Sng Wei Qi Amos

1005952

50.042 Foundations of Cybersecurity Lab Report 3

**Part 1: Hashing Using MD5**

1. How does the length of the hash correspond to the input string

The length of the hash remains constant at 32 characters.

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

No.

3. What are the issues related to the cryptographic weakness of MD5?

i. **Collision Vulnerability**: It is possible to find two different inputs that produce the same MD5 hash (a collision). This undermines the integrity verification and authentication capabilities of MD5.

ii. **Pre-image Attacks**: Although not as straightforward as collision attacks, finding an input that matches a given hash (pre-image attack) is feasible with sufficient computational resources, weakening MD5's security.

iii. **Second Pre-image Attacks**: Finding another input that produces the same hash as a given input is also relatively easier with MD5 compared to stronger hash functions.

iv. **Speed and Efficiency:** MD5’s computational simplicity, while advantageous for speed, makes it more susceptible to brute-force attacks.

v. **Lack of Future-proofing:** MD5 is not secure against evolving computational capabilities and advances in cryptanalysis, making it unsuitable for modern applications that require long-term security.

**Part 2: Break Hashes with Brute Force**

Total time taken = 59.40768504142761 seconds

Average time to crack each string = 11.8 seconds

Yes it is possible to amortize the process using the following strategies:

1. Parallel Processing: Distribute the work of generating and hashing strings across multiple cores or even machines.

2. Incremental Computation: Store partially cracked results, allowing you to continue from the last known state.

3. Caching and Reuse: Use a cache to store hashes and their corresponding strings, avoiding repeated computations.

4. Dynamic Adjustment of Workload: Change the charset or string length dynamically based on the progress and the rate of successful matches.

**Part 3: Salt**

1. What are the observed differences between your ease of cracking the salted vs the unsalted plaintexts?

It takes a significantly larger amount of computation to crack the salted plaintexts compared to the unsalted plaintexts.

2. Report the difference in time taken to crack the salted and the unsalted hash values.

Total time taken to crack all 15 salted text: 2158.5144486427307 seconds

Difference in time: 2099.10676360130309 seconds

3. The cracking strategies are largely the same, other than the fact that for the salted text, the length that we set the itertools function to is 6 instead of 5. This significantly increases the number of permutations of the given charset, by 36 times as there are now 36\*\*6 permutations of the charset instead of 36\*\*5 permutations.

**Part 4: Hash Breaking Competition**

What is the approach you used to crack the hashes?

I used John the Ripper, a widely used hash cracking software available in kali linux to crack the hashes.

I also used multithreading to speed up the process.

The following is the command I used:

john --format:md5-raw hashes.txt --fork=8

How you decided or designed your approach?

I did some research into what tools security experts use when trying to crack hashes and found that many of them recommended John the Ripper.

Recognising that this software is widely used and established to be one of the best, I decided to make use of this tool too.

Main challenges and limitations of your approach?

It was challenging due to the limited memory I had in my VirtualBox kali Linux, and the limited CPU threads. If there were more CPU threads available I would be able to crack more passwords.

How many hashes out of the total did you manage to crack (there is no prize)

I cracked 51/ 150