Project

Cracking and Analysis of the Top 10k Password Hashes From Have I Been Pwned (HIBP)

Date: 12th Aug 2024

By:
Mohammed Amer
Cybersecurity analyst
mdamer@protonmail.com
https://mdamer.in

```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 100 (SHA1)
Hash.Target......: hashes.txt
Time.Started.....: Mon Aug 12 19:00:19 2024 (10 secs)
Time.Estimated...: Mon Aug 12 19:00:29 2024 (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1406.1 kH/s (0.88ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 9424/9998 (94.26%) Digests
Remaining........: 574 (5.74%) Digests
Recovered/Time...: CUR:N/A,N/A,N/A AVG:N/A,N/A,N/A (Min,Hour,Day)
Progress.........: 14344385/14344385 (100.00%)
Rejected.........: 3094/14344385 (0.02%)
Restore.Point....: 14344385/14344385 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[21217265626f756e642121] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1..: Temp: 69c Util: 45%
```

1. Introduction

Password breaches are a major cybersecurity threat. Services like Have I Been Pwned (HIBP) [1] play a crucial role in protecting users by alerting them about breached passwords and enabling insightful analyses of big datasets of passwords.

My objective with this project was to test and showcase my skills in SQL, password cracking and data analysis.

2. Downloading the Hashes

HIBP enables users to download the full database of hashed passwords hashed in SHA1. As the official downloader [2] was breaking downloads, I used an unofficial downloader instead [3]. The file size was 37.6 GB uncompressed. At the time of downloading there were 936,493,903 hashes and the number of times the password appeared in breaches (prevalence). Each line contained [hash]:[prevalence].

The file looked like this:

```
0001400440AD1D2FABA84FED4436280C74D48A94:4
00014004FB6394EB0CBE5BC4DCB0DBD57329DF1B:5
000140128B8CB046BD41D7CA2EAA509074D4203D:2
0001401AC642959A7E052D917232ADBD2FFE5FAC:1
...
```

3. Splitting the Hashes Into Smaller Files

When I tried to build a database, I got some errors, so I had to split the text file containing the hashes into smaller, manageable chunks. I used Linux's split utility to do so.

```
split -b 5G hashes.txt
```

4. Building a SQL Database

I used a Python script [5] mentioned in an article [6], with slight modifications, to read the text file, create an SQLite database, build an index for fast searching and later allow for sorting of prevalence.

This article introduced me to the concept of B-tree used in SQL index. Using it, querying turned out to be blazing fast.

One thing to note is that the database is 2.4x bigger than the source, with a size of 89.1 GB due to the index.

5. Extracting the Top 10k Hashes

I used the following SQL query to extract the top 10k hashes, based on the prevalence of the breaches:

```
CREATE TABLE top_hashes AS
SELECT hash, password, prevalence
FROM hashes
ORDER BY prevalence DESC
LIMIT 10000;
```

And then I used the software DB Browser for SQLite to graphically export the top 10k hashes to a CSV file.

6. Preparing the Hashes for Cracking

Using LibreOffice Calc, I manually extracted only the hashes (and left the prevalence data) and saved them as top_hashes.txt.

7. Cracking the Hashes Using Hashcat

I used Hashcat and tried various attack methods to crack these hashes.

7.1 Brute-Force Attack

I first tried brute-force to crack the smaller passwords:

```
hashcat -m 100 -a 0 hashes.txt -O --potfile-path=pot -o
cracked.txt --increment --increment-min=0 –increment-max=7 ?l?l?l?
l?l?l?l
```

`-O` uses the optimised kernel

`--increment` mode tries charsets of increasing lengths

I tried various charsets in an incrementing fashion, like lowercase, uppercase, and digits, and these combined with symbols. When I felt that the attack required a lot of time to complete, I switched to wordlists.

7.2 Wordlists Attack

I used multiple wordlists like RockYou (older version) [7], names [8] and cities [9], which I found through [10].

```
hashcat -m 100 -a 0 -O --potfile-path=pot -o cracked.txt
hashes.txt rockyou.txt
```

7.3 Combination Attack

I used the combination attack by appending some of the wordlists to themselves.

```
hashcat -m 100 -a 1 -O --potfile-path=pot -o cracked.txt
hashes.txt rockyou.txt rockyou.txt
```

7.4 Hybrid Attack

I used Hashcat's attack modes 6 and 7 to try wordlists with common prefixes and suffixes like `?d?d?d?d` and `?s`.

Using these different attacks, I was able to crack 9656 hashes out of 10k.

8. Using Online Rainbow Tables

For the remaining 344 hashes, I used the site hashes.com [11] to check the hashes against their database. I successfully retrieved 317 out of 344 hashes. Overall, 20 were left uncracked.

Note: The statistics don't tally up, as mentioned above. I am unable to resolve the discrepancy.

9. Updating the SQL Database With the Plaintext Passwords

I converted the cracked hashes to uppercase using LibreOffice Calc using the `UPPER()` function and saved the file using CSV format.

Next, I imported the file into DB Browser for SQLite as a separate table and merged it with the original table with the hashes and the prevalence:

```
CREATE TABLE final AS
SELECT top_hashes.hash, cracked.password, top_hashes.prevalence
FROM top_hashes LEFT JOIN cracked
ON top_hashes.hash = cracked.hash;
```

10. Statistical Analysis of the Passwords

I used Calc and Google Sheets to do some analysis, like calculating the length of passwords, checking the different charsets (character sets) present, calculating character size and entropy and classifying the passwords based on the charset. The spreadsheet looked like this:

Password	length	lowercase	uppercase	digit	special	Character size	Entropy	Charset
123456	6	0	0	1	0	10	19.93	digit
123456789	9	0	0	1	0	10	29.9	digit
password	8	1	0	0	0	26	37.6	lower
12345678	8	0	0	1	0	10	26.58	digit
111111	6	0	0	1	0	10	19.93	digit
qwerty123	9	1	0	1	0	36	46.53	lower_digit
1q2w3e	6	1	0	1	0	36	31.02	lower_digit

10.1 Formulas Used:

Checking the lowercase charset:

```
=IF(SUMPRODUCT(--ISNUMBER(FIND(MID(A2,ROW(INDIRECT("1:" & LEN(A2))),1),"abcdefghijklmnopqrstuvwxyz")))>0,1,0)
```

Calculate character size:

```
=C2*26 + D2*26 + E2*10 + F2*10
```

Note: For future projects, it can also be considered to use a bigger character size for symbols, as per Hashcat's documentation.

Calculate entropy:

```
=LOG(G2, 2) * B2
```

11. Results

11.1 Basic Summary

Total hashes		10000
Cracked	Total	9973
	Hashcat	9656
	Online rainbow tables	317
Not cracked		20

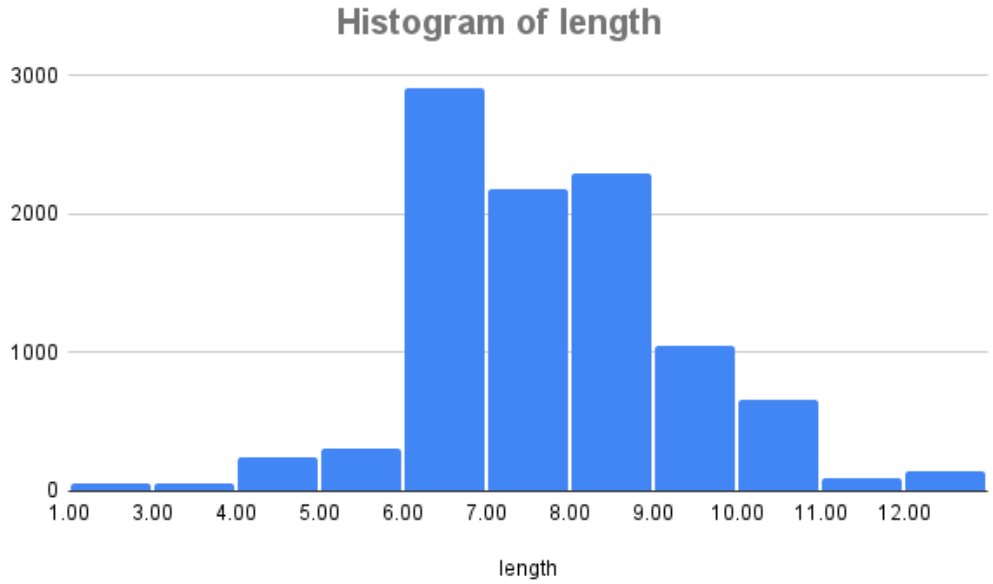
Note: There were some anomalies in the data, which I was unable to resolve. That is the reason for some of the discrepancies in the statistics. For example, there were multiple occurrences of 0 in the passwords cracked by Hashcat. And there seemed to be multiple passwords in the database with the same hashes.

11.2 Length of Passwords

Mean	7.34
Median	7
Minimum	1
Maximum	114
Standard deviation	3.00

The bar graph below was generated after removing 1% of outliers, due to the extreme values like `$HEX[d0bfd197d085d0bfd197d085d0bfd197d085d0bfd197d085d0bfd197d085d0bfd197d085d0bfd197d085d0bfd197d085d0bfd197d085].`

I tried to convert the hex value to ASCII, but only after the statistics were generated. But, the output was gibberish in this case. This could be explored in later projects.

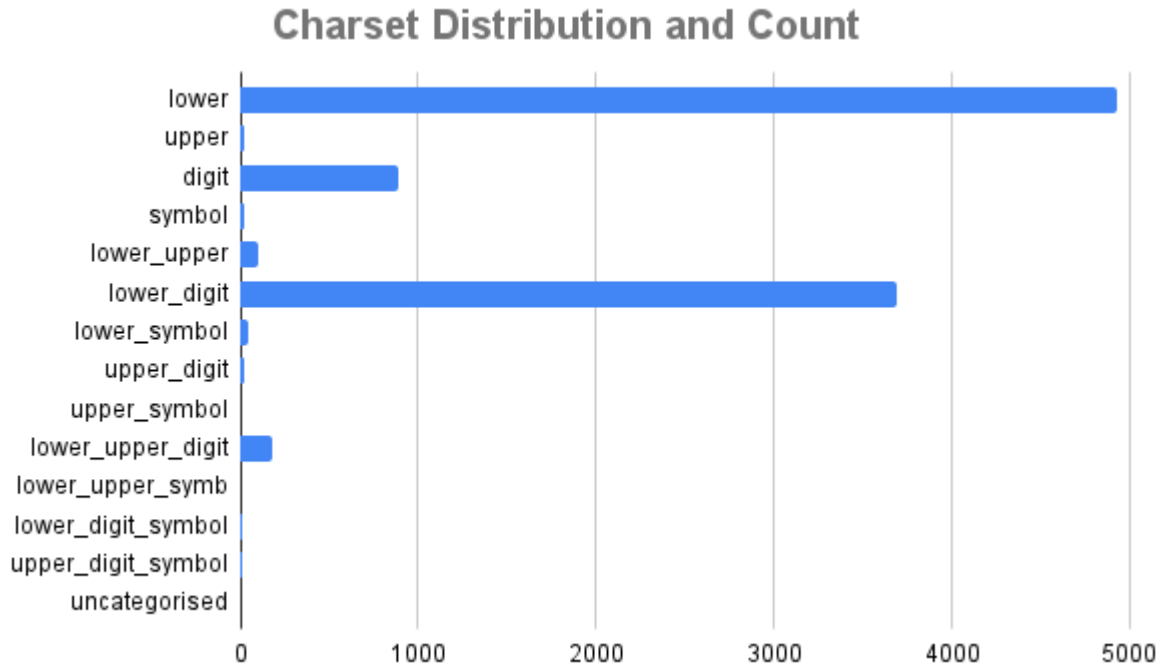


11.3 Charset

Note: lower here means that only lowercase letters were used and so on for other charsets.

Charset	Count
lower	4923
upper	20
digit	891
symbol	21
lower_upper	104
lower_digit	3688
lower_symbol	39
upper_digit	18
upper_symbol	2
digit_symbol	14
lower_upper_digit	180

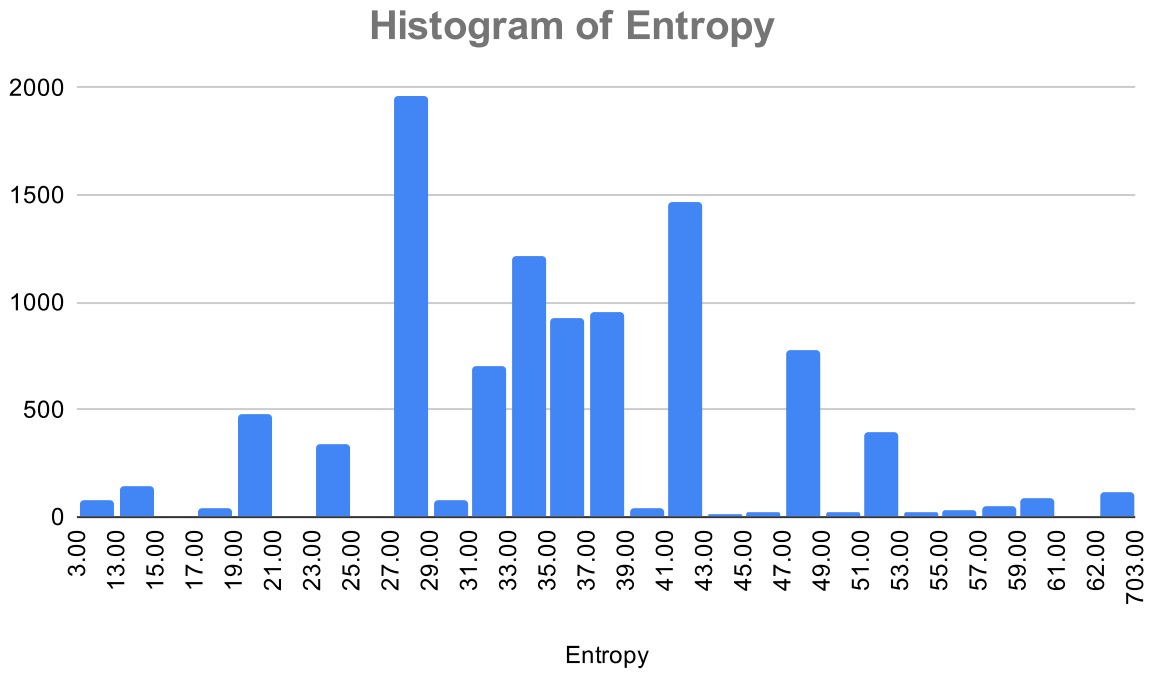
lower_upper_symbol	1
lower_digit_symbol	15
upper_digit_symbol	6
uncategorised	0



11.4 Entropy

Minimum	3.32
Maximum	703.37
Mean	35.65
Median	33.22
Standard deviation	18.39

Note: In the following histogram, 1% outliers were removed.



12. Conclusion

Overall, it was challenging, but fun project. It allowed me to hone many of my skills, especially handling large datasets and the nuances of password cracking. I believe that such analyses of leaked passwords, especially on bigger subsets, could have a significant benefit to the cybersecurity world and the users, by generating insights into the passwords used.

13. References

1. https://haveibeenpwned.com/
2. https://github.com/HaveIBeenPwned/PwnedPasswordsDownloader
3. https://github.com/easybill/easypwned
4. https://medium.com/analytics-vidhya/creating-a-local-version-of-the-haveibeenpwned-password-database-with-python-and-sqlite-918a7b6a238a
5. https://github.com/ChrisInmodis/Haveibeenpwned-Local-Version/tree/main/English
6. https://medium.com/analytics-vidhya/creating-a-local-version-of-the-haveibeenpwned-password-database-with-python-and-sqlite-918a7b6a238a
7. https://github.com/kkrypt0nn/wordlists/blob/main/wordlists/famous/rockyou.zip
8. https://github.com/kkrypt0nn/wordlists/blob/main/wordlists/names/names.txt
9. https://github.com/kkrypt0nn/wordlists/blob/main/wordlists/security_question_answers/cities.txt
10. https://github.com/kkrypt0nn/wordlists/tree/main

11. https://hashes.com/en/decrypt/hash