Password Cracking Report

- Tool Selection:
    - For cracking passwords for this project I chose Hashcat as my tool of choice. I was cracking passwords on my brother's windows computer because of the powerful RTX 4080 gpu he has in it which allowed me to reach faster cracking speeds than I would've been able to reach normally on my laptop. I did briefly look at John the Ripper as a possible tool but Hashcat not only was easier for me to install and set up, but came with more rulesets to use that I believed would be useful.
- Average Password Checking:
    - When it came to the password cracking speed I was able to achieve, the speed varied greatly depending on what type of attack I was performing. Like I stated above I was able to perform these attacks with an Nvidia GeForce RTX 4080 with 16 GB of VRAM. This allowed me to far surpass the baseline of 10,000 checks per second, and even the gpu baseline of 25M checks per second. The. Primary split in times were between dictionary/rule based attacks, and brute force attacks.
    - Dictionary/Rule based attacks:
        - These attacks were performed much slower than brute force attacks and only reached speeds of 2.4-2.5 million hashes/second. This speed drop is primarily due to the computational overhead of applying transformation rules and MD5 crypt's 1,000 iterations per hash.
    - Brute Force Attacks:
        - Brute force attacks were able to be performed much quicker and fully utilize the power of the gpu. When it came to brute force attacks I was cracking passwords at a speed of ~42 million hashes a second. The speeds were technically lower when brute focing 2 and 3 character alphanumeric, but this was due to these operations being done instantly.
- Guessing Strategies:
    - Initial Dictionary Attack:
        - The first attack was a simple one that performed an attack based on the provided wordlist.txt. This attack pretty much instantly cracked 5 passwords all of which being words from the wordlist.
    - Low-End Lowercase and Alphanumeric Brute Force:
        - After the initial dictionary attack I went on to try and brute force the lowercase and alphanumeric possibilities for the lower character numbers, 2-5 chars. When it came to the lowercase brute forcing I attacked using a mask ?l?l… for lowercase characters only. This cracked only a single password for each of the 2-5 character counts (4 passwords in total). All 4 of these were pretty much blink and you miss it and took less than 2 minutes combined. For alphanumeric I used a similar mask technique ?1?1… where ?1=?l?u?d, meaning it scanned lowercase, uppercase, and digits. I did this for character counts 2-5 and got 4 more passwords, again one for each character count. The lower 3 took no time at all but 5 characters did take a few minutes.
    - High-End Lowercase and Alphanumeric Brute Force:

        - This was very similar to the lowercase in the sense I used a mask for both categories, but this time I knew I could not run through every possibility. To start I did lowercase 6 character which took only a little longer than 5 character alphanumeric. This scored me another password before moving on to 6 character alphanumeric. I let this run for the full run time of 1.5 hours which awarded me with another password. After this I did the same for character counts 7 and 8. 7 lowercase I ran all the way through which did not give me a password for the first time. After that I ran 8 lowercase and 7 and 8 alphanumeric for about 3 hours each and stopped after none of those gave me any results.
    - L33T Speak and Dive Rules:
        - Dive Rule from Hashcat:
            - To start I ran the dive rule combined with the dictionary which cracked my first L33t password.
        - L33T Speak Rule from Hashcat:
            - After that, I tried to use the rule given to me from Hashcat regarding l33t speak, using the wordlist.txt as the source, but this did not give me any passwords. After that I combined that rule with the dive rule which resulted in me getting 3 more passwords all l33t speak.
        - New Rule:
            - After this I searched online for more rules for Hashcat and in their GitHub found the rule unix-ninja-leetspeak.rule which was a more comprehensive l33t speak rule that was able to crack the last password.
- Passwords:
    -

User	Password	Category	Strat
user 6	forwarding	Dictionary	Dictionary Attack
user7	increased	Dictionary	Dictionary Attack
user14	longitude	Dictionary	Dictionary Attack
user16	revolutionary	Dictionary	Dictionary Attack
user19	cincinnati	Dictionary	Dictionary Attack
user15	ta	Lowercase (2)	Lowercase Brute Force ?l?l
user20	lsv	Lowercase (3)	Lowercase Brute Force ?l?l?l
user12	uddg	Lowercase (4)	Lowercase Brute Force ?l?l?l?l
user5	lkyru	Lowercase (5)	Lowercase Brute Force ?l?l?l?l?l
user18	ugrknq	Lowercase (6)	Lowercase Brute Force ?l?l?l?l?l?l
user10	SJ	Alphanumeric (2)	Alphanumeric Brute Force ?1?1
user17	lr4	Alphanumeric (3)	Alphanumeric Brute Force ?1?1?1
user9	31w4	Alphanumeric (4)	Alphanumeric Brute Force ?1?1?1?1
user11	2c8zc	Alphanumeric (5)	Alphanumeric Brute Force ?1?1?1?1?1
user1	7OgTHT	Alphanumeric (6)	Alphanumeric Brute Force ?1?1?1?1?1?1
user13	c0mp4ni35	L33T	Dictionary + dive.rule
user2	41t3rn4t3	L33T	Dictionary + dive.rule + l33t rule
user8	31iz4b3th	L33T	Dictionary + dive.rule + l33t rule
user4	int3rn4ti0n411y	L33T	Dictionary + dive.rule + l33t rule
user3	14b0r4t0ri35	L33T	Dictionary + unix-ninja-leetspeak.rule

- Additional Questions:
- Question 1:
    - Assumption:
        - For this problem as well as question 2 I am assuming I am only attempting to crack one password of each length. The times I calculate here are much lower about 20x lower than the actual time it took to run alphanumeric brute force on a 6 character password. But that is because for this project it had to run each possibility for each of the 20 hashed passwords with different salts. For these questions I am assuming one password to not muddy the math. Although if someone wanted they could just multiply the possibilities by the number of passwords they are cracking.
    - Password Cracking Alphanumeric on my setup: 42 MH/s
        - Alphanumeric passwords have a character set size of 62 possible characters.
        - To get the possible passwords I simply take that 62 and raise it to the power of the number of characters in the password.
        - My cracking speed was 42 million hashes a second.
        - To get the max cracking time I simply divide the solution from the raising 62 to the power of the amount of characters by my cracking speed.
            - 6 Character: 62^6 = 56,800,235,584
                - Divide: 56,800,235,584 / 42M = 1,352 seconds
                - Time: 22.5 minutes
            - 8 Character: 62^8 = 218,340,105,584,896
                - Divide: 218,340,105,584,896 / 42M = 5,198,573 seconds
                - Time: 86,642.9 minutes, 1,444.05 hours, 60.17 days
            - 10 Character: 62^10 = 839,299,365,868,340,224
                - Divide: 839,299,365,868,340,224 / 42M = 19,983,318,234 seconds
                - Time: 333,055,304 minutes, 5,550,922 hours, 231,288.41 days, 633.7 years
            - 12 Character: 62^12 = 3,226,266,762,397,899,821,056
                - Divide: 3,226,266,762,397,899,821,056 / 42M = 76,815,875,295,188 seconds
                - Time: 1,280,264,588,253 minutes, 21,337,743,137 hours, 889,072,630 days, 2,435,815.43 years
- Question 2:
    - Same parameters only difference is cracking speed is now 2.5 billion passwords a second.
    - The guess will be for max time still just like in question 1.
        - 6 Characters: 62^6 = 56,800,235,584
            - Divide by 2.5 B = 22.72 seconds
            - Time: 22.72 seconds
        - 8 Characters: 62^8 = 218,340,105,584,896
            - Divide by 2.5 B = 87,336.04 seconds
            - Time: 1,455.6 minutes, 24.26 hours, 1.01 days
        - 10 Character: 62^10 = 839,299,365,868,340,224
            - Divide by 2.5 B = 335,719,746.35 seconds
            - Time: 5,595,329.11 minutes, 93,255.49 hours, 3,885.65 days, 10.64 years
        - 12 Character: 62^12 = 3,226,266,762,397,899,821,056

        - Divide by 2.5 B = 1,290,506,704,959.16 seconds
        - Time: 21,508,445,082.65 minutes, 358,474,084,710.88 hours, 14,936,420.2 days, 40,921.15 years
- Question 3:
    - I do not believe the typical password meter is a good indication of security because of the main category many of them do not measure, being password patterns. A typical password meter only requires a password be 8 characters, include at least one uppercase letter, lowercase letter, number, and symbol to be considered strong. The problem with that is two-fold, firstly 8 characters is far too little characters to be considered secure in the modern age of password cracking. A hacker with even just one high end gpu could crack an 8 character password with just a day or two at most, with more gpus that time only goes down. A password should never be under 12 characters to prevent brute force hacking and even passwords that long must still not follow a pattern that a hacker could use password cracking rules to exploit. If the password you use has regular English words in it, or even names followed by numbers it is not as secure as it needs to be to protect your privacy.
- Question 4:
    - Yes the use of SHA-512 over MD5 for hashing passwords does improve password security in a number of ways. The first reason is that SHA-512 uses 5,000 rounds oh hashing by default which is a large improvement over MD5's 1,000 rounds. This makes brute force attacks take roughly 5x as long which adds up quickly the second a password reaches over 8 characters in size. The second reason is that SHA-512 uses 512 bit hashes instead of 128 bits. This reason alone provides much stronger password security through increasing the difficulty of collision attacks.
- Question 5:
    - Yes, using a salt increases password security greatly through prevention of using precomputed hash tables during password cracking. Using a salt prevents a hacker from cracking multiple passwords every time they crack a single one since even identical passwords will have different hashes.
- Question 6:
    - No it does not because as the paper I did on passwords shows many sites are prone to getting their password data leaked, which would allow an attacker to perform an offline attack against the hashed passwords to figure out what they are without using the online website.