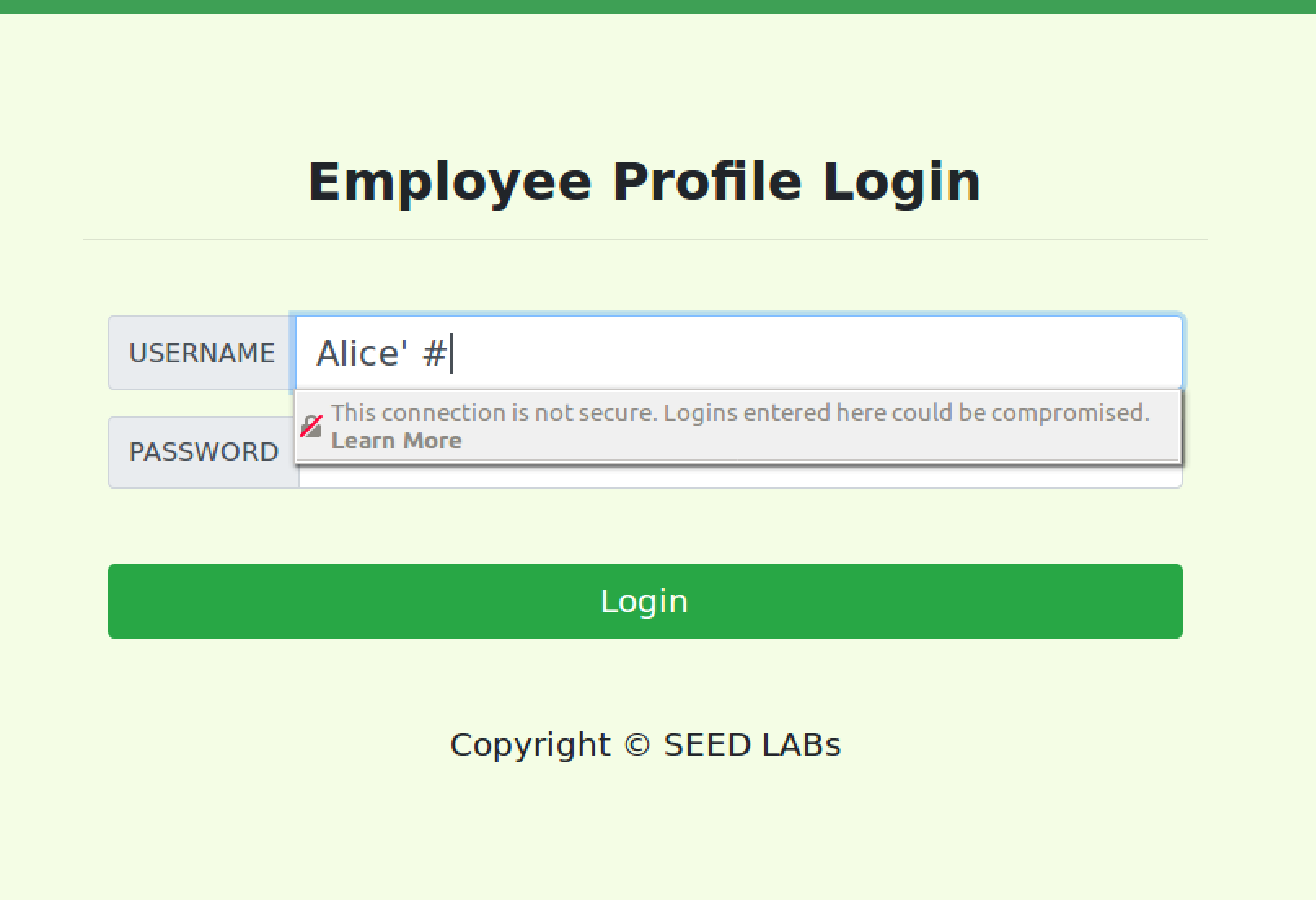


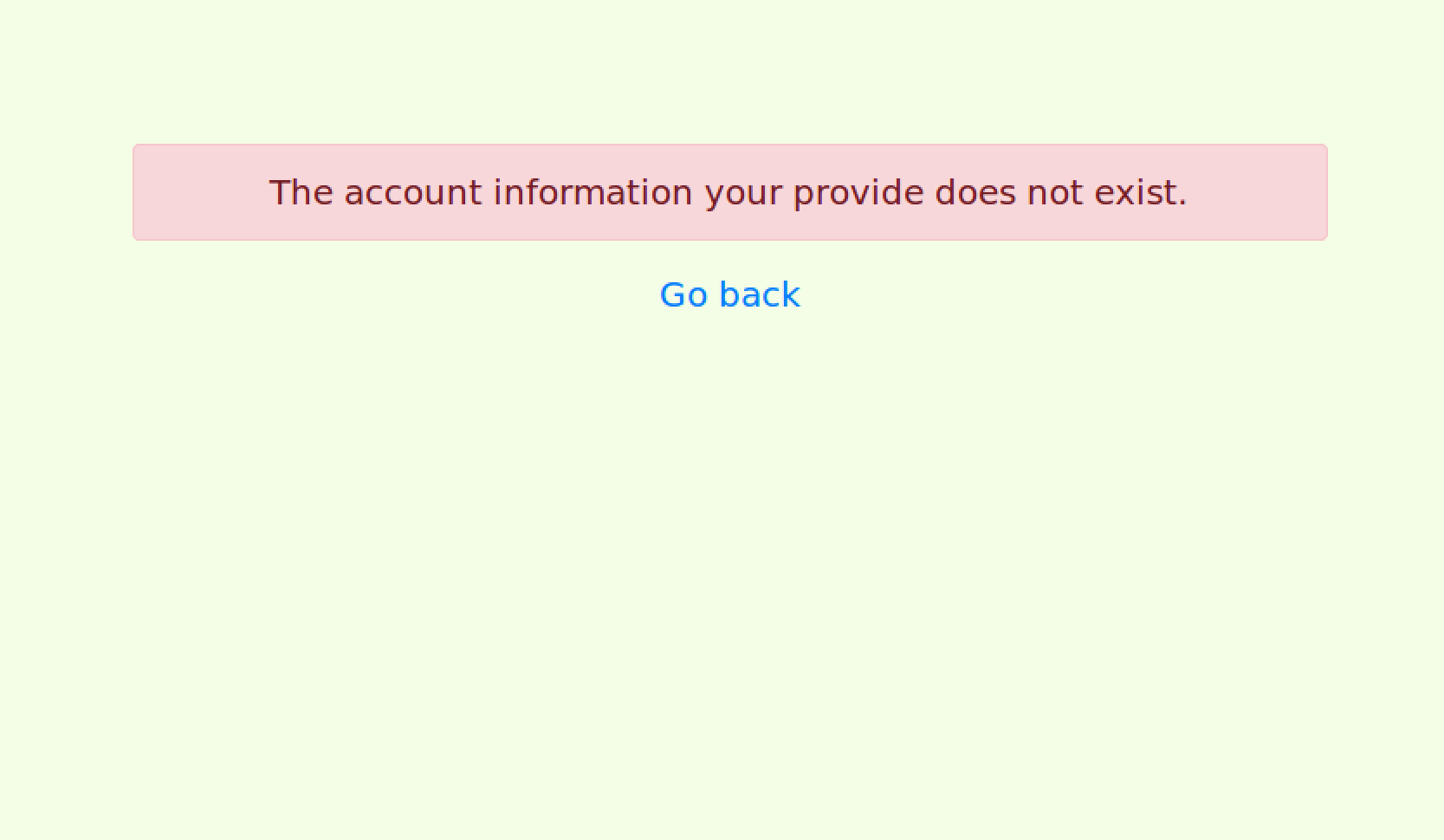


Commands and Results:









Comments:

The main issue with the site was that it was running unsafe code from the file unsafe\_home.php. The safe and nonvulnerable code is located in saf\_home.php. So all that was needed to do was change the bad code for the safe code and then the vulnerabilities were gone. As shown, I tried to do the injection again but it did not work.