Project 2 - SQL Injection Lab

Cybersecurity COSC 4363

Dr. Xiagya Lui

Andrew Kalathra

11/5/2023

1. **Overview**

* This lab is meant to explain and show how a vulnerable website can be targeted maliciously through the use of SQL Injections

1. **Lab Environment**

* I used the Ubuntu Seed (16.04 – 32bit) provided to run as a virtual machine.
* The virtual machine software was Virtual Box, specifically the 7.0.12 edition (latest edition still works for this lab, though it is not recommended).
* The vulnerable website used was <https://www.SEEDLabSQLInjection.com>
* This website specifically only runs on the Ubuntu system, and all system files are available to view on the system.
* Proof of completion can be told by the folder present in the upper portion of the screen labelled ‘Andrew Kalathra’.

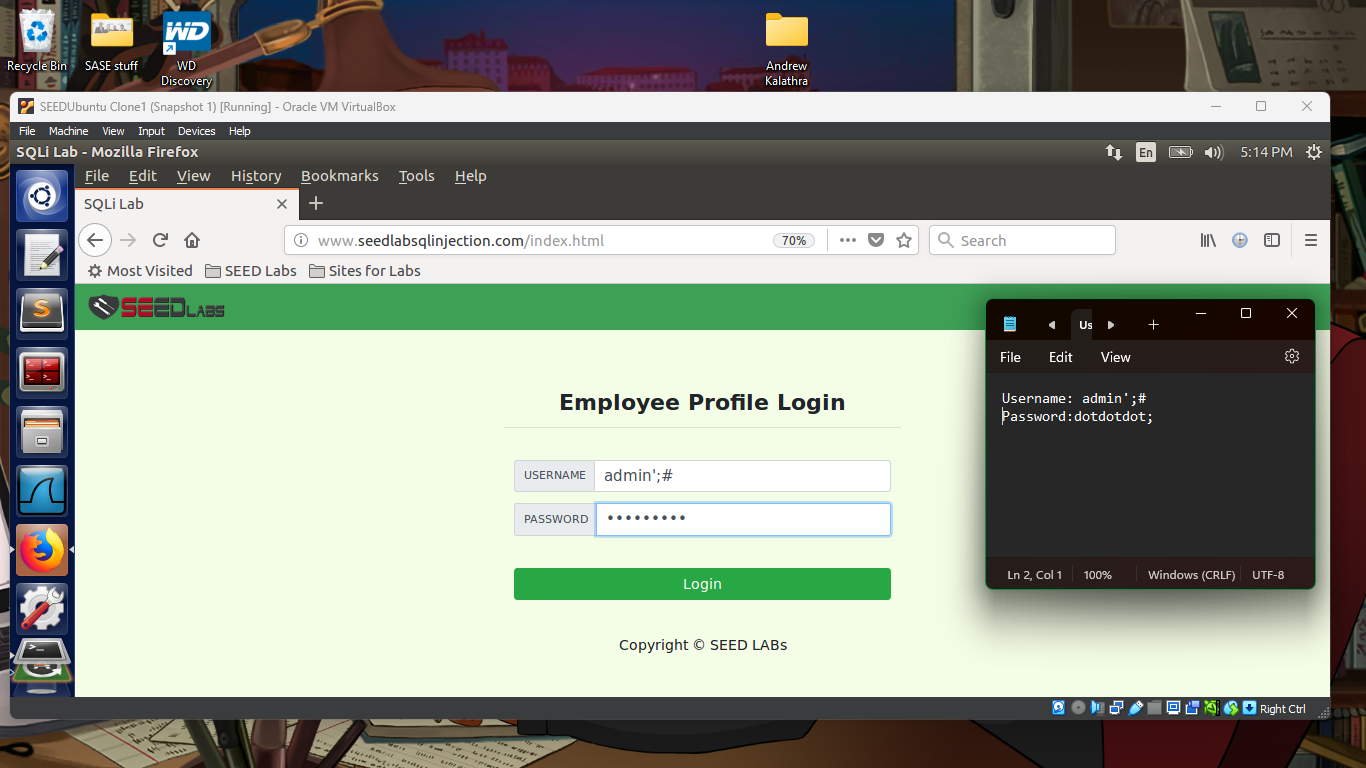
1. **Lab Tasks**

* **3.1 Getting Alice**

A screenshot of a computer

Description automatically generated

* + While in the SQL database, we are able to access all the credentials of the employees.
  + Specifically, we were tasked with accessing Alice’s information, as shown above.
* **3.2 SQL Injection Attack on SELECT Statement**
  + **2.1 SQL Injection Attack from Webpage**



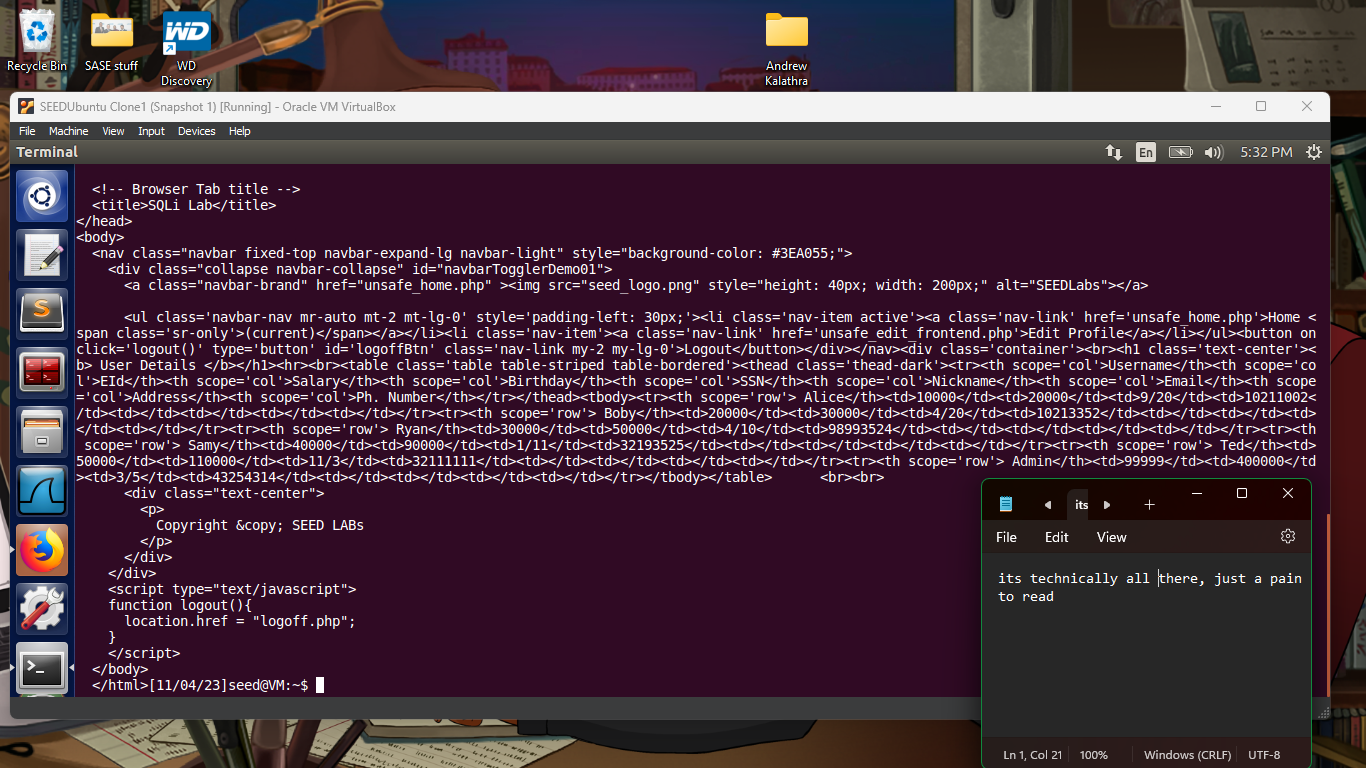
* + - First, we needed to finish the string, so we needed to use ‘ (single quotation)
    - We then needed to finish the statement, (though this is not necessary) so we use ; (semicolon)
    - To comment out the rest of the line, we use # (hash tag)

A screenshot of a computer

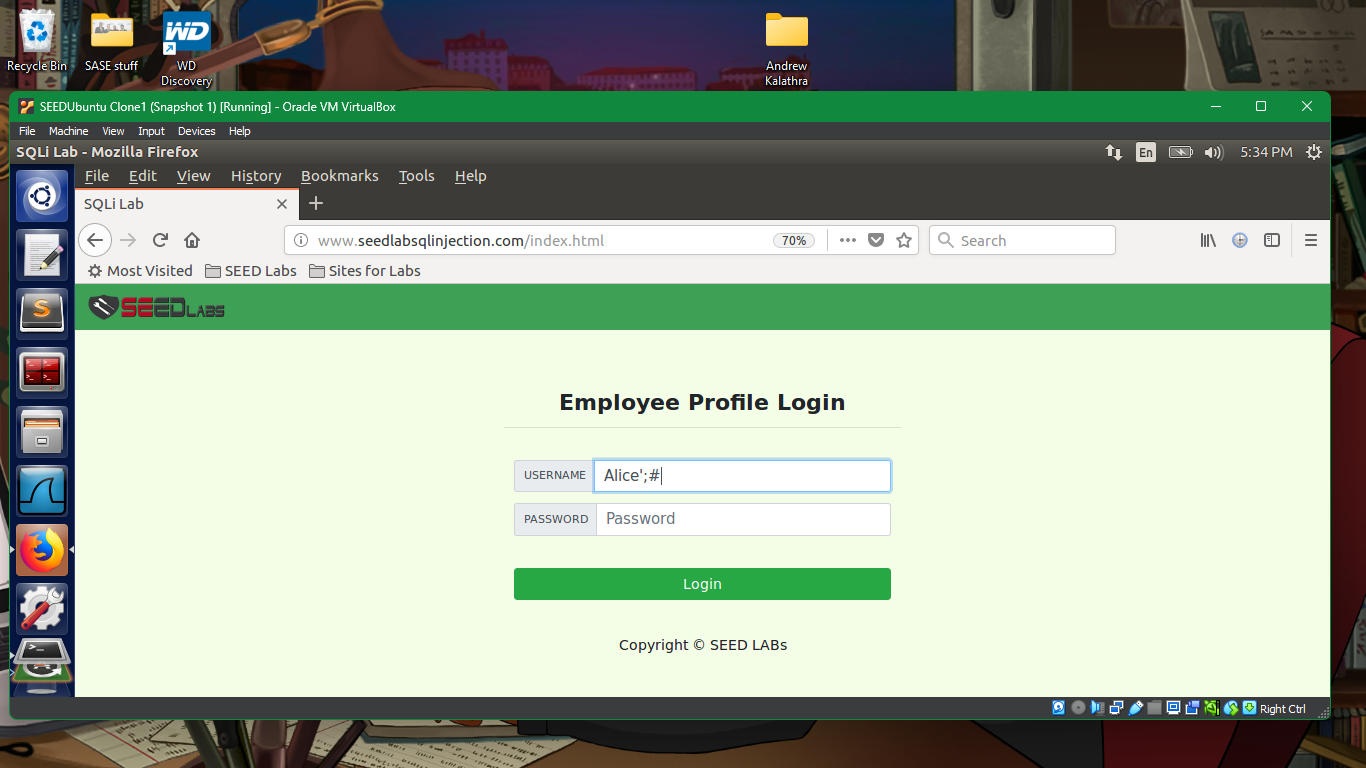
Description automatically generated

* + **2.2 SQL Injection Attack from Command Line**





* + - This was a simple curl statement.
    - The output was not what was expected, which was a clean diagram/chart of the data.
    - Instead, an html version of the page was given.
    - The data is still readable, but it is a nuisance to read.
  + **2.3 Append a new SQL statement**
    - Many attempts were made for this, however, none were successful.
    - This is because the PHP extension does not allow multiple queries in one. So appending an SQL statement only returns an error.
    - If one was able to do this, however, the appended portion would follow the URL
* **3.3 SQL Injection Attack on UPDATE Statement**
  + **3.1 Modify Your Own Salary**



* + - First, we need to login to Alice’s account. We can use the same technique we used to get into the admin’s account.

A screenshot of a computer

Description automatically generated

* + - Next, we need to Edit Alice’s Profile.

A screenshot of a computer

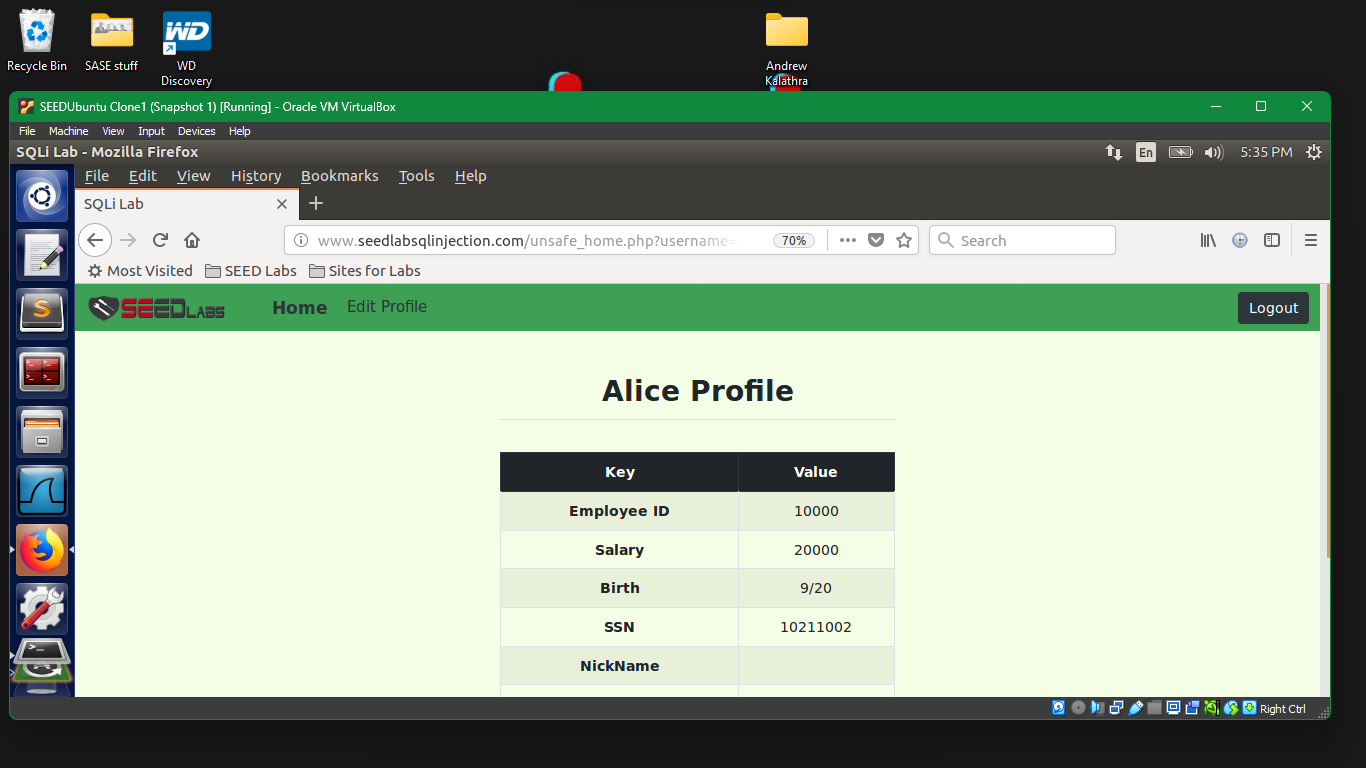
Description automatically generated

* + - In Alice’s profile, we can enter our desired nickname.
    - This update field also allows us to enter a new field, salary!

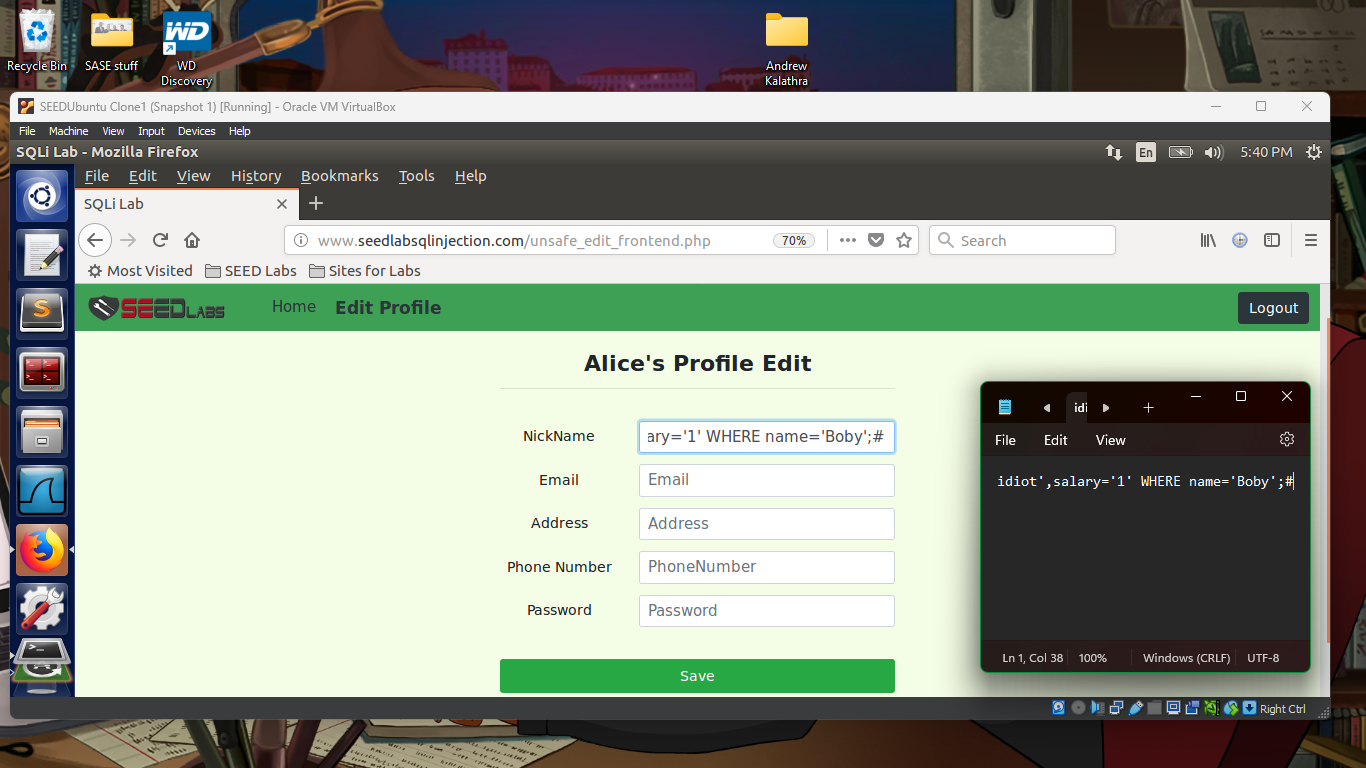
A screenshot of a computer

Description automatically generated

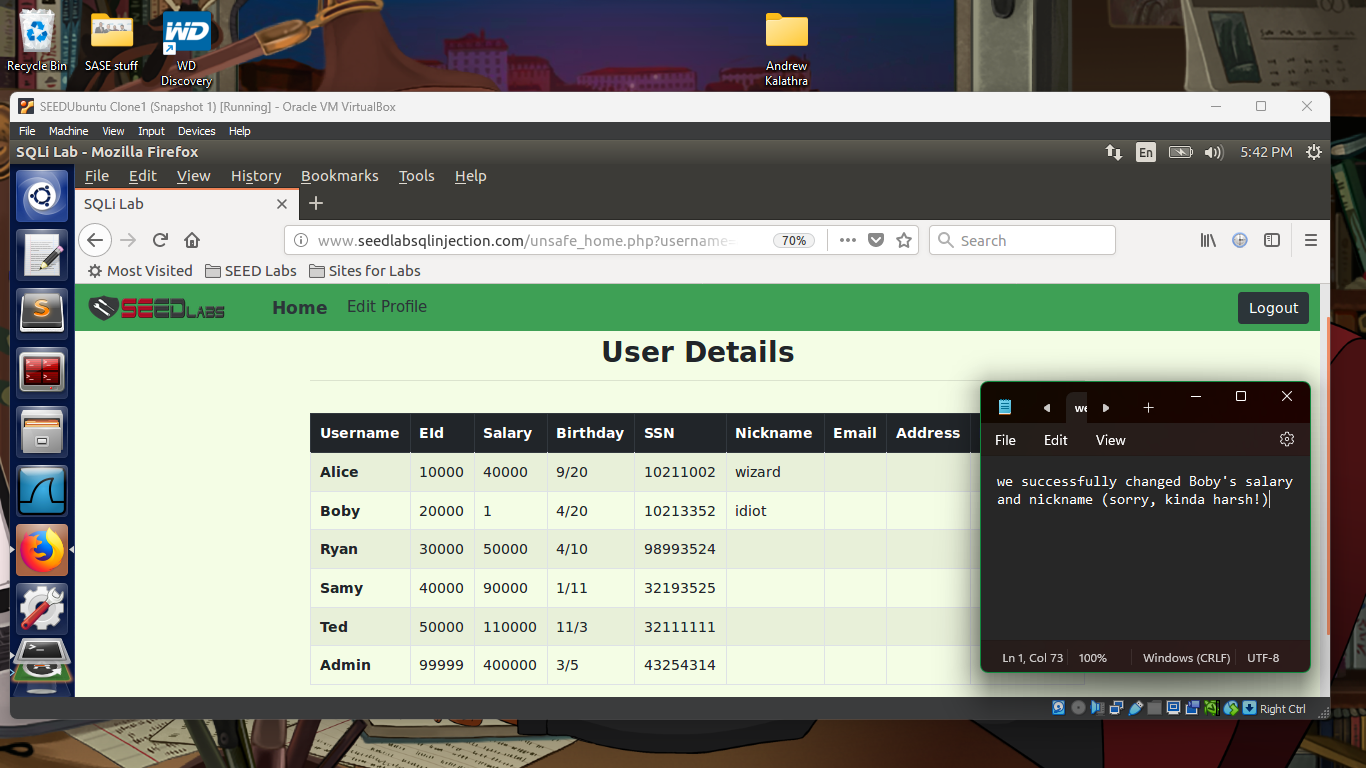
* + - We know this attack worked as we have now doubled Alice’s salary
  + **3.2 Modify Other People’s Salary**



* + - While still in Alice’s profile, we can try using the update field in other ways.



* + - Adding our own WHERE statement will allow us to change which entry is actually being updated.



* + - By logging into the admin’s account, we can tell that Boby’s salary is now much less that he ever wanted to see it.
  + **3.3 Modify Other People’s Password**

A screenshot of a computer

Description automatically generated

* + - Since we are messing with a one-way hash, we need to know what the hash is going to convert the password into.
    - By using an online tool, we can see what our hash function does to our initial password string.

A screenshot of a computer

Description automatically generated

* + - Using a different field this time (for variety), we can edit the WHERE statement to change which account is affected.
    - Alongside this, the additional password field we added must also take in the hash string and not directly the password string.

A computer screen shot of a login page

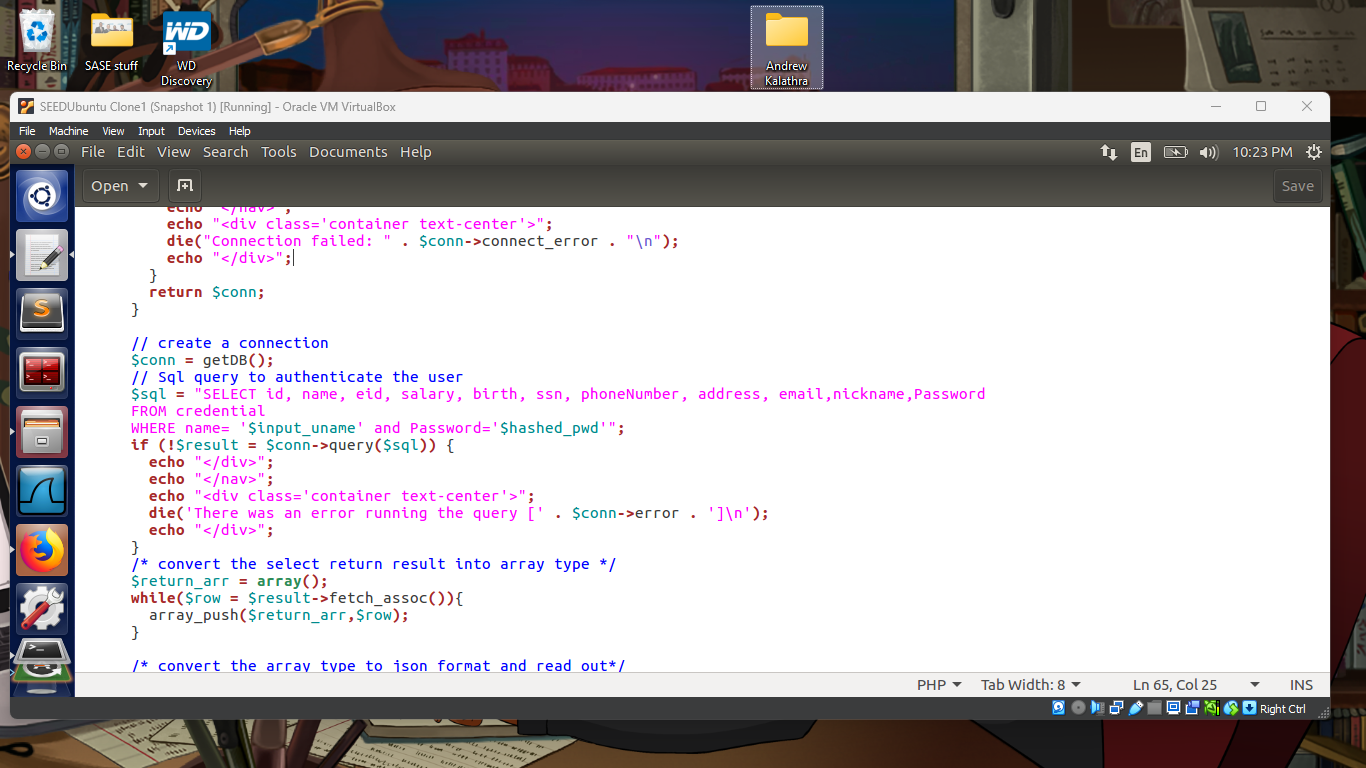
Description automatically generated

* + - Now its time to try to see if we can get into Boby’s account with our newly set password.

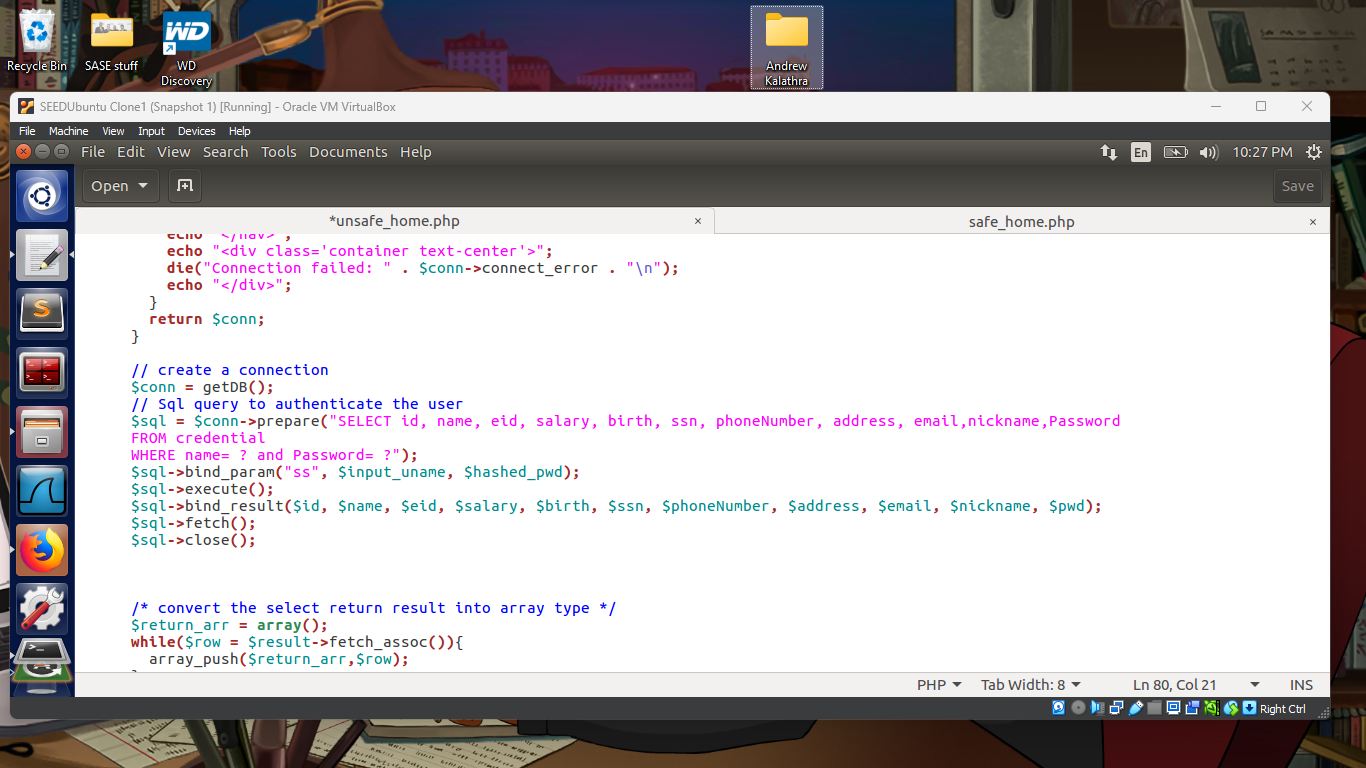
A screenshot of a computer

Description automatically generated

* + - We can see by our successful login that we have successfully changed Boby’s password.
* **3.4 Countermeasure – Prepared Statement**



* + In the unsafe\_home.php, we can locate the exact part of the unsafe code. By changing this, we could then try out and see if SQL Injections would make this website vulnerable once more.



* + As we do not have admin rights to edit the files, we cannot actually save the files. If we were allowed to, this is what is should have been changed to.
  + The assumption is that this would have prevented an SQL Injection Attack as the prepared statements would have already been compiled. Then when any data is put in their designated fields, the data would go straight to the pre-compiled query. No matter if this was a partial SQL statement, it would not be read as one, but rather as just plain data.
  + An additional note, we also tried to change the linking of the website, but to no avail, as we did not have admin rights to change the files themselves.

1. **Conclusion**

* In conclusion, we were able to understand and experiment with SELECT and UPDATE statements with read data.
* We were able to practice and explore the vulnerabilities of some websites through the use of SQL Injections.
* Finally, we were able to discuss a countermeasure to prevent this, that being prepared statements.