Security Tools Lab 1

Assignment 2 – Hashing & passwords

1. Objectives

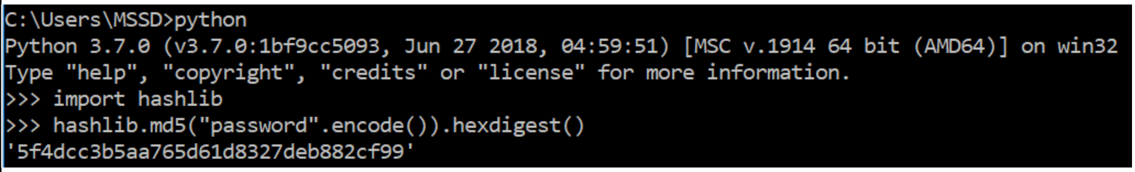
- Hash password using MD5
- Crack MD5 hashes using brute-force and rainbow tables
- Strengthen MD5 hash using salt and crack again the salted hashes by rainbow tables and rule-based extension of dictionary attack using hashcat
- Compete in the hash breaking competition

2. Setup

- This lab can be done in Windows or Linux. We'll be using mostly Windows to do the demo.
- Note that in this lab you should use python3

3. Hashing password using MD5

To warm up, compute a couple of MD5 hashes of strings of your choice using python's command line. use import hashlib module and its md5() function. For example:



```
C:\Users\MSSD>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hashlib.md5("password".encode()).hexdigest()
'5f4dcc3b5aa765d61d8327deb882cf99'
```

You can also use the provided demo file called hashing_demo.py

Observe the length of the output, and whether it depends on length of input.

4. Brute-Force VS Dictionary Attack

For this exercise, use the 15 hash values from the <STUDENT_ID>-hash5.txt

Create a Python 3 script called md5_lab1.py, which will find the corresponding input plaintext that was used for making the hash values in file <STUDENT_ID>-hash5.txt.

Consider only passwords with 5 characters (lowercase and/or numeric characters).

To help reduce the search space we provide a dictionary with newline separated common words in words5.txt. Use the dictionary as the first resort.
It might not be enough to crack all hashed plaintexts, then compute a hash value for each possible combination of lowercase letters, and then with each possible combination from union of all lowercase letters and digits.

Take note of the computation time of your algorithm to reverse all 15 hashes. You will need it in later step. Consider bash utility time (only for Linux users) or the timeit python module: https://docs.python.org/3.6/library/timeit.html
Final script should support three mandatory arguments:
-i <INPUT_FILE>, -w <DICTIONARY-_FILE>, and -o <OUTPUT_FILE>
Respect the format: one hash/plaintext/dictionary entry per line

5. Creating Rainbow Tables

Install the program rainbowcrack : http://project-rainbowcrack.com/

Use rtgen (http://project-rainbowcrack.com/generate.htm) to generate rainbow tables with the characteristics shown below.

– Five characters input
– Only lower case letters and numeric characters.
– Chain length is 3800.
– Chain number is 600000.
– Part index is 0.
– Table index is 0.

The chain length and chain number values maybe suboptimal? If yes, find better ones.

Use rtsort to sort the rainbow table to make searchable by rcrack.

Use rcrack (http://project-rainbowcrack.com/crack.htm) to crack the list of fifteen passwords from hash5.txt

You can see that there are 36^5 ~= 60M combinations for brute force attack.
and the number of plaintexts covered by rainbow table is 3800*600000 = 2,280,000,000.
So, the ratio is ~ 40:1. Try to decrease size of the rainbow table e.g., to ratio 20, 10 and 5, and observe whether all hashes are cracked (use chain number parameter).

6. Salting

Extend your Python script to append one random lowercase character as salt value to all the elements of the list of passwords you recovered in the previous part of this exercise.

Rehash the password using MD5, and store the newly hashed passwords into a new file called salted6.txt (remember to store the new password as well, maybe in a pass6.txt file). The functional definition of our salt strategy is the following:

saltedhash(password) = hash(password||salt), where operator || represents concatenation.

Generate a new rainbow table using **rtgen** (with new parameters) to break the hash values. As before, sort the table using **rtsort**.

Compare the timing of the new table generation and lookup vs the previous values
Try to break as many salted hashes as possible.

In your writeup explain the differences between salted and non-salted rcrack strategies and compare the timings.

7. Hashcat

For windows users,
Download hashcat tool https://hashcat.net/files/hashcat-5.1.0.7z

Extract to a folder and navigate to that folder to find your hashcat executable. For Windows 64 bit, it will be hashcat64.exe

You can test by running the following command in the directory.

- hashcat -m 0 -a 0 -o cracked.txt salted6.txt /usr/share/wordlists/rockyou.txt (Linux)
- hashcat64 -m 500 -D 1 -o cracked.txt example500.hash example.dict (Windows)

Open the file cracked.txt to reveal the cracked hash.

$1$uOM6WNc4$r3ZGeSB11q6UUSILqek3J1:hash234

Here, -m is the mode of hashcat (500 = md5crypt),), -D is the device type (1 = CPU), -a is the attack mode (0 = straight mode), -o is the output file, and then following files contain hash values and dictionary, respectively.
Note that you may try to use the -force if hashcat will complain about OpenCL drivers/unsupported OS. See https://hashcat.net/wiki/doku.php?id=hashcat for quick documentation and parameter description.

First, try to write simple hashcat **rule-based attack** using dictionary, which will exploit the knowledge of how salting was performed. You should consider -r parameter for rule file and -m equal to 0, as you are cracking raw hash of MD5.

See description for creating rules at URL https://hashcat.net/wiki/doku.php?id=rule_based_attack, and please consider using maskprocessor that can generate rule set according to input mask – https://github.com/hashcat/maskprocessor/releases. For example, to generate rules file covering salted passwords from a dictionary by salting strategy hash(password||any_digit), use:

- mp64 '$?d' -o digits.rule

where -o specifies output file for generated rules, operator $ represents concatenation, and ?d is the wildcard representing all digits.

Nice tutorial how to crack password by hashcat using rules is at URL https://labs.mwrinfosecurity.com/blog/a-practical-guide-to-cracking-password-hashes/

How many passwords did you crack? Why you did not crack some passwords?

Second, try to execute **mask attack** that considers knowledge of character set in particular positions (including knowledge of how salting was performed) and also leverages parallelism of you CPU/GPU thanks to openCL library.
See URL https://hashcat.net/wiki/doku.php?id=mask_attack for quick introduction.
For example, to crack passwords having 3 upper case characters followed by 2 digits, hashcat can be run as:

- hashcat -D 1 -m 0 -a 3 some_hashes.txt ?u?u?u?d?d (Linux)
- hashcat64 -D 1 -m 0 -a 3 some_hashes.txt ?u?u?u?d?d (Windows)

Compare the timings with brute force that you implemented in your custom python script. If your computer has Intel graphic card, then you can also try parameter -D 2 and take note about the differences in timings.

Compare timings with Rainbowcrack and include it into the previous writeup. Also mention rules and commands that you used for cracking the hashed and salted passwords.

8.	**Hash breaking competition** We provide a list of hashes in hashes.txt They are of various difficulty – not all are equally hard. There are no easy rules about length or characters allowed anymore! Implement an optimized script and try to reverse as many of those hashes as possible. You can also use other tools as you want (hashcat, rainbowcrack) Write a short explanation on the approach you use to crack those passwords. Submit the answers as a CSV file called competition.csv containing two columns. The first column is the md5 hash of the password you break, and the second column is the plain text password.
9.	**Hand-in** Submit your md5_lab1.py script that breaks the supplied hash values in hash5.txt, generate the salted hashes and the relevant files. Put your username and mention the timings in your header. Prepare the writeups for all sections where explanations are requested. Include your conclusions and learning points. Include the found plaintexts/hashes (competition.csv) in your writeup.