Cybersecurity

Rainbow Table Attack Lab

Contributions by Wilson Innovative Solutions LLC

CYBER.ORG

Rainbow Table Attack Lab

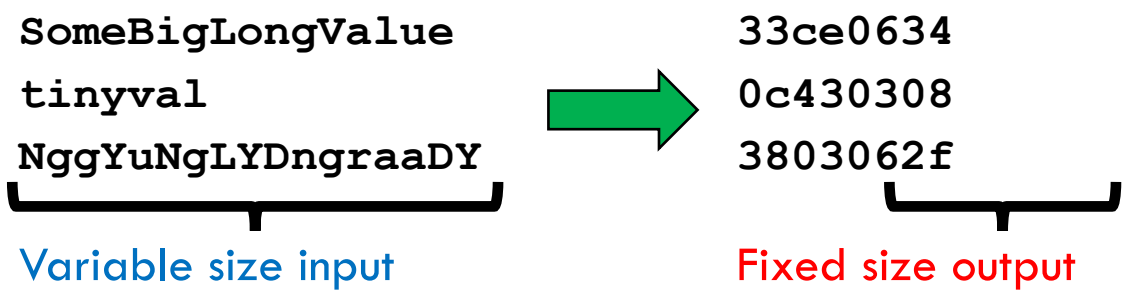
- In this lab students will perform password cracking via the use of rainbow tables.
- Materials needed
  - Kali Linux
- Software Tools used
  - rainbowcrack (Password Cracking Tool)

Objectives Covered

- Security+ Objectives (SY0-601)
  - Objective 1.2 - Given a scenario, analyze potential indicators to determine the type of attack.
    - Password attacks
      - Rainbow tables
  - Objective 4.1 - Given a scenario, use the appropriate tool to assess organizational security.
    - Password crackers

What is a Hash?

- A hashing algorithm is an algorithm that converts input data (or a message) of varying size to a hash output of a fixed size
- A hash is a one-way function, impossible to revert.
- Generally, the longer the fixed output the less possibility of collisions (two inputs producing the same output), thus the more secure the hashing algorithm



What is a Rainbow Table?

- Pre-calculated series of hashes using known hashing algorithms
- Commonly used for cracking passwords
  - Find the matching hash string of text
  - Look up the input text that gave the result
  - *Voila!* There's the password/input string
- Rainbow tables are application-specific
  - Built for each different application or OS
  - No one table for all uses

Plaintext	MD5 Checksum
Alice	64489c85dc2fe0787b85cd87214b3810
Bob	2fc1c0beb992cd7096975cfebf9d5c3b
Carol	150c16d9d096e70af3596111d7402397
Dave	083d9a270e6e16b2fbb08d35067aae5f

How does a Rainbow Table work?

- Get the first **x** characters of a hash
  - Hash these characters
- Get the first **x** characters of *that* hash
  - Hash *these* characters
- Do this repeatedly…
  - This creates a "chain"
  - Each chain can be referred to as a color "red" (first hash), "orange" (second hash), "yellow" (third hash), etc.
- After obtaining enough chains, they create a table
  - A table of all the colors... like a rainbow. Hence a "rainbow table".
- Only store the plaintext and final hash value for each chain
  - All values in between plaintext and final hash can be re-computed as needed

How does a Rainbow Table work?

- To use the table, take the first **x** characters of the target hashed password and look for a match in the table.
  - If a match is not found, take the first **x** characters, hash, and search again
  - If a match *is* found, you know the plaintext at the front of that chain is part of the target password – this narrows the search by **x** characters.
    - Take the next **x** characters and start the process again
- It is a narrowing down of the thousand and millions of possibilities

Rainbow Tables vs. Brute Force

- Advantages of a Rainbow Table
  - No need to match the whole string, looking for parts
  - Not trying all values, only searching a table (fast)
  - Can be done offline
    - System does not know attempts are being made to crack the password of its users!
- Advantages of a Brute Force
  - Does not need to store the large Rainbow Table dataset
    - Which can be large! Can be *gigs* of text or even terabytes
  - Works for all passwords, just takes time (*lots and lots and lots of time*)

The Rainbow Table Attack Lab

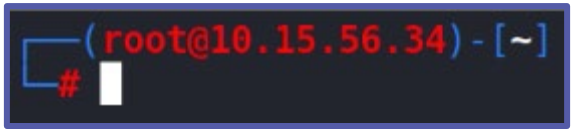
- Log into Kali Linux
- Create Rainbow Table
- Create Hashes from example passwords
- Use Rainbowcrack to crack a hash
- Use Rainbowcrack to crack a file of hashes
- Observe the results

Log into Kali Linux

- Open the Kali Linux Environment
- Open Terminal
- Login as the root user with the following command:

`sudo su -`

- Notice the command prompt is now **`root@kali`**



Create Rainbow Table

- Type the following command*:

  `rtgen -h`

- Read the options available when using this command to create a rainbow table
- Type the following command:

`rtgen md5 loweralpha 1 5 0 16000 16000 0`

- This will create a rainbow table using the MD5 hash algorithm with a hash length of 16 based on input restricted to 5 characters that are lowercase letters
- *This will take time!*

```
┌──(root@10.15.85.23)-[~]
└─# rtgen -h
RainbowCrack 1.8
Copyright 2020 RainbowCrack Project. All rights reserved.
http://project-rainbowcrack.com/

usage: rtgen hash_algorithm charset plaintext_len_min plaintext_len_max tab
le_index chain_len chain_num part_index
       rtgen hash_algorithm charset plaintext_len_min plaintext_len_max tab
le_index -bench

hash algorithms implemented:
    lm HashLen=8 PlaintextLen=0-7
    ntlm HashLen=16 PlaintextLen=0-15
    md5 HashLen=16 PlaintextLen=0-15
    sha1 HashLen=20 PlaintextLen=0-20
    sha256 HashLen=32 PlaintextLen=0-20

examples:
    rtgen md5 loweralpha 1 7 0 1000 1000 0
    rtgen md5 loweralpha 1 7 0 -bench
```

```
┌──(root@10.15.85.23)-[~]
└─# rtgen md5 loweralpha 1 5 0 16000 16000 0
rainbow table md5 loweralpha#1-5 0 16000x16000 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 - 5
reduce offset:          0x00000000
plaintext total:        12356630

sequential starting point begin from 0 (0x0000000000000000)
generating...
16000 of 16000 rainbow chains generated (0 m 35.8 s)
```

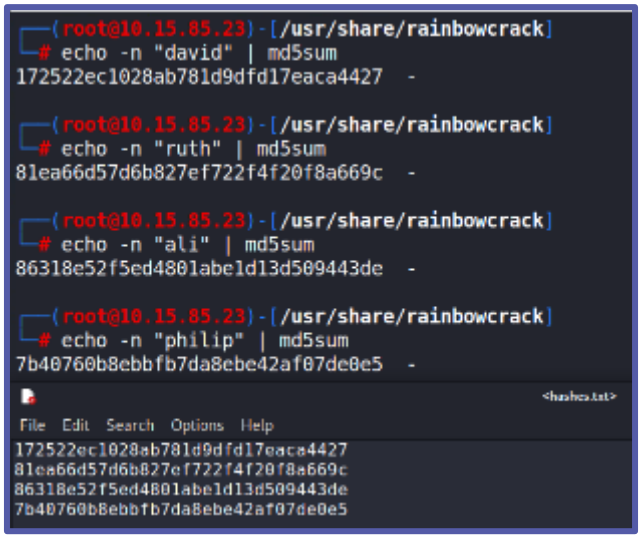
*Rainbow Crack is already installed on the CYBER.ORG Range
Use the following command to install (if not installed):

`sudo apt-get install rainbowcrack`

CYBER.ORG

Create Hashes

- Navigate to the folder with Rainbowcrack
  `cd /usr/share/rainbowcrack`
- Create a sample hash for a 5 character lowercase input by using the following command:
  `echo -n "david" | md5sum`
- Repeat this process three more times for other inputs.
- Create a new file called "*hashes.txt*" in a text editor
  `leafpad hashes.txt`
- Copy and paste each output into the "*hashes.txt*" file
- Save each hash on a new line



Crack a Hash using Rainbowcrack

- First run the following command to sort all *.rt* tables in the current directory to make binary search possible

  `rtsort .`

- Copy the MD5 hash output from the previous command:

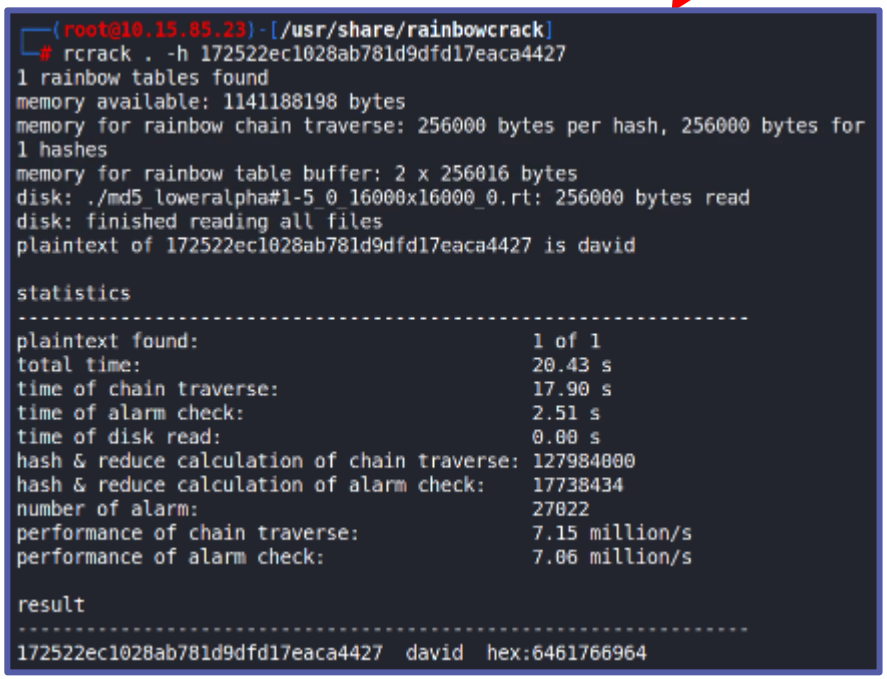
  `echo -n "`*`<name>`*`" | md5sum`

- Crack the hash using the command

  `rcrack . -h` *`<MD5 hash>`*

- Observe the output with the plaintext answer shown for the matching hash

Hash from "david"



Crack a file of hashes using Rainbowcrack

- Crack multiple hashes at once stored in a file using the command:

  `rcrack . -l` ***`<filename>`***

```
┌──(root@10.15.85.23)-[/usr/share/rainbowcrack]
└─# rcrack . -l hashes.txt
1 rainbow tables found
memory available: 1140827750 bytes
memory for rainbow chain traverse: 256000 bytes per hash, 1024000 bytes for
 4 hashes
memory for rainbow table buffer: 2 x 256016 bytes
disk: ./md5_loweralpha#1-5_0_16000x16000_0.rt: 256000 bytes read
disk: finished reading all files
plaintext of 86318e52f5ed4801abe1d13d509443de is ali
plaintext of 81ea66d57d6b827ef722f4f20f8a669c is ruth
plaintext of 172522ec1028ab781d9dfd17eaca4427 is david

statistics
----------------------------------------------------------------
plaintext found:                                3 of 4
total time:                                     239.35 s
time of chain traverse:                         81.87 s
time of alarm check:                            157.29 s
time of disk read:                              0.00 s
hash & reduce calculation of chain traverse:    511936000
hash & reduce calculation of alarm check:       945326335
number of alarm:                                222139
performance of chain traverse:                  6.25 million/s
performance of alarm check:                     6.01 million/s

result
----------------------------------------------------------------
172522ec1028ab781d9dfd17eaca4427  david  hex:6461766964
81ea66d57d6b827ef722f4f20f8a669c  ruth  hex:72757468
86318e52f5ed4801abe1d13d509443de  ali  hex:616c69
7b40760b8ebbfb7da8ebe42af07de0e5  <not found>  hex:<not found>
```

Observe the Results

- The rainbow table created, solved 3 out of 4 hashes
- The one plaintext it did not find was for "*philip*" which is more than 5 characters
- If the word is between 1-5 characters in length, the table can solve ~100% of the password
- The more rainbow tables we generate, and the longer they are, the more possibilities to crack the password – however long tables require a LOT of space!

```
┌──(root@10.15.85.23)-[/usr/share/rainbowcrack]
└─# rcrack . -l hashes.txt
1 rainbow tables found
memory available: 1140827750 bytes
memory for rainbow chain traverse: 256000 bytes per hash, 1024000 bytes for
 4 hashes
memory for rainbow table buffer: 2 x 256016 bytes
disk: ./md5_loweralpha#1-5_0_16000x16000_0.rt: 256000 bytes read
disk: finished reading all files
plaintext of 86318e52f5ed4801abe1d13d509443de is ali
plaintext of 81ea66d57d6b827ef722f4f20f8a669c is ruth
plaintext of 172522ec1028ab781d9dfd17eaca4427 is david

statistics
----------------------------------------------------------------
plaintext found:                        3 of 4
total time:                             239.35 s
time of chain traverse:                 81.87 s
time of alarm check:                    157.29 s
time of disk read:                      0.00 s
hash & reduce calculation of chain traverse: 511936000
hash & reduce calculation of alarm check:    945326335
number of alarm:                        222139
performance of chain traverse:          6.25 million/s
performance of alarm check:             6.01 million/s

result
----------------------------------------------------------------
172522ec1028ab781d9dfd17eaca4427  david  hex:6461766964
81ea66d57d6b827ef722f4f20f8a669c  ruth  hex:72757468
86318e52f5ed4801abe1d13d509443de  ali  hex:616c69
7b40760b8ebbfb7da8ebe42af07de0e5  <not found>  hex:<not found>
```

How to Defend against Rainbow Table Attacks

- Salt those passwords!
  - A salt is string of characters added to a password before it is hashed
  - Using a unique salt for each user makes using a rainbow table more difficult.
    - The rainbow table has to be recomputed for each user.
    - If a password is found, which part is the hash and which is the password?
- Key Stretching
  - “Hashing the hash”
  - Hashed values are hashed multiple times to increase the computation time required to hash each password
- How else can you defend against Rainbow Tables?