# Lab15 Password Cracking Using John the Ripper and Hashcat

## Lab Learning Objectives

- Learn how to benchmark each algorithm supported by John and Hashcat
- Use John and Hashcat to crack LANMAN and/or NT hashes from Windows
- Use John and Hashcat to crack Linux hashes

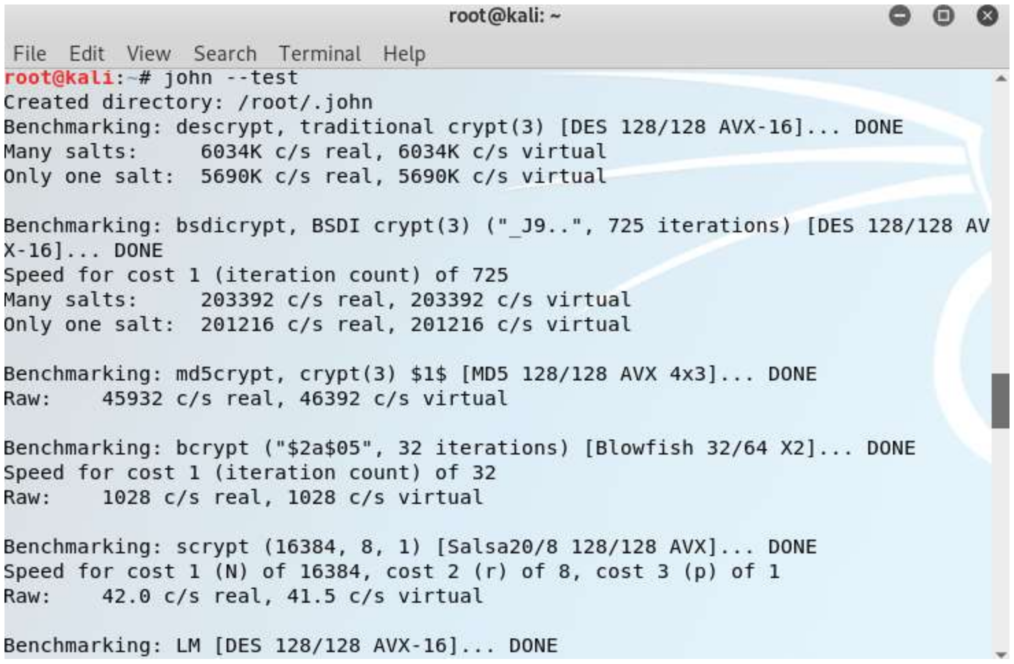
## Lab Setup

In this lab, you will use the Kali Linux virtual machine.

## Lab Instructions

1. We will start the lab to test the speed of each algorithm John will run on your particular computer hardware. Bring on a terminal on Kali Linux machine and run

**# john --test**



```
root@kali:~# john --test
Created directory: /root/.john
Benchmarking: descrypt, traditional crypt(3) [DES 128/128 AVX-16]... DONE
Many salts:     6034K c/s real, 6034K c/s virtual
Only one salt:  5690K c/s real, 5690K c/s virtual

Benchmarking: bsdicrypt, BSDI crypt(3) ("_J9..", 725 iterations) [DES 128/128 AV
X-16]... DONE
Speed for cost 1 (iteration count) of 725
Many salts:     203392 c/s real, 203392 c/s virtual
Only one salt:  201216 c/s real, 201216 c/s virtual

Benchmarking: md5crypt, crypt(3) $1$ [MD5 128/128 AVX 4x3]... DONE
Raw:    45932 c/s real, 46392 c/s virtual

Benchmarking: bcrypt ("$2a$05", 32 iterations) [Blowfish 32/64 X2]... DONE
Speed for cost 1 (iteration count) of 32
Raw:    1028 c/s real, 1028 c/s virtual

Benchmarking: scrypt (16384, 8, 1) [Salsa20/8 128/128 AVX]... DONE
Speed for cost 1 (N) of 16384, cost 2 (r) of 8, cost 3 (p) of 1
Raw:    42.0 c/s real, 41.5 c/s virtual

Benchmarking: LM [DES 128/128 AVX-16]... DONE
```

2. Next, we will use John to crack Linux password. To save your time, **you can download the passwd and shadow files from the Blackboard** and save them into the /tmp folder. We then use the unshadow script from John to combine account information from /etc/passwd with password information from /etc/shadow and store the result in passwdhash.txt.

**# unshadow /tmp/passwd /tmp/shadow > /tmp/passwdhash.txt**

We now run John against the combined file.

**# john /tmp/passwdhash.txt**

Let John run for a couple of minutes. It should crack the frank password because it is merely the login username backward and the monk password because master1is in the dictionary.

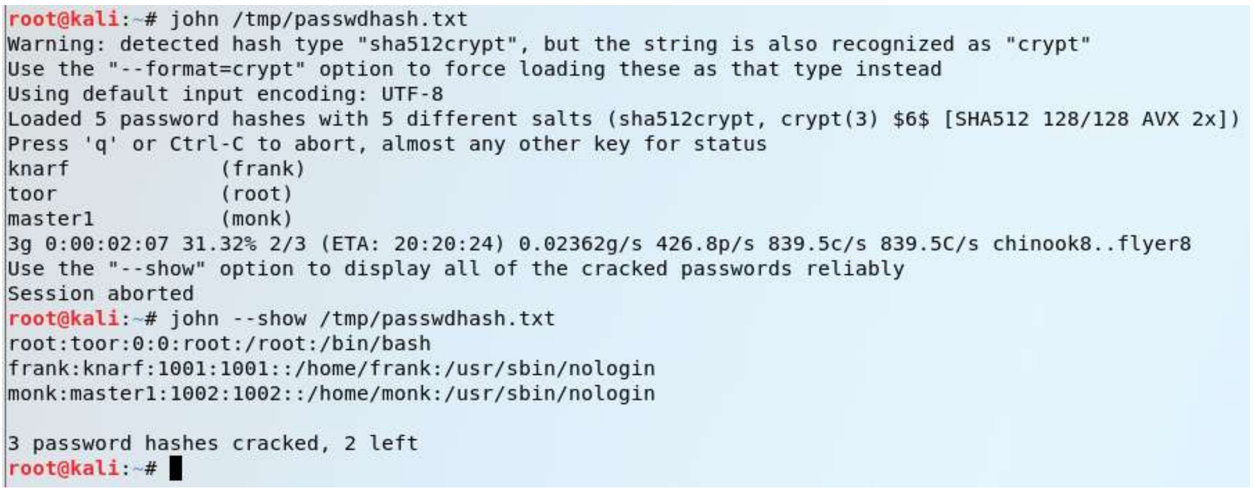
Next, we use --show option to review cracked hashes

**# john --show /tmp/passwdhash.txt**

When John cracks some password hashes, it automatically stores the cracked password hashes to a file named john.pot in a hidden directory .john under your home directory. Let's review the john.pot file by running

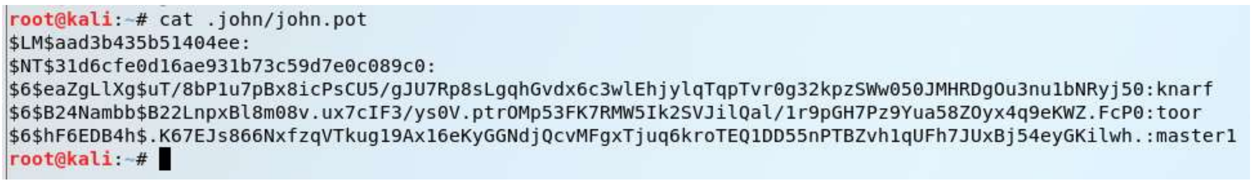
**# cat .john/john.pot**

We can find that the john.pot file only includes the password format, hash and cracked password. No login name is included. **(Screenshot #1)**



```
root@kali:~# john /tmp/passwdhash.txt
Warning: detected hash type "sha512crypt", but the string is also recognized as "crypt"
Use the "--format=crypt" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 5 password hashes with 5 different salts (sha512crypt, crypt(3) $6$ [SHA512 128/128 AVX 2x])
Press 'q' or Ctrl-C to abort, almost any other key for status
knarf            (frank)
toor             (root)
master1          (monk)
3g 0:00:02:07 31.32% 2/3 (ETA: 20:20:24) 0.02362g/s 426.8p/s 839.5c/s 839.5C/s chinook8..flyer8
Use the "--show" option to display all of the cracked passwords reliably
Session aborted
root@kali:~# john --show /tmp/passwdhash.txt
root:toor:0:0:root:/root:/bin/bash
frank:knarf:1001:1001::/home/frank:/usr/sbin/nologin
monk:master1:1002:1002::/home/monk:/usr/sbin/nologin

3 password hashes cracked, 2 left
root@kali:~# █
```

**(Screenshot #2)**



```
root@kali:~# cat .john/john.pot
$LM$aad3b435b51404ee:
$NT$31d6cfe0d16ae931b73c59d7e0c089c0:
$6$eaZgLlXg$uT/8bP1u7pBx8icPsCU5/gJU7Rp8sLgqhGvdx6c3wlEhjylqTqpTvr0g32kpzSWw050JMHRDgOu3nu1bNRyj50:knarf
$6$B24Nambb$B22LnpxBl8m08v.ux7cIF3/ys0V.ptrOMp53FK7RMW5Ik2SVJilQal/1r9pGH7Pz9Yua58ZOyx4q9eKWZ.FcP0:toor
$6$hF6EDB4h$.K67EJs866NxfzqVTkug19Ax16eKyGGNdjQcvMFgxTjuq6kroTEQ1DD55nPTBZvh1qUFh7JUxBj54eyGKilwh.:master1
root@kali:~# █
```

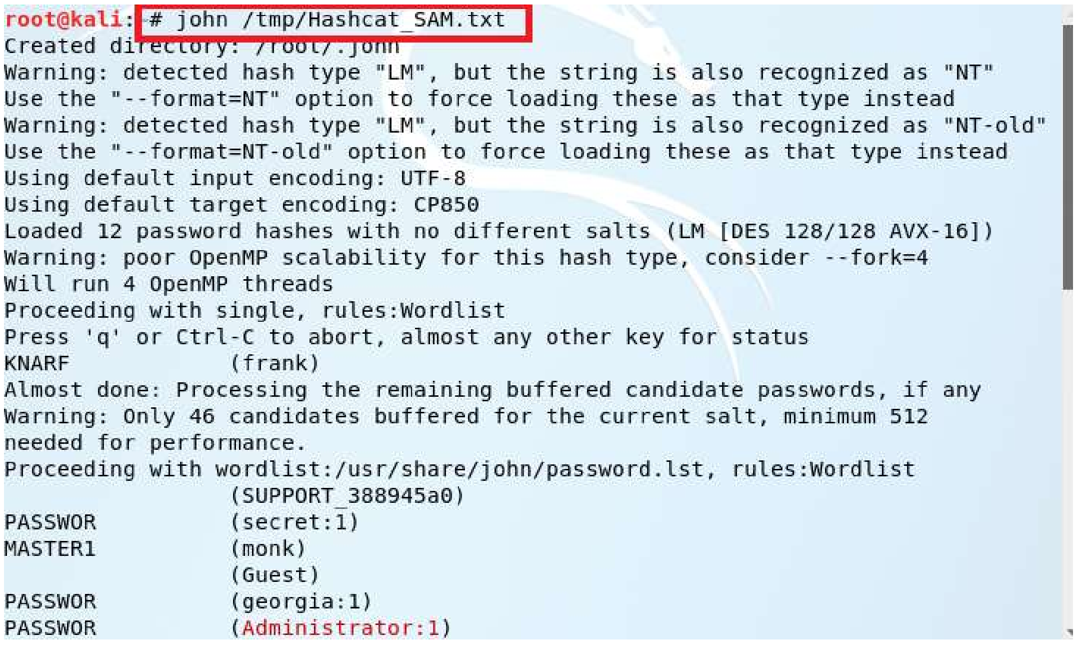
3. Next, we will use John to crack the passwords dumped from the Windows XP machine. Download the Hashcat_SAM.txt file from the Blackboard. Copy the hash file to the /tmp folder.

**# cp ~/Downloads/Hashcat_SAM.txt /tmp/**

By default, John will focus on the LM hashes. Run John against the hash file.

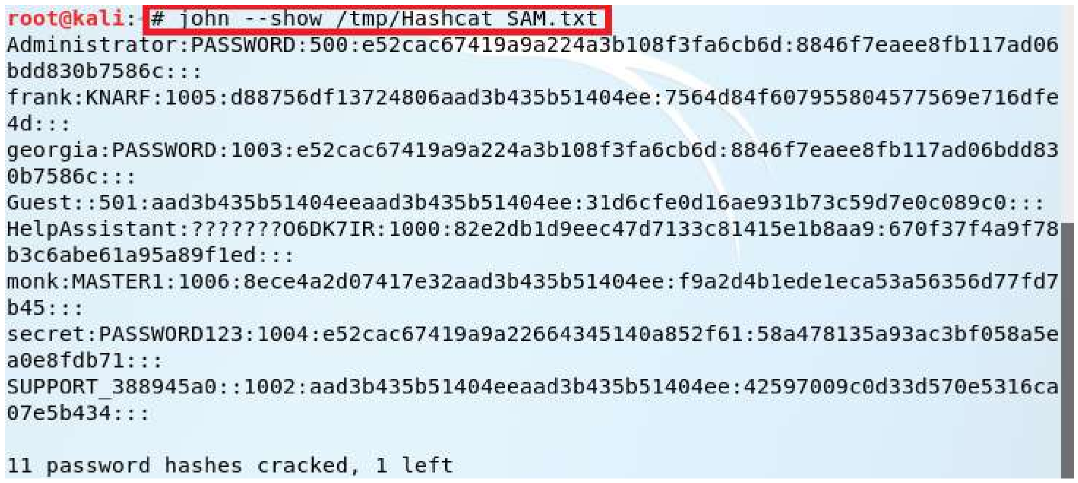
**# john /tmp/Hashcat_SAM.txt**

Note that the passwords that John cracks are all in CAPS. In addition, it is worth noting that John cracks the first 7 characters of a LM password separately from the second 7 characters, treating each half as though it was a different password. The first half of the password is indicated by the (username:1) and the second half is (username:2). After running John for several minutes, stop it by pressing CTRL-C.



```
root@kali:~# john /tmp/Hashcat_SAM.txt
Created directory: /root/.john
Warning: detected hash type "LM", but the string is also recognized as "NT"
Use the "--format=NT" option to force loading these as that type instead
Warning: detected hash type "LM", but the string is also recognized as "NT-old"
Use the "--format=NT-old" option to force loading these as that type instead
Using default input encoding: UTF-8
Using default target encoding: CP850
Loaded 12 password hashes with no different salts (LM [DES 128/128 AVX-16])
Warning: poor OpenMP scalability for this hash type, consider --fork=4
Will run 4 OpenMP threads
Proceeding with single, rules:Wordlist
Press 'q' or Ctrl-C to abort, almost any other key for status
KNARF            (frank)
Almost done: Processing the remaining buffered candidate passwords, if any
Warning: Only 46 candidates buffered for the current salt, minimum 512
needed for performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
                 (SUPPORT_388945a0)
PASSWOR          (secret:1)
MASTER1          (monk)
                 (Guest)
PASSWOR          (georgia:1)
PASSWOR          (Administrator:1)
```

We can run the --show option to view the full password cracked by John. This command searches inside the john.pot file for the hashes in Hashcat_SAM.txt so that it can print the full passwords associated with the users. **(Screenshot #3)**



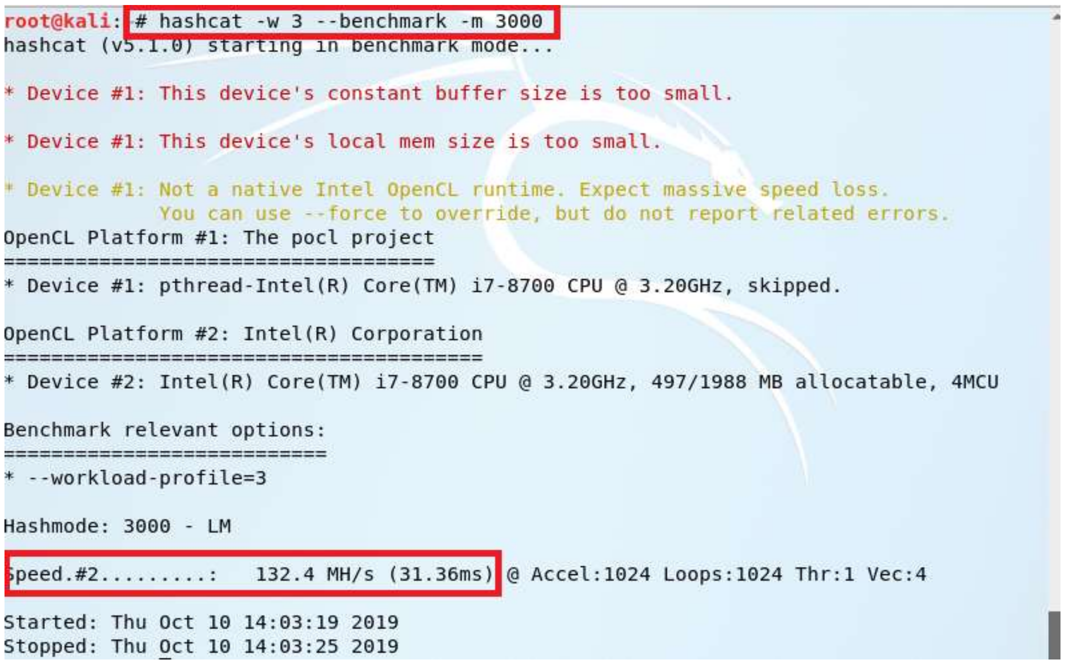
```
root@kali:~# john --show /tmp/Hashcat_SAM.txt
Administrator:PASSWORD:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06
bdd830b7586c:::
frank:KNARF:1005:d88756df13724806aad3b435b51404ee:7564d84f607955804577569e716dfe
4d:::
georgia:PASSWORD:1003:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd83
0b7586c:::
Guest::501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:???????O6DK7IR:1000:82e2db1d9eec47d7133c81415e1b8aa9:670f37f4a9f78
b3c6abe61a95a89f1ed:::
monk:MASTER1:1006:8ece4a2d07417e32aad3b435b51404ee:f9a2d4b1ede1eca53a56356d77fd7
b45:::
secret:PASSWORD123:1004:e52cac67419a9a22664345140a852f61:58a478135a93ac3bf058a5e
a0e8fdb71:::
SUPPORT_388945a0::1002:aad3b435b51404eeaad3b435b51404ee:42597009c0d33d570e5316ca
07e5b434:::

11 password hashes cracked, 1 left
```

4. Next we will use Hashcat to crack the same hash file. First, we need to prepare the wordlist provided by Hashcat.

**# cp /usr/share/doc/hashcat-data/examples/example.dict /tmp/**

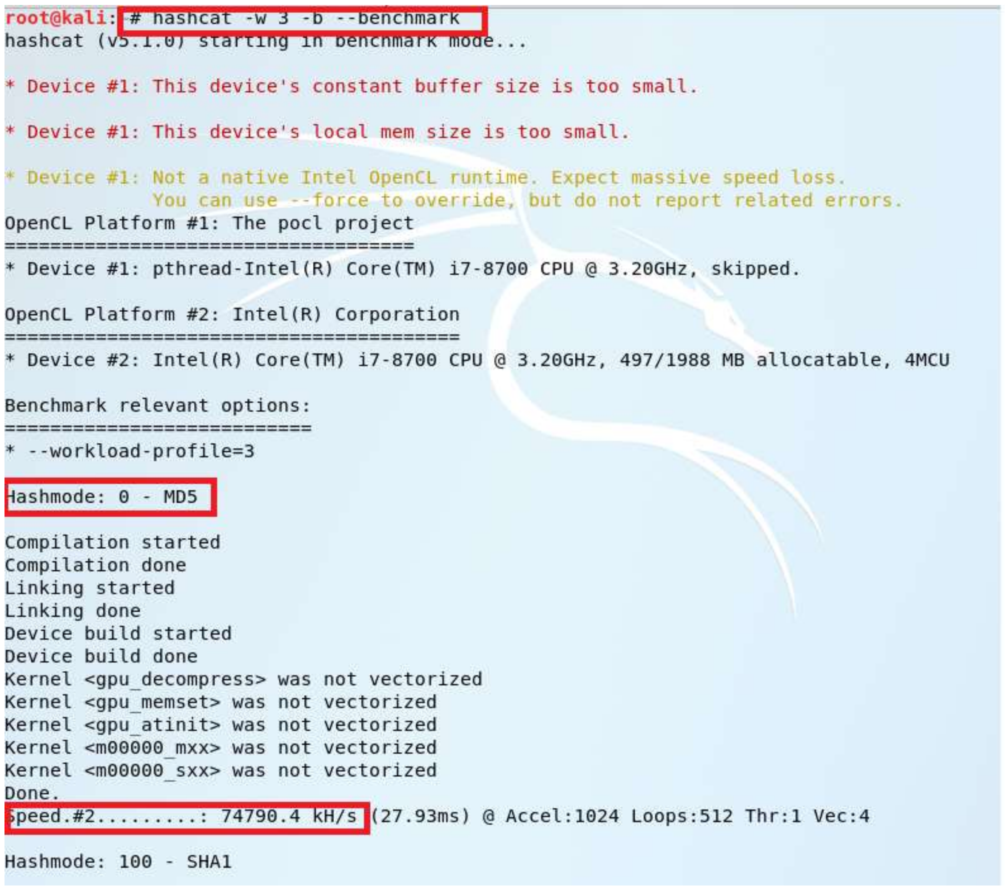
Now, let's conduct some performance measures of Hashcat for some common hash algorithms. We will test the performance for cracking hash type -m 3000, also known as LM.

**# hashcat -w 3 --benchmark -m 3000**


```
root@kali:~# hashcat -w 3 --benchmark -m 3000
hashcat (v5.1.0) starting in benchmark mode...

* Device #1: This device's constant buffer size is too small.

* Device #1: This device's local mem size is too small.

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, skipped.

OpenCL Platform #2: Intel(R) Corporation
========================================
* Device #2: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 497/1988 MB allocatable, 4MCU

Benchmark relevant options:
===========================
* --workload-profile=3

Hashmode: 3000 - LM

Speed.#2.........:   132.4 MH/s (31.36ms) @ Accel:1024 Loops:1024 Thr:1 Vec:4

Started: Thu Oct 10 14:03:19 2019
Stopped: Thu Oct 10 14:03:25 2019
```

You can see the performance in MegaHashes per second (MH/s). We can also review Hashcat's capability to show performance benchmarks for all of its hash algorithms. Since it will take quite a while to run through all the algorithms supported by Hashcat, we just let it run a few algorithms and then hit CTRL-C to stop it.

**# hashcat -w 3 -b --benchmark-all**



```
root@kali:~# hashcat -w 3 -b --benchmark
hashcat (v5.1.0) starting in benchmark mode...

* Device #1: This device's constant buffer size is too small.

* Device #1: This device's local mem size is too small.

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, skipped.

OpenCL Platform #2: Intel(R) Corporation
========================================
* Device #2: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 497/1988 MB allocatable, 4MCU

Benchmark relevant options:
===========================
* --workload-profile=3

Hashmode: 0 - MD5

Compilation started
Compilation done
Linking started
Linking done
Device build started
Device build done
Kernel <gpu_decompress> was not vectorized
Kernel <gpu_memset> was not vectorized
Kernel <gpu_atinit> was not vectorized
Kernel <m00000_mxx> was not vectorized
Kernel <m00000_sxx> was not vectorized
Done.
Speed.#2.........: 74790.4 kH/s (27.93ms) @ Accel:1024 Loops:512 Thr:1 Vec:4

Hashmode: 100 - SHA1
```

Now we will crack the LM hashes (-m 3000) use the example.dict as our dictionary. The results will be stored in a file named crackedpw.txt and we will save this file in the /tmp folder.

**# hashcat -w 3 -a 0 -m 3000 -o /tmp/crackedpw.txt /tmp/Hashcat_SAM.txt /tmp/example.dict**

- If there is a memory related error, stop Kali linux and increase memory more than 4096MB. Then restart Kali linux.
- If you get Status “Exhausted” as a result and cannot find .hashcat folder, update apt-get and install hashcat again

When Hashcat finishes running, let’s look at our results. First, we check the crackedpw.txt file

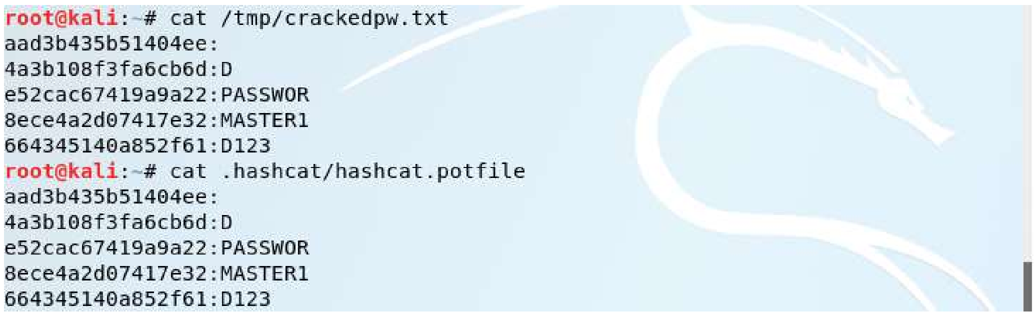
**# cat /tmp/crackedpw.txt**

You can see that Hashcat successfully cracked several of our passwords, include the blank password which has a LM hash starting with aad3b43…Let’s also look at our potfile

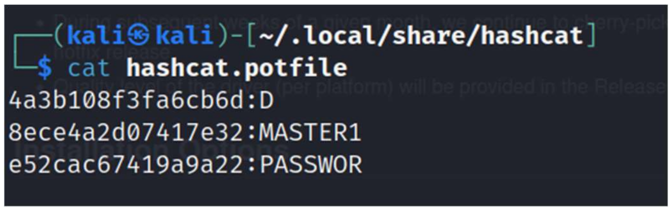
**# cat .hashcat/hashcat.potfile**

If you cannot find potfile from the location, find the Hashcat result from /home/kali/.local/share/hashcat/hashcat.potfile directory. And the Hashcat result could be a little bit different from students.

**(Screenshot #4, need both commands in one screenshot)**



```
root@kali:~# cat /tmp/crackedpw.txt
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
root@kali:~# cat .hashcat/hashcat.potfile
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
```

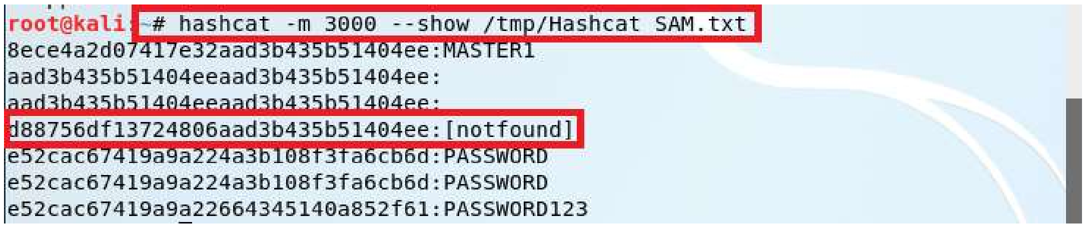


```
┌──(kali㉿kali)-[~/.local/share/hashcat]
└─$ cat hashcat.potfile
4a3b108f3fa6cb6d:D
8ece4a2d07417e32:MASTER1
e52cac67419a9a22:PASSWOR
```

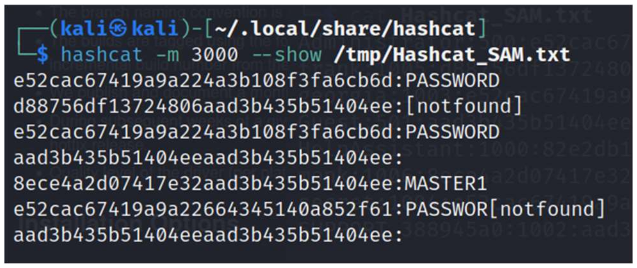
We can see that our hashcat.potfile has the exact same content as the crackedpw.txt file. That's because we have run Hashcat only once. If we were to crack additional passwords, the hashcat.potfile will still have all the results from all of Hashcat's runs, each appended to the other. In other words, hashcat.potfile is Hashcat's long term memory of what it has already cracked. In addition, just like John, Hashcat cracks the first 7 characters of a LM password separately from the second 7 characters, treating each half as though it was a different password. Also the hashcat.potfile does not display login name.

It is not surprising that Hashcat also includes a --show option to display which hashes from a given hashfile that have already been cracked because they appear in the potfile. It is worth noting that you must specify the hash type using the -m option as Hashcat cannot automatically identify hash type as John does. Again, Hashcat result could be a little bit different from students.

**# hashcat -m 3000 --show /tmp/Hashcat_SAM.txt**



```
root@kali:~# hashcat -m 3000 --show /tmp/Hashcat SAM.txt
8ece4a2d07417e32aad3b435b51404ee:MASTER1
aad3b435b51404eeaad3b435b51404ee:
aad3b435b51404eeaad3b435b51404ee:
d88756df13724806aad3b435b51404ee:[notfound]
e52cac67419a9a224a3b108f3fa6cb6d:PASSWORD
e52cac67419a9a224a3b108f3fa6cb6d:PASSWORD
e52cac67419a9a22664345140a852f61:PASSWORD123
```



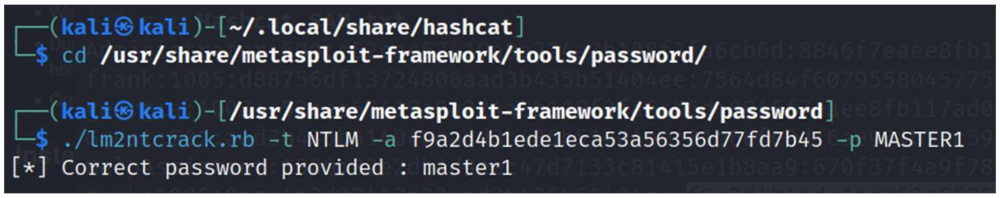
```
┌──(kali㉿kali)-[~/.local/share/hashcat]
└─$ hashcat -m 3000 --show /tmp/Hashcat_SAM.txt
e52cac67419a9a224a3b108f3fa6cb6d:PASSWORD
d88756df13724806aad3b435b51404ee:[notfound]
e52cac67419a9a224a3b108f3fa6cb6d:PASSWORD
aad3b435b51404eeaad3b435b51404ee:
8ece4a2d07417e32aad3b435b51404ee:MASTER1
e52cac67419a9a22664345140a852f61:PASSWOR[notfound]
aad3b435b51404eeaad3b435b51404ee:
```

Furthermore, we can find that monk's password has been cracked (MASTER1). However, the password hash we cracked is the LM hash. The cracked password is all in CAPs. Unfortunately, such password cannot be used in the environment as it does not have the correct case. We can use a ruby script to obtain the correct case for the password. First, we need to change directory to where the script is located by typing

**# cd /usr/share/metasploit-framework/tools/password/**

After that, run the lm2ntcrack.rb script. Secret’s password was cracked correctly.

**# ./lm2ntcrack.rb -t NTLM -a monk’s_NT_Hashes -p MASTER1**



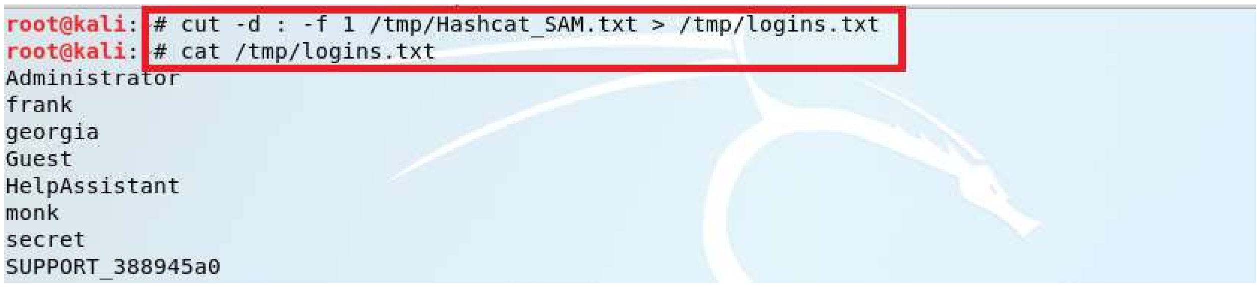
```
┌──(kali㉿kali)-[~/.local/share/hashcat]
└─$ cd /usr/share/metasploit-framework/tools/password/

┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/password]
└─$ ./lm2ntcrack.rb -t NTLM -a f9a2d4b1ede1eca53a56356d77fd7b45 -p MASTER1
[*] Correct password provided : master1
```

If you are careful enough, you will find that Hashcat did not crack the password for the frank account, which John did crack in our earlier lab in step 3. This is because Hashcat does not utilize login names in its cracking operation like John does. Frank has a password of knarf which is frank spelled backward. We can approximate John’s use of a login name as a potential password by creating a small dictionary file that includes login names from the sam file. First, let’s review the contents of the sam file.

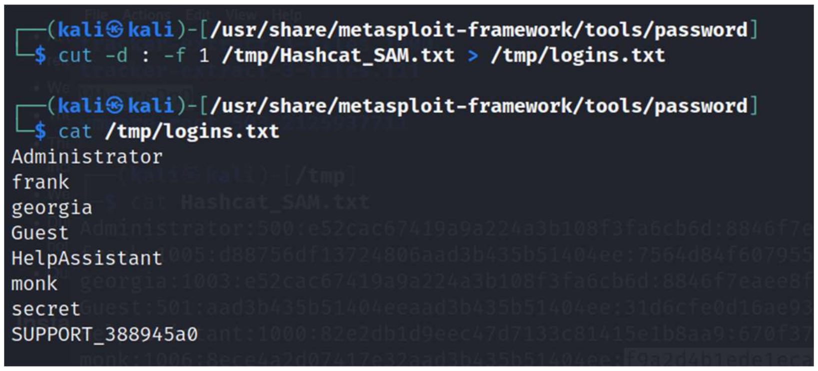
**# cat /tmp/Hashcat_SAM.txt**

We can see the login name is the first item in this colon delimited file. We will extract the login names by using the cut command. We save the results in a file called logins.txt under /tmp folder

**# cut -d : -f 1 /tmp/Hashcat_SAM.txt > /tmp/logins.txt**



```
root@kali:~# cut -d : -f 1 /tmp/Hashcat_SAM.txt > /tmp/logins.txt
root@kali:~# cat /tmp/logins.txt
Administrator
frank
georgia
Guest
HelpAssistant
monk
secret
SUPPORT_388945a0
```

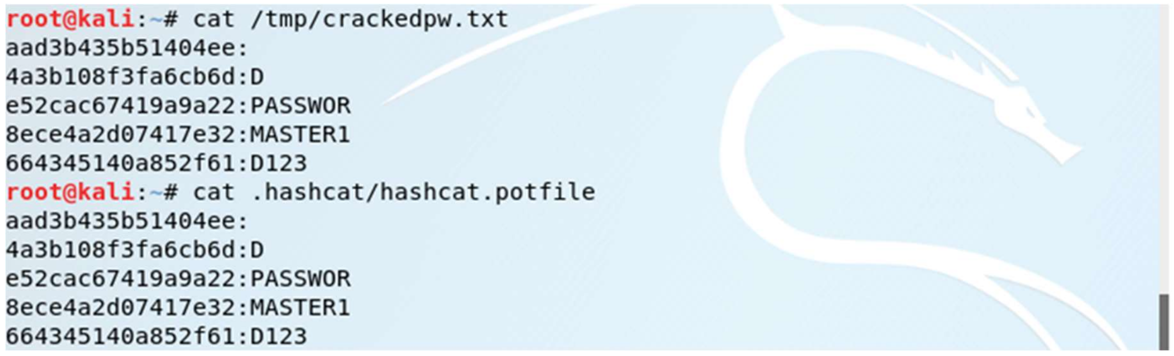


```
┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/password]
└─$ cut -d : -f 1 /tmp/Hashcat_SAM.txt > /tmp/logins.txt

┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/password]
└─$ cat /tmp/logins.txt
Administrator
frank
georgia
Guest
HelpAssistant
monk
secret
SUPPORT_388945a0
```

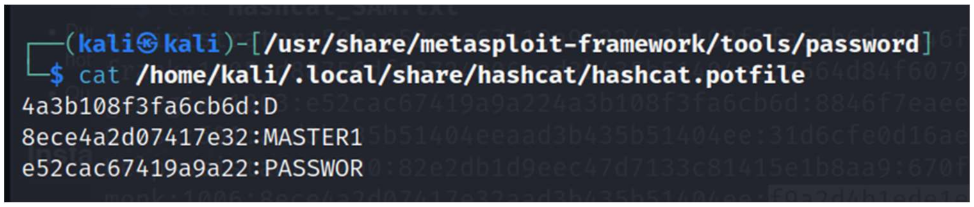
Next, we will re-run Hashcat by adding the logins.txt as a second dictionary in the command line. Hashcat conveniently allows us to append multiple dictionary files to its command line.

**# hashcat -w 3 -a 0 -m 3000 -o /tmp/crackedpw.txt /tmp/Hashcat_SAM.txt /tmp/example.dict /tmp/logins.txt**

After Hashcat finishes running, look at the crackedpw.txt and hashcat.potfile one more time. Unfortunately, we still did not crack the password for the frank account.



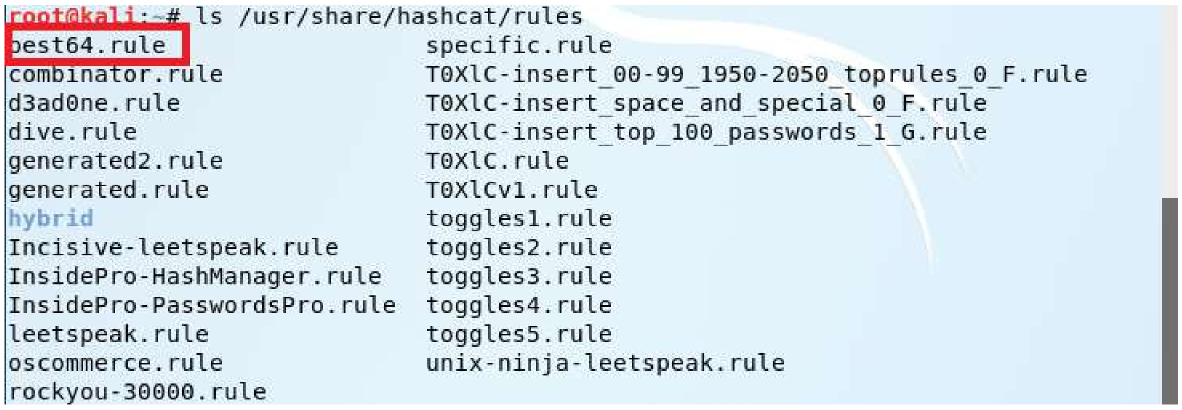
```
root@kali:~# cat /tmp/crackedpw.txt
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
root@kali:~# cat .hashcat/hashcat.potfile
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
```



```
┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/password]
└─$ cat /home/kali/.local/share/hashcat/hashcat.potfile
4a3b108f3fa6cb6d:D
8ece4a2d07417e32:MASTER1
e52cac67419a9a22:PASSWOR
```

This is because although we did use a dictionary with the word frank in it, Hashcat does not mangle words like John does. We need to apply such an operation manually by specifying the rule files. We will use one of the most useful rule best64.rule. Kali stores all the Hashcat rules inside the /usr/share/hashcat/rules folder. Review the available rules by typing

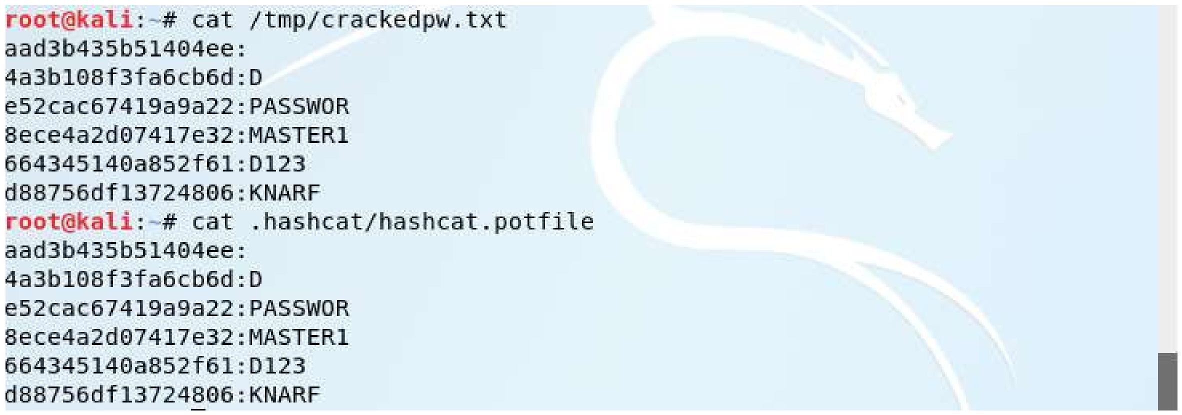
**# ls /usr/share/hashcat/rules**



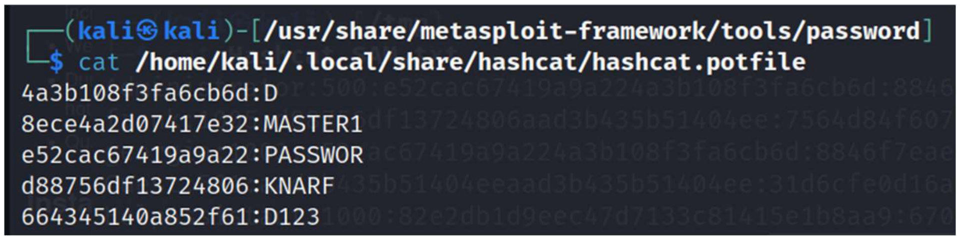
```
root@kali:~# ls /usr/share/hashcat/rules
best64.rule                  specific.rule
combinator.rule              T0XlC-insert_00-99_1950-2050_toprules_0_F.rule
d3ad0ne.rule                 T0XlC-insert_space_and_special_0_F.rule
dive.rule                    T0XlC-insert_top_100_passwords_1_G.rule
generated2.rule              T0XlC.rule
generated.rule               T0XlCv1.rule
hybrid                       toggles1.rule
Incisive-leetspeak.rule      toggles2.rule
InsidePro-HashManager.rule   toggles3.rule
InsidePro-PasswordsPro.rule  toggles4.rule
leetspeak.rule               toggles5.rule
oscommerce.rule              unix-ninja-leetspeak.rule
rockyou-30000.rule
```

Let’s crack the hash file one more time by specifying the rule using the -r option

**# hashcat -w 3 -a 0 -m 3000 -o /tmp/crackedpw.txt /tmp/Hashcat_SAM.txt /tmp/example.dict /tmp/logins.txt -r /usr/share/hashcat/rules/best64.rule**



```
root@kali:~# cat /tmp/crackedpw.txt
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
d88756df13724806:KNARF
root@kali:~# cat .hashcat/hashcat.potfile
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
d88756df13724806:KNARF
```



```
┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/password]
└─$ cat /home/kali/.local/share/hashcat/hashcat.potfile
4a3b108f3fa6cb6d:D
8ece4a2d07417e32:MASTER1
e52cac67419a9a22:PASSWOR
d88756df13724806:KNARF
664345140a852f61:D123
```

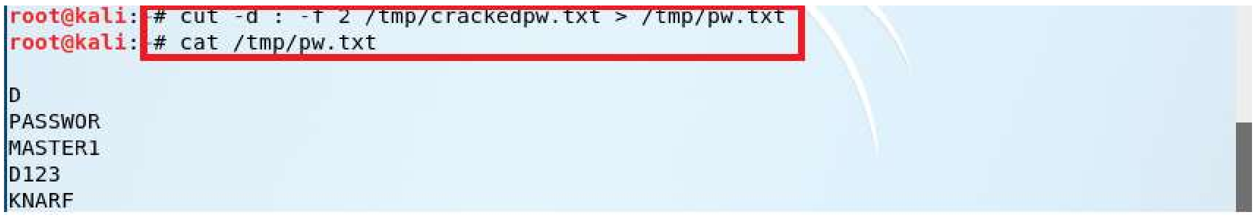
Finally, we successfully crack the password for the frank account. By doing this lab, we learn that adding a login name file to our dictionary is useful with Hashcat, and applying a rule can be even more helpful. With both efforts, we are able to successfully crack frank's password, mimicking the capability of John. You may ask then what is the benefit to use Hashcat? Remember that Hashcat can support way more hash algorithms than John can. In addition, on some hardware, Hashcat has better performance.

5. Next, we will use Hashcat to crack some Linux hash files. We will use the shadow copy from step 2. As we all know that it can be very helpful to take already cracked passwords and add them to a dictionary file so that we do not have to apply word mangling rules to them again before rediscovering the password when is tis hashed using a different algorithm. In other words, we have already mangled some words and cracked their hashes, so why mangle those words again if we come up against the same password with a different hash algorithm? Let's review the cracked passwords stored in the crackedpw.txt file.

**# cat /tmp/crackedpw.txt**

We can see that the cracked passwords are stored as the second item in this colon delimited file. We will exact the cracked passwords by using the cut command. We save the results in a file called pw.txt under /tmp folder

**# cut -d : -f 2 /tmp/crackedpw.txt > /tmp/pw.txt**



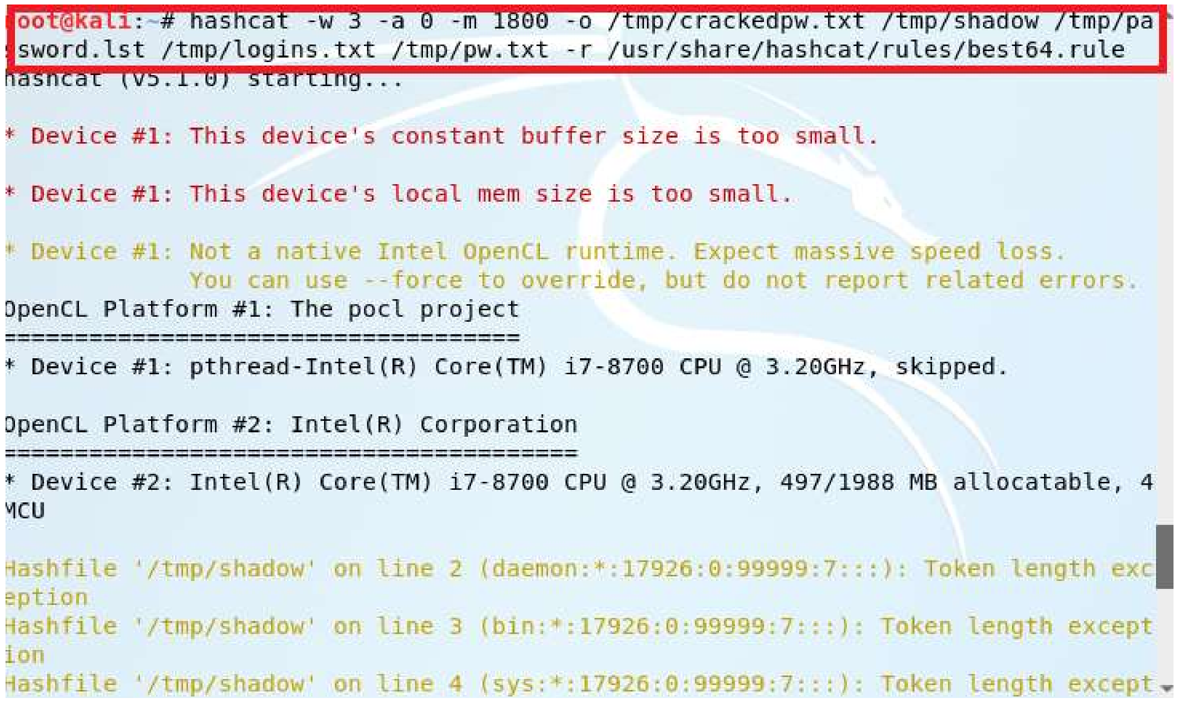
```
root@kali:~# cut -d : -f 2 /tmp/crackedpw.txt > /tmp/pw.txt
root@kali:~# cat /tmp/pw.txt

D
PASSWOR
MASTER1
D123
KNARF
```

We will use John's dictionary password.lst (with much less words) instead of Hashcat's standard example.dict file just to make the lab go more quickly while still getting successful results. How many words are included in the password.lst file? (**Question 8**). Let's copy the password.lst file to the /tmp folder.

**# cp /usr/share/john/password.lst /tmp/**

Since Linux use salted SHA512 for its password hashes. We will use hash type of 1800 when we run the Hashcat. Hashcat has the capability to parse shadow file directly and there is no need to combined it with /etc/passwd. Why? Remember, Hashcat does not utilize login names or GECOS fields in its cracking operations so that we do not need that information from /etc/passwd.

**# hashcat -w 3 -a 0 -m 1800 -o /tmp/crackedpw.txt /tmp/shadow /tmp/password.lst /tmp/logins.txt /tmp/pw.txt -r /usr/share/hashcat/rules/best64.rule**



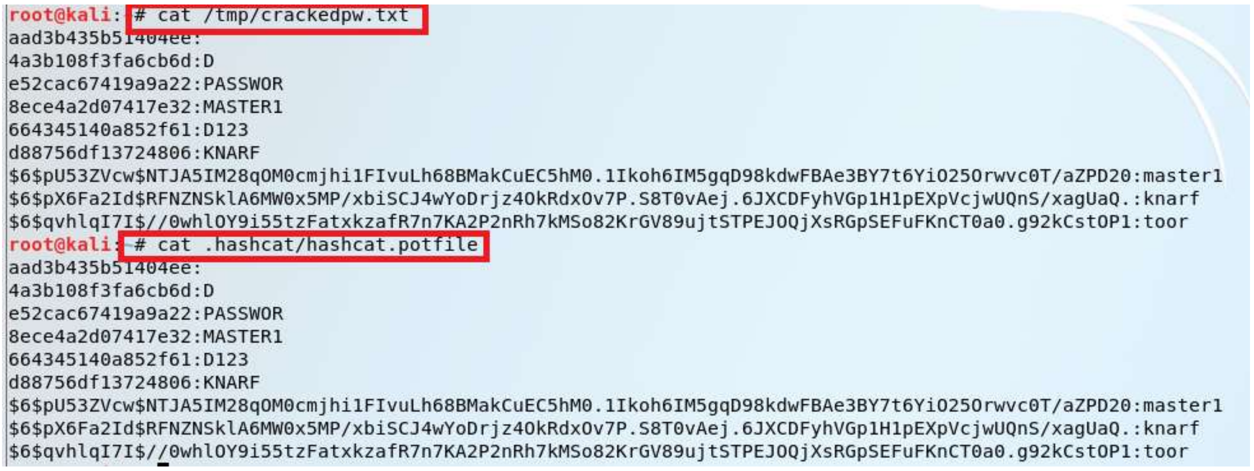
```
oot@kali:~# hashcat -w 3 -a 0 -m 1800 -o /tmp/crackedpw.txt /tmp/shadow /tmp/pa
sword.lst /tmp/logins.txt /tmp/pw.txt -r /usr/share/hashcat/rules/best64.rule
hashcat (v5.1.0) starting...

* Device #1: This device's constant buffer size is too small.

* Device #1: This device's local mem size is too small.

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, skipped.

OpenCL Platform #2: Intel(R) Corporation
========================================
* Device #2: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 497/1988 MB allocatable, 4
MCU

Hashfile '/tmp/shadow' on line 2 (daemon:*:17926:0:99999:7:::): Token length exc
eption
Hashfile '/tmp/shadow' on line 3 (bin:*:17926:0:99999:7:::): Token length except
ion
Hashfile '/tmp/shadow' on line 4 (sys:*:17926:0:99999:7:::): Token length except
```

As Hashcat runs, you can see that it is parsing the file to look for the hash type of 1800. When it sees a line without such hash, Hashcat will move on. When it sees a line with the $6$ hash, it will try to crack it. When Hashcat runs, you can type s key to get the status. It takes a few minutes for Hashcat to finish running. Let's review the results.

**# cat /tmp/crackedpw.txt**

**# cat .hashcat/hashcat.potfile**

**(Screenshot #5, need both commands in one screenshot)**



```
root@kali:~# cat /tmp/crackedpw.txt
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
d88756df13724806:KNARF
$6$pU53ZVcw$NTJA5IM28qOM0cmjhi1FIvuLh68BMakCuEC5hM0.1Ikoh6IM5gqD98kdwFBAe3BY7t6YiO25Orwvc0T/aZPD20:master1
$6$pX6Fa2Id$RFNZNSklA6MW0x5MP/xbiSCJ4wYoDrjz4OkRdxOv7P.S8T0vAej.6JXCDFyhVGp1H1pEXpVcjwUQnS/xagUaQ.:knarf
$6$qvhlqI7I$//0whlOY9i55tzFatxkzafR7n7KA2P2nRh7kMSo82KrGV89ujtSTPEJOQjXsRGpSEFuFKnCT0a0.g92kCstOP1:toor
root@kali:~# cat .hashcat/hashcat.potfile
aad3b435b51404ee:
4a3b108f3fa6cb6d:D
e52cac67419a9a22:PASSWOR
8ece4a2d07417e32:MASTER1
664345140a852f61:D123
d88756df13724806:KNARF
$6$pU53ZVcw$NTJA5IM28qOM0cmjhi1FIvuLh68BMakCuEC5hM0.1Ikoh6IM5gqD98kdwFBAe3BY7t6YiO25Orwvc0T/aZPD20:master1
$6$pX6Fa2Id$RFNZNSklA6MW0x5MP/xbiSCJ4wYoDrjz4OkRdxOv7P.S8T0vAej.6JXCDFyhVGp1H1pEXpVcjwUQnS/xagUaQ.:knarf
$6$qvhlqI7I$//0whlOY9i55tzFatxkzafR7n7KA2P2nRh7kMSo82KrGV89ujtSTPEJOQjXsRGpSEFuFKnCT0a0.g92kCstOP1:toor
```

From the results, we can find out Hashcat successfully cracked the passwords for frank and monk. However, it was not able to crack the password for susan.

6. Now that we have finished this lab, let's shred the various copies of password files we left on the systems. Shred overwrites the file with alternating zeros and ones three times so that they cannot be recovered. The --remove option makes shred delete the file when it is done shredding it.

```
# cd /tmp
# shred --remove passwd
# shred --remove shadow
# shred --remove passwdhash.txt
# shred --remove crackedpw.txt
# shred --remove Hashcat_SAM.txt
# shred --remove pw.txt
```

## Lab Report

- please include your name and 700# at the beginning of your report
- please upload your report to the Blackboard by the due date
- only word or pdf format is acceptable

1. Please screenshots (**Screenshot #1~#5**, 4 points each)