Cracking **Passwords** Like A Boss

Jeff Deifik

jeff@deifik.com

About Me

- MS in Cybersecurity, CISSP, C|CISO
- Software for first e-commerce system (from 1985-1995)
- Software for the first orbiting radio telescope satellite
- Software for the most advanced pulse oximeter
- Cybersecurity for government satellite ground control, balancing sound cybersecurity with cost and schedule.
- Interest in the intersection of cybersecurity and software development began with white hat password cracking over 30 years ago.

Three ways to Crack Passwords

- CPU - john the ripper - https://github.com/openwall/john

- GPU - hashcat - https://github.com/hashcat/hashcat

- Rainbow Tables - rcracki_mt - https://github.com/foreni-packages/rcracki_mt

  only works with non saltged passwords

How Passwords Are Stored

- Plaintext passwords are converted into a cryptographically strong hash and the hash is stored.
- For example md5sum of Password123 is a907ac8f85bbece3069a52a39947b287
- Modern hashing adds 'salt' to the plaintext password so that if two people use Password123 when the salt is added, different hashes are generated. This makes using rainbow tables or precomputing hashed passwords impractical. Salt was first used for passwords in 1979 and was 12 bits long. Modern linux systems use 128 bits of salt.

Cryptographic Hashing

- There are many ways to hash a password, with wildly varying speeds. You want a very slow, resource intensive hash so that performing a brute force attach is impractical.
- (current microsoft windows) NTLM - 24,934 MH/s
- (most current linux systems) bcrypt - 13,094 H/s

How long does a password need to be?

- It all depends on the hashing method used to store the password.
- Since bcrypt is 1.9 million times slower than NTLM, assuming you have 100 passible password characters, a password would need to be about 3 characters longer using NTLM than bcrypt for comparable security.
- If you are just using lowercase then the password would need to be 4.4 characters longer using NTLM than bcrypt for comparable security.

Password manager and cryptographically strong passwords

- If you want to be very safe, store all your passwords in a password manager on your own computer.
- keepass - https://keepass.info/
- https://keepass.info/help/v2/license.html - GPL2
- Make sure you use a different password for every website.
- Since you don't have to memorize the passwords, I recommend using 20 character long passwords

When you know something about the password format or length

- If you know the password is 8 characters long and the first 3 characters are digits, you can do a 'mask' attack using the mask ?d?d?d?a?a?a?a?a
- This is much faster than a brute force attack.

Brute Force Attach

- If you know the password is 10 characters long and nothing else the 'mask' is ?a?a?a?a?a?a?a?a?a?a
- This will try all passwords of a given length, but it is the most inefficient way to crack passwords.
- You will want to use HashCat and a big GPU
  - 8 chars upper lower number - 18.340,105,584,896 @229 days on 3060ti
  - 12 char upper lower number - 3,226,266,762,397,899,821,056 @12,791,288 years
  - 16 char upper lower number - 47,672,401,706,823,533,450,263,330,816 years

Dictionary Attack

- There are many so called dictionaries of known passwords available. The best one I have found is RockYou2021. You can quickly test hashes against password dictionaries. This is the most efficient way to crack passwords. You can get 10% to 50% of passwords this way.

Rule Based Attack

- This is what you do after you have done a pure dictionary attach. There are many different rules included with JohnTheRinner and Hashcat. You generally apply the rules to a password dictionary.
- Depending on the hashing method, the size of the dictionary, and the number of hashes you are trying to crack, it can take minutes to years to do a rule based attack.
- You will not get all the passwords, but you can get as much as 90% of them, using good rules and a good dictionary.

Tools

- John The Ripper
  - Infrequent official releases, Many unofficial releases
  - Poor Graphical Processor Unit (GPU) windows support
  - Easy to make custom rules
  - Good mailing list support
- HashCat
  - 7.0 released Aug 2025
  - Great GPU acceleration
  - Primitive rule syntax
  - Dictionary attacks takes a lot of memory

Wordlists

- Some very high quality
- Most stuffed full of junk and require editing
  - Very long lines, often thousands of characters long
  - Non ASCII letters
  - Separators that are not newlines
  - Since they are big, specialized tools are needed
- Rockyou2021 is a bit big, but very high quality

My Custom Password Tools

- short -s 40 foo > foo.40 short -l 41 foo > foo.41
  - splits foo into 2 files, 40 chars and shorter, and 41 chars and longer
- msort -l foo > foo.l - Sorts foo by line length
- ascii-lines -p foo > foo.p
  - only outputs lines of foo compromised solely of printable ascii characters
- multi-merge foo.1 foo.2 foo.3 > foo.123
  - merges any number of sorted files into a big sorted file
- sample -10000 foo > foo.10k
  - outputs one line every 10000 lines, for sampling foo
- line_len foo - Prints line length counts
- count foo - Character frequency count
- pw_stats – Shows stasicists on passwords
- pw_unhex – Removes hex encoding from found passwords

Standard Wordlist Tools

- gnu sort
  - You generally want to process sorted wordlists
  - Works with files bigger than RAM using tmp files
- uniq
  - Remove duplicate words
- comm
  - Removes duplicate words in different files
- emacs
  - The one true editor, regular expressions, can process gigabyte files

Relative Hashing Speed

- NTLM Speed 41,825.0 MH/s ☺
- md5 Speed 24,943.1 MH/s
- LM Speed 18,382.7 MH/s
- descrypt Speed 906.7 MH/s
- SHA1 Speed 788.2 MH/s
- scrypt Speed 435.1 kH/s
- WPA2 Speed 396.8 kH/s
- bcrypt Speed 13094 H/s
  https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40

Salt

- 1979 - Unix 12 bits, 4,096 different salts

  https://spqr.eecs.umich.edu/courses/cs660sp11/papers/10.1.1.128.1635.pdf
- 1980’s - Unix 48 bits, 281,474,976,710,656
- 1996 - bcrypt 128 bits, 3.4 x 10^38 salts
- Argon2 128 bits, 3.4 x 10^38 salts
- Descrypt uses 12 bits of salt
- LM and NTLN doesn’t use salt ☺

Password Statistics on 1061.0m - Length

Length:

1: 0.0 % (276)	2: 0.0 % (19905)	3: 0.0 % (405k)
4: 0.2 % (1,830k)	5: 0.7 % (7,602k)	6: 5.5 % (58,667k)
7: 8.4 % (88,972k)	8: 26.4 % (279,591k)	9: 14.6 % (155,327k)
10: 16.4 % (174,317k)	11: 9.0 % (95,195k)	12: 6.3 % (67,173k)
13: 4.0 % (41,969k)	14: 2.7 % (28,177k)	15: 2.5 % (26,807k)
16: 1.5 % (15,595k)	17: 0.6 % (6,146k)	18: 0.5 % (4,821k)
19: 0.3 % (3,046k)	20: 0.2 % (2,390k)	21: 0.1 % (848k)
22: 0.1 % (695k)	23: 0.0 % (403k)	24: 0.0 % (404k)
25: 0.0 % (224k)	26: 0.0 % (224k)	27: 0.0 % (138k)
28: 0.0 % (2572)	29: 0.0 % (2071)	30+: 0.0 % (11179)

Password Statistics on 1061.0m - Characters

all lower: 16.1 % (170,962k)
all upper: 0.6 % (6,854k)
all digit: 7.9 % (83,592k)
all special: 0.0 % (60368)
all lower digit: 41.9 % (444,844k)
all alpha: 2.5 % (26,758k)
all alpha digit: 14.2 % (151,069k)
all lower special: 2.3 % (23,993k)
all upper digit: 3.0 % (31,748k)
all digit special: 0.5 % (5584375)
all akoha special: 0.6 % (6,169k)
all lower digit special: 4.7 % (49,710k)
all alpha digit special: 5.0 % (53,131k)
Has control char: 0.1 % (897818)
Has 8 bit asciil: 0.1 % (675318)

Password Statistics on 1061.0m – String Classes

String Classes:

All alpha: 19.3 % (204,575k)

Alphas + Numbers: 35.9 % (380,608k)

Numbers + Alphas: 6.6 % (70,361k)

Alphas + Specials: 0.7 % (7,391k)

Alphas + Numbers + Alphas: 6.7 % (71,374k)

Numbers + Alphas + Numbers: 2.2 % (23,253k)

Alphas + Specials + Alphas: 1.6 % (17,420k)

Password Statistics on 380.6m – <alpha><num>

<alpha> then 1 <numbers> 12.9 % (48,985k)
<alpha> then 2 <numbers> 23.0 % (87,610k)
<alpha> then 3 <numbers> 15.7 % (59,916k)
<alpha> then 4 <numbers> 25.2 % (96,027k)
<alpha> then 5 <numbers> 5.7 % (21,557k)
<alpha> then 6 <numbers> 8.4 % (31,881k)
<alpha> then 7 <numbers> 3.3 % (12,542k)
<alpha> then 8 <numbers> 3.4 % (12,990k)
<alpha> then 9 <numbers> 1.2 % (4,535k)
<alpha> then 10 <numbers> 1.2 % (4,560k)

Control chars in passwords

nul [0]=751	soh [1]=3815	stx [2]=3302	etx [3]=3572
eot [4]=4466	enq [5]=3282	ack [6]=3292	bel [7]=3151
bs [8]=3863	ht [9]=140k	lf [10]=8	vt [11]=3406
ff [12]=3973	cr [13]=29,284k	so [14]=4017	si [15]=3224
dle [16]=4361	dc1 [17]=4117	dc2 [18]=4404	dc3 [19]=4184
dc4 [20]=3756	nak [21]=4065	syn [22]=4407	etb [23]=4746
can [24]=5130	em [25]=5257	sub [26]=5075	esc [27]=5080
fs [28]=4952	gs [29]=3620	rs [30]=3825	us [31]=4539
del [127]=4428			

Defense

- Don’t use NTLM
- 2 factor authentication
  - What you have - Titan security key, yubikey, smartcard
  - What you are - Fingerprint, Face ID
- Use cryptographically strong random passwords
- Use a password manager
  - keepass, 1password, bitwarden
- I wrote a password generator, here is some output:

password is K)dE;pN%(]R~H6L-11!R bits 129
password is GAw->8k?+Qou#(*#L:Z0 bits 129
password is YmytLWazQ[g{0R@}l2ha bits 129
password is _a^W9h8[J~jsO)*6ahaQ bits 129
password is [q;)y_):BTJAfHZU)7.* bits 129

Advanced Cracking Details

Cracking 1,297,509,369 Passwords

- Dump from Have I Been Pwned
- Good news – they are NTLM format
- Bad news – 1,297,000,000
- This requires a Big Data approach and lots of RAM
- Started with 128gb and went to 256gb
  - Generally needs server grade hardware for lots of RAM
- Limited RAM means I could only run a few threads at the beginning
- I have found 80.7%

Rainbow Tables

- Doesn’t play nice with salt
- Very very fast ☺
- Works with LM, NTLM, MD5, etc.
- Defcon data duplication village – 6tb drives
  - freerainbowtables.com GSM A51 and MD5 hash tables
  - more rainbowtables, lanman, mysqlsha1, ntlm, and some word lists
- Best used with a small number of hashes

Starting to Crack - Using Rainbow Crack

- I tried Rainbow Crack 1.8
- Used NTLM loweralpha-space 9 char rainbow table
  - 43 gigabytes
- Unable to get it working
  - Complex process to convert downloaded tables to rainbow table
  - Unable to crack a known hash
  - **Uses 160kbytes per hash ☹**
  - **Therefore on 936m passwords, 150,000 gigabytes RAM required ☹**
  - Contacted project rainbow crack Sep 26 – no response

Starting to Crack - Using Rainbow Crack

- I tried rcracki_mt (0.7.0) (works with rti2 files)
  - It actually works, unlike rainbow crack
- Used NTLM loweralpha-space 9 char rainbow table
  - 35 gigabytes

Takes 6 seconds per file (SATA SSD), 84 files (504 sec per hash)

- 900m passwords will take 16,000 years ☹
- Good for cracking a few passwords, bad for millions

Starting to crack - Using Hashcat

- My hashcat machines has 16gb of ram.
- When I ran hashcat on 1m passwords:

hashcat.exe -m 1000 ..\pwned_pw_pruned_ntlm.rawest.1m ..\dictionaries\rock.dic (3.9mbyte dictionary)

Host memory required for this attack: 667 MB

Therefore on 936m passwords, 624 gigabytes RAM required ☹

Using Hashcat – Got VRAM ?

- Mask attack i.e. all 8 char (lower, upper, number) passwords
- Between 120 m and 150 m hashes runs out of GPU memory (8gb)
  - Bought a Nvidia 5060ti 16gb of VRAM – works with 233m hashes
  - 48% faster than the 3060ti cracking NTLM hashes
- 8 char (lower, upper, number) takes 9 days, 19 hours to run

John the Ripper – Got DRAM ?

- Using JTR rules
- Started using JTR / default dictionary & rules
- Using –fork option consumes a lot of ram – typically 30gb per fork
- Upgraded from 128gb to 256gb
- Running 6 forks currently
- Found 487,193,352 passwords in 12 days
- Lots more work to do

More JTR

- JTR default dictionary and rules
  - Found 154m passwords
- JTR incremental attack (which never finishes)
  - Total found 325m passwords
- JTR using rockyou2021 wordlist
  - Found 156m passwords (already found with incremental ☹)
- Got total 256gb ram
- JTR –fork=6 default wordlist & rules
  - Found 15m more passwords

More JTR

- JTR --fork=7 apply rules twice
  - Found 36m more passwords
- JTR –fork=8 apply rules to rockyou2021
  - Found 265m passwords in less than an hour ☺
- JTR --fork=18 rules on rockyou2021
  - Now we can use more threads, as only 140m unfound passwords
  - found @11m passwords in 3 days

More JTR

- JTR brute force lower/number up to len=9
  - Brute force all lower/number up to 9 len
  - found @3.6m passwords in @4 days
- JTR rules using 811m found passwords as dictionary
  - Found 16m passwords in @7 days
  - Will take years to finish ☹
- JTR apply rules twice on 811m found passwords
  - Will take years to finish ☹

More JTR – control characters

- john.exe --fork=10 --format=NT --verbosity=2 --no-log --wordlist=\pw-crack\dictionaries\rockyou2021.dic --rules=rep_control_1 \pw-crack\pwn_ntlm.129m.rawest
  - Replace a control char into rockyou2021
- john.exe --fork=22 --format=NT --verbosity=2 --no-log --wordlist=\pw-crack\dictionaries\rockyou2021.dic --rules=ins_control_1 \pw-crack\pwn_ntlm.115m.rawest
  - Insert a control char into rockyou2021
  - Found 8m (@105k tabs, @7.9m cr)

More JTR – control char rules

From solar designer:

```
# Overstrike any one character
[List.Rules:rep_control_1]
# Trivial
# o[0-9A-Z][\x7f\x80\x01-\x1f]
# Optimized
->\r[1-9A-ZZ] >\p[0-9A-Z] o\0[\x7f\x80\x01-\x1f] Q

# Insert any one character
[List.Rules:ins_control_1]
# Trivial
# i[0-9A-Z][\x7f\x80\x01-\x1f]
# Optimized
->\r[2-9A-ZZZ] >\p1[0-9A-Z] i\0[\x7f\x80\x01-\x1f]
```

Hashcat

- Brute force attack
  - lower, upper, number, special — len 7 3.7 days
  - lower, upper, number — len 8 @10 days
    - 6m passwords
  - lower, number — len 9 5.3 days
    - 1.9m passwords
  - upper, number — len 9 5.3 days
    - 1.1m passwords found
  - Lower, number — len 10 – 180 days
    - 2.46m passwords found