SE507μ Hacking Password Hashes
with Rainbow Tables

Session 2

Rainbow Table Attack and Password Database Protection

Sébastien Combéfis

Winter 2020

Objectives

- Understand the rainbow table data structure

  Long chains of data interleaved with their hashes

- Reverse an hash function with rainbow table attack

  Thanks to a time versus memory trade-off

- Evaluate the speedup improvement and memory cost

  Comparing rainbow table with brute force and dictionary attacks


Rainbow Table

Rainbow Table (1)

- Rainbow table is a precomputed table to reverse hash function

  Invented by Oechslin based on simpler algorithm by Hellman

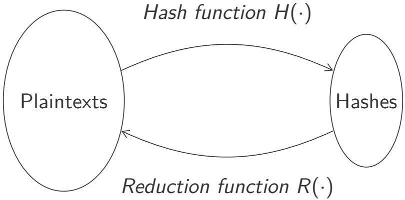
- Rainbow table exploits the space-time trade-off
  - More efficient in time than using brute-force attack
  - Less consuming in storage space than using lookup table

- Work in the reverse order of data hashing process

  Going from the hashes to the original plaintexts

Hash and Reduction Function

- Rainbow tables are built thanks to two operations
  - **Hash** function maps a plaintext to the corresponding hash
  - **Reduction** function maps a hash to a possible plaintext



Hash Chain

- Reduction function maps hash value back to plaintext value

  Inverting the domain and codomain of the hash function

- Building chains of alternating plaintexts and hashes

  By alternating between hash and reduction functions

$$\textbf{abc} \xrightarrow{H} \text{81AB20} \xrightarrow{R} \text{vch} \xrightarrow{H} \text{ECA760} \xrightarrow{R} \text{ben} \xrightarrow{H} \textbf{76B7C4}$$

Collision (1)

- Two hash chains may collide with each other

  Reduction function may generate plaintext already encountered

- Collisions add redundancy in data encoded in rainbow table

  Too many collisions result in “loosing” space

$$\begin{tikzcd}
\textbf{abc} \arrow[r, "H"] & \text{81AB20} \arrow[dr, "R"] & & & & & \\
& & \text{vch} \arrow[r, "H"] & \text{ECA760} \arrow[r, "R"] & \text{ben} \arrow[r, "H"] & \textbf{76B7C4} \\
\textbf{def} \arrow[r, "H"] & \text{2CD8A4} \arrow[ur, "R"'] & & & & &
\end{tikzcd}$$

Rainbow Table Attack

Collision (2)

- Important to avoid collision in a rainbow table

  Redundancy in hash chains wastes space in the table

- Rainbow table uses a set of reduction functions R_j

  Reduction function may generate plaintext already encountered

- Each reduction function associated to a different colour...

  ...hence the name rainbow table

$$\textbf{abc} \xrightarrow{H} \text{81AB20} \xrightarrow{R_1} \text{vch} \xrightarrow{H} \text{ECA760} \xrightarrow{R_2} \text{sam} \xrightarrow{H} \textbf{1FE75C}$$

Rainbow Table Attack

- Building a big rainbow table with many hash chains
  - Considered plaintexts are passwords from stolen database
  - Only need to store the first and last element of the chain
  - The whole chain can be recomputed on-demand if necessary
- Space-time trade-off to improve search performance
  - Using more memory to improve speed
  - 10^{12} hashes can be stored with only 10^6 chains stored

$$\mathbf{P_1} \xrightarrow{H} H_1 \xrightarrow{R_1} P_2 \xrightarrow{H} \cdots \xrightarrow{R_{k-1}} P_k \xrightarrow{H} H_k$$

Finding Password Algorithm

- Iterate to find password P corresponding to hash value H
  1. Check if H is the endpoint of a chain $(P_{i,1}, H_{i,k})$
  2. Recompute the chain $P_{i,1} \rightarrow H_{i,1} \rightarrow ... \rightarrow P_{i,k} \rightarrow H$
  3. A corresponding password for the given H is $P_{i,k}$
- If not found, reduce H and then hash the result to obtain H'
  1. Check if H' is the endpoint of a chain $(P_{i,1}, H_{i,k})$
  2. Recompute $P_{i,1} \rightarrow H_{i,1} \rightarrow ... \rightarrow P_{i,k-1} \rightarrow H \rightarrow P_{i,k} \rightarrow H'$
  3. A corresponding password for the given H is $P_{i,k-1}$

Finding Password Example (1)

- Let's assume a rainbow table with three hash chains

  $(abc\ , 1FE75C)$
  $(def\ , 76B7C4)$
  $(ghi\ , A928B0)$

- Searching for the password corresponding to the $ECA760$ hash

  Search failed, the hash is not an endpoint of any hash chain

Finding Password Example (2)

- Search one layer before the end one by reducing and hashing

  $ECA760 \xrightarrow{R_2} sam \xrightarrow{H} 1FE75C$

- Searching for the password corresponding to the $ECA760$ hash

  Search succeeded, the new hash is an endpoint of an hash chain

- Recomputing the hash chain corresponding to $(abc, 1FE75C)$
  - Chain reconstructed until finding the $ECA760$ hash
  - abc $\xrightarrow{H}$ 81AB20 $\xrightarrow{R_1}$ vch $\xrightarrow{H}$ ECA760
  - The password corresponding to $ECA760$ is *vch*

Existing Rainbow Table

- Several rainbow tables are available for download

  For example on the RainbowCrack project website

RainbowCrack　Home　Documentation　Rainbow Tables　Performance　Buy Rainbow Tables　Contact

List of Rainbow Tables

This page lists the rainbow tables we generated.

LM rainbow tables speed up cracking of password hashes from Windows 2000 and Windows XP operating system.
NTLM rainbow tables speed up cracking of password hashes from Windows Vista and Windows 7 operating system.
MD5 and SHA1 rainbow tables speed up cracking of MD5 and SHA1 hashes, respectively.

The largest rainbow tables here are ntlm_mixalpha-numeric#1-9, md5_mixalpha-numeric#1-9 and sha1_mixalpha-numeric#1-9. Each has a key space of 13,759,005,997,841,642 (i.e., $2^{53.6}$).

Benchmark result of each rainbow table is shown in last column of the list below. We generate hashes of random plaintexts and crack them with the rainbow table and rcrack/rcrack_cuda/rcrack_cl program. rcrack program uses CPU for computation and rcrack_cuda/rcrack_cl program uses NVIDIA/AMD GPU.

Video demonstration of some rainbow tables on YouTube :

- Hash Cracking with Rainbow Table ntlm_ascii-32-95#1-8
- Hash Cracking with Rainbow Table md5_ascii-32-95#1-8
- Hash Cracking with Rainbow Table sha1_ascii-32-95#1-8

Perfect rainbow tables are rainbow tables without identical end points, produced by removing merged rainbow chains in normal rainbow tables. To achieve same success rate, perfect rainbow tables are smaller and faster to lookup than non-perfect rainbow tables. In lists below, parameters of non-perfect rainbow tables are in gray.

Rainbow Tables

LM Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
lm_ascii-32-65-123-4#1-7	ascii-32-65-123-4	1 to 7	7,555,858,447,479	99.9 %	27 GB 32 GB	Perfect Non-perfect	Perfect Non-perfect

NTLM Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance

Protection

Salt

- Using large salts protects against rainbow table attack

  Concatenated with password before being hashed in database

- Two main possibilities to use salt with passwords
  - saltedhash(password) = hash(password || salt)
  - saltedhash(password) = hash(hash(password) || salt)

- Each user's password is hashed uniquely
  - Makes precomputation attacks very difficult
  - Old UNIX passwords with 12-bit salt requires 4096 tables

Key Stretching

- Combine salt, password and intermediate hash values

  Makes brute-force attacks more time consuming

- Obtaining an enhanced hash from a possibly weak one

  Requires a little bit more time for users to log in

Slow Hash Function

- Better to choose a computationally expensive hash function

  To make it slower for the rainbow table attack to succeed

- Two slow common password hashing algorithms using salt
  - Bcrypt designed to protect against rainbow table attack
  - PBKDF2 used for WPA2 Wi-Fi routers

Password Choice

- Important to choose a strong password

  So that there is no rainbow tables containing the password

- Let's try with the SHA-256 of "**password**" and "**$a;Z2b**"

  For example using https://md5decrypt.net/en/Sha256

References

- Jason Beneducci (2019). *The Land of "Rainbow Tables"*, September 10, 2019. https://medium.com/@jasonbeneducci/rainbow-tables-d88b99c35d61
- Andy O'Donnell (2019). *Rainbow Tables: Your Password's Worst Nightmare*, November 18, 2019. https://www.lifewire.com/rainbow-tables-your-passwords-worst-nightmare-2487288
- Don Donzal (2006). *Tutorial: Rainbow Tables and RainbowCrack*, November 5, 2006. https://www.ethicalhacker.net/columns/gates/tutorial-rainbow-tables-and-rainbowcrack
- Gibson Research Corporation (2012). *How Big is Your Haystack? ...and how well hidden is YOUR needle?*, March 28, 2012. https://www.grc.com/haystack.htm

Credits

- Velvet Elevator (Pandy Farmer), July 30, 2009, https://www.flickr.com/photos/elainelope/3840340711.
- Camron Flanders, May 29, 2012, https://www.flickr.com/photos/camflan/7302192024.
- kismihok, September 5, 2013, https://www.flickr.com/photos/kismihok/9686252463.