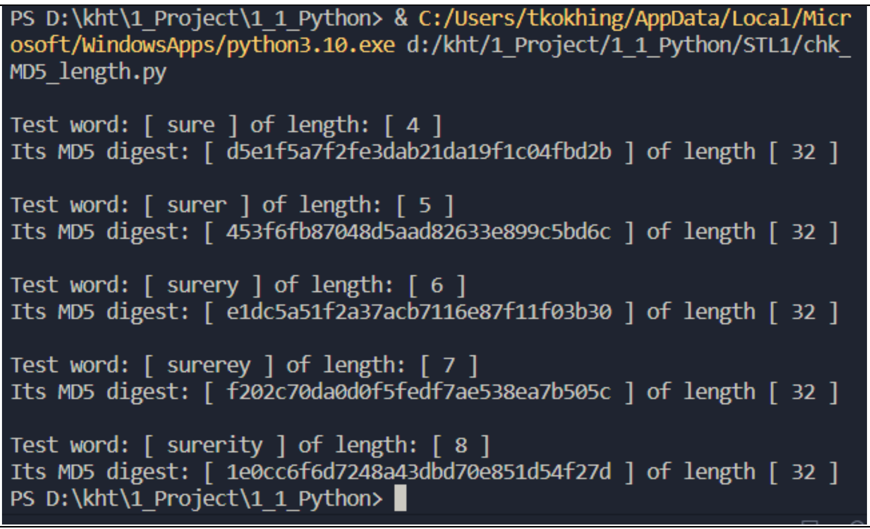
51.506 Security Tools Lab 1			
ASSIGNMENT 1 - PASSWORD AND HASHING			
		tkokhing	: 10***92

1. Hashing password using MD5

a) Using words of length 4 to 8, the respective hashed outputs the **same length of 32** as shown below. **# Filename: chk_MD5_length.py**



```
PS D:\kht\1_Project\1_1_Python> & C:/Users/tkokhing/AppData/Local/Micr
osoft/WindowsApps/python3.10.exe d:/kht/1_Project/1_1_Python/STL1/chk_
MD5_length.py

Test word: [ sure ] of length: [ 4 ]
Its MD5 digest: [ d5e1f5a7f2fe3dab21da19f1c04fbd2b ] of length [ 32 ]

Test word: [ surer ] of length: [ 5 ]
Its MD5 digest: [ 453f6fb87048d5aad82633e899c5bd6c ] of length [ 32 ]

Test word: [ surery ] of length: [ 6 ]
Its MD5 digest: [ e1dc5a51f2a37acb7116e87f11f03b30 ] of length [ 32 ]

Test word: [ surerey ] of length: [ 7 ]
Its MD5 digest: [ f202c70da0d0f5fedf7ae538ea7b505c ] of length [ 32 ]

Test word: [ surerity ] of length: [ 8 ]
Its MD5 digest: [ 1e0cc6f6d7248a43dbd70e851d54f27d ] of length [ 32 ]
PS D:\kht\1_Project\1_1_Python> █
```

2. BRUTEFORCE VS DICTIONARY ATTACK

a) Dictionary Attack. Based on the provided Dictionary word5.txt, 6 out of 15 MD5 digests in the 27.txt, were found in 1.04 sec. **# Filename: output.txt**

```
Words found using provided Dictionary
Hash: 61bec295a8e120781f6a8a548f463e2e -> password: omtoa
Hash: 51bc5c6e1535b8b0a426f79ce4a446a9 -> password: cbsan
Hash: c796d948a14fd11223c18ae53c96f5f6 -> password: omype
Hash: 3d380d3fb519934f1cbf8315efb460f6 -> password: daich
Hash: c99b7512b9f54cfcaf35dc3a5451f00c -> password: akesq
Hash: ab302e86e62dc30aee5e48f666d5afbf -> password: acyce
Time taken is: 1.0477560999970592 for [6] out of 15 found.
```

b) Bruteforce Attack (lowercase alphabet only). My python scripted was written such that it filtered out those hashes that were not cracked by dictionary attack to be sent for bruteforce attack., i.e. on the remaining 9 hashes, as there is no meaning to re-run those hashes that were already crack via dictionary attack. This also saves that computation time. 4 more MD5 digests in the 27.txt, were found in 27.80 sec. **# Filename: output.txt (appended)**

```
New Words found using Bruteforce with lowercase
Hash: 61d50ae2cec24a824eade9d8cbd53930 -> password: aorcw
Hash: fe63521f3357e43e75eafb990f3dd932 -> password: dtomn
Hash: 905c0b375c43e93b7f3b4be831166b62 -> password: lrbye
Hash: 0d4f747734acf9363740c494df90716f -> password: mscye
Time taken is: 27.80564080000113 for [4] out of 15 found.
```

c) Bruteforce Attack (lowercase alphabet and digits). Using the same logic as above, only 5 hashes were left and they were successfully cracked in 126.45 sec. **# Filename: output.txt (appended)**

```
New Words found using Bruteforce with lowercase and digits
Hash: 669141e1b9639f5e793aaab1e69da0c4 -> password: alho0
Hash: 0033a4d0f4d4a839f96e779a528827d5 -> password: alty0
Hash: 4d3c873538081b41753a95ef677e4918 -> password: h0srd
Hash: 9febc0a4c110fc05a2585f652b006bd8 -> password: u9are
Hash: 655a58e886bdc3494047772d19a421a6 -> password: 7at7l
Time taken is: 126.450713000002 for [5] out of 15 found.
```

d) **As expected**, Dictionary Attack took the shortest time of 1.04 sec, followed by bruteforce using Lowercase Alphabets (27.80 sec) and Lowercase Alphabets with Digits (126.45 sec) from the above 3 stages of cracking the passwords. The **longer durations are congruent with the computation** as shown:

   i. Dictionary Attack has a specific list (500,000) of words to hash and compared.

   ii. Bruteforce using Lowercase Alphabets has to cycle through all 26^5 of words. This is 23.76 (= $\frac{26^5}{500000}$) times more than Dictionary list with the timing of 27.80 sec.

   iii. Bruteforce using Lowercase Alphabets and digits has to cycle through all 36^5 of words. This is 120.93 (= $\frac{36^5}{500000}$) times more than Dictionary list with the timing of 126.45 sec.

e) The rule to bruteforce attack is a linear attack by stepping through the characters sequentially, but the time required is not linear but increases as shown in the above results. With increased in the number of characters, the increment in the duration would be more visible in terms of **exponential increment**.

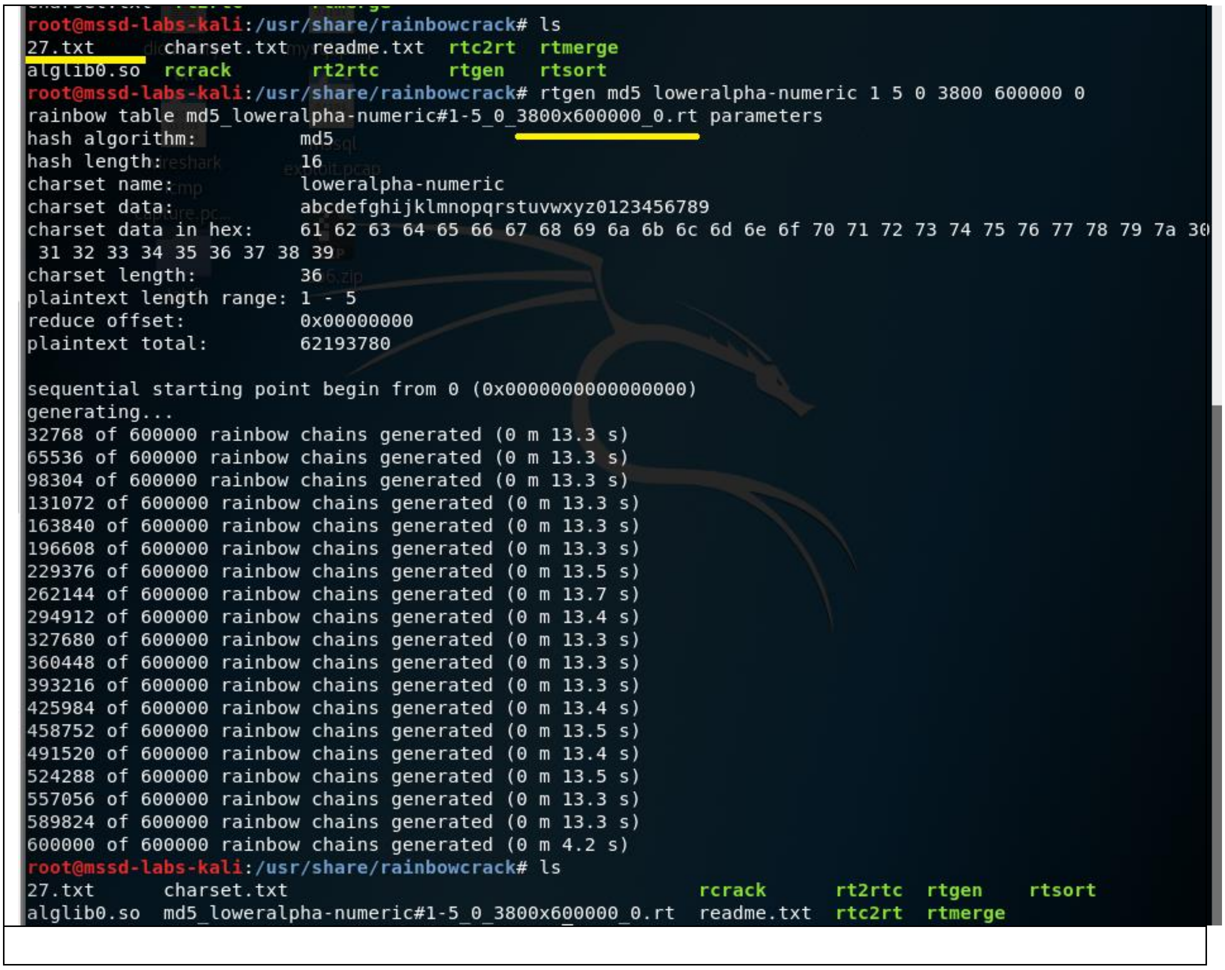
f) By using the combination of dictionary and bruteforce attacks, **all 15 passwords were cracked with the total time of 155.30 secs**. The used of combination of attacks also mirrored the way hashcat multi-attack modes that could be utilized under ruled-based settings.

3. USE OF RAINBOW TABLE

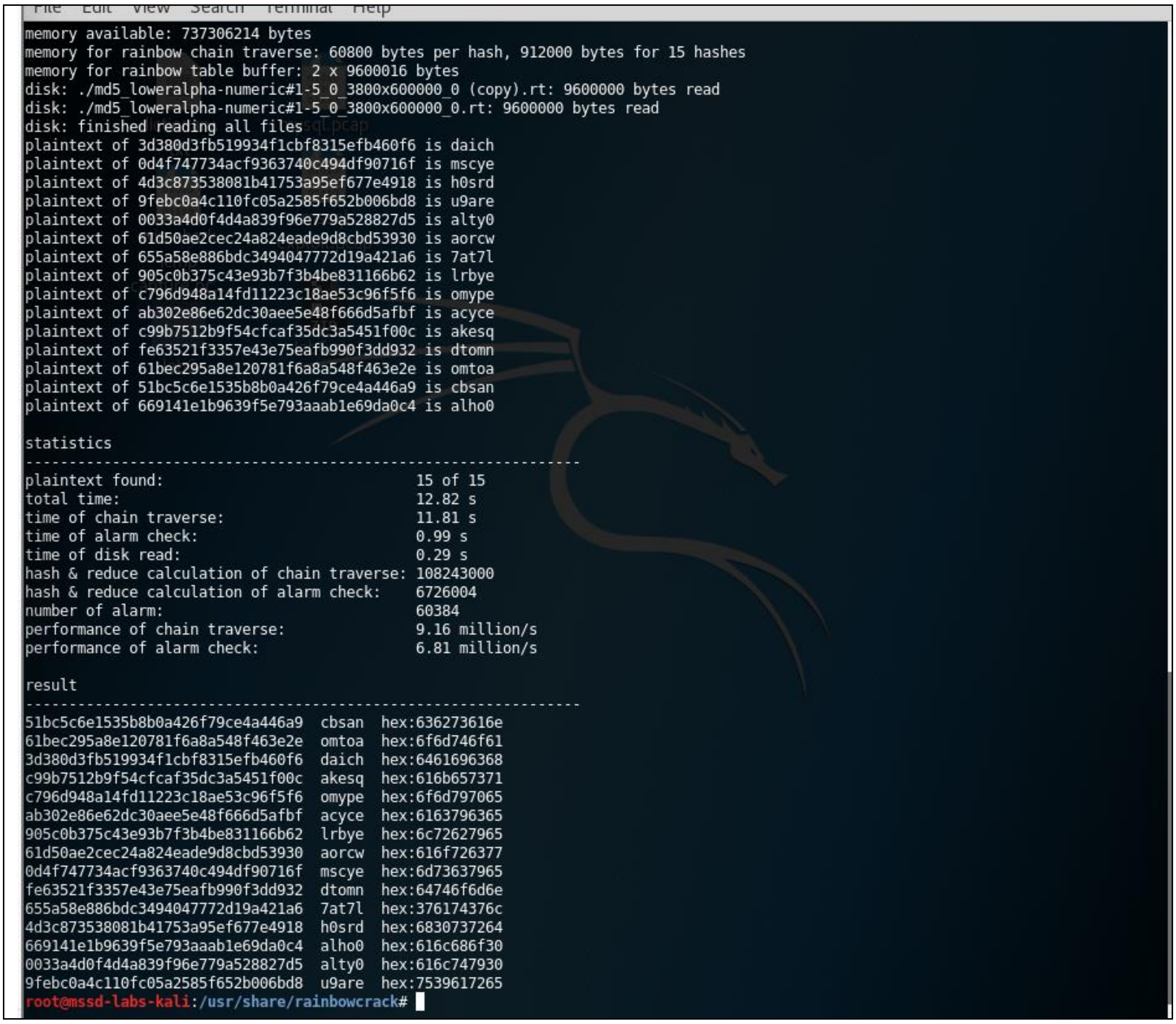
a) Rainbow Table of 5 Characters. The used of Rainbow Table was explored with 6 settings, with chain length as 3800 but with the chain number varying from 80K to 1260K. This was to produce a ratio from 5 to 80 when compared with the number of combinations (36^5 ~= 60M, for 5-character space) that bruteforce could generate.

b) From the ratio of 20 and onwards, all 15 passwords (blue box)) were cracked in 12.68 sec being the fastest. Consequently, a longer time was also required to generate the respective rainbow tables based on the different settings.

Chain Length	Chain Num	RT Size	Ratio (Approx)	RT Generation Time (Sec)	Crack Total Time (Sec)	Number of Password Cracked)
3800	80000	304M	5	34	18.50	14 (93.3%)
3800	160000	608M	10	66.4	18	14 (93.3%)
3800	320000	1216M	20	133	12.68	15 (100%)
3800	600000	2280M	40	244.9	12.82	15 (100%)
3800	960000	3648M	60	400.2	13.00	15 (100%)
3800	1260000	4788M	80	522.1	12.53	15 (100%)



```
root@mssd-labs-kali:/usr/share/rainbowcrack# ls
27.txt      charset.txt  readme.txt  rtc2rt  rtmerge
alglib0.so  rcrack       rt2rtc      rtgen   rtsort
root@mssd-labs-kali:/usr/share/rainbowcrack# rtgen md5 loweralpha-numeric 1 5 0 3800 600000 0
rainbow table md5_loweralpha-numeric#1-5_0_3800x600000_0.rt parameters
hash algorithm:         md5
hash length:            16
charset name:           loweralpha-numeric
charset data:           abcdefghijklmnopqrstuvwxyz0123456789
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 30
 31 32 33 34 35 36 37 38 39
charset length:         36
plaintext length range: 1 - 5
reduce offset:          0x00000000
plaintext total:        62193780

sequential starting point begin from 0 (0x0000000000000000)
generating...
32768 of 600000 rainbow chains generated (0 m 13.3 s)
65536 of 600000 rainbow chains generated (0 m 13.3 s)
98304 of 600000 rainbow chains generated (0 m 13.3 s)
131072 of 600000 rainbow chains generated (0 m 13.3 s)
163840 of 600000 rainbow chains generated (0 m 13.3 s)
196608 of 600000 rainbow chains generated (0 m 13.3 s)
229376 of 600000 rainbow chains generated (0 m 13.5 s)
262144 of 600000 rainbow chains generated (0 m 13.7 s)
294912 of 600000 rainbow chains generated (0 m 13.4 s)
327680 of 600000 rainbow chains generated (0 m 13.3 s)
360448 of 600000 rainbow chains generated (0 m 13.3 s)
393216 of 600000 rainbow chains generated (0 m 13.3 s)
425984 of 600000 rainbow chains generated (0 m 13.4 s)
458752 of 600000 rainbow chains generated (0 m 13.5 s)
491520 of 600000 rainbow chains generated (0 m 13.4 s)
524288 of 600000 rainbow chains generated (0 m 13.5 s)
557056 of 600000 rainbow chains generated (0 m 13.3 s)
589824 of 600000 rainbow chains generated (0 m 13.3 s)
600000 of 600000 rainbow chains generated (0 m 4.2 s)
root@mssd-labs-kali:/usr/share/rainbowcrack# ls
27.txt      charset.txt                               rcrack      rt2rtc  rtgen    rtsort
alglib0.so  md5_loweralpha-numeric#1-5_0_3800x600000_0.rt  readme.txt  rtc2rt  rtmerge
```



```
memory available: 737306214 bytes
memory for rainbow chain traverse: 60800 bytes per hash, 912000 bytes for 15 hashes
memory for rainbow table buffer: 2 x 9600016 bytes
disk: ./md5_loweralpha-numeric#1-5_0_3800x600000_0 (copy).rt: 9600000 bytes read
disk: ./md5_loweralpha-numeric#1-5_0_3800x600000_0.rt: 9600000 bytes read
disk: finished reading all files
plaintext of 3d380d3fb519934f1cbf8315efb460f6 is daich
plaintext of 0d4f747734acf9363740c494df90716f is mscye
plaintext of 4d3c873538081b41753a95ef677e4918 is h0srd
plaintext of 9febc0a4c110fc05a2585f652b006bd8 is u9are
plaintext of 0033a4d0f4d4a839f96e779a528827d5 is alty0
plaintext of 61d50ae2cec24a824eade9d8cbd53930 is aorcw
plaintext of 655a58e886bdc3494047772d19a421a6 is 7at7l
plaintext of 905c0b375c43e93b7f3b4be831166b62 is lrbye
plaintext of c796d948a14fd11223c18ae53c96f5f6 is omype
plaintext of ab302e86e62dc30aee5e48f666d5afbf is acyce
plaintext of c99b7512b9f54cfcaf35dc3a5451f00c is akesq
plaintext of fe63521f3357e43e75eafb990f3dd932 is dtomn
plaintext of 61bec295a8e120781f6a8a548f463e2e is omtoa
plaintext of 51bc5c6e1535b8b0a426f79ce4a446a9 is cbsan
plaintext of 669141e1b9639f5e793aaab1e69da0c4 is alho0

statistics
----------------------------------------------------------------
plaintext found:                             15 of 15
total time:                                  12.82 s
time of chain traverse:                      11.81 s
time of alarm check:                         0.99 s
time of disk read:                           0.29 s
hash & reduce calculation of chain traverse: 108243000
hash & reduce calculation of alarm check:    6726004
number of alarm:                             60384
performance of chain traverse:               9.16 million/s
performance of alarm check:                  6.81 million/s

result
----------------------------------------------------------------
51bc5c6e1535b8b0a426f79ce4a446a9  cbsan  hex:636273616e
61bec295a8e120781f6a8a548f463e2e  omtoa  hex:6f6d746f61
3d380d3fb519934f1cbf8315efb460f6  daich  hex:6461696368
c99b7512b9f54cfcaf35dc3a5451f00c  akesq  hex:616b657371
c796d948a14fd11223c18ae53c96f5f6  omype  hex:6f6d797065
ab302e86e62dc30aee5e48f666d5afbf  acyce  hex:6163796365
905c0b375c43e93b7f3b4be831166b62  lrbye  hex:6c72627965
61d50ae2cec24a824eade9d8cbd53930  aorcw  hex:616f726377
0d4f747734acf9363740c494df90716f  mscye  hex:6d73637965
fe63521f3357e43e75eafb990f3dd932  dtomn  hex:64746f6d6e
655a58e886bdc3494047772d19a421a6  7at7l  hex:376174376c
4d3c873538081b41753a95ef677e4918  h0srd  hex:6830737264
669141e1b9639f5e793aaab1e69da0c4  alho0  hex:616c686f30
0033a4d0f4d4a839f96e779a528827d5  alty0  hex:616c747930
9febc0a4c110fc05a2585f652b006bd8  u9are  hex:7539617265
root@mssd-labs-kali:/usr/share/rainbowcrack# 
```

4. EFFECTS OF SALTING

a) Rainbow Table of 6 Characters. To investigate the time required for longer password, salt was added using a random alphabet (highlighted in RED) to the 15 5-character passwords.

cbsanm
omtoac
daichs
akesqo
omypen
acycey
lrbyes
aorcwr
mscyeh
dtomnk
7at7lb
h0srdt
alho0i
alty0o
u9ares

b) For better comparison, the ratio used 6-character (36^5 ~= 220M) followed the same ratio used for 5-character.

Chain Length	Chain Num	RT Size	Ratio (Approx)	RT Generation Time (Sec)	Crack Total Time (Sec)	Number of Password Cracked)
3800	285000	1083M	5	199.4	24.39	3 (20%)
3800	580000	2204M	10	399.9	26.20	8 (53%)
3800	1160000	4408M	20	538.7	17.62	10 (67%)
3800	2320000	8816M	40	969.7	19.87	13 (87%)
3800	3510000	13,338M	60	1516.3	20.04	14 (93%)
3800	4680000	17,784M	80	1981.2	20.25	15 (100%)

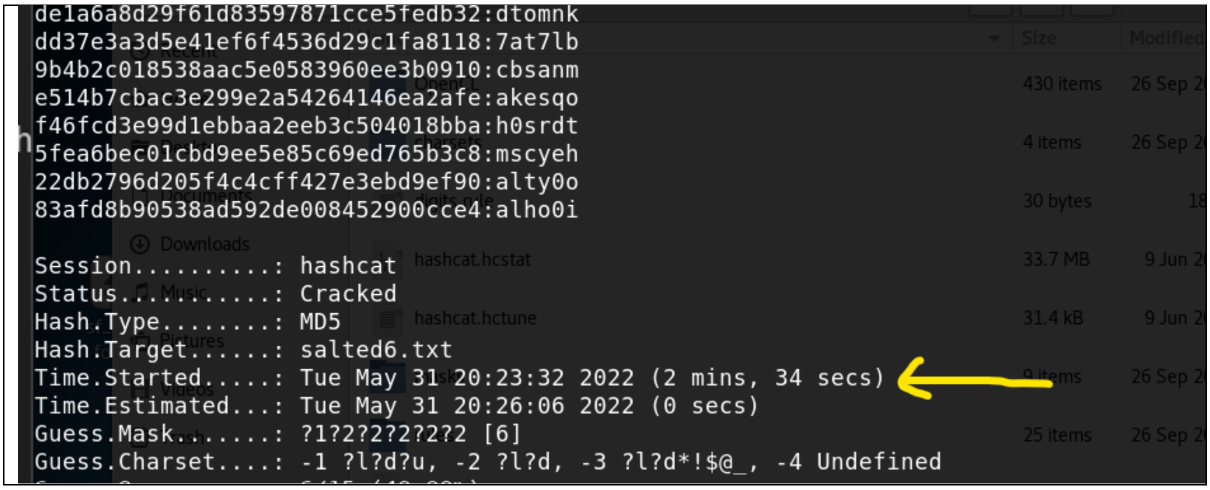
c) Comparison between 5- and 6-character password. Although all 15 passwords were cracked for 6-character password, but it was fulfilled **only at the ratio of 80**. This implies that there would need a larger pool passwords, increased from 1216M to 17,784M, due to the increment of 1 character space.

5. HASHCAT

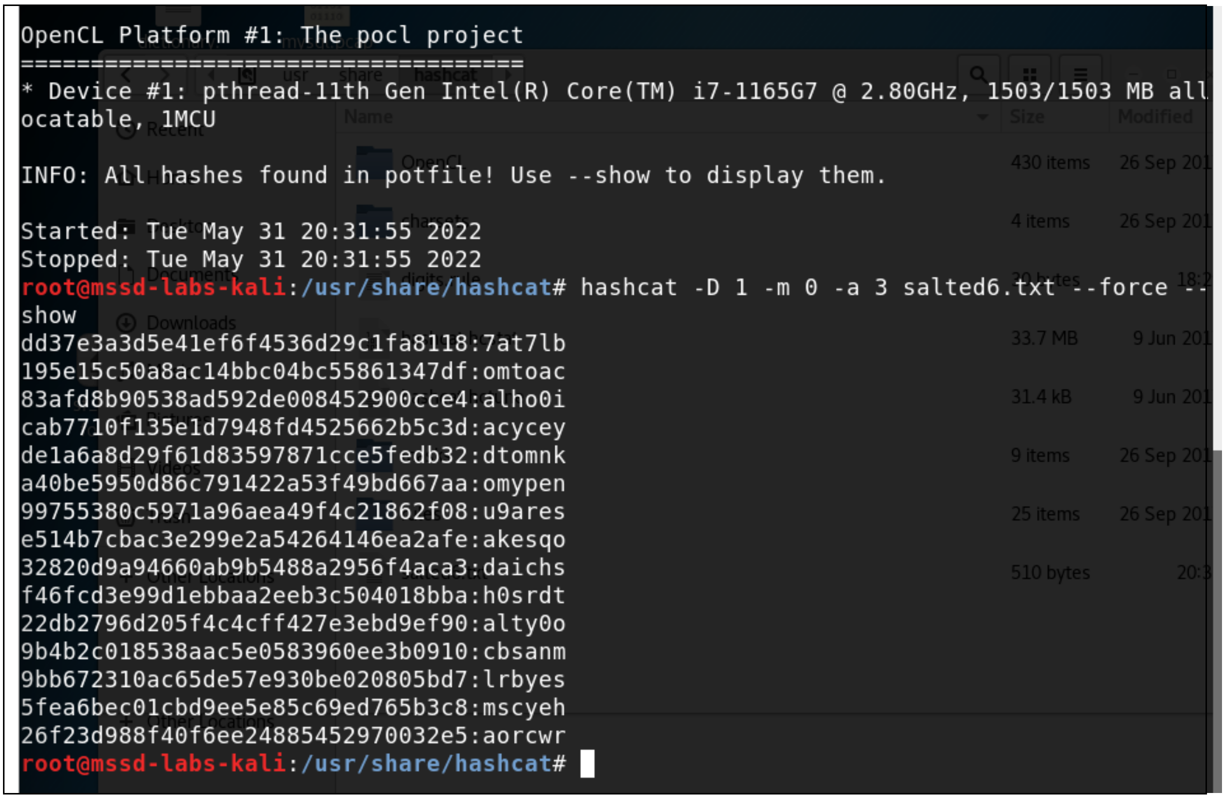
a) Rockyou As Dictionary. For the 15 salted hashes, Hachcat **did not crack** any password using the Rockyou.txt, thus **no cracked.txt was generated**.

b) Rule-based and Mask Attack. To understand both of its ability to crack password, the first attempt was to perform without any flags, and progressively adding flags according to the pass6.txt.

i. All 15 salted hashes were cracked in 154 sec (or 2 min 34 sec) without specifying any flag options.
➔hashcat -D 1 -m 0 -a 3 salted6.txt.

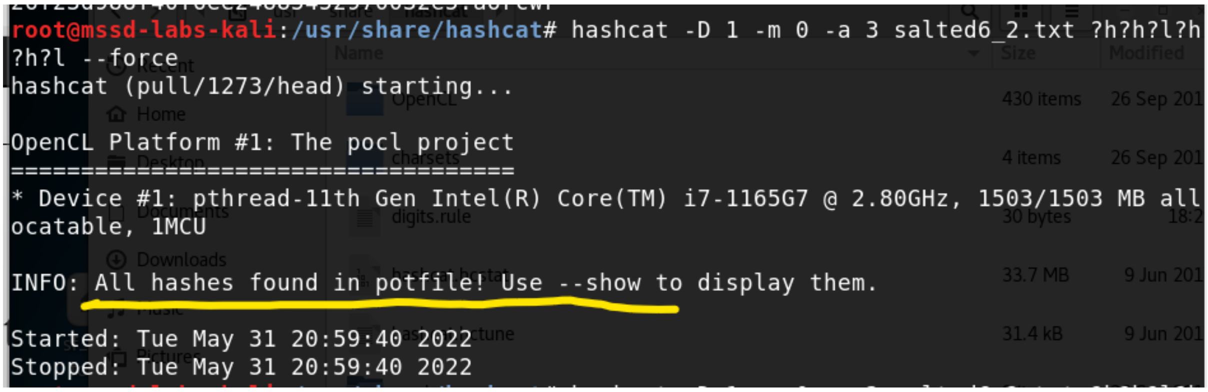


```
de1a6a8d29f61d83597871cce5fedb32:dtomnk
dd37e3a3d5e41ef6f4536d29c1fa8118:7at7lb
9b4b2c018538aac5e0583960ee3b0910:cbsanm
e514b7cbac3e299e2a54264146ea2afe:akesqo
f46fcd3e99d1ebbaa2eeb3c504018bba:h0srdt
5fea6bec01cbd9ee5e85c69ed765b3c8:mscyeh
22db2796d205f4c4cff427e3ebd9ef90:alty0o
83afd8b90538ad592de008452900cce4:alho0i

Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: salted6.txt
Time.Started.....: Tue May 31 20:23:32 2022 (2 mins, 34 secs)
Time.Estimated...: Tue May 31 20:26:06 2022 (0 secs)
Guess.Mask.......: ?1?2?2?2?2?2 [6]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
```



```
OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, 1503/1503 MB all
ocatable, 1MCU

INFO: All hashes found in potfile! Use --show to display them.

Started: Tue May 31 20:31:55 2022
Stopped: Tue May 31 20:31:55 2022
root@mssd-labs-kali:/usr/share/hashcat# hashcat -D 1 -m 0 -a 3 salted6.txt --force --
show
dd37e3a3d5e41ef6f4536d29c1fa8118:7at7lb
195e15c50a8ac14bbc04bc55861347df:omtoac
83afd8b90538ad592de008452900cce4:alho0i
cab7710f135e1d7948fd4525662b5c3d:acycey
de1a6a8d29f61d83597871cce5fedb32:dtomnk
a40be5950d86c791422a53f49bd667aa:omypen
99755380c5971a96aea49f4c21862f08:u9ares
e514b7cbac3e299e2a54264146ea2afe:akesqo
32820d9a94660ab9b5488a2956f4aca3:daichs
f46fcd3e99d1ebbaa2eeb3c504018bba:h0srdt
22db2796d205f4c4cff427e3ebd9ef90:alty0o
9b4b2c018538aac5e0583960ee3b0910:cbsanm
9bb672310ac65de57e930be020805bd7:lrbyes
5fea6bec01cbd9ee5e85c69ed765b3c8:mscyeh
26f23d988f40f6ee2488545297003 2e5:aorcwr
root@mssd-labs-kali:/usr/share/hashcat#
```

ii. In the second attempt, hashcat could not perform any more cracking because the all 6-character passwords were already found and written to the potfile.



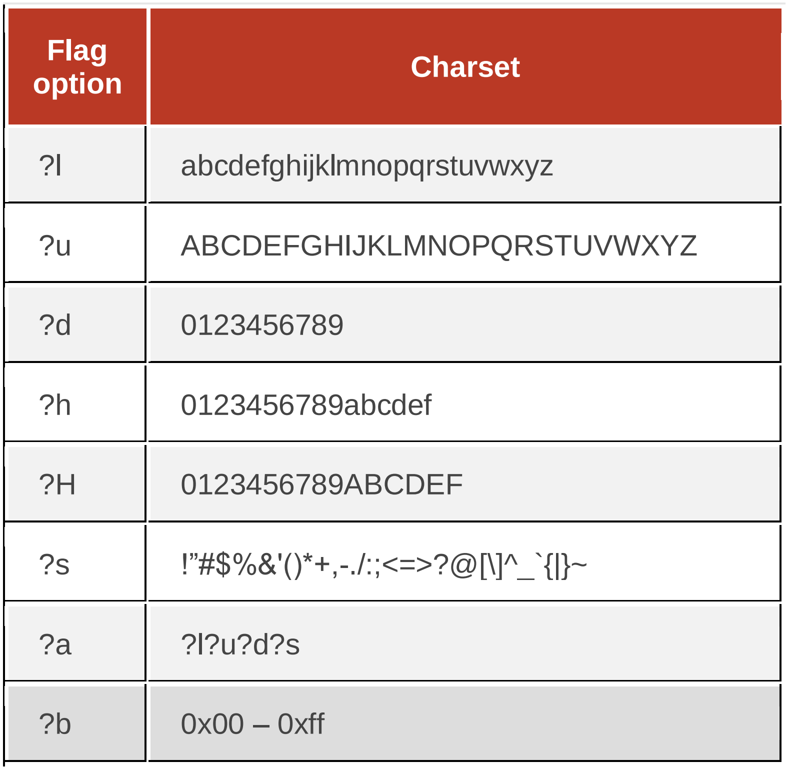
```
root@mssd-labs-kali:/usr/share/hashcat# hashcat -D 1 -m 0 -a 3 salted6_2.txt ?h?h?l?h
?h?l --force
hashcat (pull/1273/head) starting...

OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, 1503/1503 MB all
ocatable, 1MCU

INFO: All hashes found in potfile! Use --show to display them.

Started: Tue May 31 20:59:40 2022
Stopped: Tue May 31 20:59:40 2022
```

iii. Therefore, the progressive attempts could not be fulfilled. Nevertheless, the commands to showcase hashcat ability to shorten the cracking timing with known positions of the character as shown:

iv. There were designed as such because position 1, 2, 4 and 5 having alphanumeric (lowercase characters) while only position 3 and 6 were solely alphabets (lowercase characters).

➔ hashcat -D 1 -m 0 -a 3 salted6.txt ?h?h?l?h?h?l

v. The knowledge these flags were referenced from https://linuxhint.com/hashcat-tutorial/



Flag option	Charset
?l	abcdefghijklmnopqrstuvwxyz
?u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d	0123456789
?h	0123456789abcdef
?H	0123456789ABCDEF
?s	!”#$%&'()*+,-./:;<=>?@[\]^_`{|}~
?a	?l?u?d?s
?b	0x00 – 0xff

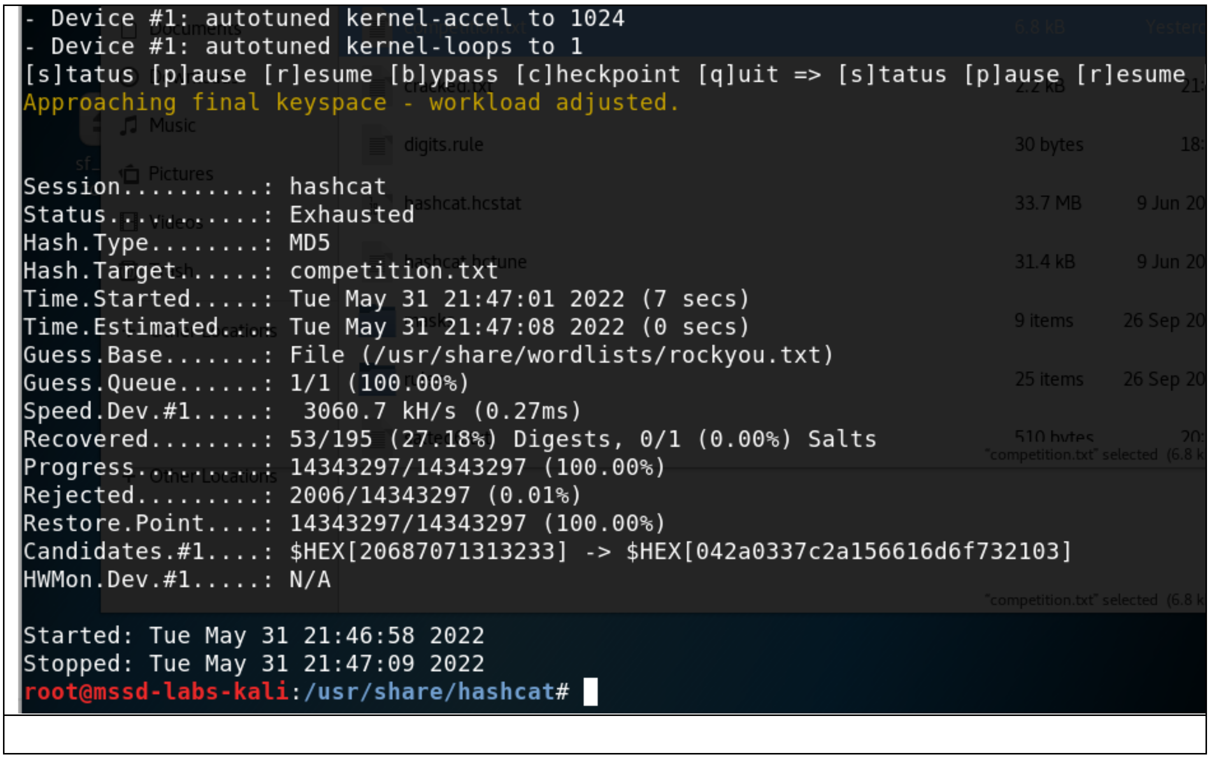
c) Comparison Between Rainbow Table and Hashcat. Although rainbow table computes in a shorter time, it however requires the generation of the table hence the total timing would 2001.45 sec (1981.2 sec + 20.25 sec). Therefore, hashcat outshined the cracking of passwords with the timing of 154 sec.

6. COMPETITION

a) STEP 1. With the knowledge that the potfile will record passwords that were already cracked, thus the **quickest way to eliminate** the any easily available and common passwords would be to use Dictionary Attack.

b) Thus, using the provided rockyou.txt, the following command were executed.

```
⇨ hashcat -m 0 -a 0 -o cracked.txt competition.txt /usr/share/wordlists/rockyou.txt
```

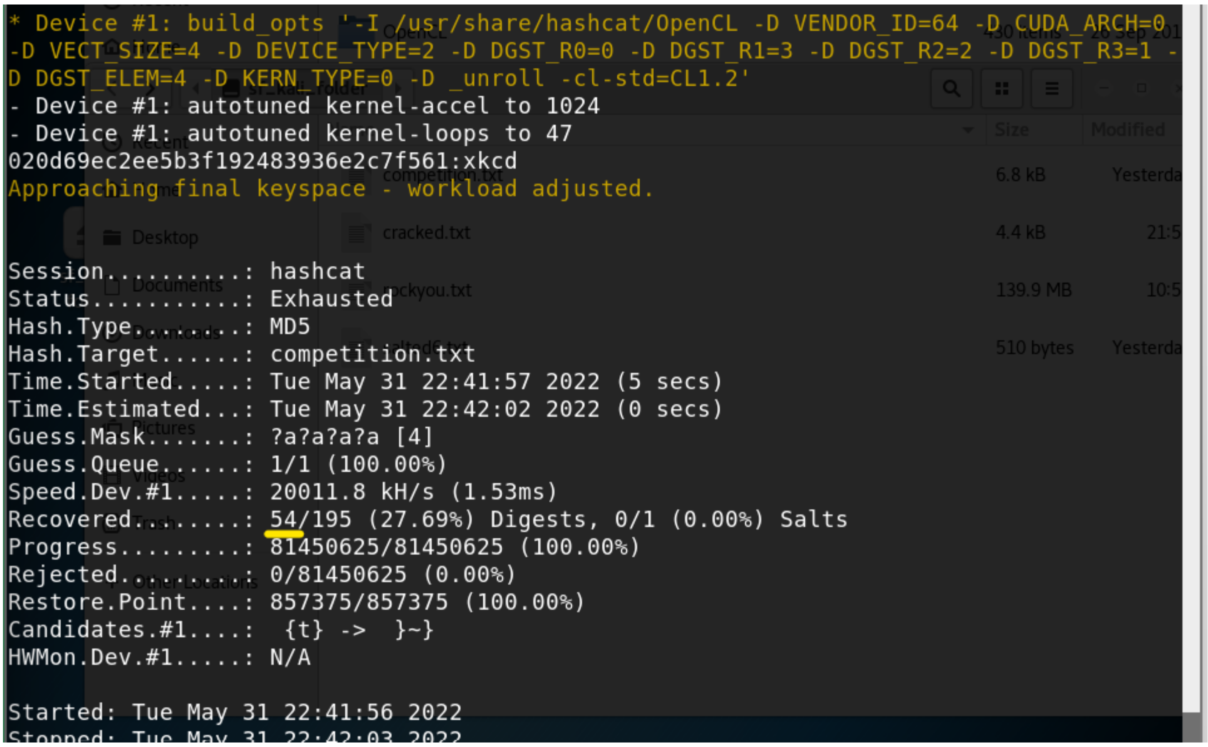


```
- Device #1: autotuned kernel-accel to 1024
- Device #1: autotuned kernel-loops to 1
[s]tatus [p]ause [r]esume [b]ypass [c]heckpoint [q]uit => [s]tatus [p]ause [r]esume
Approaching final keyspace - workload adjusted.

Session..........: hashcat
Status...........: Exhausted
Hash.Type........: MD5
Hash.Target......: competition.txt
Time.Started.....: Tue May 31 21:47:01 2022 (7 secs)
Time.Estimated...: Tue May 31 21:47:08 2022 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:  3060.7 kH/s (0.27ms)
Recovered........: 53/195 (27.18%) Digests, 0/1 (0.00%) Salts
Progress.........: 14343297/14343297 (100.00%)
Rejected.........: 2006/14343297 (0.01%)
Restore.Point....: 14343297/14343297 (100.00%)
Candidates.#1....: $HEX[206870713132333] -> $HEX[042a0337c2a156616d6f732103]
HWMon.Dev.#1.....: N/A

Started: Tue May 31 21:46:58 2022
Stopped: Tue May 31 21:47:09 2022
root@mssd-labs-kali:/usr/share/hashcat#
```

c) STEP 2. The cracked passwords were examined. The shortest and longest passwords were asdf and password1234567890. With no special characters involved, the next step would be mask attack including special characters.

i. With 4-characters using ?a. One more password (increment from 53 to 54) was cracked.

⇨ hashcat -D 1 -m 0 -a 3 competition.txt ?a?a?a?a



```
* Device #1: build_opts '-I /usr/share/hashcat/OpenCL -D VENDOR_ID=64 -D CUDA_ARCH=0
-D VECT_SIZE=4 -D DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=3 -D DGST_R2=2 -D DGST_R3=1 -
D DGST_ELEM=4 -D KERN_TYPE=0 -D _unroll -cl-std=CL1.2'
- Device #1: autotuned kernel-accel to 1024
- Device #1: autotuned kernel-loops to 47
020d69ec2ee5b3f192483936e2c7f561:xkcd
Approaching final keyspace - workload adjusted.

Session..........: hashcat
Status...........: Exhausted
Hash.Type........: MD5
Hash.Target......: competition.txt
Time.Started.....: Tue May 31 22:41:57 2022 (5 secs)
Time.Estimated...: Tue May 31 22:42:02 2022 (0 secs)
Guess.Mask.......: ?a?a?a?a [4]
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....: 20011.8 kH/s (1.53ms)
Recovered........: 54/195 (27.69%) Digests, 0/1 (0.00%) Salts
Progress.........: 81450625/81450625 (100.00%)
Rejected.........: 0/81450625 (0.00%)
Restore.Point....: 857375/857375 (100.00%)
Candidates.#1....:  {t} ->  }~}
HWMon.Dev.#1.....: N/A

Started: Tue May 31 22:41:56 2022
Stopped: Tue May 31 22:42:03 2022
```

ii. <u>With 6-characters using ?a</u>. One more password (increment from 54 to **56 passwords**) was cracked.

⇨ hashcat -D 1 -m 0 -a 3 competition.txt ?a?a?a?a?a?a

d) <u>Summing Up</u>. **56 out of 201 hashes were cracked**.

7. CONCLSION

In conclusion, it is noted that the increased in sampling space via increasing the character space in incomparable to the increase in the length of the password. Therefore, for greater security of password should having a longer password than having fanciful (combination of alphanumeric and special characters) passwords.

Summing up, based on the results the approach taken should to achieve an optimum timing should be as such:

a) Dictionary attack using popular and common passwords (as done in STEP 1 of competition). This is due to that fact that there is a time saving in generating the list of passwords, such as in bruteforce, prior to hashing them for comparison.

b) When the above is unsuccessful, Hashcat attack should be used as it is a hybrid and thus optimize all other techniques.