Lab 1 Report

**Introduction**

Lab 1 consisted of a problem that we had to handle that consisted of finding the password for 100 accounts for some kind of software that had been hashed and stored in a TXT file. This TXT file contained information such as the accounts user ID, salt value (a string added to the actual password in order to perform a hashing algorithm), and finally a hashed password that we had to compare with our version of it, this was all to be done using recursion.

**Solution**

The solution to this problem was to generate every possible numerical password combination that had the requirement of being 3 – 7 integers long, this solution included the use of two For loops in order to achieve the desired length and every possible combination. There were two parameters that were received, on being the string to compare the passwords to (S), and the other being the length of the password required at that specific iteration (N). One For loop had the task to check every combination using the digits from 0 to 9 using a method called range and incremented with every recursive call. The other for loop would increment the desired length of the password and that specific instance, if so, it would call the method named “check” which had multiple tasks. The check method would handle tasks like reading and storing all information from the TXT file, it would also concatenate (alongside the method call concatenate) the generated password with the salt value in order to perform the hashing algorithm included in the method hash\_with\_sha256. After hashing the generated password, it would compare those to the stored hashed passwords from the TXT files to see if a possible password had been found. Finally printing all possible passwords was the last task of this program.

**Experimental Results**

The results yielded by the program had every possible combination checked and allowed the comparison of those generated to those existing to see the ones that were the true password. This experiment allowed for such passwords to be generated.

* User21 password is: 113 HASHED = 41bc3fa390da1d3a8cac896958f073fad5bda7fc1f61b64a8395a66509e31aac
* User4 password is: 942 HASHED = 5372c991ea3b3c9b8eb0e094188d985dba415099687c3d60ca230aedcc3b740b

**CONCLUSIONS/ LEARNING OUTCOME**

This project did not only allow me to experience some recursion methods and a text file reading process, but it was a great introduction and learning experience for the Python language. I had used python once before in my coding career but never to the extent required in this program. Alongside python being a learning experience, I got to realize and understand a concept that I had heard about before, Hashing passwords. Security is one of the most important topics in today’s world as everything is connected by servers and the internet, and preventing someone’s information to be accessible by anyone, simply by hashing, is a great method and a true eye opener to what is out there for programmers to use as tools.

**Appendix / Source Code**

# Federico Marin (ID: 88736754)  
# Lab 1 option B  
# 9/9/2019  
   
import hashlib  
  
def hash\_with\_sha256(str):  
 hash\_object = hashlib.sha256(str.encode('utf-8'))  
 hex\_dig = hash\_object.hexdigest()  
 return hex\_dig  
  
# check method allows the file to be read and to retrieve all information from text file necessary to make program  
# compile. also stores information read from text file into variables for further manipulation.  
def check(s):  
 # try/catch was not necessary as file was hard coded in but inserted just to practice  
 try:  
 file = open("password\_file.txt", "r")  
 read = file.readlines()  
  
 except:  
 print("file read error")  
  
 for line in read:  
 arr = line.split(",")  
 user = arr[0]  
 salt\_value = arr[1]  
 hashedpassfromarr = arr[2].replace('\n', '')  
 newhash = concatinate(s, salt\_value) # we store the concatinated password in newhash  
 newhash = hash\_with\_sha256(newhash) # newhash is updated by calling the hash method on the concatinated password  
  
 # if statement just checks if the hashed password we created is the same as the one in the txt file.  
 if newhash == hashedpassfromarr:  
 print(user + " password is: " + s)  
 file.close()  
  
def concatinate(s, salt): # this method allows the generated password (EX: 000) to be added with the salt value  
 return s + salt  
  
# method where passwords are generated for all instances.  
def password\_generator(s, n):  
 # if checks if length of the generated password is at the correct and desired length (n). if so calls method check  
 if len(s) == n:  
 check(s)  
 return  
 # for allows the passwords to generate using numbers 0 - 9. i will increase with every iteration. recursive call  
 for i in range(10):  
 password\_generator(s + str(i), n)  
  
# for loop in main allows the length of the password to be set without hard coding it in, range method allows #'s 3-7  
# to be used  
def main():  
 for n in range(3, 8):  
 password\_generator("", n)  
 hex\_dig = hash\_with\_sha256('This is how you hash a string with sha256')  
 print(hex\_dig)  
  
main()

**Academic Honesty**

I, Federico Marin, certify that this project is entirely my own work, I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.