Lexy Andershock
Applied Cryptography
COSC 583
November 6th, 2025

Project: Password Cracking

In this project, I utilized John the Ripper (JTR) to crack a given list of passwords. I chose JTR over Hashcat because I thought it have better performance on my MacOS. It is easy to define rulesets, masks, and dictionaries to test different password structures. Masks, specifically, are one of the fastest ways to implement brute-force attacks, according the JTR documentation.

1. Uppercase + Lowercase + Digits of Length 2-8

Cracking Strategy

The strategy employed is selective brute force through custom masking (combination of lowercase, uppercase, and digits). I used this tool because mask mode is the fastest cracking mode in JTR and allows me to test for specific use cases. In this case, I tested the strings from length 2 to 6 consisting of the custom character set.

Script

```
CHARSET='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
./john --format=md5crypt --mask='?d?1?1?1?1?1' -1="$CHARSET" --progress-every=30 --
session='case' shadow.txt
```

Average Rate of Guess and Check

- 25,021 passwords/second

Passwords Cracked

1. user2……..7OgTHT
2. user9……..31w4
3. user10……SJ
4. user11……2c8zc
5. user17……lr4

2. Lowercase Passwords of Length 2-8

Cracking Strategy

The strategy employed is selective brute force through custom masking (all lowercase). In this case, I tested the strings from length 2 to 6 consisting of all lowercase letters.

Script

```
./john --format=md5crypt --mask='?l?l?l' --progress-every=30 --session='lowercase'
shadow.txt
```

Average Rate of Guess and Check

- 18,080 passwords/second

Passwords Cracked

1. user5……lkyru
2. user12……..uddg
3. user15……..ta
4. user18……..ugrknq
5. user20……..lsv

3. Using a Wordlist

Cracking Strategy

The strategy used is a checking from a wordlist (called a dictionary attack). This checks simple, memorable, or commonly used passwords.

Script

```
./john --format=md5crypt --wordlist=wordlist.txt shadow.txt
```

Average Rate of Guess and Check

- 24,855 passwords/second

Passwords Cracked

1. user6……...forwarding
2. user7……...increased
3. user14…….longitude
4. user16…….revolutionary
5. user19…….cincinnati

4. Using a Permuted Wordlist with Leet-Speak

Cracking Strategy

The strategy used is a checking from a wordlist (called a dictionary attack). The difference here is that the original dictionary values are transformed to slightly more complex passwords. In this case, we checked for l33t speak, which is a common substitution.

Script

```
./john --format=md5crypt --wordlist=wordlist.txt --rules=l33t shadow.txt
./john --format=md5crypt --wordlist=wordlist.txt --external=Leet shadow.txt
```

Average Rate of Guess and Check

- 24,109 passwords/second

Passwords Cracked

1. user1……..41t3rn4t3
2. user3……..14b0r4t0ri35
3. user4……...int3rn4ti0n411y
4. user8……...31iz4b3th
5. user13…….c0mp4ni35

Additional Questions

1. The time estimates for cracking alphanumeric passwords are below. The word space is $26 + 26 + 10 = 62$ and the average password guessing rate is 82,900 passwords/second (measured recently for just 6-character masks).

    a. 6-character: $62^6 \text{ passwords} \Big/ 82{,}900\ passwords\ /\ sec = 685{,}166\ sec \approx 7.9\ days$
    b. 7-character: $62^7 \text{ passwords} \Big/ 82{,}900\ passwords\ /\ sec = 42{,}480{,}273\ sec \approx 492\ days$
    c. 8-character: $62^8 \text{ passwords} \Big/ 82{,}900\ passwords\ /\ sec = 2{,}633{,}776{,}907\ sec \approx 30{,}484\ days$
    d. 8-character: $62^{10} \text{ passwords} \Big/ 82{,}900\ passwords\ /\ sec = 1.01e13\ sec \approx 321{,}037\ years$
    e. 12-character: $62^{12} \text{ passwords} \Big/ 82{,}900\ passwords\ /\ sec = 3.9e16\ sec \approx 1{,}234{,}068{,}129\ years$

2. If we were using an RTX 3080 with a password guessing rate of 2.5 billion passwords/second, how long would it take to crack alphanumeric passwords?

    a. 6-character: $62^6 \text{ passwords} \Big/ 2.5e9\ passwords\ /\ sec = 22.7\ sec$
    b. 7-character: $62^7 \text{ passwords} \Big/ 2.5e9\ passwords\ /\ sec = 1{,}408\ sec \approx 23\ mins$
    c. 8-character: $62^8 \text{ passwords} \Big/ 2.5e9\ passwords\ /\ sec = 87{,}336\ sec \approx 24\ hours$
    d. 10-character: $62^{10} \text{ passwords} \Big/ 2.5e9\ passwords\ /\ sec = 335{,}719{,}746 \text{ sec} \approx 10.6 \text{ years}$
    e. 12-character: $62^{12} \text{ passwords} \Big/ 2.5e9\ passwords\ /\ sec = 1.3e12\ sec \approx 40{,}922\ years$

3. The password meter is an okay indicator of actual security. The strength of the password depends on which attack it is subject to. For example, wordlists and permutations are very easy to check, so even if a password is long and uses many symbols, like *P@55w0rd123!*, it is very likely to be guessed because it's commonly used. For preventing brute-force attacks, the minimum password length, per my recommendation, should be 10 characters. Even with fast GPUs, which attackers would likely have if they were conducting large-scale brute force attacks and/or sponsored by a nation state, 10 years is not a feasible time frame to crack passwords.

4. Yes, using SHA-512 would be more secure than using MD5. This is because MD5 uses a simple algorithm with fast operations like AND and XOR to compute the hash. This means that it has a fast hash construction, so attackers can try billions of passwords a second. SHA-512, on the other hand, is complex, meaning it slows down attacks.

5. Using a salt increases security, because even the same password will produce different hashes. This means the attacker cannot use previous information to crack passwords faster; they must try to break every single password instead.

6. Knowing that online attacks of this nature are not possible, I think it increases the importance of offline password attack protection. Users shouldn't worry about making their online passwords incredibly strong to resist brute-force attacks, but they do need to be aware of offline password

attack protections since these are most likely the avenues that attacker will use if they were to take a brute-force or wordlist approach.