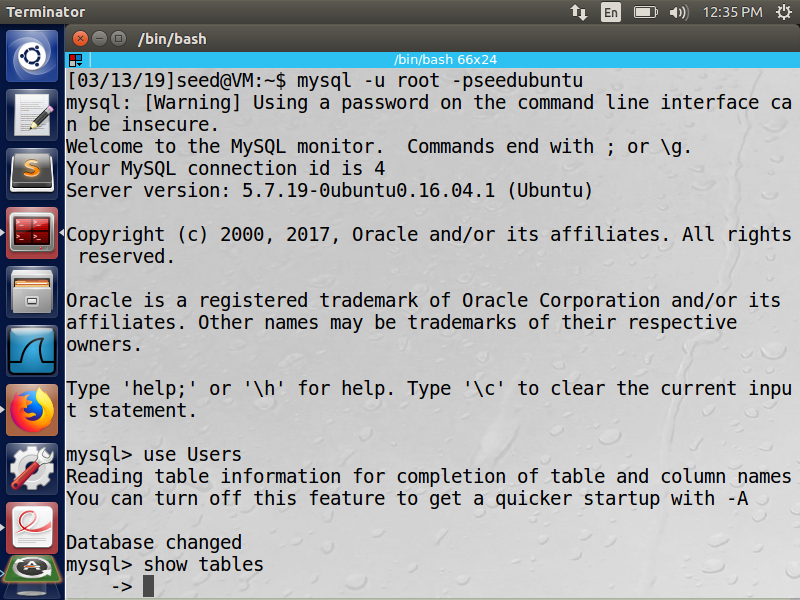
Steven Arce

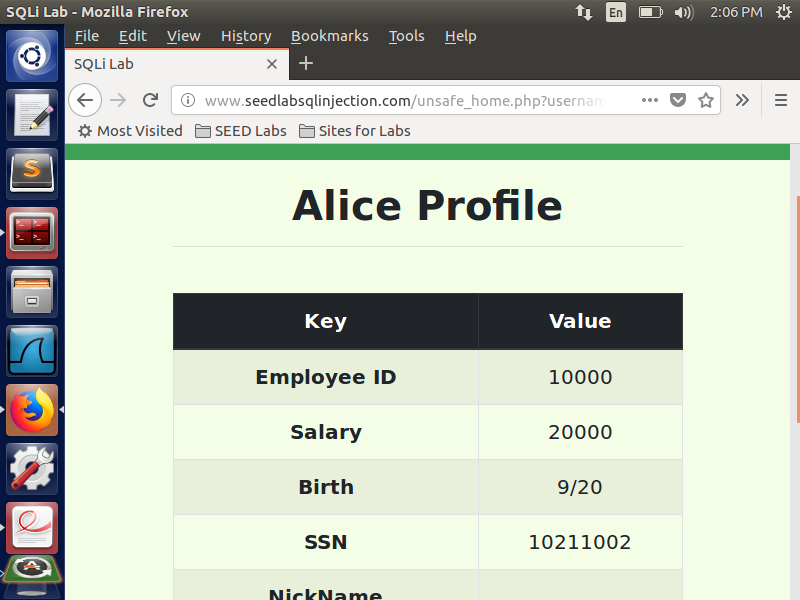
3/13/19

Computer Network Security

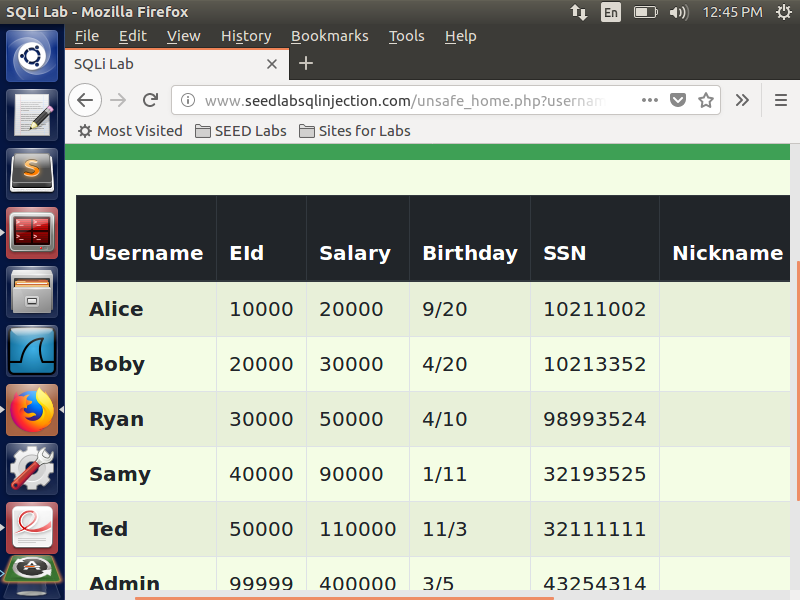
SQL Injection Lab



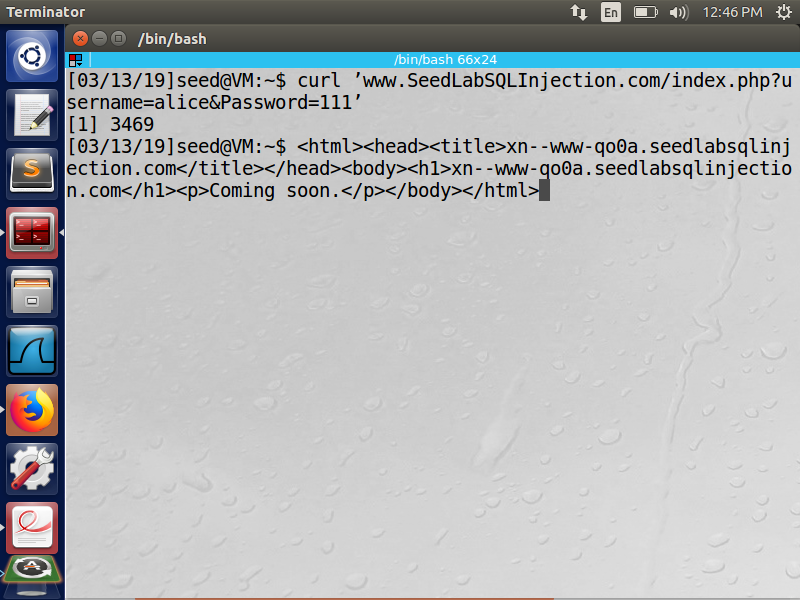
Our first goal is to connect to the database via MySQL so the Virtual Machine can access the table.



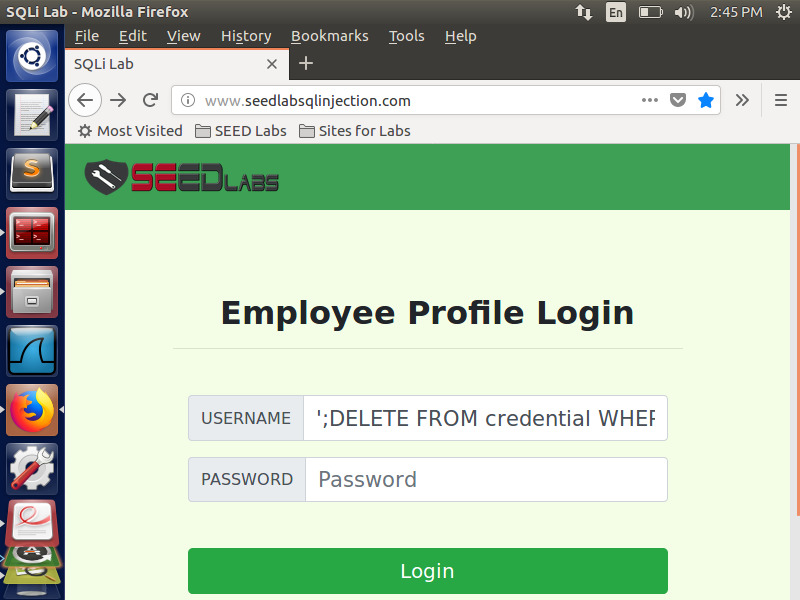
At the login page, I typed in a’ OR 1 # in order to log into Alice’s account.



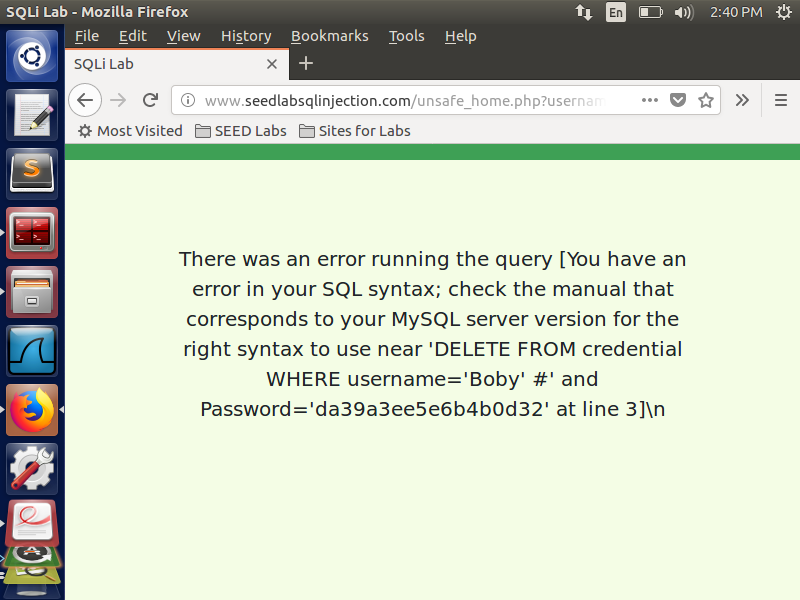
Alternatively, I can type in admin’ # to access the admin account, which revals all of the data in the credentials table.



To access Alice’s page without using the login screen, I head to the terminal to input a curl command that shows me her username and password.



Here, I typed into the username field ‘; DELETE FROM credential WHERE username=’Boby’ # in order to delete him from the table.



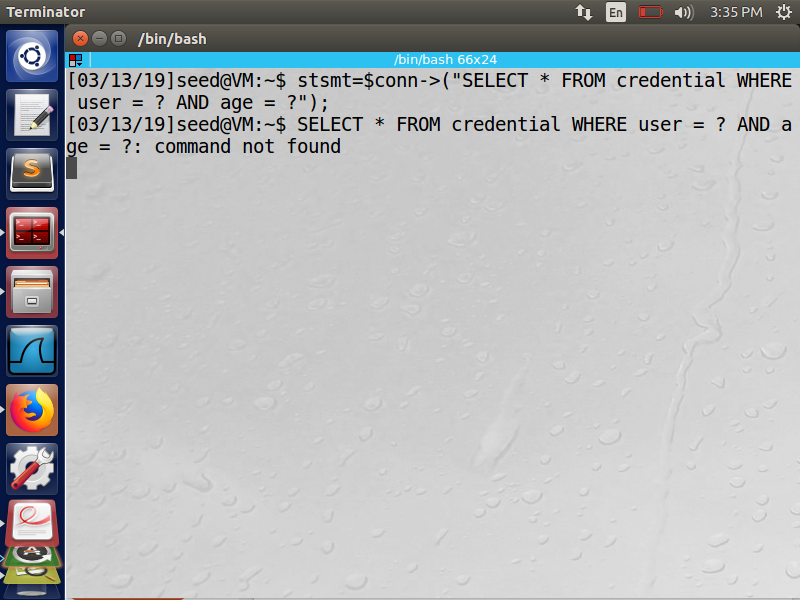
This among many other attempts did not work. Either I’m doing something wrong, or... yeah, I’m doing something wrong.

The next task sees us using our sql injection knowledge to practical (or dare I say monetary) use. Unfortunately, the only way to accomplish this attack is from the “Edit Profile” page, which is nowhere to be seen whether logged in or logged out. In loo of this, I will provide my answers to the queries here without pictures.

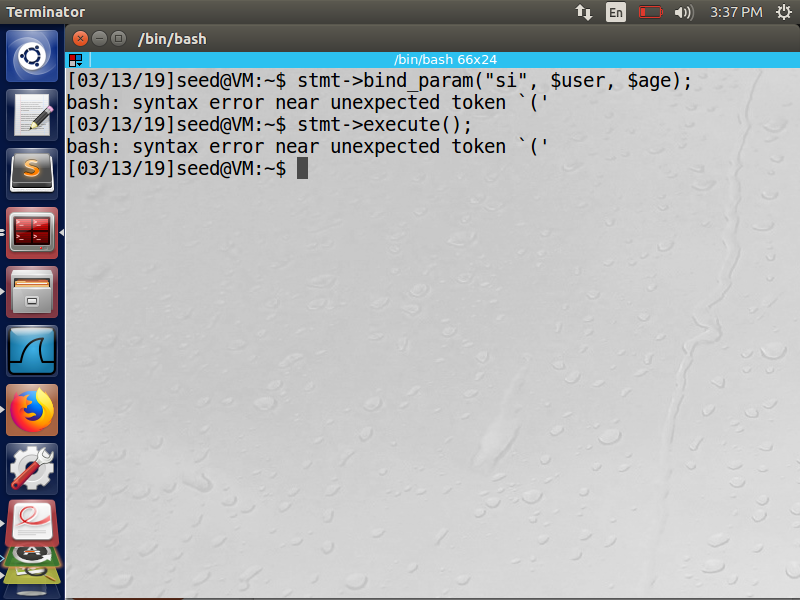
If I wants to increase my (or in this case Alice’s) salary, I would type into the “Nickname” field: a’ salary=10000000000000000000 #. This should update the salary to much better levels.

To decrease my boss’ salary, I could do one of two things: 1) I could use the skills Alice clearly has to log in as Boby and edit his salary from within his own account, applying the similar code a’, salary=19999 #; 2) tweak the query to enter into my own account a’; SET salary = 19999 WHERE username = ‘Boby’ #. Either way, I win.

For the third task, changing Boby’s password should be somewhat similar. Alter my latter example text to say a’; SET password = ‘u\_sucketh’ WHERE username = ‘Boby’ #. Then log into his account and let the games begin.



Now, we attempt to remedy the issue using a prepared statement. This also didn’t seem to be working out for me. Perhaps there was a connection issue between the database and the virtual machine that occurred along the way.



Once (or should I say if) the previous prepared statement entry worked, I could bind and execute that statement in order to stop the issue from occuring any further. Now, the database can tell the difference between a legitimate entry and a query in disguise.

This lab was very eye-opening. Again, the simplicity of the attack surprises me, as all it takes is one line of code or even a few words and a # to compromise the security of database info. However, what amazes me more is that this attack is based on a common oversight which everyone who ever tried to create a database has done (including myself). This is why I’m very glad I chose this lab, so that I can remember to implement prepared staments in the queries of my databases in the future.