Casey Provitera, cbp21006, Giovanni Donato

***Q1:***

CODE:

import time

import subprocess

start = time.time()

passwords = open('MostCommonPWs',"r")

passwordsList = []

for line in passwords:

passwordsList.append(line.strip())

i = 0

while(True):

output = subprocess.run(["python3", "Login.pyc", "SkyRedFalcon914", passwordsList[i]], capture\_output=True)

if ("successful" in str(output)):

print (f"Password for SkyRedFalcon914 is: {passwordsList[i]}")

break

else:

if (i == len(passwordsList)-1):

print ("Password Not Found")

break

i+=1

end = time.time()

final = end-start

print (f"start time was: {start}")

print (f"end time was: {end}")

print (f"total time was: {final}")

END CODE

A screenshot of a computer code

Description automatically generated

The code for question one is simply trying to guess the password for only the user SkyRedFalcon914 by looping through a given file of passwords (MostCommonPwS) and trying to login with each.

This question went pretty smoothly. The biggest holds up were:

1. figuring out how to read text from a file and turn into an iterable list that can be indexed.
2. Learning how to use the subprocess module to run command line calls in a python file

***Q2:***

CODE

import time

import subprocess

start = time.time()

passwords = open('MostCommonPWs',"r")

passwordsList = []

for line in passwords:

passwordsList.append(line.strip())

members = open('gang', "r")

gangList = []

for member in members:

gangList.append(member.strip())

gangList.remove("SkyRedFalcon914")

for member in gangList:

for word in passwordsList:

output = subprocess.run(["python3", "Login.pyc", member, word], capture\_output=True)

if ("successful" in str(output)):

print (f"{member} has the password of {word}")

end = time.time()

final = end-start

print (f"start time was: {start}")

print (f"end time was: {end}")

print (f"total time was: {final}")

A screenshot of a computer code

Description automatically generated

Similar to 1 the code in 2 is testing the passwords from the MostCommonPWs file. But this time the code is attempting to login to multiple users from the file gang.

There were no hold ups on this question since it is basically Q1 with a nested for loop instead of a single for loop.

***Q3:***

CODE

import time

import subprocess

start = time.time()

passwords = open('PwnedPWs100k',"r")

passwordsList = []

for line in passwords:

passwordsList.append(line.strip())

members = open('gang', "r")

gangList = []

for member in members:

gangList.append(member.strip())

gangList.remove('SkyRedFalcon914')

gangList.remove('SkySilverWolf337')

flag = False

answers = []

for member in gangList:

for word in passwordsList:

output = subprocess.run(["python3", "Login.pyc", member, word], capture\_output=True)

print(f"{output}")

if ("successful" in str(output.stdout)):

print (f"{member} has the password of {word}")

print ("\n\n\n\n")

answers.append(str(output))

flag = True

break

if (flag):

break

end = time.time()

final = end-start

print (f"The start time was: {start}")

print (f"The end time is: {end}")

print (f"The total time is: {final}")

print ("\n\n\n\n")

for value in answers:

print (f"{value}")

***A screen shot of a computer

Description automatically generated***

The code for Q3 is almost identical to the code from Q2. The main difference is now the password list we are pulling passwords from is now PwnedPWs100k. As the name would imply this list has 100k passwords so it takes much longer to run than Q2. I removed the users whose passwords had already been found since this function takes so long.

The biggest hold up for Q3 is its own runtime. I needed to open a tmux session for the code to complete and then it took 26 hours, and that was only to find the first password since I made it stop as soon as it found one.

***Q4:***

CODE

import time

import subprocess

start = time.time()

passwords = open('PwnedPWfile',"r")

MemberPossiblePass = []

gangFile = open('gang', 'r')

gangMembers = []

for line in gangFile:

gangMembers.append(line.strip())

for line in passwords:

check = line.split(',')

if (check[0] in gangMembers):

MemberPossiblePass.append(line.strip().split(','))

for pair in MemberPossiblePass:

output = subprocess.run(["python3", "Login.pyc", pair[0], pair[1]], capture\_output=True)

print (f"{output}")

if ("successful" in str(output.stdout)):

print (f"{pair[0]} has the password of {pair[1]}")

end = time.time()

final = end-start

print (f"start time was: {start}")

print (f"end time was: {end}")

print (f"total time was: {final}")

A black screen with white text

Description automatically generated

In this question we are given a file PwnedPWfile that contains users and a leaked possible password of theirs, but the list we are given contains a lot of users that do not exist in the gang file. The code first checks if the user exists in gang, if they do they are added to a list with their leaked password to be tested to see if that is their correct password.

The biggest hold up for this question was realizing that a lot of the users in the given PwnedPWfile do not exist in gang but when that was realized, and I wrote the code to only test the users that do exist in gang then code ran much quicker.

***Q5:***

CODE

import time

import subprocess

import hashlib

start = time.time()

Hashed = open('HashedPWs',"r")

gang = open('gang', "r")

hundoK = open('PwnedPWs100k',"r")

HundoK = []

for line in hundoK:

HundoK.append(line.strip())

extraNums = []

num =00

for x in HundoK:

while (num<=99):

if (num<10):

extraNums.append(x+'0'+str(num))

else:

extraNums.append(x+str(num))

num+=1

num =00

gangMembers = []

for line in gang :

gangMembers.append(line.strip())

MemberPossibleHash = []

for line in Hashed:

check = line.split(',')

if (check[0] in gangMembers):

MemberPossibleHash.append(line.strip().split(','))

for word in extraNums:

hash=hashlib.sha256()

hash.update(bytes(word,'utf-8'))

guess=hash.hexdigest()

for pair in MemberPossibleHash:

if (pair[1] == guess):

output = subprocess.run(["python3", "Login.pyc", pair[0], word], capture\_output=True)

if ('successful' in str(output.stdout)):

print(f"{pair[0]} has the password of {word}")

end = time.time()

final = end-start

print (f"The start time was: {start}")

print (f"The end time is: {end}")

print (f"The total time is: {final}")

A screenshot of a computer screen

Description automatically generated

In this question we are given a file the contains users and the value of the password they might be using after it was hashed, similar to 4 there are many users that are not in gang so we only worry about the ones in gang like in Q4. We then need to add the digest 00-99 to each password in PwnedPws100k since the instructions said people can sometimes be lazy and add characters like this to passwords to get around when they are forced to make new passwords. Then similarly to Q4 with the file HashedPWs we are given we need to go through it and only pull out only the users that are in the file gang. Once both those steps are completed we loop through the new list of passwords that came from adding the digits to PwnedPws100k, for each of those passwords we hash it using a hash function from the hashlib module. If that hash matches the hashed value that came from HashPWs then we attempt to log in with the unhashed password and its corresponding user.

There were a few roadblocks in the question since it involved very new modules and logic. First understanding how to properly hash, to figure this out I went back to the file LoginTemplate.py to see how it hashed functions. Secondly adding the digits to each password in PwnedPws100k took a little bit of figuring out since if you start the digits you want to add to the passwords at ‘00’ when you add it to the passwords only 1 zero shows up. To get around this I have that code that when the number you want to add to the password is <10 we add the character ‘0’ as well as the number that is to be added.

***Q6:***

CODE:

import time

import subprocess

import hashlib

import csv

start = time.time()

salted = open('SaltedPWs',"r")

gang = open('gang', "r")

hundoK = open('PwnedPWs100k',"r")

HundoK = []

for line in hundoK:

HundoK.append(line.strip())

extraNums = []

for x in HundoK:

num=0

while (num<10):

extraNums.append(x+str(num))

num+=1

gangMembers = []

for line in gang :

gangMembers.append(line.strip())

MemberPossibleHash = []

for line in salted:

check = line.split(',')

if (check[0] in gangMembers):

MemberPossibleHash.append(line.strip().split(','))

#print(f"{MemberPossibleHash}")

# idea is that you add the salt to the unhashed passowrds then hash that combination

# if that combination hashes to what is currently stored then try the original password to log in

found = []

for word in extraNums:

for set in MemberPossibleHash:

hash=hashlib.sha256()

hash.update(bytes(set[1]+word,'utf-8'))

guess=hash.hexdigest()

if (set[2]==guess):

output = subprocess.run(["python3", "Login.pyc", set[0], word], capture\_output=True)

if ('successful' in str(output.stdout)):

print(f"{set[0]} has the password of {word}")

found.append([set[0], word])

# save a file with the newly discovered passowords

toClear = open('foundPasswords', 'w')

toClear.close()

with open('foundPasswords', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['This file contains username and password pair for gang members'])

for line in found:

writer.writerow([line])

end = time.time()

final = end-start

print (f"The start time was: {start}")

print (f"The end time is: {end}")

print (f"The total time is: {final}")

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generatedA black screen with white text

Description automatically generated

The logic in this question is similar to Q5 and Q4 since there are users that are not in gang in the file SaltedPWs that we are given so we only deal with the ones in gang, but since it includes ‘salts’ and we only need to add 1 digit to the passwords it is slightly different. Adding the single digit is easy since we can set the value to add to 0 and incriment it to 9 for each password with no hastle. What the ‘salt’ changes is that now instea of hashing the password we want we have to hash the salt concatonated with the password we want to try. Then if the hash of that is the same of the hash in the file we are given we then try the password and user pair in attempt to login. One more thing for this question is that we are suppose to create a file that has all the users and passwords we find that work. That is done at the very end of the function with a simple function from the csv module.

The hold ups on this question were similar to Q5 since hashing is always a little starnge but once I realized on the hash related lines need to be back to back to one another the code worked. I had a small hold up with creating the file to house all the passwords we find that work but that was simple after I looked up some examples.

The last part of this question is to describe why having ‘salts’ in real world would make it harder to break into a system. This is simply because having the salt means attackers need to have 1 additional piece of information to break into a system. Also a salt can be added to a password prior to its hash in any position relative to the password, so the attacker would also need to know where to add the salt to the password to find the proper hash.

Q7:

A screenshot of a computer

Description automatically generated

Q8:

Plain Text:

In my opinion this is one of the worst exposures because it let out so much sensitive data, that only the police departments normally have access to. The fact that all of the data was given out by a single employees password means that there were very few firewalls/authentication factors in place blocking access from section to section.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Hashed Unsalted:

This is a very bad instance of hashed passwords being leaked because it says in the article that the leak was occurring for over 5 years. Since slack is a communications application, it is possible that sensitive data was sent or received in slack while the data leak was occurring. This gives anyone who breached the system the ability to find and use whatever they want over the 5-year span. The only positive is that since the passwords were hashed even though they were leaked attackers still need to put in effort to reverse the hash function.

A screenshot of a computer

Description automatically generated

Q9:

Something you know – refers to a password

Something you have – refers to a phone or device

Something you are – refers to a finger print of genetic signature

Significant website that does not support 2FA: Best Buy does not support 2FA this is significant because people put their credit card information into this website.

Website which does support 2FA: Many social media platforms support 2FA including Facebook, and Twitter (now known as X).

I do use 2FA on accounts besides UCONN, I had to use it at my internship during the fall semester. I do not believe my parents use it but my mom might for work.