

## Lab 6.6.4 Password Cracking using Rainbow Tables

*From TestOut CompTIA Security+ Course*

In this lab I will be analyzing a network interface to ascertain if a SYN Flood attack is occurring on “my network” as a hypothetical corporate IT Admin. I will be using Wireshark to listen in on packets and deduce a conclusion.

### **The scenario for this lab is as follows:**

“A recent breach of a popular third-party service has exposed a password database. The security team is evaluating the risk of the exposed passwords for the company. The password hashes are saved in the root user’s home directory, /root/captured\_hashes.txt. You want to attempt to hack these passwords using a rainbow table. The password requirements for your company are as follows:

- The password must be 12 or more characters in length.
- The password must include at least one uppercase and one lowercase letter.
- The password must have at least one of these special characters: !, ", #, \$, %, &, \_, ', \*, or @.
- All passwords are encrypted using a hash algorithm of either md5 or sha1.

In this lab, your task is to:

- Create md5 and sha1 rainbow tables using rtgen.
- Sort the rainbow tables using the rtsort command.
- Crack the hashes using the rcrack command. You can run rcrack on an individual hash or the hash file (/root/captured\_hashes.txt).
- Answer the questions.

The type of charset that can be used to create a rainbow table is stored in the /usr/share/rainbowcrack/charset.txt file. This file can be viewed using the cat command. “

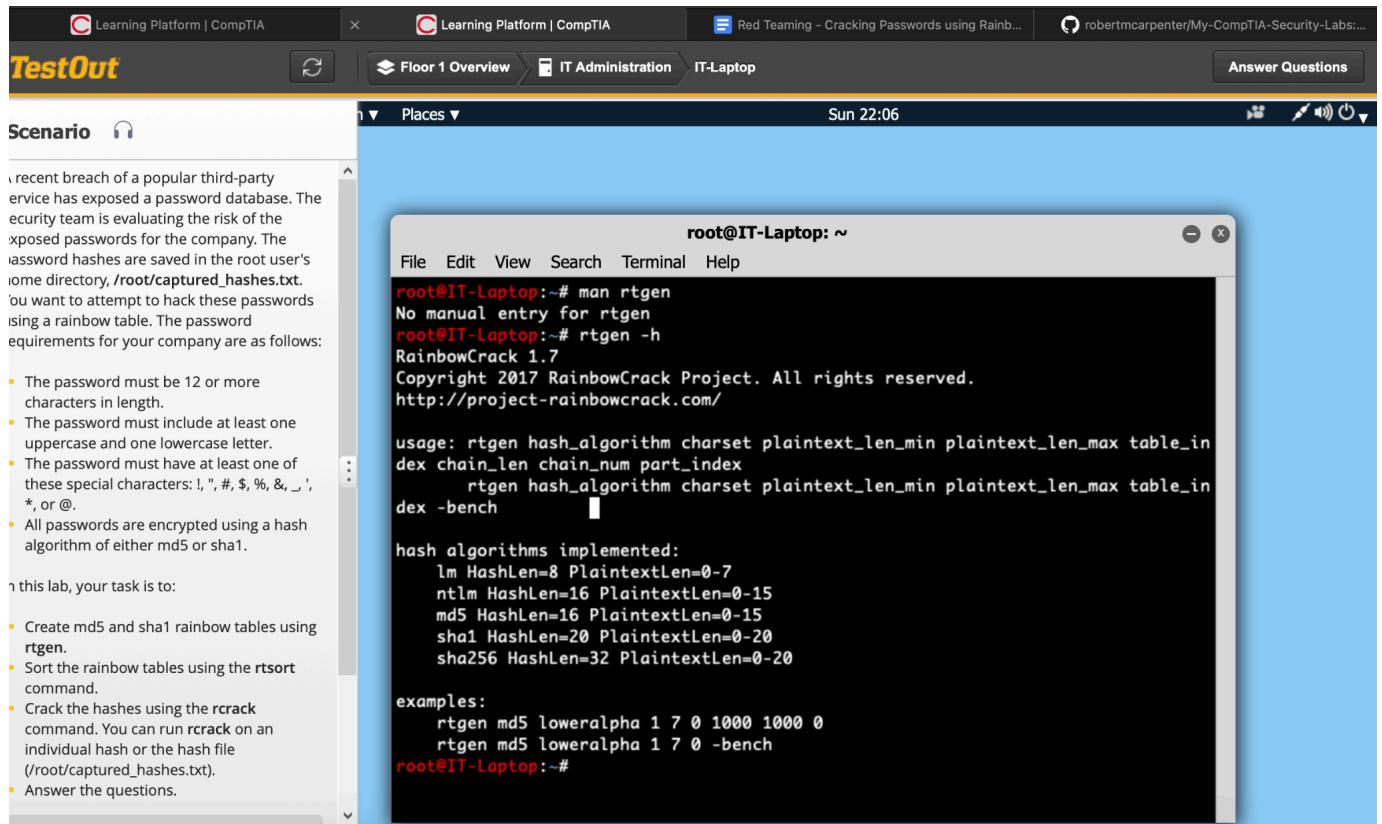
Since we are working with Rainbow Tables we will need to generate some hashes. A Rainbow Table by definition is a table with varying hashes according to certain character sets.

Our constraints in this lab is that the passwords are:

- The password must be 12 or more characters in length.
- The password must include at least one uppercase and one lowercase letter.
- The password must have at least one of these special characters: !, ", #, \$, %, &, \_, ', \*, or @.
- All passwords are encrypted using a hash algorithm of either md5 or sha1.

Based off these constraints I will create a Rainbow Table around them.

The binary to do this on my Kali Linux box is **rtgen**. To see what this command can do I will issue the command: **rtgen -h**



```
root@IT-Laptop: ~
File Edit View Search Terminal Help
root@IT-Laptop:~# man rtgen
No manual entry for rtgen
root@IT-Laptop:~# rtgen -h
RainbowCrack 1.7
Copyright 2017 RainbowCrack Project. All rights reserved.
http://project-rainbowcrack.com/

usage: rtgen hash_algorithm charset plaintext_len_min plaintext_len_max table_index chain_len chain_num part_index
      rtgen hash_algorithm charset plaintext_len_min plaintext_len_max table_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@IT-Laptop:~#
```

As we can see , the syntax for inputting the **rtgen** command is listed above:

**Rtgen [hash\_algorithm] [charset] [plaintext\_len\_min] [plaintext\_len\_max] [table\_index] [chain\_len] ...**

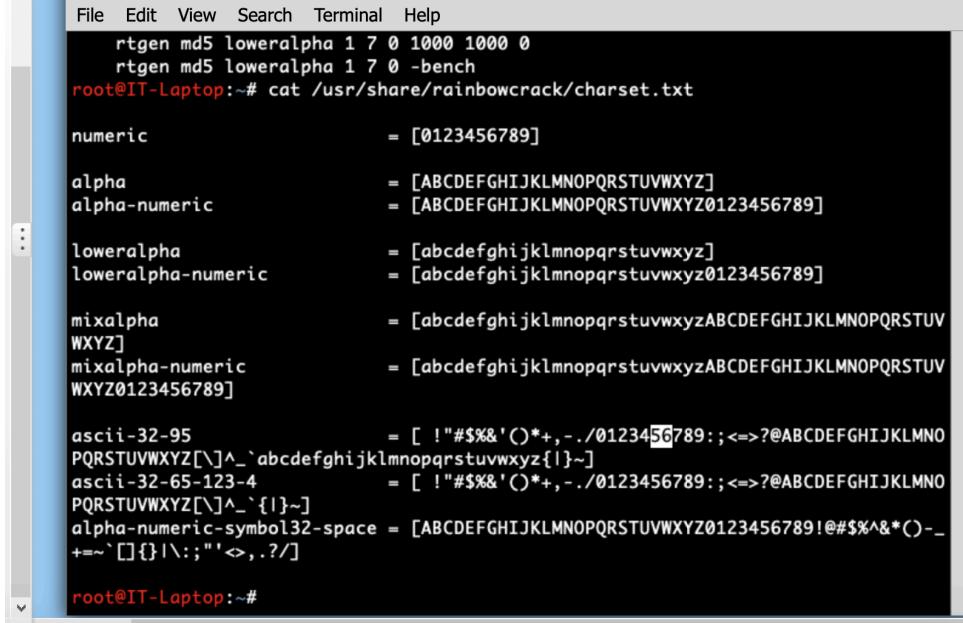
- The password must be 12 or more characters in length.
- The password must include at least one uppercase and one lowercase letter.
- The password must have at least one of these special characters: !, ", #, \$, %, &, \_, ', \*, or @.
- All passwords are encrypted using a hash algorithm of either md5 or sha1.

I repasted the constraints again for quick reference. Looking at them in order I can tell that:

- **[plaintext\_len\_min]** is **12**: Since minimum password length is 12
- **[charset]** is ascii-32-95: Because it's A-z , a-z, and special characters as well. The great thing about ASCII is that we can manually specify these

Sat December 14th 2024

special characters using this standard. (See Ascii Table below). The Lab also gives us a hint that the available character sets are in  
`/usr/share/rainbowcrack/charset.txt`



```
root@IT-Laptop: ~
File Edit View Search Terminal Help
rtgen md5 loweralpha 1 7 0 1000 1000 0
rtgen md5 loweralpha 1 7 0 -bench
root@IT-Laptop:~# cat /usr/share/rainbowcrack/charset.txt

numeric = [0123456789]

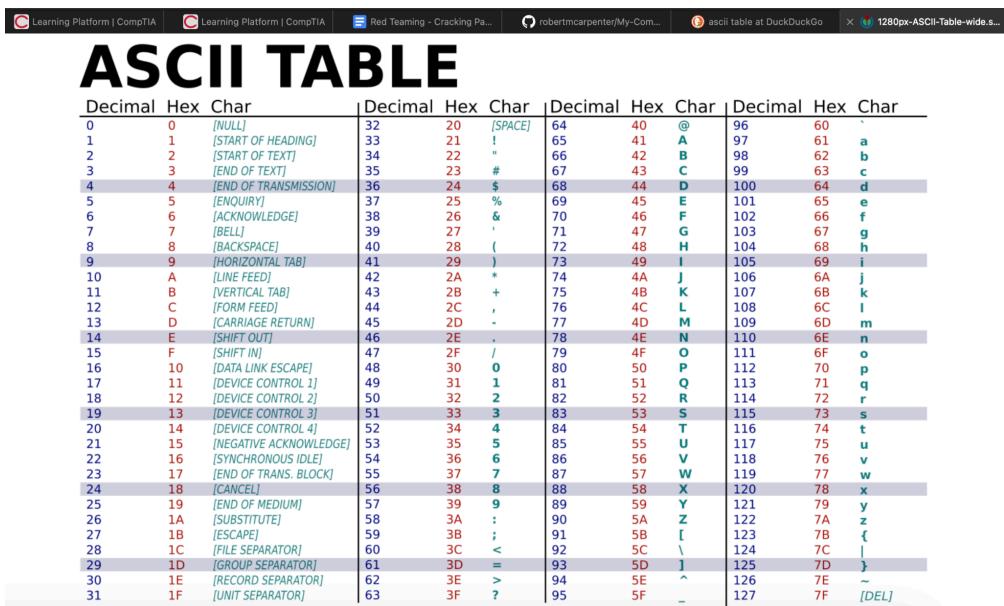
alpha = [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
alpha-numeric = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]

loweralpha = [abcdefghijklmnopqrstuvwxyz]
loweralpha-numeric = [abcdefghijklmnopqrstuvwxyz0123456789]

mixalpha = [abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]
mixalpha-numeric = [abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]

ascii-32-95 = [ !"#$%&'()*,.-./0123456789:;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[{}]{|^~}={=~`[]{}|\\";""<,<?.?/]]

root@IT-Laptop:~#
```



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	-	127	7F	[DEL]

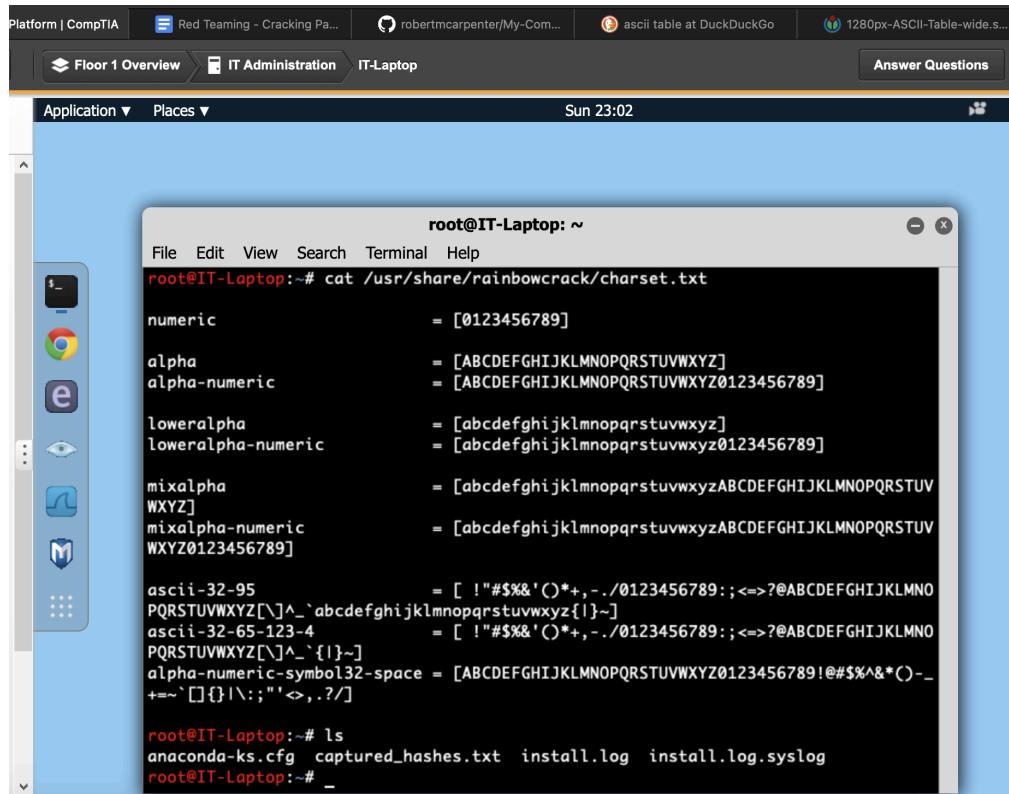
Sat December 14th 2024

- [algorithm] is SHA1 and md5 : Because we need to generate 2 Rainbow Tables, one sha1 and md5 per the lab's instructions.

To generate a sha1 rainbow table I will use:

```
rtgen sha1 ascii-32-95 1 20 0 1000 1000 0
```

Before, I will verify that I'm in the correct directory I want to be, which is the same directory as the captured hashes from the data breach.



```
root@IT-Laptop:~# cat /usr/share/rainbowcrack/charset.txt
numeric = [0123456789]
alpha = [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
alpha-numeric = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]
loweralpha = [abcdefghijklmnopqrstuvwxyz]
loweralpha-numeric = [abcdefghijklmnopqrstuvwxyz0123456789]
mixalpha = [abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]
mixalpha-numeric = [abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]
ascii-32-95 = [ !#$%&'()*+,-./@^_`{|}~-]
PQRSTUVWXYZ[\]^_`{[]~-}
ascii-32-65-123-4 = [ !#$%&'()*+,-./@^_`{|}~-]
PQRSTUVWXYZ[\]^_`{[]~-}
alpha-numeric-symbol32-space = [ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!@#$%^&*()_-+=~`{|}!`;"'<,.?/]

root@IT-Laptop:~# ls
anaconda-ks.cfg captured_hashes.txt install.log install.log.syslog
root@IT-Laptop:~#
```

Indeed I am! I will generate the sha1 and md5 tables with these two commands:

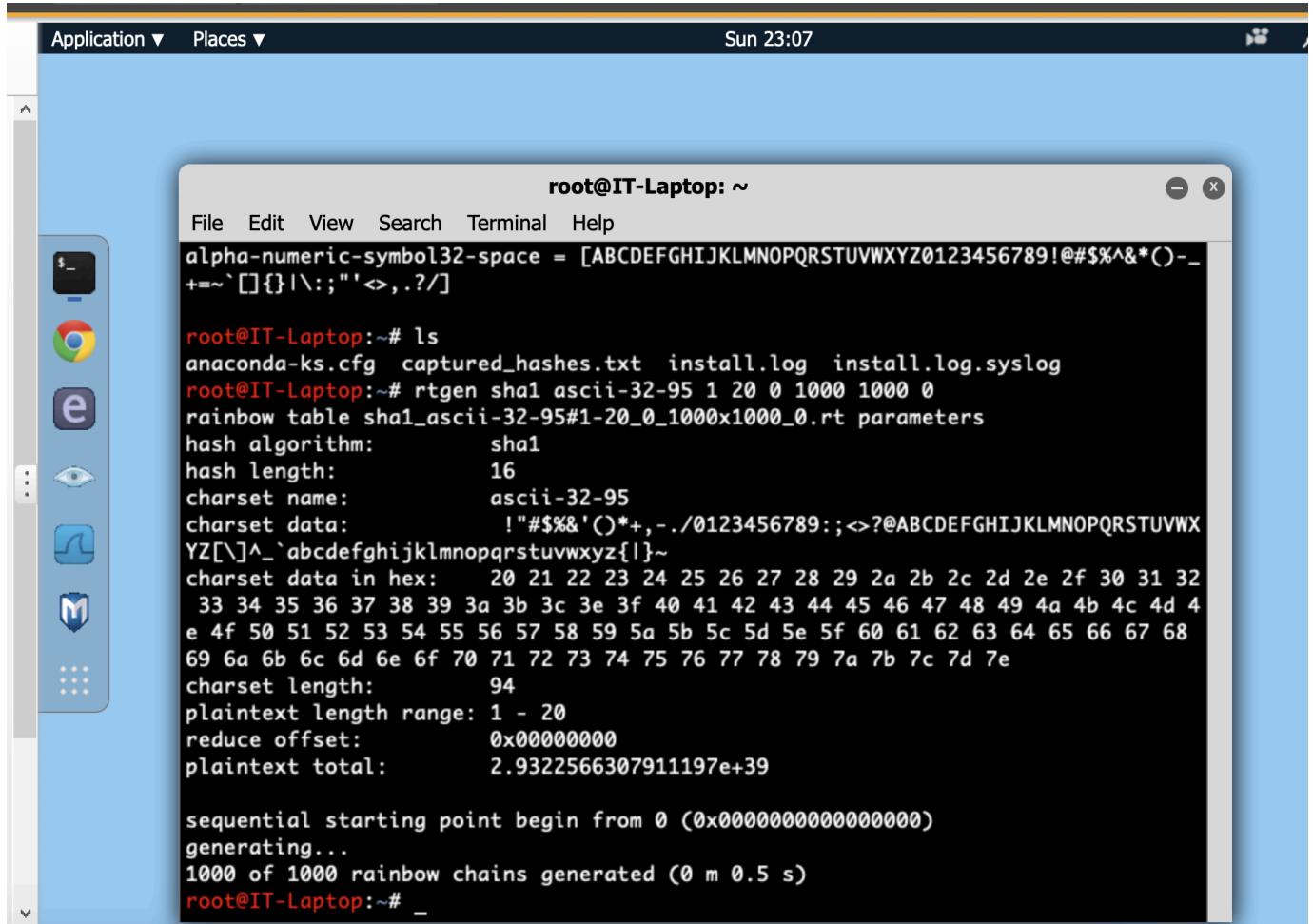
```
rtgen sha1 ascii-32-95 1 20 0 1000 1000 0
```

```
rtgen md5 ascii-32-95 1 20 0 1000 1000 0
```

Robert Carpenter

[github.com/robertmcarpenter](https://github.com/robertmcarpenter)

Sat December 14th 2024



The screenshot shows a terminal window titled "root@IT-Laptop: ~". The terminal displays the following command and its output:

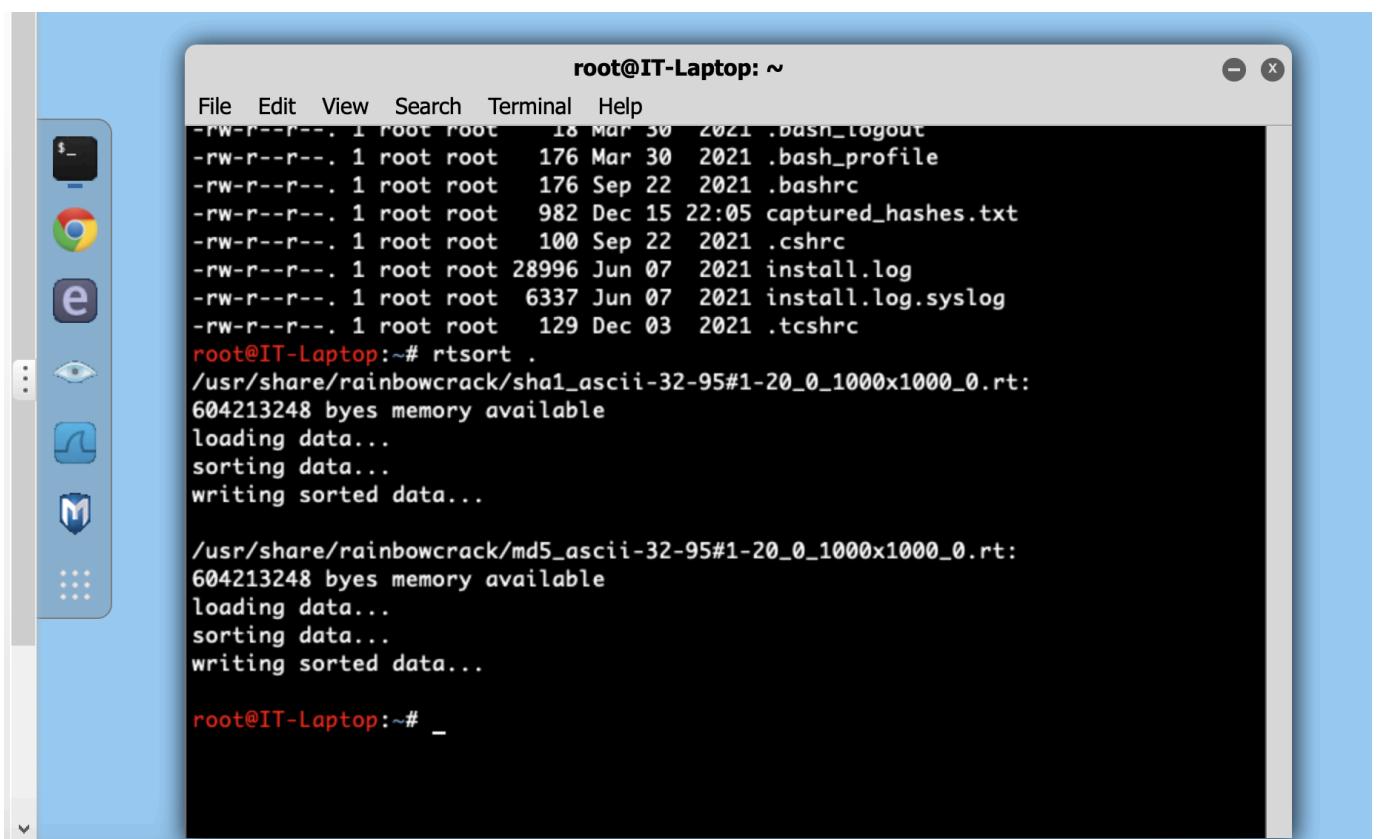
```
alpha-numeric-symbol32-space = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()_-+=~`[]{}|\:;'"<>,.?/] 

root@IT-Laptop:~# ls
anaconda-ks.cfg  captured_hashes.txt  install.log  install.log.syslog
root@IT-Laptop:~# rtgen sha1 ascii-32-95 1 20 0 1000 1000 0
rainbow table sha1_ascii-32-95#1-20_0_1000x1000_0.rt parameters
hash algorithm:      sha1
hash length:        16
charset name:       ascii-32-95
charset data:        !"#$%&'()*+,.-./0123456789:;,<>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{!}~
charset data in hex:   20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32
                      33 34 35 36 37 38 39 3a 3b 3c 3e 3f 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4
                      e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 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 7b 7c 7d 7e
charset length:      94
plaintext length range: 1 - 20
reduce offset:        0x00000000
plaintext total:      2.9322566307911197e+39

sequential starting point begin from 0 (0x0000000000000000)
generating...
1000 of 1000 rainbow chains generated (0 m 0.5 s)
root@IT-Laptop:~# _
```

Success! I've generated the sha1 table. Now let's generate the md5 table.

After generating both tables I need sort them. The default directory/path to rainbow tables is **/usr/share/rainbowcrack**. In order to sort the table I need to use the **rtsort** binary as part of the rainbowcrack package.



The screenshot shows a terminal window titled "root@IT-Laptop: ~". The terminal displays the following command and its execution:

```
root@IT-Laptop:~# rtsort .
/usr/share/rainbowcrack/sha1_ascii-32-95#1-20_0_1000x1000_0.rt:
604213248 bytes memory available
loading data...
sorting data...
writing sorted data...

/usr/share/rainbowcrack/md5_ascii-32-95#1-20_0_1000x1000_0.rt:
604213248 bytes memory available
loading data...
sorting data...
writing sorted data...

root@IT-Laptop:~# _
```

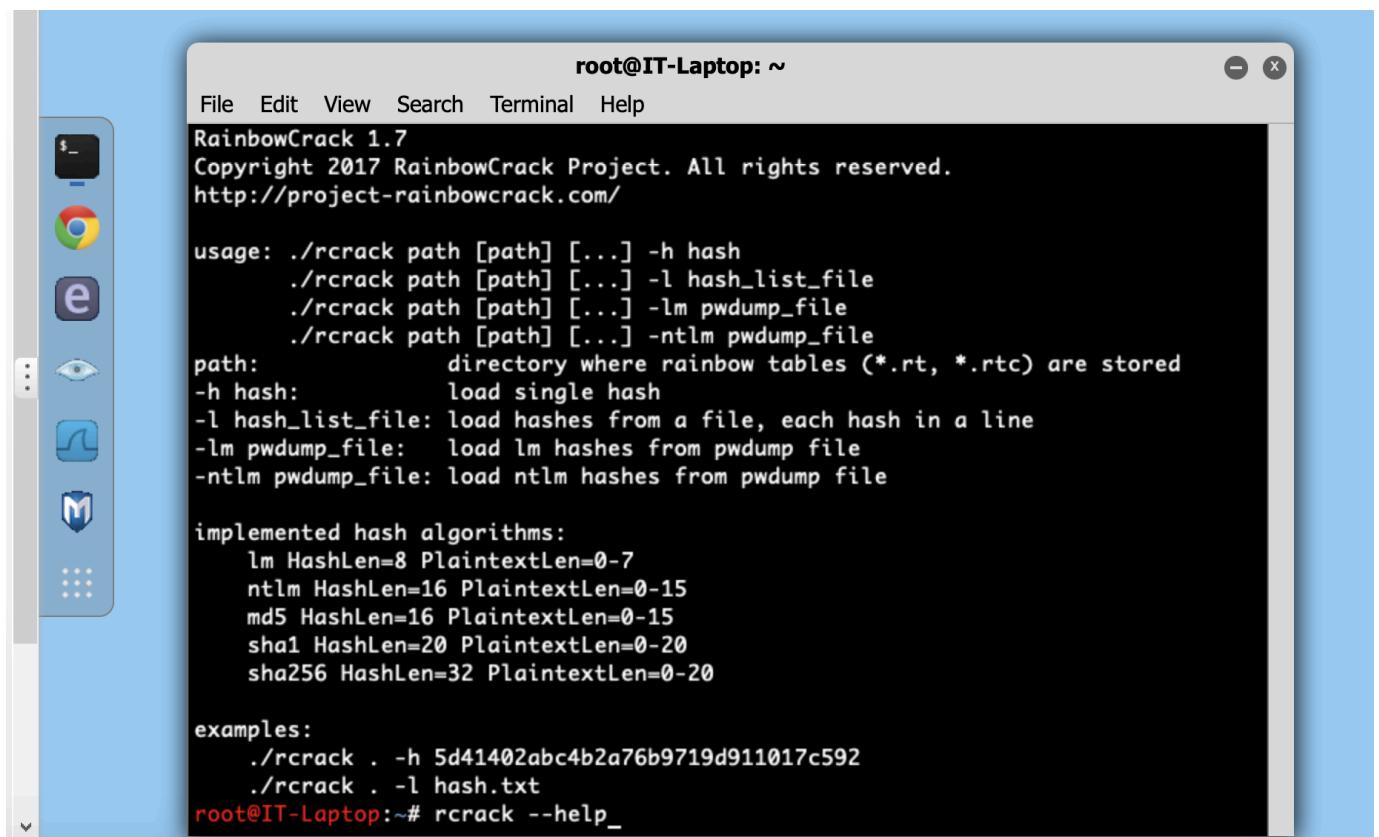
The terminal window is part of a desktop environment, as evidenced by the application dock on the left containing icons for a file manager, browser, terminal, eye, and other utilities.

Notice how **rtsort** has sorted both the md5 and sha1 tables. To execute the actual crack using the rainbow tables I need to use **the binary “rcrack -l [path/to/hashfile]”** To check the options and flags for the **rcrack** command I can use **rcrack –help**.

Robert Carpenter

[github.com/robertmcarpenter](https://github.com/robertmcarpenter)

Sat December 14th 2024



The screenshot shows a terminal window titled "root@IT-Laptop: ~". The window contains the help output for the RainbowCrack 1.7 tool. The text is as follows:

```
RainbowCrack 1.7
Copyright 2017 RainbowCrack Project. All rights reserved.
http://project-rainbowcrack.com/

usage: ./rcrack path [...] -h hash
       ./rcrack path [...] -l hash_list_file
       ./rcrack path [...] -lm pwdump_file
       ./rcrack path [...] -ntlm pwdump_file
path:          directory where rainbow tables (*.rt, *.rtc) are stored
-h hash:       load single hash
-l hash_list_file: load hashes from a file, each hash in a line
-lm pwdump_file:  load lm hashes from pwdump file
-ntlm pwdump_file: load ntlm hashes from pwdump file

implemented hash algorithms:
    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

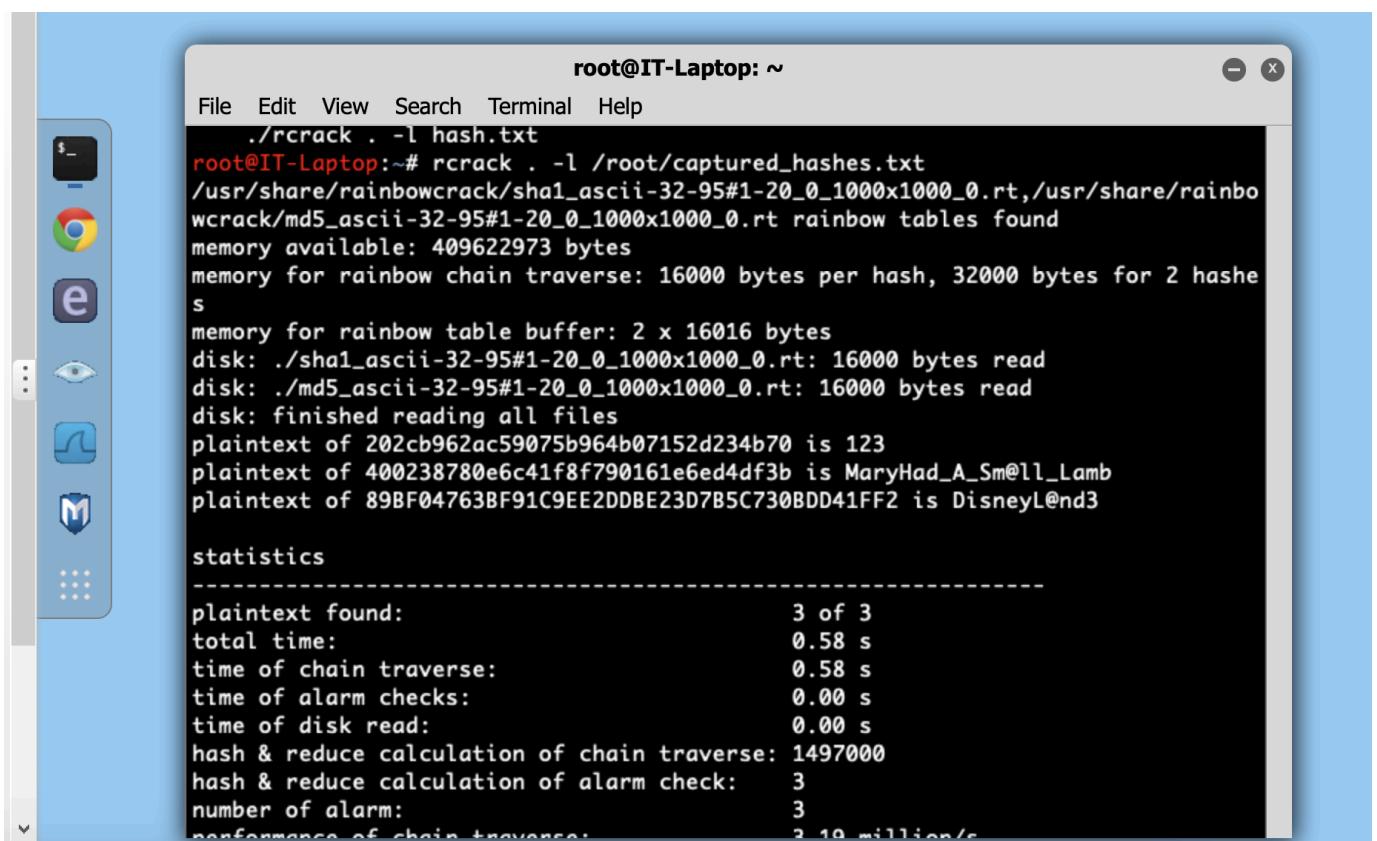
examples:
    ./rcrack . -h 5d41402abc4b2a76b9719d911017c592
    ./rcrack . -l hash.txt
root@IT-Laptop:~# rcrack --help_
```

I'll need to use `-l` which is a hash list file. I can feed in the `captured_hashes.txt` to the command to use the Rainbow tables on them.

To crack the hashes I will issue:

```
rcrack . -l /root/captured_hashes.txt
```

Robert Carpenter  
[github.com/robertmcarpenter](https://github.com/robertmcarpenter)  
Sat December 14th 2024



```
root@IT-Laptop: ~
File Edit View Search Terminal Help
./rcrack . -l hash.txt
root@IT-Laptop:~# rcrack . -l /root/captured_hashes.txt
/usr/share/rainbowcrack/sha1_ascii-32-95#1-20_0_1000x1000_0.rt,/usr/share/rainbo
wcrack/md5_ascii-32-95#1-20_0_1000x1000_0.rt rainbow tables found
memory available: 409622973 bytes
memory for rainbow chain traverse: 16000 bytes per hash, 32000 bytes for 2 hashe
s
memory for rainbow table buffer: 2 x 16016 bytes
disk: ./sha1_ascii-32-95#1-20_0_1000x1000_0.rt: 16000 bytes read
disk: ./md5_ascii-32-95#1-20_0_1000x1000_0.rt: 16000 bytes read
disk: finished reading all files
plaintext of 202cb962ac59075b964b07152d234b70 is 123
plaintext of 400238780e6c41f8f790161e6ed4df3b is MaryHad_A_Sm@ll_Lamb
plaintext of 89BF04763BF91C9EE2DBE23D7B5C730BDD41FF2 is DisneyL@nd3

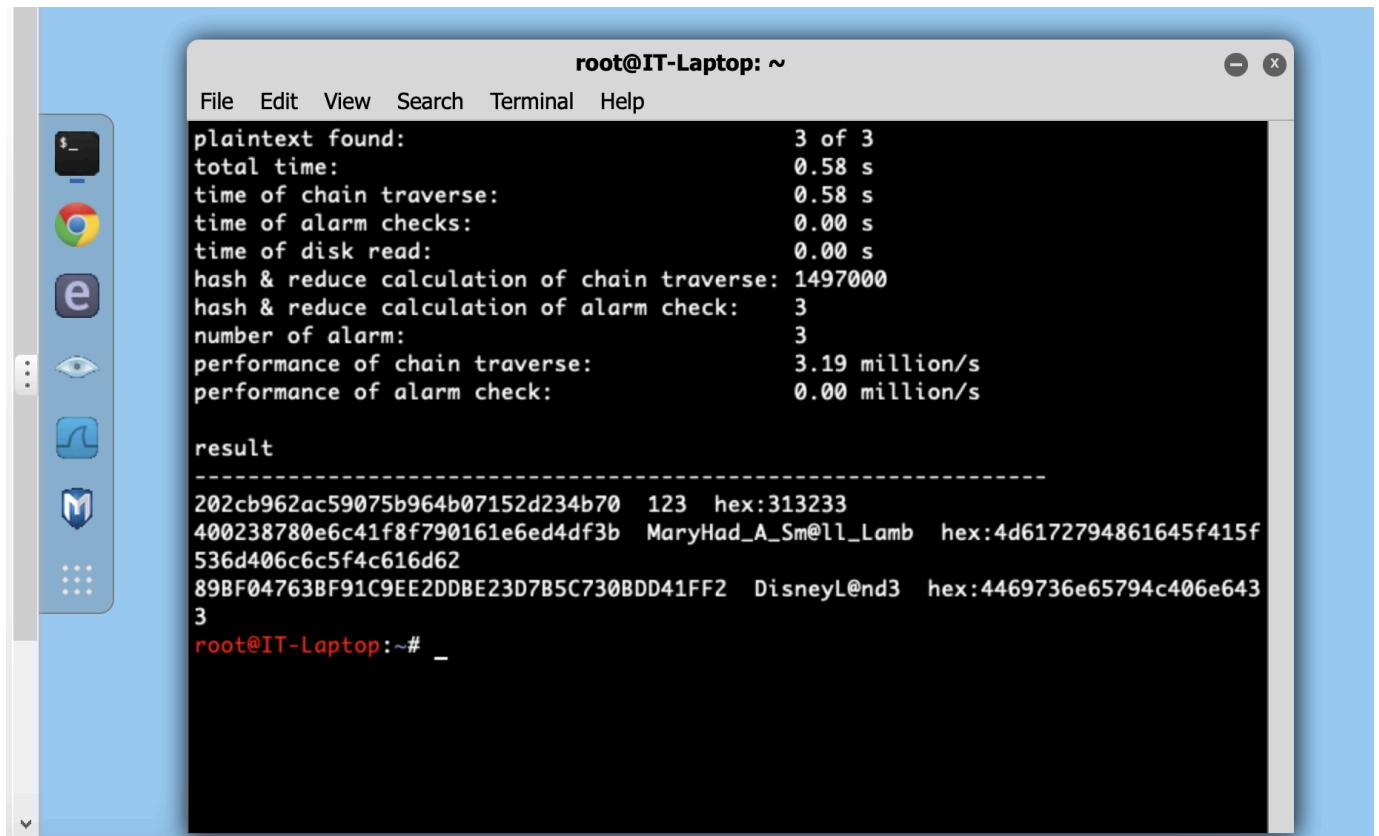
statistics
-----
plaintext found: 3 of 3
total time: 0.58 s
time of chain traverse: 0.58 s
time of alarm checks: 0.00 s
time of disk read: 0.00 s
hash & reduce calculation of chain traverse: 1497000
hash & reduce calculation of alarm check: 3
number of alarm: 3
performance of chain traverse: 3.19 million/s
```

Looks like we've found some matches! I'll screenshot the second half of this output because it's quite a lot.

Robert Carpenter

[github.com/robertmcarpenter](https://github.com/robertmcarpenter)

Sat December 14th 2024



The screenshot shows a terminal window titled "root@IT-Laptop: ~". The window contains the following output:

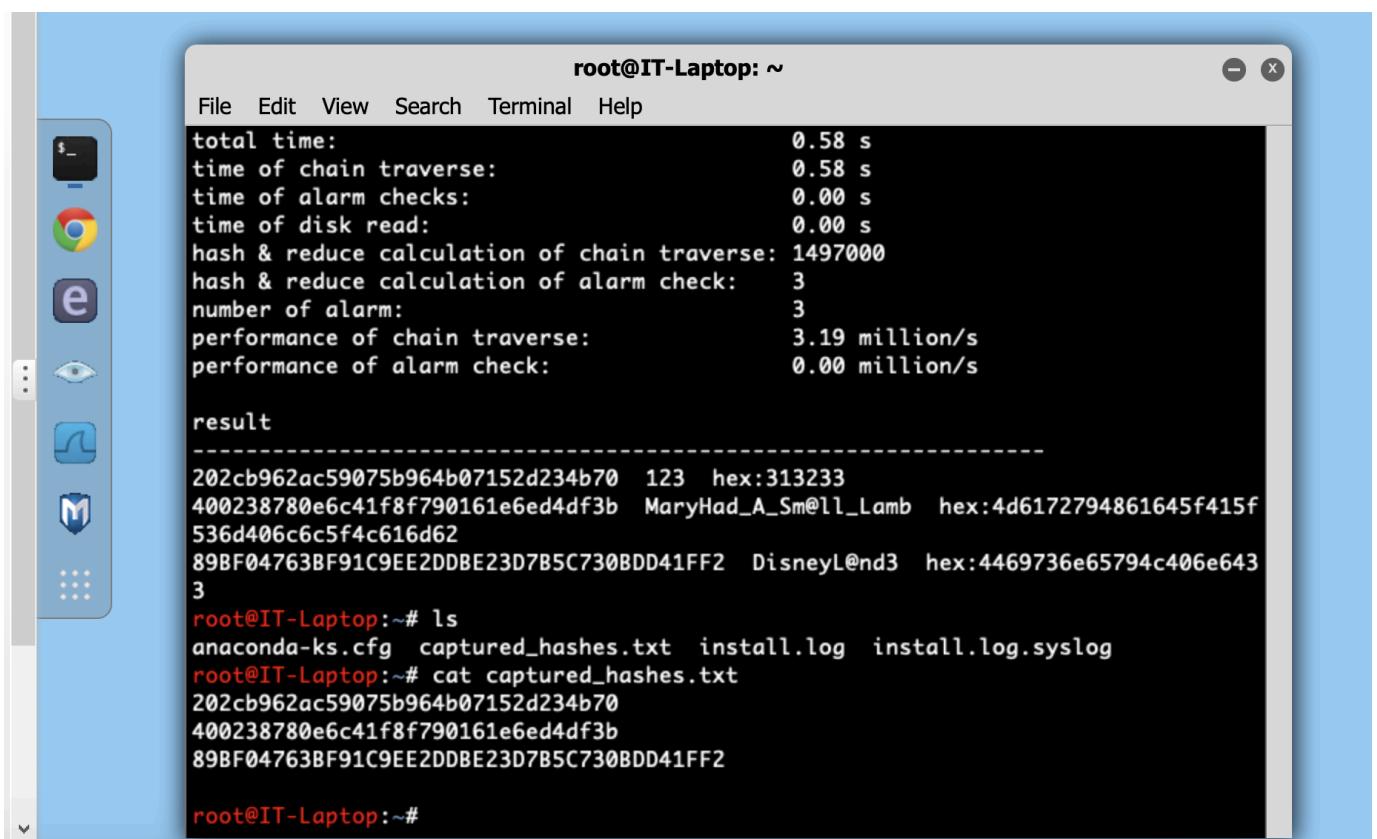
```
root@IT-Laptop: ~
File Edit View Search Terminal Help
plaintext found: 3 of 3
total time: 0.58 s
time of chain traverse: 0.58 s
time of alarm checks: 0.00 s
time of disk read: 0.00 s
hash & reduce calculation of chain traverse: 1497000
hash & reduce calculation of alarm check: 3
number of alarm: 3
performance of chain traverse: 3.19 million/s
performance of alarm check: 0.00 million/s

result
-----
202cb962ac59075b964b07152d234b70 123 hex:313233
400238780e6c41f8f790161e6ed4df3b MaryHad_A_Sm@ll_Lamb hex:4d6172794861645f415f
536d406c6c5f4c616d62
89BF04763BF91C9EE2DDBE23D7B5C730BDD41FF2 DisneyL@nd3 hex:4469736e65794c406e643
3
root@IT-Laptop:~# _
```

The terminal window has a light blue background and a dark blue header bar. On the left side, there is a vertical dock with several icons: a file icon, a browser icon, an "e" icon, an eye icon, a gear icon, and a "M" icon.

**BINGO!!!!** We've successfully cracked the hashes. Let's verify that we've cracked all the hashes by printing out the `captured_hashes.txt`

Robert Carpenter  
[github.com/robertmcarpenter](https://github.com/robertmcarpenter)  
Sat December 14th 2024



root@IT-Laptop: ~

```
total time: 0.58 s
time of chain traverse: 0.58 s
time of alarm checks: 0.00 s
time of disk read: 0.00 s
hash & reduce calculation of chain traverse: 1497000
hash & reduce calculation of alarm check: 3
number of alarm: 3
performance of chain traverse: 3.19 million/s
performance of alarm check: 0.00 million/s

result
-----
202cb962ac59075b964b07152d234b70 123 hex:313233
400238780e6c41f8f790161e6ed4df3b MaryHad_A_Sm@ll_Lamb hex:4d6172794861645f415f
536d406c6c5f4c616d62
89BF04763BF91C9EE2DBBE23D7B5C730BDD41FF2 DisneyL@nd3 hex:4469736e65794c406e643
3

root@IT-Laptop:~# ls
anaconda-ks.cfg captured_hashes.txt install.log install.log.syslog
root@IT-Laptop:~# cat captured_hashes.txt
202cb962ac59075b964b07152d234b70
400238780e6c41f8f790161e6ed4df3b
89BF04763BF91C9EE2DBBE23D7B5C730BDD41FF2

root@IT-Laptop:~#
```

We indeed cracked all of them. Let's answer the questions now:

Robert Carpenter

[github.com/robertmcarpenter](https://github.com/robertmcarpenter)

Sat December 14th 2024

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window has a title bar 'Floor 1 Overview' and 'IT Adminstr'. The main area of the terminal displays a password cracking session. It shows the following text:

```
File Edit View S
plaintext of 202
plaintext of 400
plaintext of 89B
statistics
-----
plaintext found:
total time:
time of chain tr
time of alarm ch
time of disk rea
hash & reduce co
hash & reduce co
number of alarm:
performance of c
performance of a
result
-----
202cb962ac59075b
400238780e6c41f8
536d406c6c5f4c61
89BF04763BF91C9E
3
root@IT-Laptop:~#
```

To the right of the terminal, there is a sidebar titled 'Lab Questions' with a refresh icon. It contains four questions:

1. What is the password for hash 202cb962ac59075b964b07152d234b70?  
Answer: 123
2. What is the password for hash 400238780e6c41f8f790161e6ed4df3b?  
Answer: MaryHad\_A\_Sm@ll\_Lamb
3. What is the password for hash 89BF04763BF91C9EE2DDBE23D7B5C730BDD41FF2?  
Answer: DisneyL@nd3
4. How many of the passwords found meet the company's password requirements?  
Options:
  - 0
  - 1
  - 2
  - 3

Only one of the passwords matches the company's password policy. This now concludes this lab.

Robert Carpenter

[github.com/robertmcarpenter](https://github.com/robertmcarpenter)

Sat December 14th 2024

The screenshot shows a cybersecurity lab interface. A modal window is open, displaying a question and the user's answer. The question is: "2. What is the password for hash 100000700...?". The user's answer is "123". Below the question, there is a "Print" icon and a close "X" button. The main interface shows a "Lab Report" section with a progress bar indicating 7/7 (100%) completed. The report also includes a "Time Spent: 08:04" and a "Score: 7/7 (100%)". The "TASK SUMMARY" section lists required actions: "Create rainbow tables" (with a "Show Details" link), "Sort the rainbow tables using rtsort", and "Crack the hash using rcrack . -l or rcrack . -h". It also includes a question: "Q1: What is the password for hash 202cb962ac59075b964b07152d234b70?". The user's answer is "123", and the correct answer is also "123". A "Score Lab" button is visible in the bottom right corner. At the bottom, a terminal window shows the command "root@IT-Laptop:~# \_".