BruteForce - RainbowTables and Dictionary Attacks
An Analysis

Network Security: Assignment-1

Practical

---

1. Rainbow Tables

Overview

A powerful tool for decrypting passwords.A **rainbow table** is a precomputed **table** for caching the output of cryptographic hash functions, usually for cracking password hashes. **Tables** are usually used in recovering a key derivation function (or credit card numbers, etc.) up to a certain length consisting of a limited set of characters.

Unlike bruteforce attack, which works by calculating the hash function of every string present with them, calculating their hash value and then compare it with the one in the computer, at every step. A rainbow table attack eliminates this need by already computing hashes of the large set of available strings.

Hash Function

Some hash algorithms used in Rainbow Tables are given in table 1.

Hash Algorithm	Hash Length	PlainText Length
md5	16	0-7
sha1	20	0-20
sha256	32	0-20
lm	8	0-7
ntlm		0-15

Table 1. *List of Hash Algorithms*

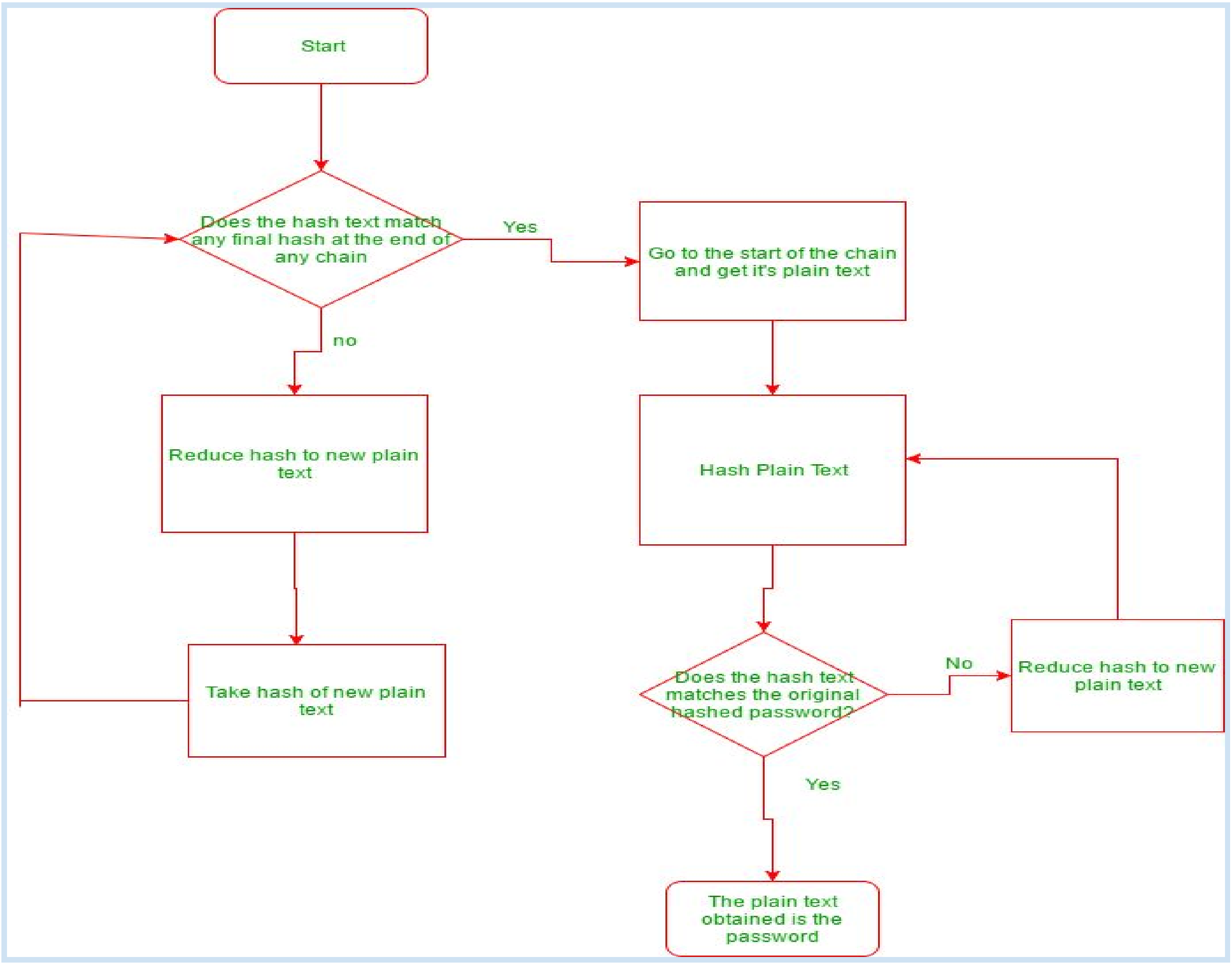
Reduction function

RainbowTable files are very large, though. So that the required storage space doesn't get out of

$$\begin{tikzcd}\texttt{aaaaaa} \arrow[r, "H"'] & \texttt{281DAF40} \arrow[r, "R"'] & \texttt{sgfnyd} \arrow[r, "H"'] & \texttt{920ECF10} \arrow[r, "R"'] & \texttt{kiebgt}\end{tikzcd}$$

hand, rainbow tables use a reduction function that change the hash value into plaintext.
Important: The reduction function doesn't reverse the hash value, so it doesn't output the original plaintext (i.e. the password) – because this isn't possible – but instead outputs a completely new one.
A new hash value is them generated from this text. In a rainbow table, this takes place not only one time, but many times, resulting in a chain. In the final table, however, only the first password and the last hash value of a chain appear.

WorkFlow

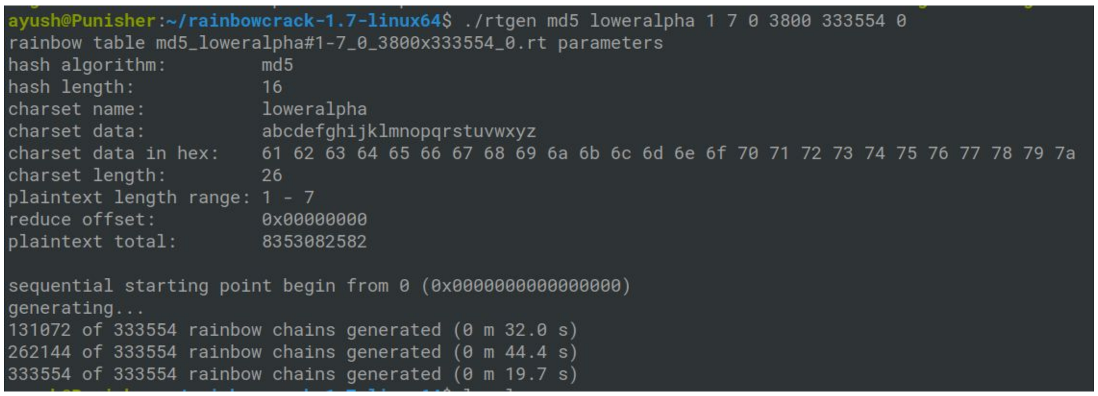

Start
Does the hash text match any final hash at the end of any chain
Yes
Go to the start of the chain and get it's plain text
no
Reduce hash to new plain text
Hash Plain Text
Take hash of new plain text
Does the hash text matches the original hashed password?
No
Reduce hash to new plain text
Yes
The plain text obtained is the password

Some Tools

- Rainbow Crack --used for practical example
- CrackStation (Online)
- John the Ripper

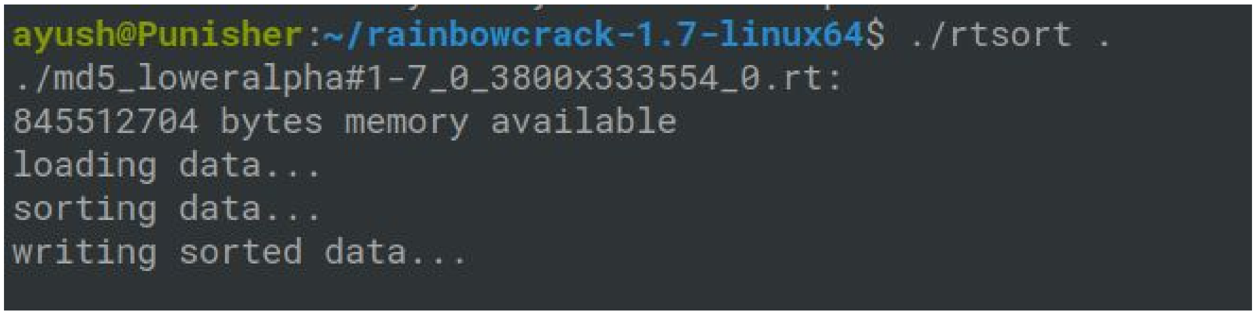
Practical Example -using rainbowCrack tool

1. Creating rainbow table
    a. **Hash** : md5sum
    b. **Charset** : loweralpha
    c. **Length password**: 1-7
    d. **Chain length**: 3800
    e. **Number of Chains** : 333554



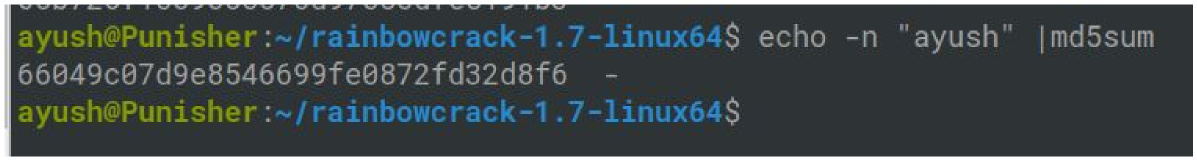
```
ayush@Punisher:~/rainbowcrack-1.7-linux64$ ./rtgen md5 loweralpha 1 7 0 3800 333554 0
rainbow table md5_loweralpha#1-7_0_3800x333554_0.rt parameters
hash algorithm:         md5
hash length:            16
charset name:           loweralpha
charset data:           abcdefghijklmnopqrstuvwxyz
charset data in hex:    61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a
charset length:         26
plaintext length range: 1 - 7
reduce offset:          0x00000000
plaintext total:        8353082582

sequential starting point begin from 0 (0x0000000000000000)
generating...
131072 of 333554 rainbow chains generated (0 m 32.0 s)
262144 of 333554 rainbow chains generated (0 m 44.4 s)
333554 of 333554 rainbow chains generated (0 m 19.7 s)
```

2. Sorting data in the generated table.



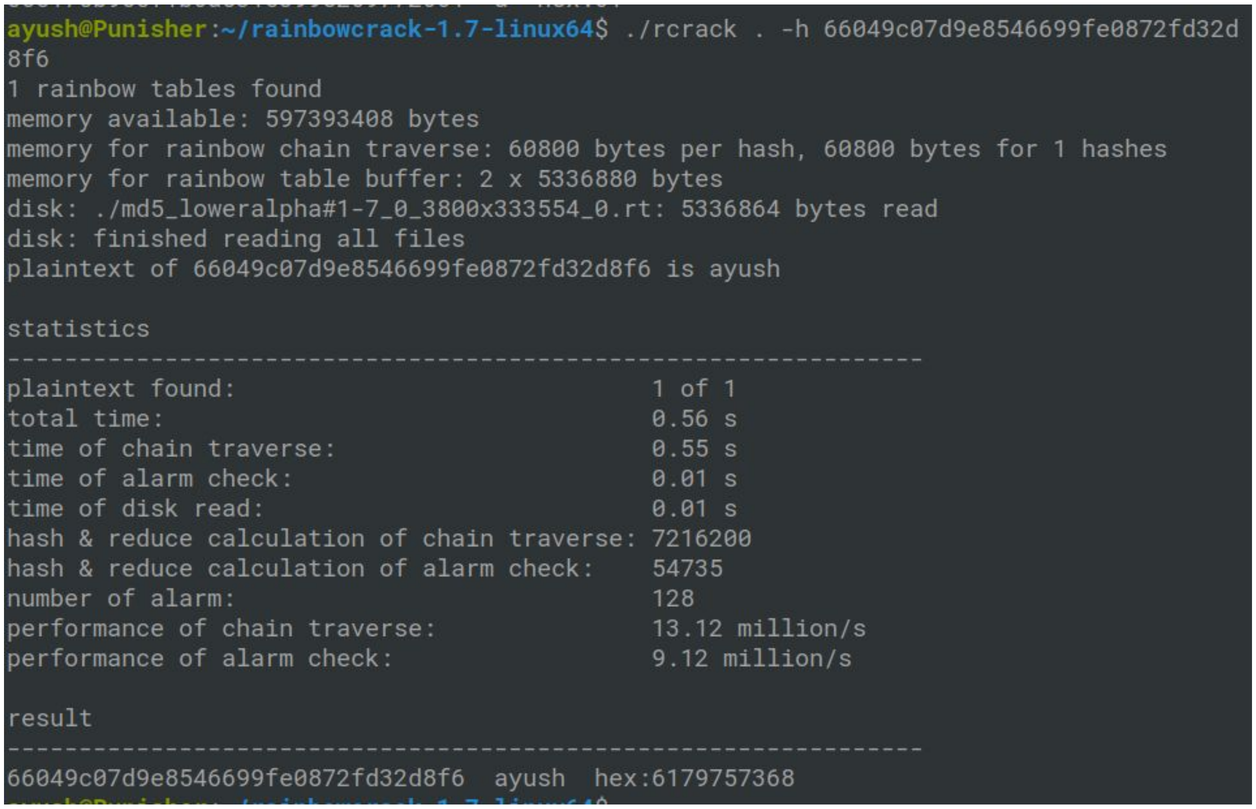
```
ayush@Punisher:~/rainbowcrack-1.7-linux64$ ./rtsort .
./md5_loweralpha#1-7_0_3800x333554_0.rt:
845512704 bytes memory available
loading data...
sorting data...
writing sorted data...
```

3. Generating hash for Plaintext : Ayush



```
ayush@Punisher:~/rainbowcrack-1.7-linux64$ echo -n "ayush" |md5sum
66049c07d9e8546699fe0872fd32d8f6  -
ayush@Punisher:~/rainbowcrack-1.7-linux64$
```

4. Using rcrack to crack hash using generated rainbow table.



```
ayush@Punisher:~/rainbowcrack-1.7-linux64$ ./rcrack . -h 66049c07d9e8546699fe0872fd32d
8f6
1 rainbow tables found
memory available: 597393408 bytes
memory for rainbow chain traverse: 60800 bytes per hash, 60800 bytes for 1 hashes
memory for rainbow table buffer: 2 x 5336880 bytes
disk: ./md5_loweralpha#1-7_0_3800x333554_0.rt: 5336864 bytes read
disk: finished reading all files
plaintext of 66049c07d9e8546699fe0872fd32d8f6 is ayush

statistics
----------------------------------------------------------------
plaintext found:                             1 of 1
total time:                                  0.56 s
time of chain traverse:                      0.55 s
time of alarm check:                         0.01 s
time of disk read:                           0.01 s
hash & reduce calculation of chain traverse: 7216200
hash & reduce calculation of alarm check:    54735
number of alarm:                             128
performance of chain traverse:               13.12 million/s
performance of alarm check:                  9.12 million/s

result
----------------------------------------------------------------
66049c07d9e8546699fe0872fd32d8f6  ayush  hex:6179757368
```

Hence Succesfully performed rainbow table attack

Time Memory Trade-off

A time-memory tradeoff is basically when you accept a longer runtime in favor of fewer memory requirements – or the other way around. A table, on the other hand, in which billions of passwords are presented together with their hash values, takes up an enormous amount of storage space, but can very quickly run decryptions. Rainbow tables represent a compromise of both. In principle, they also perform real-time calculations, but to a lesser extent, and so save a lot of storage space compared to complete tables.

Disadvantages

Rainbow table attacks, can be thwarted by the use of ***salt***, *a technique that forces the hash dictionary to be recomputed for each password sought, making precomputation infeasible*, provided that the number of possible salt values is large enough.

Salts are a random token (usually used only once) that is combined with the password before hashing.It artificially increases the length of a password in the rainbow table (so to crack a 4 character password with a 4 character salt, you'd need to generate an 8 character rainbow table).

Protective Measures

1. Don't use MD5 or SHA1 in your password hashing function. MD5 and SHA1 are outdated password hashing algorithms. Consider using more modern hashing methods like SHA2.

2. Use a cryptographic "Salt" in your password hashing routine.

2. <u>Dictionary Attack</u>

Overview

A Dictionary Attack as an attack vector used by the attacker to break in a system, which is password protected, by putting technically every word in a dictionary as a form of password for that system. This attack vector is a form of Brute Force Attack.

A dictionary attack is based on trying all the strings in a pre-arranged listing. Such attacks originally used words one would find in a dictionary (hence the phrase dictionary attack).

Working

A dictionary attack tries only those possibilities which are deemed most likely to succeed. Dictionary attacks often succeed because many people have a tendency to choose short passwords that are ordinary words or common passwords, or variants obtained, for example, by appending a digit or punctuation character.

Dictionary attacks are difficult to defeat, since most common password creations techniques are covered by the available lists, combined with cracking software pattern generation. A safer approach is to randomly generate a long password (15 letters or more) or a multi word passphrase, using a password manager program or a manual method.

Some Tools

- **John the Ripper ---**used here
- L0phtCrack
- Metasploit Project
- Ophcrack
- Hashcat
- Mentalist

Password Manager

To boost productivity of this attack. Password manager are used which customize the list based on our requirement so more contracted and efficient list can be created for lookup.

One such tool is -- mentalist. It asks various questions like name, DOB, age, phone, family, pet name, favourite food etc. and prepares a list of passwords accordingly which can be used for attack.

Protective Measures

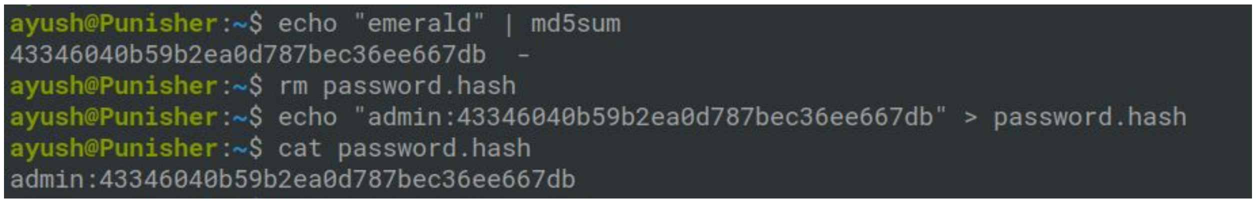
You can protect yourself from such kind of attacks by following ways:

- Choose a mix of upper and lower case letters, numbers and specials (i.e. special characters).
- Password must be a long string with more characters. The longer it is, the more time consuming it is to crack (sometimes, time to crack is in years).
- Password reset should be done after a certain period of time

Dictionary attacks are not effective against systems that make use of multiple-word passwords, and also fail against systems that use random permutations of lowercase and uppercase letters combined with numerals.

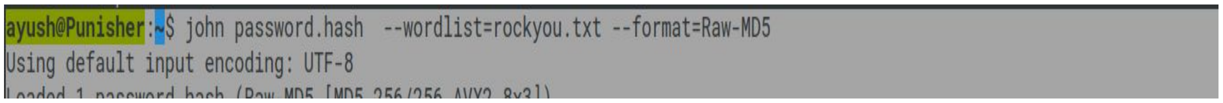
Practical Example

1. Creating hash for password "emerald" using md5sum and storing in password.hash.
   PASSWORD: **emerald**   HASH: **41ca53dbedb99b4eca36836dafb555e2**



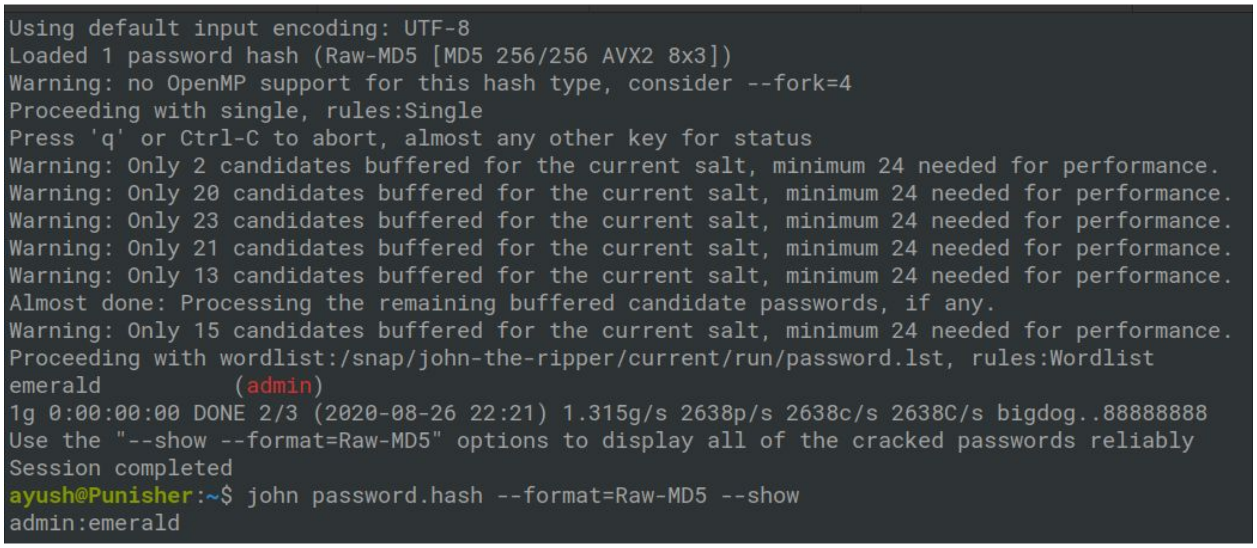
```
ayush@Punisher:~$ echo "emerald" | md5sum
43346040b59b2ea0d787bec36ee667db  -
ayush@Punisher:~$ rm password.hash
ayush@Punisher:~$ echo "admin:43346040b59b2ea0d787bec36ee667db" > password.hash
ayush@Punisher:~$ cat password.hash
admin:43346040b59b2ea0d787bec36ee667db
```

2. Initiating Dictionary attack using JOHN-THE-RIPPER tool. Wordlist applied is rockyou.txt, which has **14,341,564** unique passwords from **32,603,388** accounts.



```
ayush@Punisher:~$ john password.hash  --wordlist=rockyou.txt --format=Raw-MD5
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
```

3. Output log from john-the-ripper. For getting password use **command:** *john <passwd file> --show*


```
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 2 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 20 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 23 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 21 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 13 candidates buffered for the current salt, minimum 24 needed for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 15 candidates buffered for the current salt, minimum 24 needed for performance.
Proceeding with wordlist:/snap/john-the-ripper/current/run/password.lst, rules:Wordlist
emerald          (admin)
1g 0:00:00:00 DONE 2/3 (2020-08-26 22:21) 1.315g/s 2638p/s 2638c/s 2638C/s bigdog..88888888
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed
ayush@Punisher:~$ john password.hash --format=Raw-MD5 --show
admin:emerald
```

Hence Successfully performed dictionary attack.