

Ethan Booker

SQL Injection Attack Lab

Spring 2020

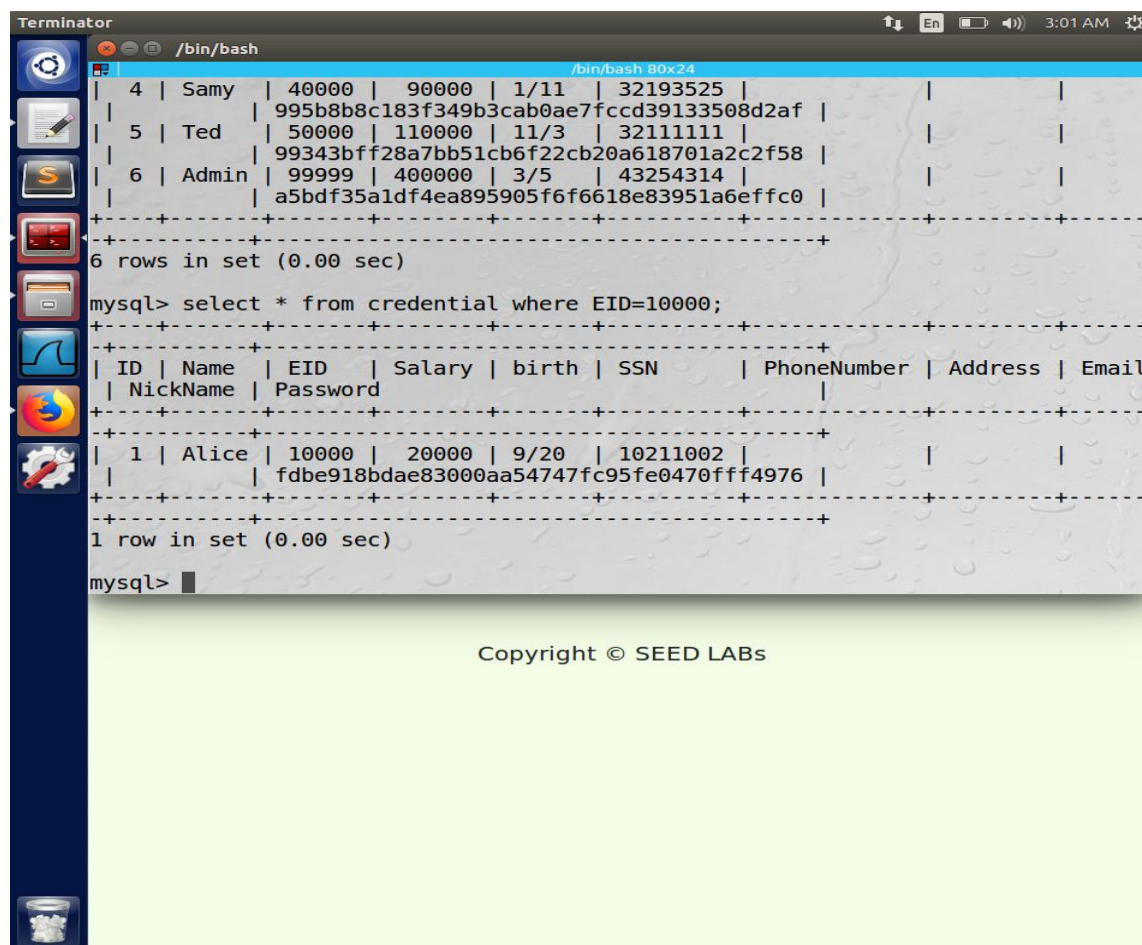
Description of overall goals of lab:

To learn how a website that uses an SQL database could be vulnerable to an attack called SQL injection, see the damage of an SQL injection, and to see how to defend such an attack.

3.1, Task 1

Description: Get familiar with SQL statements

Evidence:



The screenshot shows a Terminator terminal window with a dark theme. The terminal title is "Terminator" and the current directory is "/bin/bash". The prompt is "/bin/bash 80x24". The output shows the results of a MySQL query: "select * from credential where EID=10000;". The results are displayed in a table format with 9 columns: ID, Name, EID, Salary, birth, SSN, PhoneNumber, Address, and Email. The first row shows ID 1, Name Alice, EID 10000, Salary 20000, birth 9/20, SSN 10211002, and PhoneNumber fdbe918bdae83000aa54747fc95fe0470fff4976. The second row shows ID 4, Name Samy, EID 40000, Salary 90000, birth 1/11, SSN 32193525, and PhoneNumber 995b8b8c183f349b3cab0ae7fccd39133508d2af. The third row shows ID 5, Name Ted, EID 50000, Salary 110000, birth 11/3, SSN 32111111, and PhoneNumber 99343bff28a7bb51cb6f22cb20a618701a2c2f58. The fourth row shows ID 6, Name Admin, EID 99999, Salary 400000, birth 3/5, SSN 43254314, and PhoneNumber a5bdf35a1df4ea895905f6f6618e83951a6efffc0. The terminal also shows "6 rows in set (0.00 sec)" and "1 row in set (0.00 sec)". The prompt "mysql>" is visible at the bottom. The background of the terminal window is a light green color with a dark blue sidebar on the left containing various icons. The text "Copyright © SEED LABS" is visible at the bottom of the terminal window.

```
Terminator /bin/bash
/bin/bash 80x24
+-----+
| 4 | Samy | 40000 | 90000 | 1/11 | 32193525 | | | |
| | | 995b8b8c183f349b3cab0ae7fccd39133508d2af | | | |
| 5 | Ted | 50000 | 110000 | 11/3 | 32111111 | | | |
| | | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 | | | |
| 6 | Admin | 99999 | 400000 | 3/5 | 43254314 | | | |
| | | a5bdf35a1df4ea895905f6f6618e83951a6efffc0 | | | |
+-----+
6 rows in set (0.00 sec)

mysql> select * from credential where EID=10000;
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email |
| NickName | Password |
+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | |
| | | fdbe918bdae83000aa54747fc95fe0470fff4976 | | | |
+-----+
1 row in set (0.00 sec)

mysql>
```

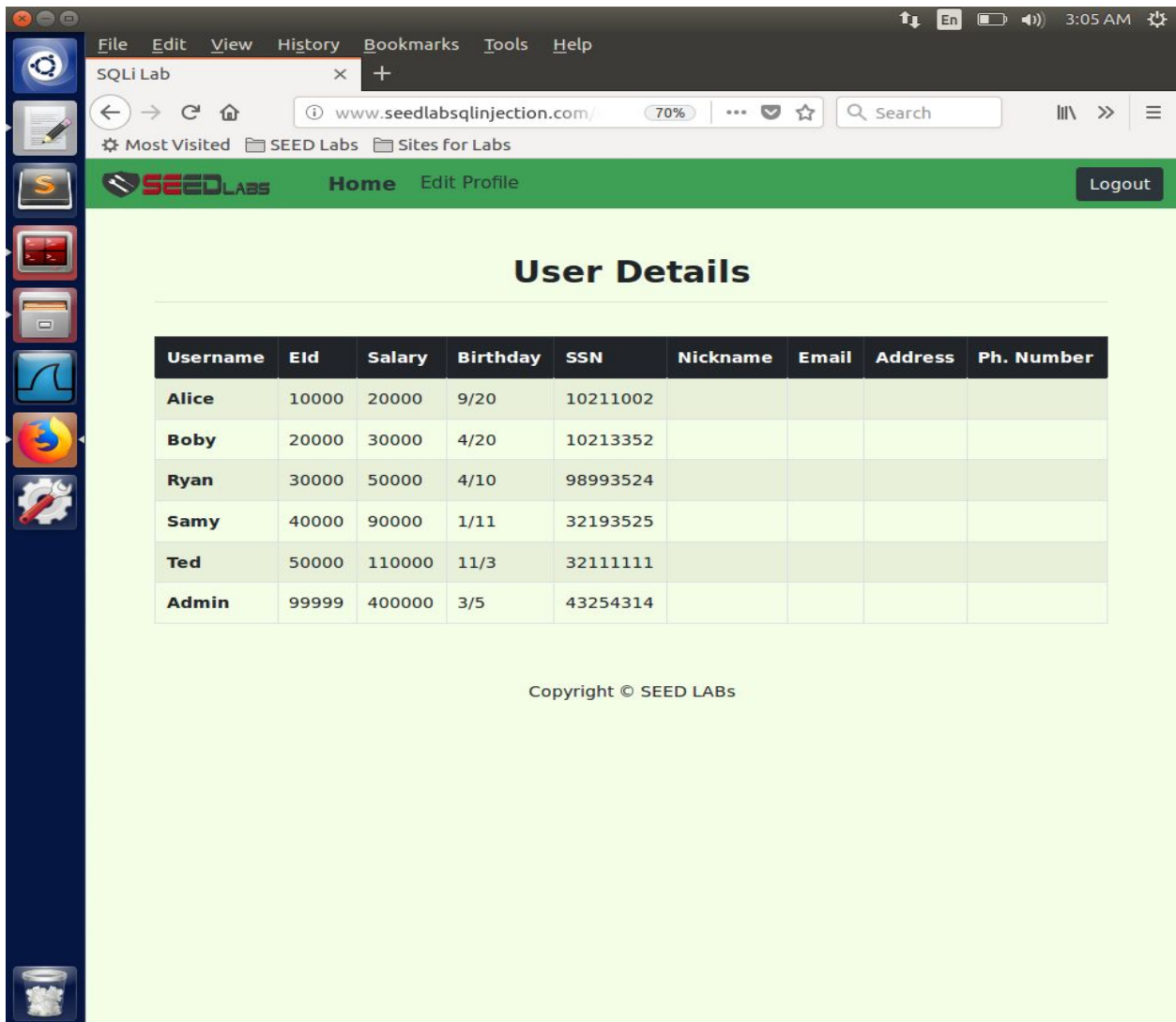
Copyright © SEED LABS

Analysis: These are simple SQL query statements and not much to say. The query used was "select * from credential where EID=10000;"

3.2, Task 2.1

Description: SQL Injection Attack on SELECT Statement

Evidence:



The screenshot shows a web browser window with the address bar displaying `www.seedlabsqlinjection.com/`. The page title is "SQLi Lab". The browser's address bar shows the URL `www.seedlabsqlinjection.com/` and the page is zoomed to 70%. The page has a green header with the "SEED LABS" logo, a "Home" link, an "Edit Profile" link, and a "Logout" button. The main content area is titled "User Details" and contains a table with user information. The table has columns: Username, Eld, Salary, Birthday, SSN, Nickname, Email, Address, and Ph. Number. The table lists six users: Alice, Bobby, Ryan, Sammy, Ted, and Admin. The Admin user has a salary of 400,000 and a birthday of 3/5. The footer of the page reads "Copyright © SEED LABS".

Username	Eld	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				

Analysis: In the Username text box I typed in ' or Name="admin";# since the single quote has special meaning in a normal statement then I am able to inject my own statements in there then get rid of any other statements afterwards using the comment symbol #. I already knew that Username mapped to the name column so I can specifically look for a name that is admin.

3.2, Task 2.2

Description: SQL Injection Attack from command line

Evidence:

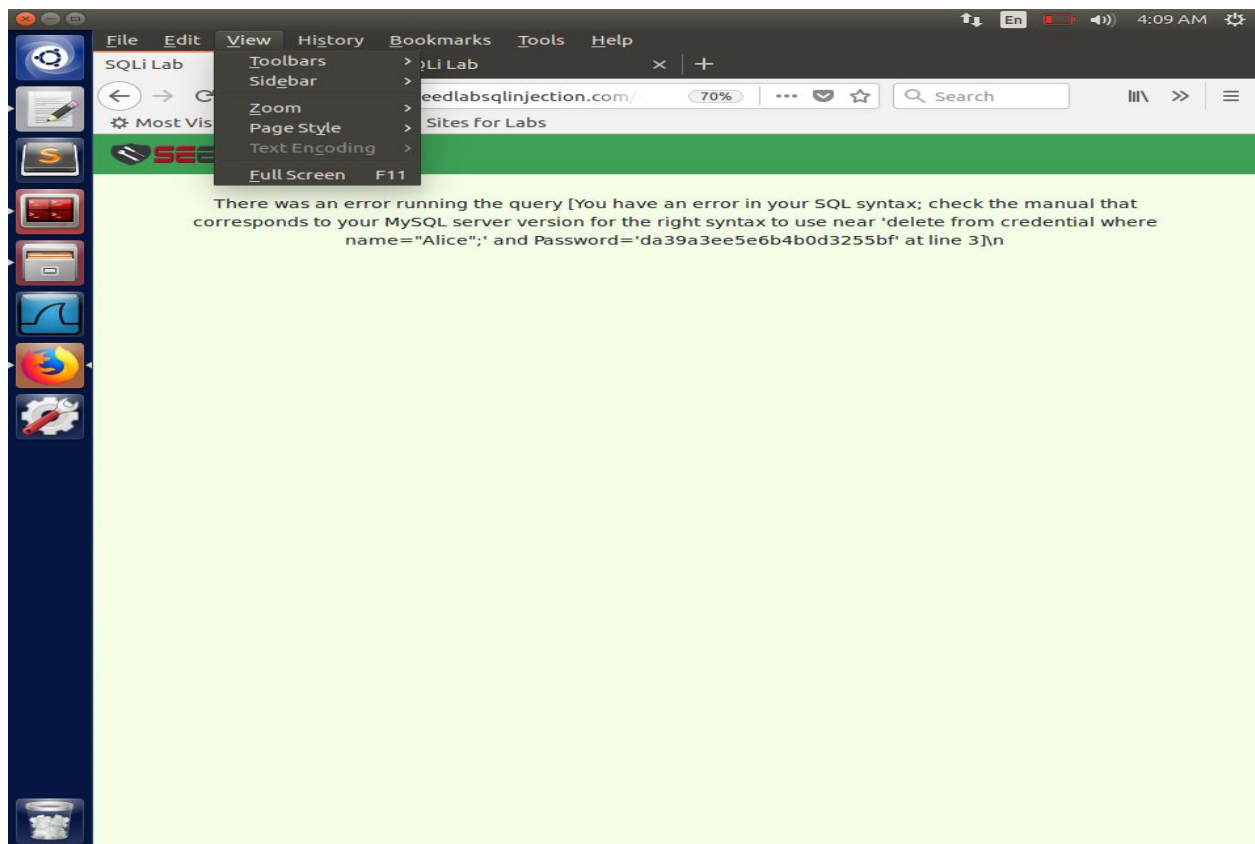
[illegible]

Analysis: I was able to obtain the url through the website's current url header. By doing that, I was able to call curl and mimic the same attack but through the command line.

3.2, Task 2.3

Description: Append a new SQL statement

Evidence:

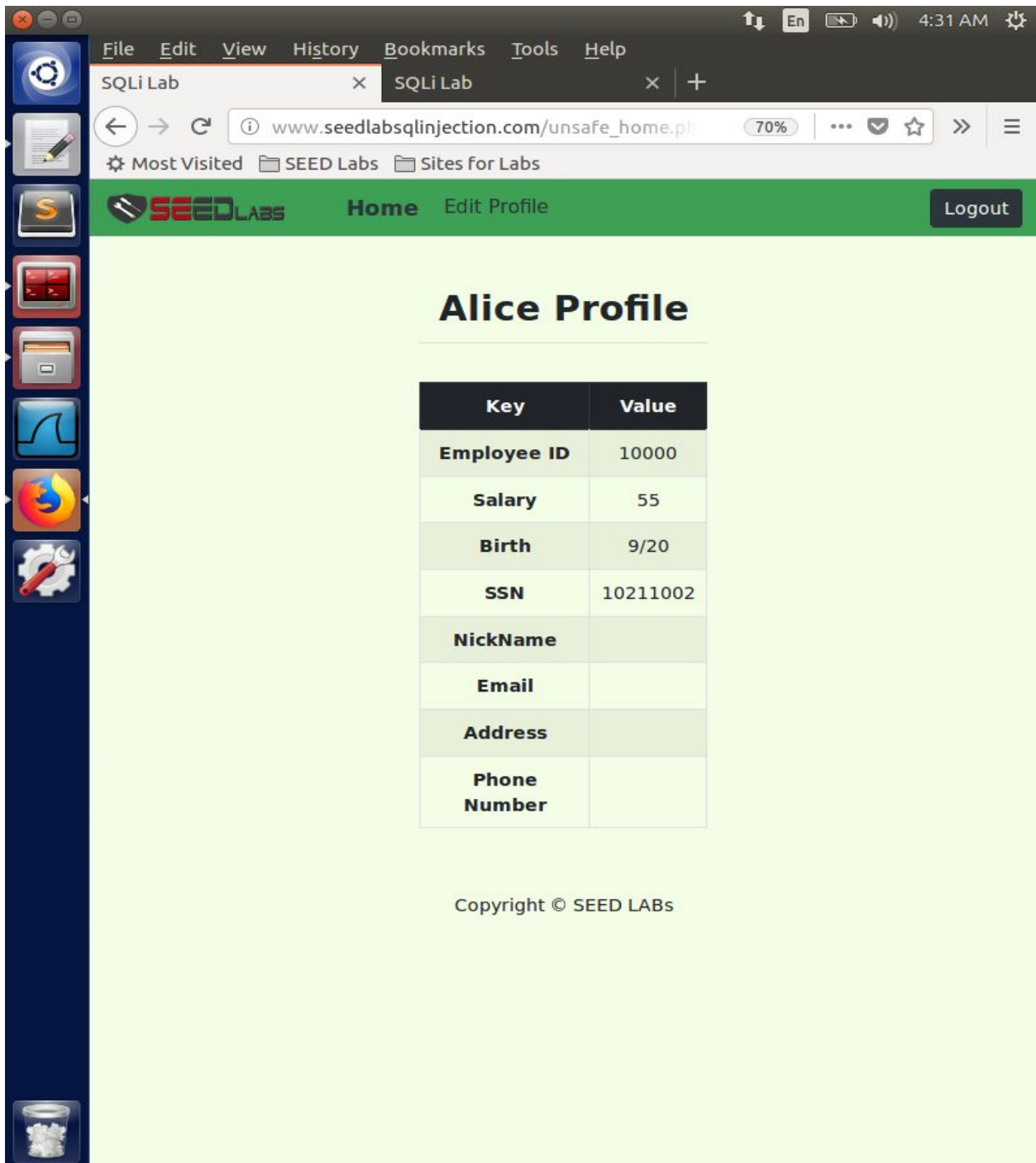


Analysis: I got an error when I tried to append another SQL statement after my first statement in logging in. This happened because after the first statement (' or 1=1;) is executed, we get a syntax error due to how SQL compiles the query. So even after adding a # after my second statement there will still be an error because you cannot execute two statements in one query.

3.3, Task 3.1

Description: Modify Alice's salary

Evidence:



The screenshot shows a web browser window with two tabs labeled 'SQLi Lab'. The address bar displays the URL 'www.seedlabsqlinjection.com/unsafe_home.ph' with a 70% zoom level. The website's header is green and contains the 'SEEDLABS' logo, 'Home' and 'Edit Profile' links, and a 'Logout' button. The main content area has a light green background and features the title 'Alice Profile' above a table of user data.

Key	Value
Employee ID	10000
Salary	55
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

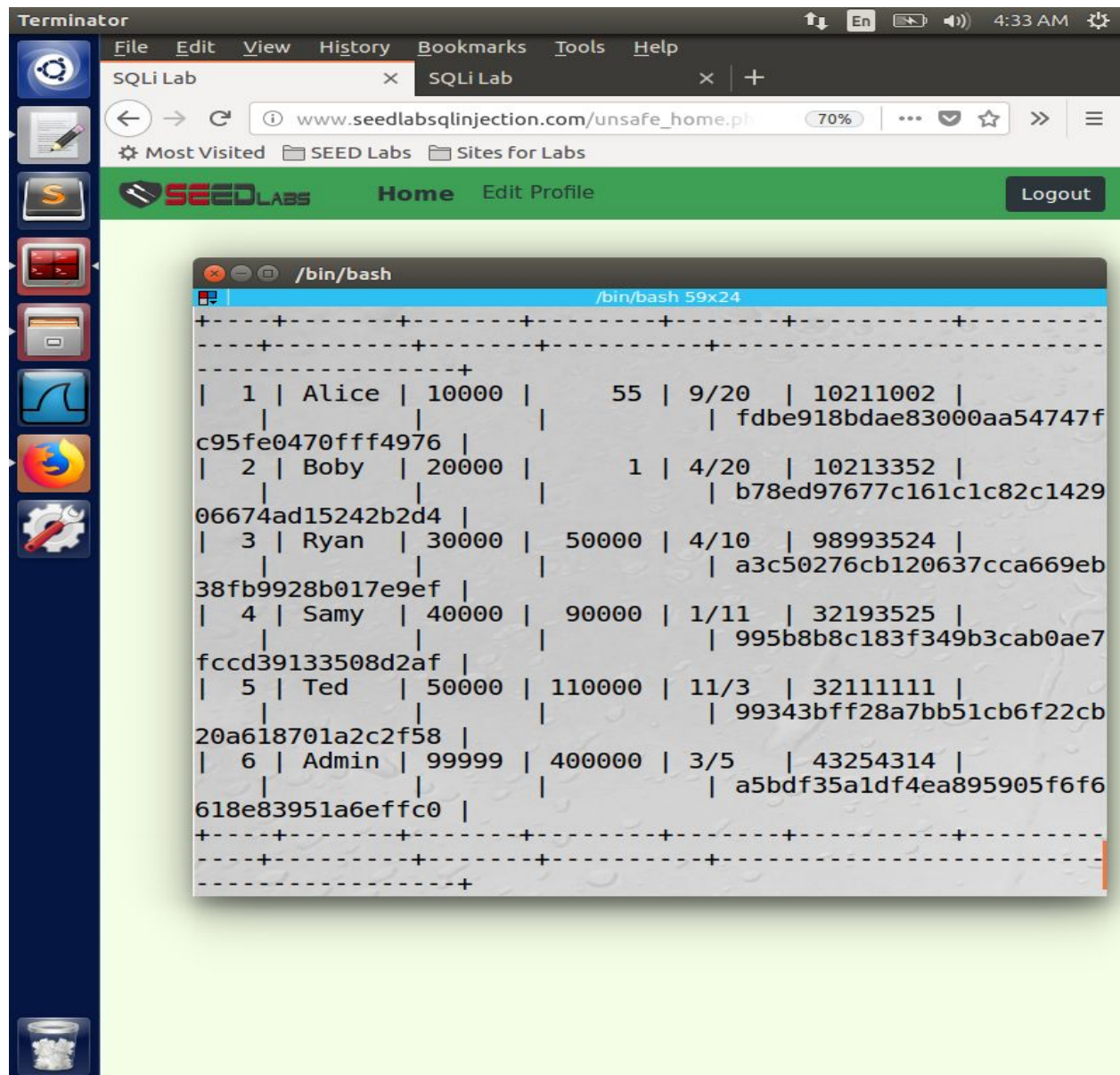
Copyright © SEED LABs

Analysis: In the NickName text box, I inputted ' , Salary="55" where Name="Alice";# '. Due to how the query was set up, I used a comma then I set which attribute I wanted to change then added the where statement to select a specific row resulting in Alice's salary being updated.

3.3, Task 3.2

Description: Modify Bobby's salary to 1.

Evidence:



The screenshot shows a Terminator window with a web browser displaying the SEED Labs website. A terminal window is open, showing the output of a database query. The query results are as follows:

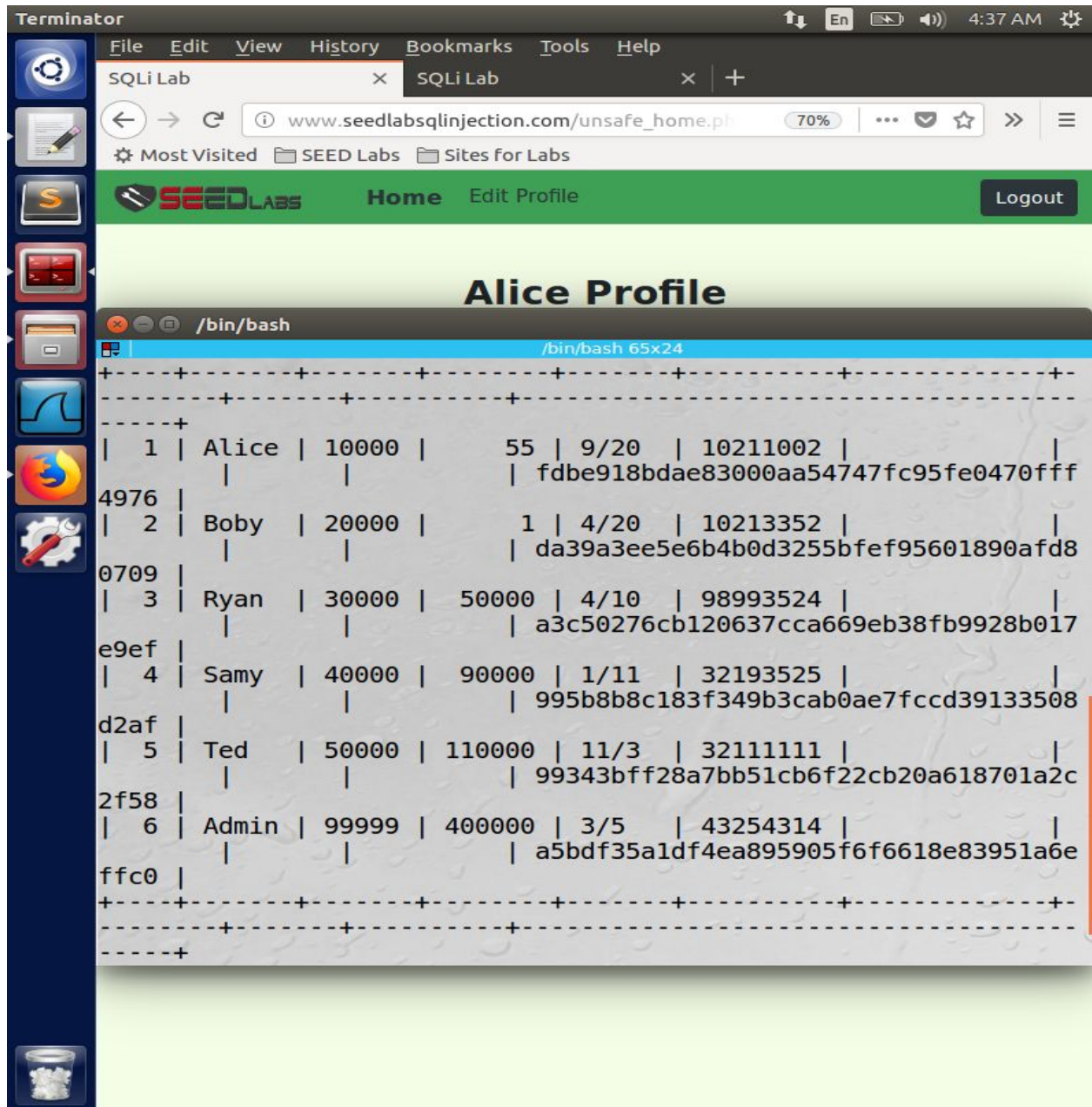
ID	Name	Salary	Age	DOB	Hash
1	Alice	10000	55	9/20	10211002 fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	1	4/20	10213352 b78ed97677c161c1c82c142906674ad15242b2d4
3	Ryan	30000	50000	4/10	98993524 a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	90000	1/11	32193525 995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	110000	11/3	32111111 99343bff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	400000	3/5	43254314 a5bdf35a1df4ea895905f6f6618e83951a6effc0

Analysis: Similar to task 3.1, I just changed the query to ' , Salary="1" where Name="Boby";# and this changed Bobby's salary. I showed the database to show the change. I did not need to leave Alice's edit profile in order to do this.

3.3, Task 3.3

Description: Modify Bobby's password

Evidence:



The screenshot shows a web browser window with the URL www.seedlabsqlinjection.com/unsafe_home.php. The page title is "Alice Profile". Below the title is a table of user profiles. A terminal window is open over the table, showing the command `echo test | openssl sha1` and its output.

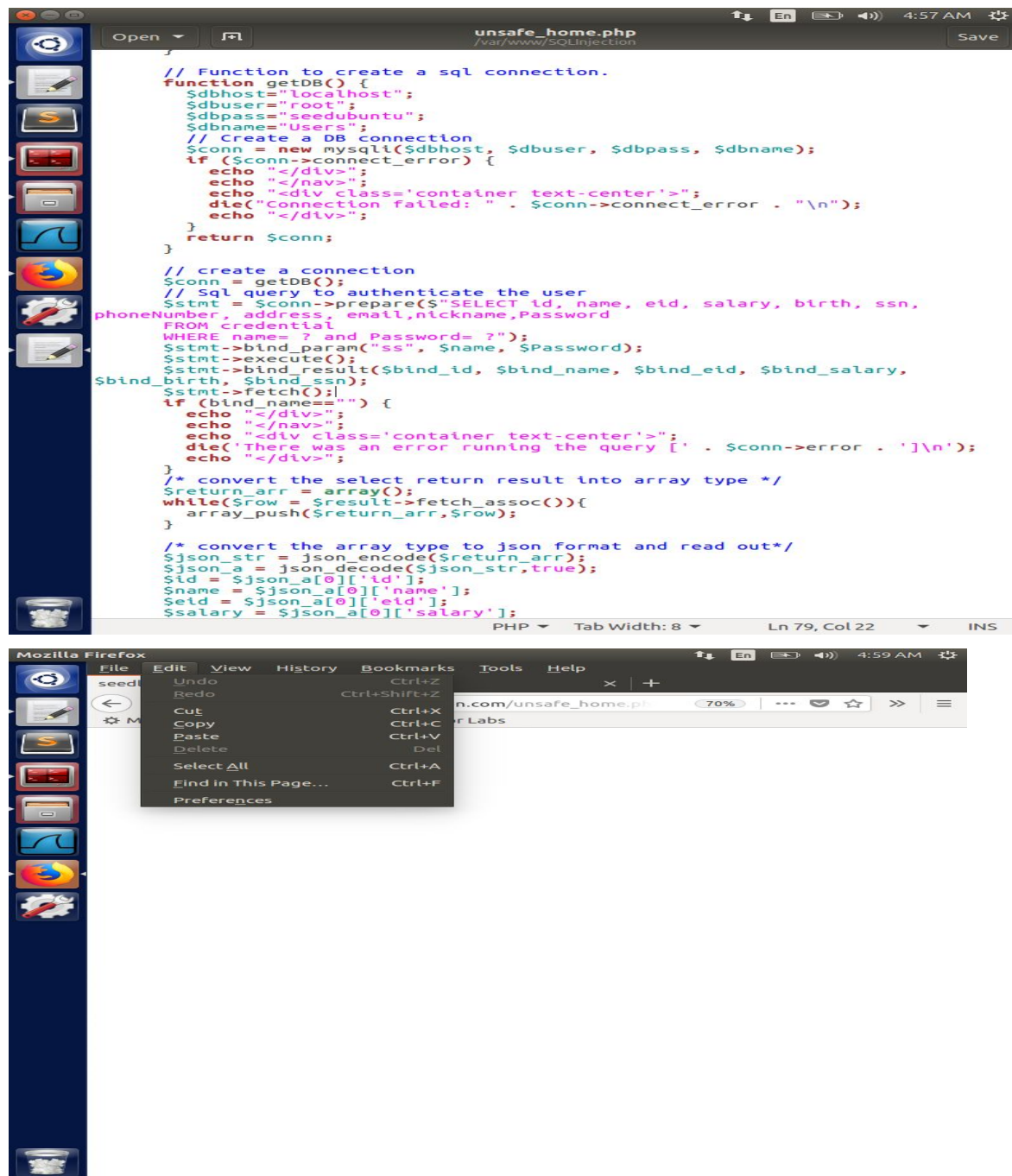
ID	Name	Salary	Age	DOB	Phone	Hash
1	Alice	10000	55	9/20	10211002	fdbe918bdae83000aa54747fc95fe0470fff
2	Boby	20000	1	4/20	10213352	da39a3ee5e6b4b0d3255bfebf95601890afd8
3	Ryan	30000	50000	4/10	98993524	a3c50276cb120637cca669eb38fb9928b017
4	Samy	40000	90000	1/11	32193525	995b8b8c183f349b3cab0ae7fccd39133508
5	Ted	50000	110000	11/3	32111111	99343bff28a7bb51cb6f22cb20a618701a2c
6	Admin	99999	400000	3/5	43254314	a5bdf35a1df4ea895905f6f6618e83951a6e

Analysis: This task is similar to task 3.2, but rather than saying Salary, I used Password="...". The hash is different and from task 3.2 for Bobby. How I obtained the hash was due to opening the terminal and typing `echo test | openssl sha1`.

3.4, Task 4

Description: Countermeasure - prepared statement

Evidence:



Analysis: By changing how the sql statement is executed by using a prepared statement in order to send only the query part without the data then the ? marks are replaced with the actual binded parameter data. The prepared statements prevent special meanings from occurring with characters as data inputs. As for the second image, this is due to me testing the login page with

the successful ' or Name="admin";# but it didn't work this time and showed a white screen due to the if (\$bind_name="") ... as this gave an error. Perhaps if this was correct then the correct screen would show up, but regardless the attack did not work anymore.

Conclusion: In this lab I learned how to apply some SQL commands through a website and through the terminal to create an SQL Injection. I have a better understanding on how to test if a website is using SQL by trying to execute a simple SQL injection statement. Most importantly, I learned how to defend against this attack using prepared statements and I have a better understanding as to why an SQL Injection is possible when they could easily just use a prepared statement or sanitize the input data.