Lab 1 Instructions

Q1:

As a cybersec researcher, you are asked to help find the password to the account of The RedFalcon, a notorious supervillain, in the darknet server of The Red Falcon's supervillain gang. The Red Falcon's username is SkyRedFalcon914. Since The Red Falcon didn't study cybersecurity and is known for villainy rather than intelligence, you decide he is likely to use one of these very common passwords. So, try common passwords against the login program of the gang's server, Login.pyc, to find out The Red Falcon's password. You can do this manually, easily, and it's not a bad idea to do so as a warm-up exercise. However, you must then write a simple Python script, Break1.py, that will find The Red Falcon's password by trying the different passwords against Login.pyc. Your program, Break1.py, should also print the time, at both the beginning and the end of the run; the runtime should be very short. To execute another program within your Python script, you can import the subprocess module and use the run function that is defined within it. Here's the documentation for that function.

Q2:

After breaking into The Red Falcon's account, you found there a file, gang, that contains all usernames in the server of the supervillain gang. Surely, a few of these guys also use one of the common passwords in the file MostCommonPWs! Extend your code from Break1.py and create a new script, Break2.py, which will find different gang member that uses one of the passwords in MostCommonPWs, by using the login program Login.pyc.

Q3:

Even when users do not use one of the obviously weak passwords such as these in MostCommonPWs, they still often use commonly used passwords. We now want to find passwords of an additional supervillain gang member, whose password is from the `dictionary' of 100,000 common passwords, which you will find in the file PwnedPWs100K (downloaded from https://haveibeenPwned.com – a useful resource, visit it!).

Extend your code from Break2.py and create a new script, Break3.py, that will try out all of the passwords in PwnedPWs100K, for gang members whose password was not found yet. Run this modified program (Break3.py) and see how long it takes to find the password for one additional gang member using Login.pyc. Try to make your program efficient! Note: while in reality the passwords in the file are sorted by popularity, we pick

them uniformly, i.e., each password is
equally likely.
**IMPORTANT: Even when programming with efficiency in mind, this script is checking a very large number of passwords and will most likely take MANY HOURS (up to 24hrs) to run and find the correct password. As a result, you will need to:

- Run your script inside a tmux session so that your VM does not stop its execution when
  your SSH session ends. Refer to our tmux tutorial on HuskyCT in the Lab1 folder.
- Test your script thoroughly before letting it run for many hours! You can even use simple print debugging statements to help with this. It's very easy for an insidious bug (e.g., forgetting to strip off newlines) to prevent the correct password from being found. I recommend making your own test version of Login.py where you hard code a chosen
  password from PwnedPWs100K (choose one very early on in the file) and test your script against that test login program first. Also, make sure your script breaks as soon as it finds the correct password to prevent any unnecessary runtime!

Q4:

In this question, you will use this technique to find the passwords of additional gang members, where the techniques of previous questions failed. In fact, these will be quite random passwords. You receive an exposed passwords file, PwnedPWfile, and will write a program, Break4.py, that will search for gang member which have an account (and password) in the file. For any gang members whose passwords are listed in PwnedPWfile, Break4 will check if the password `works' against the login program Login.py.

Q5:

You are given another exposed passwords file, HashedPWs, which contains, for each user x, the results of applying a cryptographic hash function h(.) to the password PWx of user x, i.e., h(PWx). (In the next question we will see an improved defense.)

Write a new program, Break5.py, that uses the file HashedPWs to find, as quickly as possible, the passwords of these additional gang members. It will be infeasible to test all random passwords (why?); instead, focus on gang members who pick a random password from PwnedPWs100K, and concatenate to it two random digits. (Many users do such minor tweaks to their passwords, to bypass password-choice requirements, or in the

incorrect hope that this suffices to prevent password guessing.). For gang members whose passwords you recover using HashedPWs, use Login.pyc to check if the gang member used the same password. To compute hashes: The hashes for this questions were computed with a SHA256 hashing algorithm.
Hint: think carefully how to do this question efficiently, or it may be quite slow.

Q6:

To further improve security, password files usually do not contain the hash of the password PWx of user x. Instead, the password file contains two values for each user x: a random value saltx , called salt, and the result of hashing of a combination of the password PWx and of the salt. In this question, you are given a file SaltedPWs which contains, for each user x, the pair (saltx , h(saltx + PWx)), where saltx is a random value chosen for user x. Here we use the + sign to denote concatenation.

Write a new program, Break6.py, that uses the file SaltedPWs to find the passwords of as many additional gang members as possible that use passwords from PwnedPWs100K concatenated with one random digit, as quickly as possible. Break6 should also save a file containing the names and corresponding passwords. Confirm the exposures using Login.pyc.

Q7:

The https://haveibeenPwned.com service collects passwords from the many password-file exposures, and allows users to detect when their password was exposed; it also allows service providers to detect when a user is trying to use an already-exposed password. In this question, we ask you to use this service for at least your UConn email(s); do this from home or campus – you will need to be connected to the Web.

Q8:

One may expect every `serious’ web service to use salted passwords file, no? Unfortunately, this is not the case; in fact, some services (maybe not that serious) did not even use hashing – their password files just contained passwords in the clear! Search for such incidents. You may be surprised!

Q9:

Password-based authentication relies on the user’s knowledge of the password; in general, authentication methods relying on information known to the user are referred to as something you know authentication factor. The two other main authentication factors are something you have and something you are. Search these terms online to understand them better and to identify popular websites that support other authentication factors, instead of relying on passwords or in addition to passwords (two-factor authentication, aka 2FA).