Homework 3

Exercise 1: [attack] Password Cracking

The objective of this exercise is for you to crack some passwords. The exercise consists of three parts. In each part, the setting is slightly different modifying the way the cracking should be executed.

For each part, please find the set of password to crack as a list of hash digests.

Part 1a: Brute force attack

In this part you should implement a brute-force attack. Passwords are randomly generated from the set of lowercase letters and digits ('abcd...xyz0123...9') and have length 4, 5, or 6 characters. Generated passwords are then hashed with SHA-256 and corresponding hexdigests are sent to you in the file.

The list of SHA2-56 digests you need to crack is:

- `7c58133ee543d78a9fce240ba7a273f37511bfe6835c04e3edf66f308e9bc6e5`
- `37a2b469df9fc4d31f35f26ddc1168fe03f2361e329d92f4f2ef04af09741fb9`
- `19dbaf86488ec08ba7a824b33571ce427e318d14fc84d3d764bd21ecb29c34ca`
- `06240d77c297bb8bd727d5538a9121039911467c8bb871a935c84a5cfe8291e4`
- `f5cd3218d18978d6e5ef95dd8c2088b7cde533c217cfef4850dd4b6fa0deef72`
- `dd9ad1f17965325e4e5de2656152e8a5fce92b1c175947b485833cde0c824d64`
- `845e7c74bc1b5532fe05a1e682b9781e273498af73f401a099d324fa99121c99`
- `a6fb7de5b5e11b29bc232c5b5cd3044ca4b70f2cf421dc02b5798a7f68fc0523`
- `1035f3e1491315d6eaf53f7e9fecf3b81e00139df2720ae361868c609815039c`
- `10dccbaff60f7c6c0217692ad978b52bf036caf81bfcd90bfc9c0552181da85a`

What can you say about the computational time required? Is this kind of attack parallelisable? Can you explain in one sentence the rationale behind the use of SHA-256?

Part 1b: Dictionary attack with rules

In the previous part, you implemented the brute force attack and faced one of its drawbacks. Unfortunately, people very rarely use random passwords. Instead they use some common words and sometimes modify them slightly. This is a fortunate fact for password crackers, because they can use 'dictionary attacks' to crack the passwords more efficiently than with brute-force. In this part you should implement one such dictionary attack. We generate a password by selecting a word from a large dictionary and then randomly applying some of the common user modifications:

- capitalise the first letter and every letter which comes after a digit (for example: 'com402class' becomes 'Com402Class'). If you are using Python, this is easily achieved by 'title()' function from string module ('com402class'.title() will give you 'Com402Class')
- change 'e' to '3'
- change 'o' to '0' (that's small letter 'o' to zero)
- change 'i' to '1' (For instance, 'window' can become 'W1ndow', or 'w1nd0w', ...)

Examples of dictionaries can be found online (e.g. https://wiki.skullsecurity.org/Passwords).

The list of SHA-256 digests you need to crack is:

- `2e41f7133fd134335f566736c03cc02621a03a4d21954c3bec6a1f2807e87b8a`
- `7987d2f5f930524a31e0716314c2710c89ae849b4e51a563be67c82344bcc8da`

- 076f8c265a856303ac6ae57539140e88a3cbce2a2197b872ba6894132ccf92fb
- b1ea522fd21e8fe242136488428b8604b83acea430d6fcd36159973f48b1102e
- fa5700d75974a94dd73f7c5d48e85afa87beba19b873d4eb8d411dd251560321
- 326e90c0d2e7073d578976d120a4071f83ce6b7bc89c16ecb215d99b3d51a29b
- 269398301262810bdf542150a2c1b81ffe0e1282856058a0e26bda91512cfdc4
- 4fbee71939b9a46db36a3b0feb3d04668692fa020d30909c12b6e00c2d902c31
- 55c5a78379afce32da9d633ffe6a7a58fa06f9bbe66ba82af61838be400d624e
- 5106610b8ac6bc9da787a89bf577e888bce9c07e09e6caaf780d2288c3ec1f0c

What do you observe compared to part 1a?

Part 1c: Dictionary attack with salt

In the previous part of the exercise you implemented a dictionary attack. You should notice that once you have a dictionary you can compute the hashes of all those words in it, and create a lookup table. This way, each next password you want to crack is nothing more than a query in the lookup table. To tackle this problem, passwords are usually 'salted' before hashing. Salt is exactly two characters long and it contains only hexadecimal characters. In this part of the exercise you should implement another attack using a dictionary. We generate a password by simply selecting a random word from a dictionary and appending a random salt to it. The password is then hashed with SHA-256 and hexdigest and salt are available to you. Your task is to crack the passwords using a dictionary.

The list of SHA-256 digests you need to crack and the salt (in brakets) used are:

- 962642e330bd50792f647c1bf71895c5990be4ebf6b3ca60332befd732aed56c (b9)
- 8eef79d547f7a6d6a79329be3c7035f8e377f9e629cd9756936ec233969a45a3 (be)
- e71067887d50ce854545afdd75d10fa80b841b98bb13272cf4be7ef0619c7dab (bc)
- 889a22781ef9b72b7689d9982bb3e22d31b6d7cc04db7571178a4496dc5ee128 (72)
- 6a16f9c6d9542a55c1560c65f25540672db6b6e121a6ba91ee5745dabdc4f208 (9f)
- 2317603823a03507c8d7b2970229ee267d22192b8bb8760bb5fcef2cf4c09edf (17)
- c6c51f8a7319a7d0985babe1b6e4f5c329403d082e05e83d7b9d0bf55876ecdc (94)
- c01304fc36655dd37b5aa8ca96d34382ed9248b87650fffcd6ec70c9342bf451 (7f)
- cff39d9be689f0fc7725a43c3bdc7f5be012c840b9db9b547e6e3c454a076fc8 (2e)
- 662ab7be194cee762494c6d725f29ef6321519035bfb15817e84342829728891 (24)

Why is it a good idea to salt the passwords? Estimate the complexity required in this part if the salts were not provided. What additional security countermeasures could you think off?

Note1: The SHA-256 digest of a password 'psswd' is the result of SHA-256(psswd). The digest of a salted password with salt XX is SHA-256(psswdXX).

Note2: Not all dictionaries are the same, be aware that if you implement the attack correctly but you can't crack the passwords, then you might be using a dictionary which doesn't contain all the words as the dictionary we used.

Note3: in order to check the passwords, just compute the hash of the word (with or without salt):

- macOS cli: `echo -n "psswd" | shasum -a 256`
- unix cli: `echo -n "psswd" | shasum -a 256`
- Windows: look into *Microsoft File Checksum Integrity Verifier* or use an online hasher

Note4: The attacks you implement might take some time when you run them. Depending on your hardware and your implementation, the attacks may run more than 30 minutes.

Exercise 2: [defense] bcrypt

In this exercise you will implement a simple server using Flask.

Instructions

Create a python file `server.py` using flask to satisfy the following requirements:

- The server needs to hash the password using **bcrypt** and returns the status code 200 and the digest. Make sure you enforce an UTF-8 encoding of the password **before** computing the hash.
- The server should expect a POST request on `127.0.0.1:5000` with JSON data containing fields `user` and `pass`.

Testing the solution

Evaluate your solution using the following command in the same folder as `server.py`

`docker run --read-only -v $(pwd):/app/student com402/hw3ex2`

If your implement is correct you should be prompted a `success`