FCS Lab 3 - MD5, Rainbow Tables

- Name(s): Lim Boon Han Melvin
- Student ID(s): 1005288

Part I: Hashing Using MD5

- How does the length of the hash correspond to the input string?

The hash length does not change as the last block is padded to ensure the output is 128 bits long.

- Are there any visible correlations between the hash and the input string?

No, there are no visible changes or patterns when the input string changes by 1 value (e.g a --> b) even though there are chances for collision.

- What are the issues related to the cryptographic weakness of MD5?

It is not collision resistant as multiple strings can have the same hash value, and thus is vulnerable to length extension attacks where it can find collisions using intermediate states.

Also, it is not preimage resistant as the hash value is the same for every instance of the same string, thus a brute force attack using a rainbow table can be done.

Part II: Break Hashes with Brute Force

- How much time did you take in total?

Time taken : 36.610324s

- How much time does it take to crack each string, on average?

36.610324 / 15 = 2.441s

- Is it possible to amortize (gradually write off the initial cost of) the brute forcing attempts?

Yes, but the limiting factor will be memory storage of the hashed passwords that are used in brute forcing. The initial cost will be reduced as the number of hashed password in storage increases. However this reduction will see a slower growth as the memory storage gets bottlenecked.

Part III: Creating Rainbow Tables

- Is rcrack faster/slower than your script ex2.py? By how much faster/slower is it?

Faster, it finished cracking in 5.33s, whereas ex2.py finished in 36.61s. It is faster by about 31.28s.

- Do you observe any advantages or disadvantages of using rainbowcrack?

Disadvantage : The creation of rainbow table took about 3mins, for a small input size. Thus it might take longer for commonly used criterias in other password databases.

Advantage : The GUI displays a table that shows clearly the cracked hash text and its plaintext, which makes breaking in more comfortable. Also, since it is mostly the computer doing work, we do not have to spend time coding out the brute force attack like in ex2.py

Part IV: Salt

- What is the observed differences between your ease of cracking the salted vs the unsalted plaintexts?

The process increases in difficulty significantly after the addition of a single salt, where the size of rainbow table increases by a few times and thus time taken to generate these tables increases.

- Report the difference in time observed to crack.

The time needed to crack the password is roughly the same at 5.7s, as it is mostly comparison but with a bigger table. Most of the work had already been done in the generation phase.

- Explain any differences between salted and non salted rcrack strategies.

There are not much differences as it is a brute force attack, thus there are a handful of variables like table_part, chain_length to play with that will give many variations of tables.

Part V: Hash Breaking Competition

- What is the approach you used to crack the hashes

I used a dictionary approach and an online cracking website : https://hashes.com/en/decrypt/hash.

- How you decided or designed your approach

Initially, I tried to modify the ex2.py code for brute force, but it was tough as the password length is uncertain, which makes brute forcing a bad idea and thus making rainbow tables useless as well. Thus I decided to use a dictionary attack with commonly used passwords that are hashed. After awhile, the dictionary base is limited and I went online to source for cracking websites.

- Main challenges and limitations of your approach

The dictionary is exhaustive and extensive, which made me source for a larger dictionary often.

- How many hashes did you manage to crack?

I stopped after cracking the 100th hash as it was too exhaustive.