Alex Warden
COSC583
Password Cracking Project
11/7/2025

# Password Cracking Project

For my project, I decided to use hashcat. This is mostly because when talking with the TA, they mentioned they used hashcat on Windows. Since I'm also using Windows, I thought this would be easiest.

My average rate for password checks a second was pretty high. I ran '.\hashcat.exe --benchmark -m 500', and got a rate of 10931.8 kH/s, or roughly 11 million password checks per second. I think my rate is so high due to my CPU, an AMD Ryzen 7 3800X 8-Core Processor, which has a base speed of 3.90 GHz, and my GPU, a NVIDIA GeForce RTX 3060.

I will discuss the guessing strategies I employed a bit more below when I list the specific passwords. But mostly I tried to keep it simple. We are given a word list? Probably use that first. There are some all lowercase passwords? Let's just try to brute force just the lowercase one since that will be fast. Hashcat has a built in l33t rule? Lets try that first. Another strategy that helped me a lot, which goes along with keeping it simple, was a friend told me that when he started, he noticed that passwords with length past 6 got way more time consuming to run. So when I got started, I decided I'd only go 2-6 length (besides the word list ones), and focus on the rest later. This idea helped me greatly.

Cracked Passwords: 20/20 (The number does not match the user, it's by order I got them)
1.cincinnati
2.revolutionary
3.longitude
4.forwarding
5.increased
6.ta
7.lsv
8.uddg
9.lkyru
10.ugrknq
11.SJ
12.lr4
13.31w4
14.2c8zc
15.7OgTHT
16.41t3rn4t3
17.c0mp4ni35
18.31iz4b3th
19.int3rn4ti0n411y
20.14b0r4t0ri35

The first five I got by downloading the wordlist.txt in the assignment description and running '.\hashcat.exe -m 500 -a 0 .\shadow.txt .\wordlist.txt -o cracked.txt'. This worked really well and got 5/20 done just like that.

For the next 5 I figured doing random lowercase passwords would be easy, and limiting the length to 6 for now would ensure it would be fast. So I ran '.\hashcat.exe -m 500 -a 3 .\shadow.txt ?l?l?l?l?l?l --increment --increment-min=2 -o cracked.txt'. This took a bit longer, since it had to try each combination of 26 lowercase characters for the given size, but still got me to 10/20 progress relatively fast.

For the next 5 I decided I'd use a similar approach as above, but include uppercase and numbers. Since just lowercase didn't take that long to run, I figured including some more characters and letting it run overnight couldn't hurt. This did take a while, but was worth it and got me to 15/20.

For the last 5, I figured that they had to be l33t speak, since each other methods covered 5 passwords each and this was the last method to try. I started with Hashcats built in l33t code rules combined with the assignments word list, using the command '.\hashcat.exe -m 500 -a 0 .\shadow.txt .\wordlist.txt -r .\rules\leetspeak.rule -o cracked.txt'

Unfortunately, this didn't crack any passwords. Upon looking in the .rule file, I figured I'd try my own l33t speak rule. For this version, I got rid of all symbols, and had adjusted the all case variant then ran '.\hashcat.exe -m 500 -a 0 .\shadow.txt .\wordlist.txt -r .\rules\custom.rule -o cracked.txt' This also proved fruitless. So I went to the l33t wiki page. Based on this, I adjusted b to be 8, g to be 6, and ran again.Still no luck. So I figured I'd try 5 replacements at a time, doing 0 - 4 first, and came up with this rule: so0si1ss2se3sa4. Again, nothing. But I did notice something weird. The wiki listed S > 2, but the original Hashcat rule had S > 5. This led me to believe that maybe the wiki or the Hashcat rules could be wrong. So I stared at both, and tried to make one that made the most sense to me. S felt like it should be 5 instead, so I changed that. The lowercase i being 1 also felt weird to me, and the wiki had 1 as lowercase L, so I tried that, while still trying to stick to my "Do 5 at a time" idea (which I'm now realizing I added ss5 to which makes 6 rules but oh well) . Additionally, I asked the TA for any guidance, and she suggested I ensure I was testing each combination of the rules, not just individually then all together. This was somewhat tricky, but I started with the rules so0 sl1 sr2 se3 sa4 ss5, and then created the 64 combinations of these. Creating all 64 was awful, but it payed off, because when I ran this, BOOM! Last 5 done!

Questions

1. How long would it take to crack an alphanumeric password of various lengths using my rate?

    Alphanumeric = lower, upper, digits = 26 + 26 + 10 = 62 characters.

    Possibilities = 62^L where L is the length of the password.
    Time = Possibilities / Rate (11 Million)

    Length = 6

        62^6 / 11,000,000 = 5,163.6 seconds or 1.4 hours.

    Length = 8

        62^8 / 11,000,000 = 19,849,100.5 seconds or 5,513.6 hours.

    Length = 10

        62^10 / 11,000,000 = 76,299,942,351.6 seconds or 21,194,428.4 hours.

    Length = 12

        62^12 / 11,000,000 = 293,296,978,399,809 seconds or 81,471,382,888.8 hours.

2. How long would it take to crack an alphanumeric password of various lengths using the rate 2.5 billion?

    Alphanumeric = lower, upper, digits = 26 + 26 + 10 = 62 characters.

    Possibilities = 62^L where L is the length of the password.
    Time = Possibilities / Rate (2.5 billion)

    Length = 6

        62^6 / 2,500,000,000 = 22.7 seconds

    Length = 8

        62^8 / 2,500,000,000 = 87,336 seconds or 24.2 hours.

    Length = 10

        62^10 / 2,500,000,000 = 335,719,746.3 seconds or 93,255.4 hours.

    Length = 12

        62^12 / 2,500,000,000 = 1,290,506,704,959.1 seconds or 358,474,084.7 hours.

3.

I think the password meter is a decent indicator of security, purely based on a "Time to brute force vs complexity" standpoint. But it doesn't take into account the security of how the password is being stored or possible smarter ways to get passwords. I think length 15-17 is decent, especially using numbers, uppercase letters, lowercase, and symbols. But I think you should be random about using these symbols, as predictable patterns make it easier to brute force a password. But if you have a random password of length 17, symbols numbers and upper and lower case letters, your password is going to take longer than 40,894.529513 yr assuming a rate of 2.5 billion.

4.

I would say yes, SHA-512 does improve password security. This is because it is still cryptographically stronger than MD5. However, the better improvement would come from using a slow and expensive hash to decrease the possibility of brute force attacks.

5.

Yes, salts do increase password security. Since salts allow the same password to hash to different values, this helps to prevent large scale, multi target attacks. However, it won't help make a weak password harder to brute force.

6.

No, offline protection is just as if not more important. Protections like network lag, timeouts, and throttling are non-existent in an offline attack. This is why the data itself must also have it's own protection