Hashcat:

Password cracking tool

Go to terminal

Add hashed passwords in file pass.lst (nano pass.lst) and copy paste

locate wordlists or locate common.txt or locate rockyou.txt to find predefined list of common passwords in plaintext

this command below uses hashcat to crack the passwords it takes the rockyou text passwords hashes them using md5 and compares to pass.lst. the cracked plaintext passwords are then stored in newly created rohan.txt.

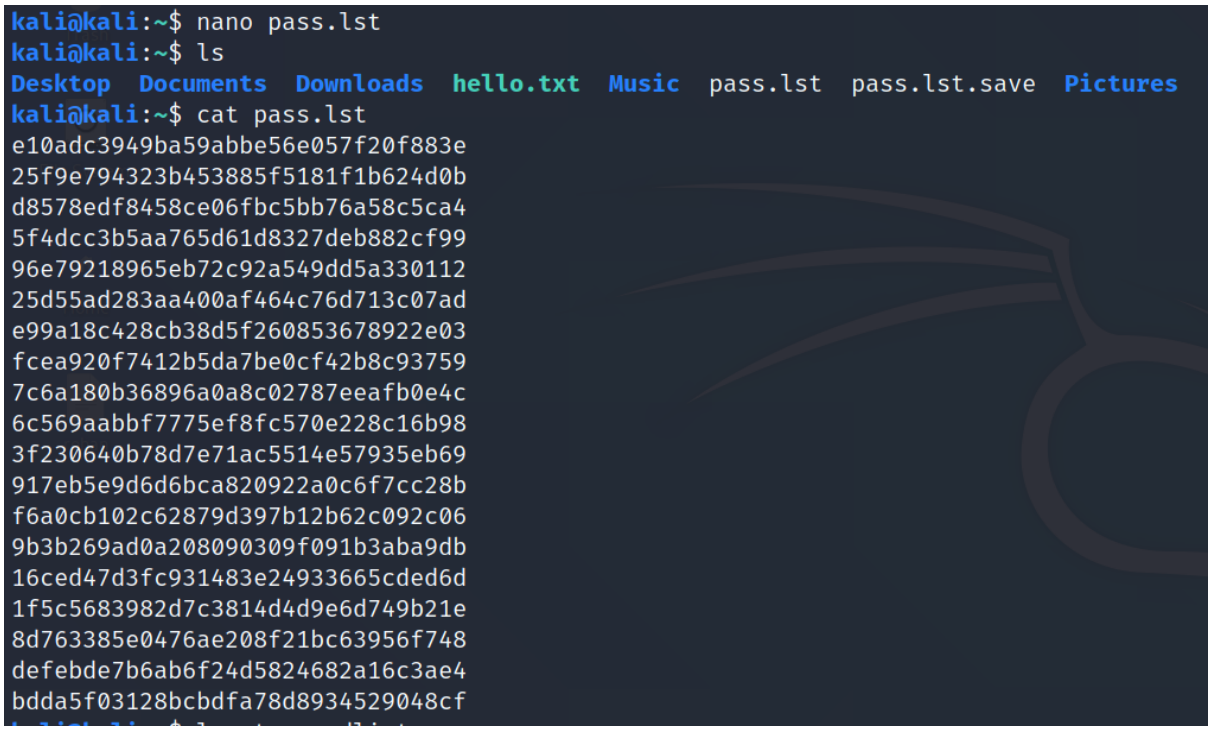
-m is used to specify hash algo 0- md5, 100-sha1, 1700-sha512

-a is used to define attack type 0-staright, 2-comnination,etc

Use man hashcat to get help

We can choose better password lists and include combination and permutation to increase the cracking capabilities.

hashcat -m 0 -a 0 -o rohan.txt pass.lst /usr/share/wordlists/rockyou.txt



```
kali@kali:~$ locate rockyou.txt
/usr/share/wordlists/rockyou.txt
kali@kali:~$ ^C
kali@kali:~$ hashcat -m 0 -a 0 -o rohan.txt pass.lst /usr/share/wordlists/rockyou.txt
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=============================================================================================================================
* Device #1: pthread-Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1429/1493 MB (512 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 19 digests; 19 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers applied:
```

```
Status...........: Exhausted
Hash.Name........: MD5
Hash.Target......: pass.lst
Time.Started.....: Sat Oct 30 04:17:00 2021 (22 secs)
Time.Estimated...: Sat Oct 30 04:17:22 2021 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   614.7 kH/s (1.10ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 13/19 (68.42%) Digests
Progress.........: 14344385/14344385 (100.00%)
Rejected.........: 0/14344385 (0.00%)
Restore.Point....: 14344385/14344385 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: $HEX[206b72697374656e616e6e65] → $HEX[042a0337c2a156616d6f732103]

Started: Sat Oct 30 04:16:51 2021
Stopped: Sat Oct 30 04:17:24 2021
kali@kali:~$ cat rohan.txt
e99a18c428cb38d5f260853678922e03:abc123
5f4dcc3b5aa765d61d8327deb882cf99:password
e10adc3949ba59abbe56e057f20f883e:123456
25f9e794323b453885f5181f1b624d0b:123456789
fcea920f7412b5da7be0cf42b8c93759:1234567
25d55ad283aa400af464c76d713c07ad:12345678
d8578edf8458ce06fbc5bb76a58c5ca4:qwerty
96e79218965eb72c92a549dd5a330112:111111
7c6a180b36896a0a8c02787eeafb0e4c:password1
6c569aabbf7775ef8fc570e228c16b98:password!
3f230640b78d7e71ac5514e57935eb69:qazxsw
f6a0cb102c62879d397b12b62c092c06:bluered
917eb5e9d6d6bca820922a0c6f7cc28b:Pa$$word1
kali@kali:~$ ^C
kali@kali:~$
```

Brief about process followed:

I used hashcat on KaliLinux to crack the passwords. Was successful in cracking 13 out of the 19 hash values given.

The hashing algorithm used was MD5. By trial and error when applying MDS as the hashing algorithm parameter I was able to crack the passwords.

The current mechanism offers a relatively low level of protection for the passwords. It does hash the passwords making the hackers job a little more tedious but also MD5 hash algorithm is a known cryptographically weak algorithm, with collisions reported and has numerous vulnerabilities. More robust algorithms like SHA-256 should be utilised as it is much more difficult to compute.

The next time a password database file leaks it should be encrypted so that the hashvalues itself should not be known to the attacker. Furthermore we can add salt values(random values) to the plaintext password and then hash them further complicating the hash value and this will also create completely different hash values even if the password is the same, which is a huge advantage and will make the attackers job that much more cumbersome. The salt values however have to be random and unique everytime, as otherwise it defeats the purpose and attackets can simply append a common list of salt values to the passwords and compute the hash values.

The passwords used in this list are very short and easy and was cracked in barely 10 seconds. It is important that passwords be 8-16 charecters long and should be alpha-numeric and consists of Uppercase and lowercase letters and special charcters as well. This password policy should be enforced in all companies and

organisations. This will simple rule will make our password extremely strong and take a hacker thousands of years to crack our passwords.

After the conducted analysis it was determined that organization uses an outdated password hashing algorithm (MD5) which offers very little protection in the event of a password database leaking. It was also determined that the current password policy is not aligned with industry best practices allowing users to have short passwords (6 characters) and reuse usernames as part of passwords.

As a result of the analysis the following uplifts are proposed to increase the overall level of password protection:

- Use a dedicated password hashing algorithm bcrypt, scrypt or PBKDF2 as this will greatly increase the time needed to crack individual passwords,
- Implement salting to prevent usage of rainbow tables to speed up cracking,
- Increase the minimum password length requirement to 10 characters – this will increase the computational effort required to crack password and will give additional time to change all passwords in the event of the password database being leaked,
- Prevent passwords to be the same as usernames or reused as part of the password – such password combination is easy to check without gaining access to the password database itself.
- It is advised to educate users on creating safe and easy to remember passwords. Having a password policy requiring long passwords with a number of special characters results in user writing passwords down or constantly resetting them. The best way to create a strong and user-friendly password is using passphrases (e.g. mygrannyschairhadstaples). The best way to create such passwords is to combine a couple of completely random word. It's also advised to use some special characters and numbers as easy to remember substitutions to expand the key space (e.g. mYgranny$cha1rhadstaples)
- Educate users on the benefits of passwords managers. Having a password manager allows having very long and completely random passwords (e.g. M>?{tk6Cfep6BrZ4J)KZWQ8j) without the need to remember/write down. A strong passphrase is still required as a master key for to access the password manager.