Project 3c. Cracking Linux password with hashcat using AWS t2.micro CPU

In this project we learn

- How to clone an AWS t2.micro instance from an image created by me from Ubuntu Linux distribution with cracking password hashcat software.
- How to crack a password using dictionary lookup.
- How to crack a password based on a known pattern.

The assignment will be graded by checking

1. Whether the learner know how to clone the instance and connect to it.
2. Whether the learner can use hashcat with the dictionary look up to find the password.
3. Whether the learner can specify the password pattern for cracking password using mask feature provided by hashcat.

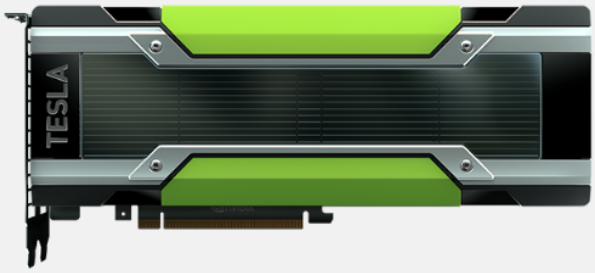
Note that since AWS Educate only allow you to run t2.micro free tier instance. We can only utilize the Xeon CPU provided to run the hashcat software to crack the password. It is much slower than using the GPU instances that are available if you have a regular AWS account. But we can still can crack the passwords for the exercises in this project within reasonable amount of time, e.g., the password in shadow2.txt with 6 digits unknown pattern was cracked in 25 second with p2.xlarge GPU but on a Xeon CPU it takes 7 mins, 46 seconds. It is almost 19 times faster with p2.xlarge GPU instance. For password in top100password list, the hashcat with Xeon CPU can crack two passwords within a second. If you have regular AWS account and willing to pay for p2.xlarge GPU instance usage (just a few minutes☺), I strongly recommend you to do that to gain the experience of speed difference.

You can find the more recent price information about AWS offering at
https://aws.amazon.com/ec2/instance-types/p2/

Name	GPUs	vCPUs	RAM (GiB)	Network Bandwidth	Price/Hour*	RI Price / Hour**
p2.xlarge	1	4	61	High	$0.900	$0.425
p2.8xlarge	8	32	488	10 Gbps	$7.200	$3.400
p2.16xlarge	16	64	732	20 Gbps	$14.400	$6.800

TESLA K80 ACCELERATOR FEATURES AND BENEFITS

- 4992 NVIDIA CUDA cores with a dual-GPU design
- Up to 2.91 teraflops double-precision performance with NVIDIA GPU Boost
- Up to 8.73 teraflops single-precision performance with NVIDIA GPU Boost
- 24 GB of GDDR5 memory
- 480 GB/s aggregate memory bandwidth
- ECC protection for increased reliability
- Server-optimized to deliver the best throughput in the data center



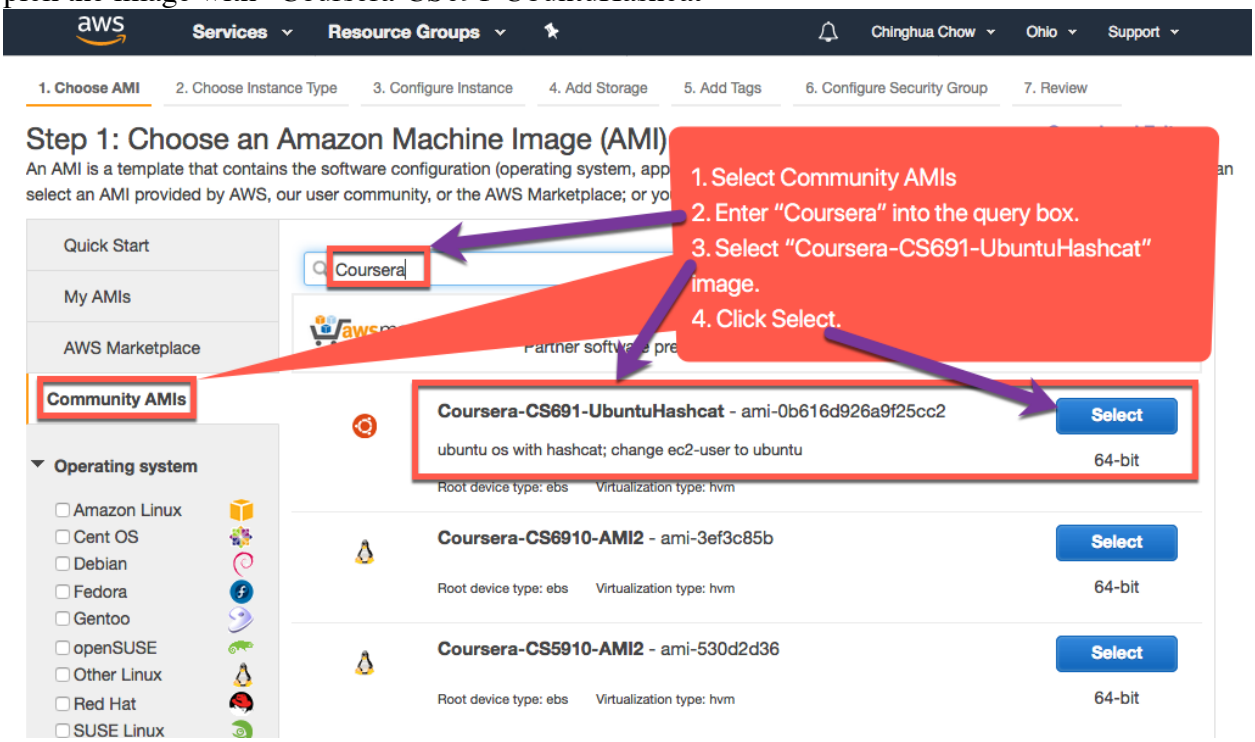
For our exercise, you only need less than an hour of time using p2.xlarge, no need to pick the bigger one. Actually each hacking session only takes about 30 seconds for the type of passwords we have. There are students reported using t2.micro with hashcat and was able to perform dictionary attack in Step 2 in very short time.
If you prepare the hash data on your own computer, then transfer the hash data to your p2 instance, you could save a lot of time.

Stop your instance right away when you finish your exercise. For p2.xlarge instance, it is almost a dollar an hour!

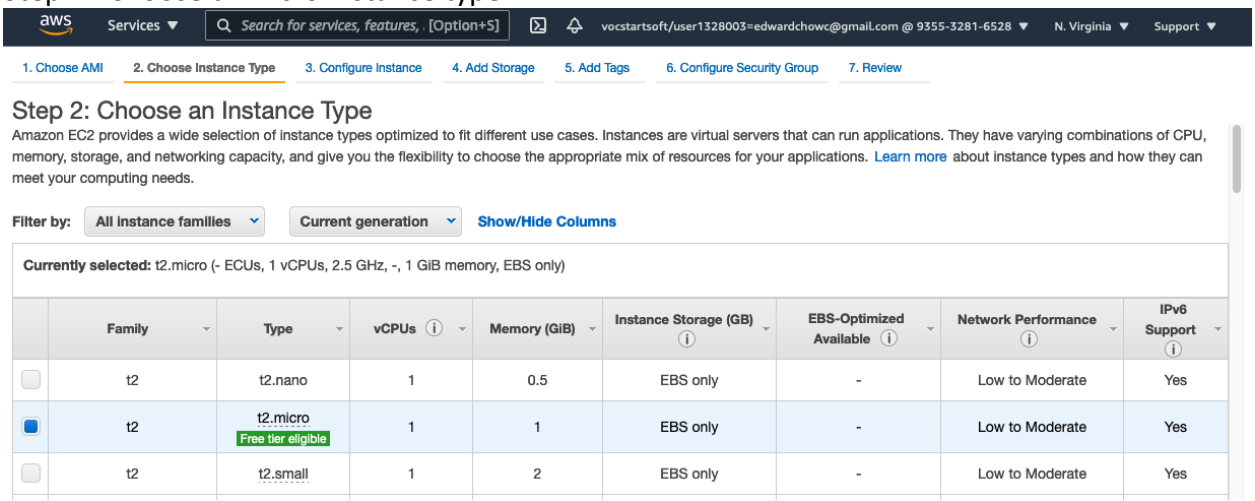
Step A. Clone the P2.xlarge instance from Coursera-CS691-UbuntuHashcat AMIs.

After login to your AWS management console, select ec2 service and click “Launch Instance”.

Step 1. Choosing AMI, Select Community AMIs, enter Coursera-CS691 to the query box, then pick the image with "Coursera-CS691-UbuntuHashcat"

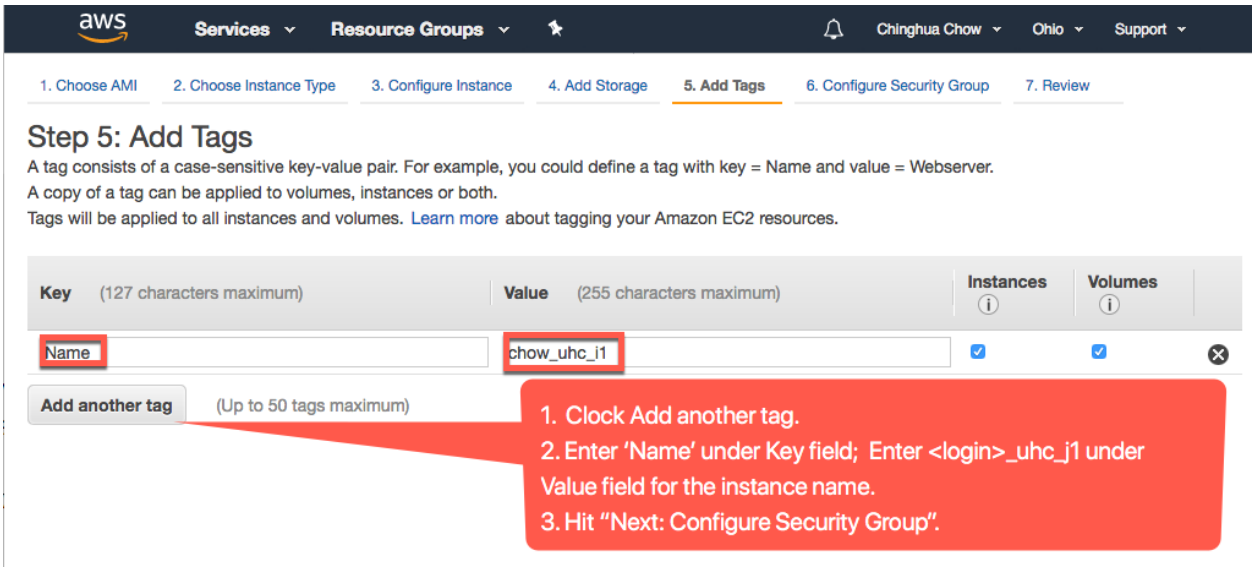


Step 2. Choose t2.micro instance type.

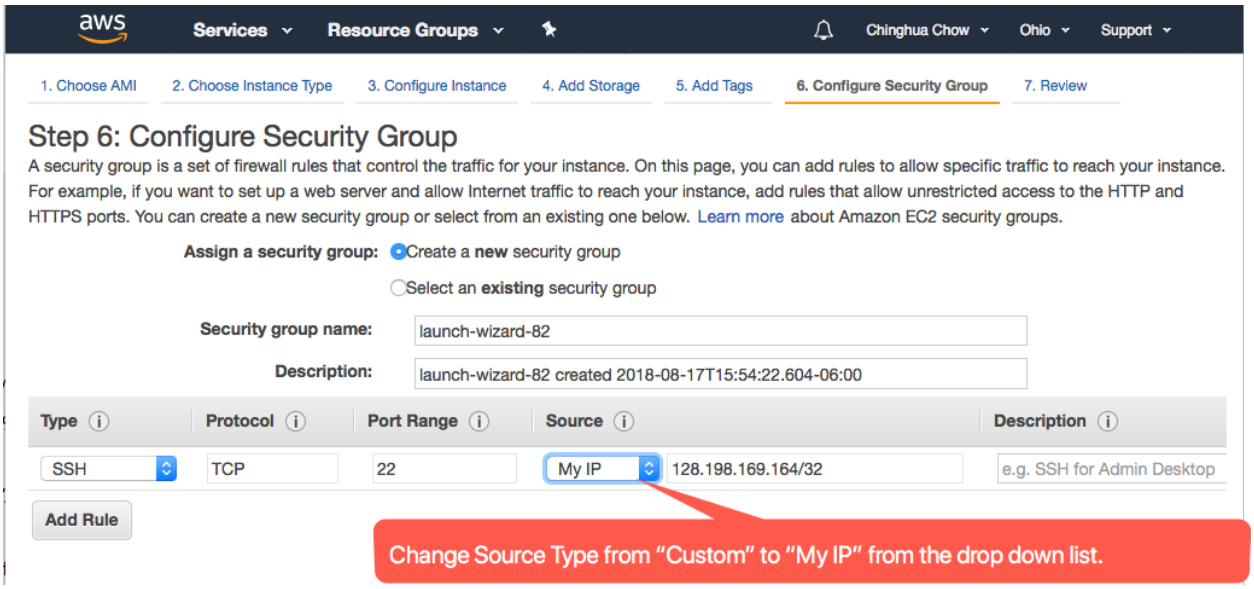


Step 3 and Step 4. Choose default settings.

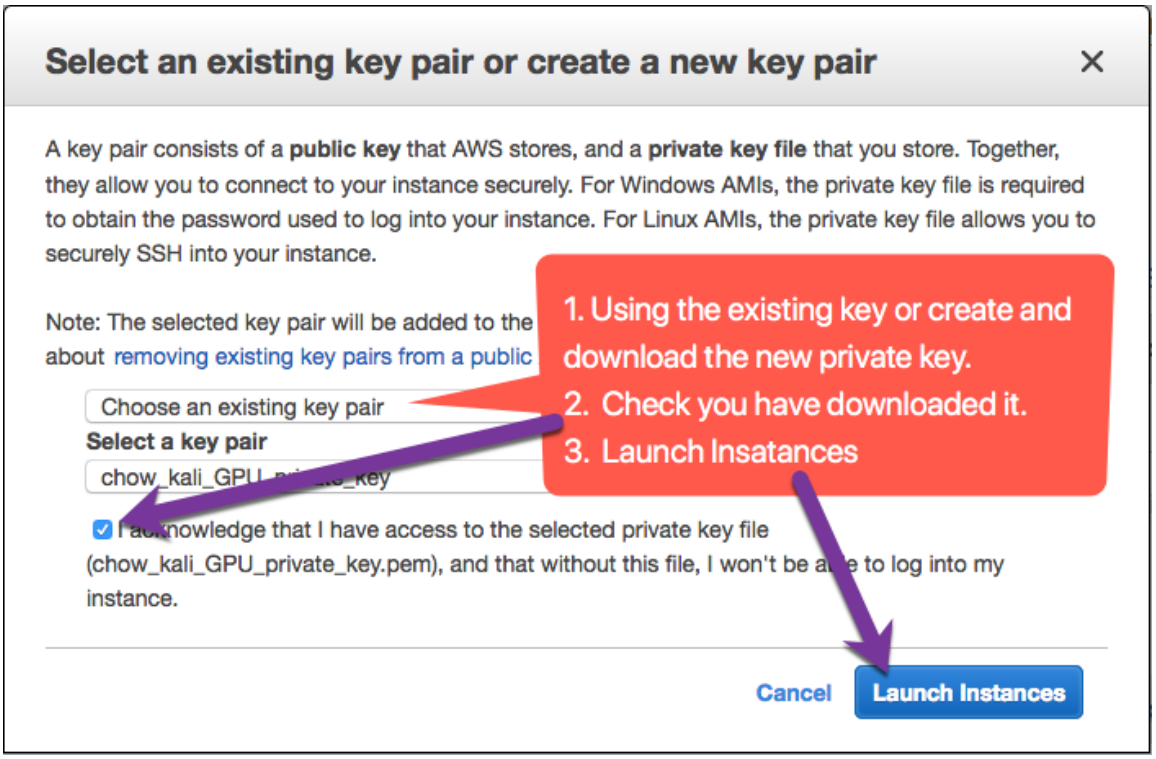
Step 5. We need to create a new tag with Name as key and <yourLogin>_uhc_i1 as value for instance name.



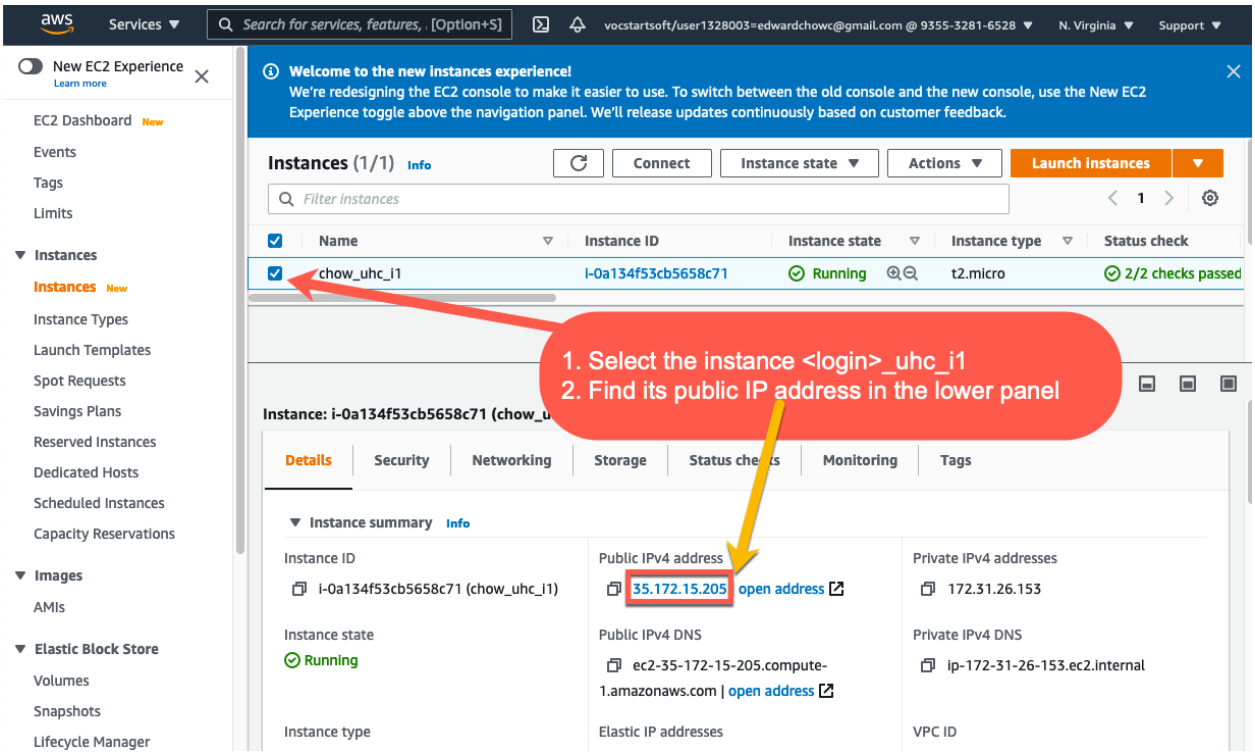
Step 6. Make sure you choose My IP for the source type. To avoid the instance being hacked.



Step 7. Review. Choose existing key pair you already have, or create a new private key for download.



Go to the instance list and select instance just created. Its name is <login>_uhc_i1.
Wait for it status to change from pending, initializing, to running.
Select it and find out its public IP address in the lower panel.



I found the chow_ubc_i1 instance is associated with 35.172.15.205 public IP address.
Note that the public IP address will change if we stop and start it again. You can create and associate it with an elastic IP address to make its public IP address "permanent".

Step B. Connect to the Instance using ssh command (Mac/Linux) or bitvise/putty (Windows).

Here we show how to connect on a maclaptop. First, make sure your private key file is only readable by you.
For example, I run the following command before I attempt to launch the ssh command.

chmod 644 chow_kali_GPU_private_key.pem

We will the use following command on a macOS or Linux.

ssh –i <private key file> ubuntu@<yourInstanceIPAddr>.

Note that on ubuntu image, the first user is ubuntu. This is unlike AMI Linux instance where the first user is ec2-user.

For Windows users, you can set up putty/bitwise. Follow the instructions in Section 3.2 of http://ciast.uccs.edu/coursera/pub/project1aV3.pdf to setup putty or bitwise for ssh access to your instance.

[en186-macl7:coursera/ec2/privateKey] cchow% ssh -i chow_kali_GPU_private_key.pem ubuntu@18.219.30.27
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-1065-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

266 packages can be updated.
0 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Aug 17 06:18:30 2018 from 75.71.209.54
ubuntu@ip-172-31-19-171:~$ ls
hcmask phpbb.txt shadow1.txt shadow2.txt top100passwd.txt

The home directory already contains a few data files for our exercises.
The instance is installed and tuned with new hashcat software package.

Step C. Benchmark Hashcat.

Normally to see if hashcat works on the ubuntu instance, type in “hashcat -b”. It will generate statistics about various hashing functions it supports.
But in our case, we do not have GPU or optimized kernel installed, therefore “hashcat --force -b” actually generate segmenation fault. We will skill this step.

For those use p2.xlarge instance, you can try “hashcat -b”. You can hit control-c key after seeing sha512 method. You do not have wait for it to complete.

Step D. Discover the password using the dictionary lookup.

Here we will demonstrate to use hashcat to discover passwords if it is within the dictionary of top100 passwords. We will select the two passwords from a popular top 100 password list. Create two user accounts on ubuntu with these two passwords. We will then extract the encrypted passwords from the /etc/shadow file, and feed them to the hashcat program with the dictionary file as inputs.

Step D1. Use a browser to access the following site and copy top100 password list from it

https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100.txt

Create a file with filename, top100passwd.txt, and save the top 100 passwords as content.
We will use this list for ***dictionary lookup password cracking***.

Step D2. Pick two of those passwords in the list and create two Linux user accounts.

In the list, I pick 666666 and access respectively for csr and nsa. You can pick different ones in the list of that top 100 passwords.
Create two users: csr and nsa using

```
ubuntu@ip-172-31-26-153:~$ sudo useradd csr
ubuntu@ip-172-31-26-153:~$ sudo useradd nsa
```

Set their passwords using

```
ubuntu@ip-172-31-26-153:~$ sudo passwd csr
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
ubuntu@ip-172-31-26-153:~$ sudo passwd nsa
Enter new UNIX password:
Retype new UNIX password:
```

Copy the /etc/shadow password file to current directory using

```
ubuntu@ip-172-31-26-153:~$ sudo cp /etc/shadow shadow.txt
ubuntu@ip-172-31-26-153:~$ sudo chown ubuntu:ubuntu shadow.txt
```

remove all other entries and only leave the last two.
It will be something like this:

```
ubuntu@ip-172-31-26-153:~$ cat shadow.txt
csr:$6$wrGtJUbA$tExdPUKg8aEXrmyJ11cUkhCQloNy.XUrzTVtSHXCClKpVAorXNC8TztKqR2wAuWbga6Y.sZWwpHXCX1RE4kpY0:17425:0:99999:7:::
nsa:$6$wfQm.MJt$NK.xz71C1q5GVix77FfTvtjg1C2KgC7CtfCeeaa.BZqh6fxBwj8txvuZvyifrdApJ8NTv/r7T/Wvd447XJgb.0:17425:0:99999:7:::
```

Step D3. Extract encrypted passwords from the password file.

Since hashcat only deal with hash password portion of the password, we remove the account name and the : after it. We also remove the ":17425:0:99999:7:::" at the end. Note that the

numbers before "0:99999:7:::" may be different. If do not remove this, the hashcat will complain that the length of the hash is too long.

The resulting file should like this:

6wrGtJUbA$tExdPUKg8aEXrmyJ11cUkhCQloNy.XUrzTVtSHXCClKpVAorXNC8TztKqR2wAuWbga6Y.sZWwpHXCX1RE4kpY0
6wfQm.MJt$NK.xz71C1q5GVix77FfTvtjg1C2KgC7CtfCeeaa.BZqh6fxBwj8txvuZvyifrdApJ8NTv/r7T/Wvd447XJgb.0

The "Features in glibc" Section in the crypt() function man page described in
http://man7.org/linux/man-pages/man3/crypt.3.html
the format of the encrypted password with three fields separated by '$': idsalt@encrypted
The first field indicates the encryption method or Linux system password mode. It has the value of 6, which indicate SHA-512 hashing method is used.
The second field is the salt. It is the 16 characters encoded in BASE64 format. It is used to defend against brute force dictionary look up attack.
The third field is the encrypted value after the plain password and salt is run through the hashing method, sometime multiple rounds.

Now let us run the hashcat command.

Step D4. Cracking password with dictionary lookup.

We will use the following command:

```
hashcat --force -m 1800 -o found1.txt shadow.txt top100passwd.txt
```

where --force indicates to hashcat we do not have special GPU device, just use CPU
-m 1800 specifies the hash is related to Linux system password mode 6 using SHA512.
-o specifies the output will be saved in found1.txt file. The first parameter is the file containing the encrypted password. The second parameter is the dictionary file.

Without force option, you will get the following error:

```
ubuntu@ip-172-31-26-153:~$ hashcat -m 1800 -o found1.txt shadow.txt top100passwd.txt
hashcat (v4.1.0) starting...

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
            You can use --force to override, but do not report related errors.
No devices found/left.

Started: Sat Mar  6 22:08:40 2021
Stopped: Sat Mar  6 22:08:40 2021
```

Here is the result:

ubuntu@ip-172-31-26-153:~$ hashcat --force -m 1800 -o found1.txt shadow.txt top100passwd.txt
hashcat (v4.1.0) starting...

OpenCL Platform #1: The pocl project
==========
* Device #1: pthread-Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz, 256/743 MB allocatable, 1MCU

Hashes: 2 digests; 2 unique digests, 2 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Uses-64-Bit

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

ATTENTION! Pure (unoptimized) OpenCL kernels selected.
This enables cracking passwords and salts > length 32 but for the price of drastically reduced performance.
If you want to switch to optimized OpenCL kernels, append -O to your commandline.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D VENDOR_ID=64 -D CUDA_ARCH=0 -D AMD_ROCM=0 -D VECT_SIZE=4 -D DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=2 -D DGST_R3=3 -D DGST_ELEM=16 -D KERN_TYPE=1800 -D _unroll'
* Device #1: Kernel m01800.080f4403.kernel not found in cache! Building may take a while...
* Device #1: Kernel amp_a0.d878b11b.kernel not found in cache! Building may take a while...
Dictionary cache hit:
* Filename..: top100passwd.txt
* Passwords.: 100
* Bytes.....: 744
* Keyspace..: 100

The wordlist or mask that you are using is too small.

This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

Session..........: hashcat
Status...........: Cracked
Hash.Type........: sha512crypt 6, SHA512 (Unix)
Hash.Target......: shadow.txt
Time.Started.....: Sat Mar 6 22:09:12 2021 (1 sec)
Time.Estimated...: Sat Mar 6 22:09:13 2021 (0 secs)
Guess.Base.......: File (top100passwd.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:   318 H/s (3.93ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered........: 2/2 (100.00%) Digests, 2/2 (100.00%) Salts
Progress.........: 200/200 (100.00%)
Rejected.........: 0/200 (0.00%)
Restore.Point....: 0/100 (0.00%)
Candidates.#1....: 123456 -> matrix
HWMon.Dev.#1.....: N/A

Started: Sat Mar 6 22:09:02 2021
Stopped: Sat Mar 6 22:09:14 2021
ubuntu@ip-172-31-26-153:~$ cat found1.txt
6a7sRdudT$zy4W8fbr3zkftZfnFBcTlILfh1siJ1J8OXAtzCy.ot9Wt6H/n3gY7AWTkr9/uEXjKPhtfyN7mMRElG.pAH.cw.:666666
6SL9vLE3j$.ofQ3DNjhQM9YF.OpzxWKqVOXgvDUZgaRJx6wm2sHh6DIoei/NrO7d4UyS2.oEYds9uTja7hieghLMmeZ80PI.:access

Found.1 content below shows that the passwords are indeed 666666 and access.

With dictionary lookup, it only takes about 1 second. Note that if you run the command again, it will return with cached results right away. This is due to hashcat software cached/kept track all cracked results. You can turn off searching the cache results with option
--potfile-disable

Step E. Cracking Linux password with known simple pattern.

Step E1. Retrieve password files with special encoding pattern.

Assume that we are able to retrieve recent Linux shadow files with the content through the vulnerability of the Linux system. Assume that the three shadow files can be downloaded with the following three wget commands:

wget http://ciast.uccs.edu/coursera/pub/shadow3
wget http://ciast.uccs.edu/coursera/pub/shadow2
wget http://ciast.uccs.edu/coursera/pub/shadow1

ubuntu@ip-172-31-26-153:~$ **cat shadow3**
amca:$6$9XO7r56HUdnP4BVM$XC47J/U9ZHyE4RL4l9P9Ps6zXZnZofraeukRZXmTCuXE2P8CUAO
nbFjbSYLAR7eReNTcOvDV45vgAzm70JCcv/:17402:0:99999:7:::
ubuntu@ip-172-31-26-153:~$ **cp shadow3 shadow3.txt**

As before, let us edit shadow3.txt file and remove other info except the hash. In this case, any of the above text marked yellow. The results should be as follows:

ubuntu@ip-172-31-26-153:~$ cat shadow3.txt
$6$9XO7r56HUdnP4BVM$XC47J/U9ZHyE4RL4l9P9Ps6zXZnZofraeukRZXmTCuXE2P8CUAOnbFjb
SYLAR7eReNTcOvDV45vgAzm70JCcv/

Step E2. Create mask file containing the pattern.

Through the email announcement to the users, we also know the password was created with #a followed by 9 digits of student ID. The question is "How can we utilize the knowledge of this pattern and hashcat to discover the password?"

The hashcat supports the pattern matching. The mask file can be created with .hcmask file extension. Each line in the mask file is one pattern. ?d present digits, ?l represent letter.
In our case, we further assume some of leading 3 digits are 101
Now the pattern in the mask file becomes

```
\#a101?d?d?d?d?d?d
```

Let us save this line as hcmask

Step E3. Cracking passwords with encoding patterns.

Now let us apply the hashcat command for the pattern search with hcmask. Know that the software will prompt you to decide whether to check status, pause, or quit, while it is running. In the session below, I type 's' a few times to see the progress. It shows how many patterns in terms of percentage have been searched.

ubuntu@ip-172-31-26-153:~$ hashcat --force -m 1800 -o found2a.txt -a 3 shadow2.txt hcmask
hashcat (v4.1.0) starting...

OpenCL Platform #1: The pocl project
==========
* Device #1: pthread-Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz, 256/743 MB allocatable, 1MCU

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

Applicable optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force
* Uses-64-Bit

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

ATTENTION! Pure (unoptimized) OpenCL kernels selected.
This enables cracking passwords and salts > length 32 but for the price of drastically reduced performance.
If you want to switch to optimized OpenCL kernels, append -O to your commandline.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D VENDOR_ID=64 -D CUDA_ARCH=0 -D AMD_ROCM=0 -D VECT_SIZE=4 -D DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=2 -D DGST_R3=3 -D DGST_ELEM=16 -D KERN_TYPE=1800 -D _unroll'
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s # you can continue to hit s to let hashcat report status; and watch the “progress:” line for the percentage of patterns compared

Session..........: hashcat
Status...........: Running
Hash.Type........: sha512crypt 6, SHA512 (Unix)
Hash.Target......: 6mvGQ9ZN.JvN8XT5F$mJC8rN3Liu4BGzX3/oWFHOIpi/AcfXQ...Me0xa.
Time.Started.....: Sat Mar 6 22:20:11 2021 (1 sec)
Time.Estimated...: Sat Mar 6 23:12:03 2021 (51 mins, 51 secs)
Guess.Mask.......: #a101?d?d?d?d?d?d [11]

Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 321 H/s (10.07ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 256/1000000 (0.03%)
Rejected.....: 0/256 (0.00%)
Restore.Point....: 256/1000000 (0.03%)
Candidates.#1....: #a101550000 -> #a101205699
HWMon.Dev.#1.....: N/A

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s

Session.....: hashcat
Status.....: Running
Hash.Type.....: sha512crypt 6, SHA512 (Unix)
Hash.Target.....: 6mvGQ9ZN.JvN8XT5F$mJC8rN3Liu4BGzX3/oWFHOIpi/AcfXQ...Me0xa.
Time.Started.....: Sat Mar 6 22:20:11 2021 (3 secs)
Time.Estimated...: Sat Mar 6 23:12:14 2021 (52 mins, 0 secs)
Guess.Mask.....: #a101?d?d?d?d?d?d [11]
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 320 H/s (10.03ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 1024/1000000 (0.10%)
Rejected.....: 0/1024 (0.00%)
Restore.Point....: 1024/1000000 (0.10%)
Candidates.#1....: #a101323234 -> #a101750123
HWMon.Dev.#1.....: N/A

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s

Session.....: hashcat
Status.....: Running
Hash.Type.....: sha512crypt 6, SHA512 (Unix)
Hash.Target.....: 6mvGQ9ZN.JvN8XT5F$mJC8rN3Liu4BGzX3/oWFHOIpi/AcfXQ...Me0xa.
Time.Started.....: Sat Mar 6 22:20:11 2021 (4 mins, 58 secs)
Time.Estimated...: Sat Mar 6 23:12:17 2021 (47 mins, 8 secs)
Guess.Mask.....: #a101?d?d?d?d?d?d [11]
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 320 H/s (10.00ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 95488/1000000 (9.55%)
Rejected.....: 0/95488 (0.00%)
Restore.Point....: 95488/1000000 (9.55%)
Candidates.#1....: #a101686299 -> #a101028645
HWMon.Dev.#1.....: N/A

Session.....: hashcat
Status.....: Cracked
Hash.Type.....: sha512crypt 6, SHA512 (Unix)
Hash.Target.....: 6mvGQ9ZN.JvN8XT5F$mJC8rN3Liu4BGzX3/oWFHOIpi/AcfXQ...Me0xa.
Time.Started.....: Sat Mar 6 22:20:11 2021 (7 mins, 46 secs)
Time.Estimated...: Sat Mar 6 22:27:57 2021 (0 secs)
Guess.Mask.....: #a101?d?d?d?d?d?d [11]
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....:   320 H/s (10.07ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 148992/1000000 (14.90%)
Rejected.....: 0/148992 (0.00%)
Restore.Point....: 148736/1000000 (14.87%)
Candidates.#1....: #a101518441 -> #a101297441
HWMon.Dev.#1.....: N/A

Started: Sat Mar 6 22:20:09 2021
Stopped: Sat Mar 6 22:27:59 2021
ubuntu@ip-172-31-26-153:~$ cat found2a.txt
6mvGQ9ZN.JvN8XT5F$mJC8rN3Liu4BGzX3/oWFHOIpi/AcfXQtaw.0EUGPXw.4PZog54yuhYh4s
ybXpMKH9sPd41QLYldXrlFmMe0xa.:#a101465920

The password #a101465920 is discovered.
Note that we are lucky here that it only takes 14.9% of the patterns comparison to find the match. Others cases may not be that quick, since on average it will take 50% of pattern matching. In worst case you need to compare all 1000000 patterns and it may take about 54 mins. Average case it will take about 27 min.

Let us repeat the process for shadow1 and shadow3, and report the passwords discovered. Also compare the time and percentage it takes to discover the passwrods.

ubuntu@ip-172-31-26-153:~$ hashcat--force -m 1800 -o found2b.txt -a 3 shadow1.txt hcmask
ubuntu@ip-172-31-26-153:~$ hashcat--force -m 1800 -o found2b.txt -a 3 shadow3.txt hcmask

For shadow1 password, we examine 23.96% of the patterns.

Deliverables: Save the above session text as hcresult.txt. Submit it as your deliverable of project3c.
Create passwd.txt which documents the five login and password cracked, submit it as the seconds deliverable of project3c.

Very important!!
Stop your instance right away when you finish your exercise. You should consider terminate it if you do not intend to crack passwords soon.

The following are the hash modes supported by hashcat: 1800 is one of them.

- [Hash modes] -

```
    # | Name                                               | Category
======+==================================================+==================
  900 | MD4                                              | Raw Hash
    0 | MD5                                              | Raw Hash
 5100 | Half MD5                                         | Raw Hash
  100 | SHA1                                             | Raw Hash
 1300 | SHA-224                                          | Raw Hash
 1400 | SHA-256                                          | Raw Hash
10800 | SHA-384                                          | Raw Hash
 1700 | SHA-512                                          | Raw Hash
 5000 | SHA-3(Keccak)                                    | Raw Hash
10100 | SipHash                                          | Raw Hash
 6000 | RipeMD160                                        | Raw Hash
 6100 | Whirlpool                                        | Raw Hash
 6900 | GOST R 34.11-94                                  | Raw Hash
11700 | GOST R 34.11-2012 (Streebog) 256-bit             | Raw Hash
11800 | GOST R 34.11-2012 (Streebog) 512-bit             | Raw Hash
   10 | md5($pass.$salt)                                 | Raw Hash, Salted and / or Iterated
   20 | md5($salt.$pass)                                 | Raw Hash, Salted and / or Iterated
   30 | md5(unicode($pass).$salt)                        | Raw Hash, Salted and / or Iterated
   40 | md5($salt.unicode($pass))                        | Raw Hash, Salted and / or Iterated
 3800 | md5($salt.$pass.$salt)                           | Raw Hash, Salted and / or Iterated
 3710 | md5($salt.md5($pass))                            | Raw Hash, Salted and / or Iterated
 2600 | md5(md5($pass))                                  | Raw Hash, Salted and / or Iterated
 4300 | md5(strtoupper(md5($pass)))                      | Raw Hash, Salted and / or Iterated
 4400 | md5(sha1($pass))                                 | Raw Hash, Salted and / or Iterated
  110 | sha1($pass.$salt)                                | Raw Hash, Salted and / or Iterated
  120 | sha1($salt.$pass)                                | Raw Hash, Salted and / or Iterated
  130 | sha1(unicode($pass).$salt)                       | Raw Hash, Salted and / or Iterated
  140 | sha1($salt.unicode($pass))                       | Raw Hash, Salted and / or Iterated
 4500 | sha1(sha1($pass))                                | Raw Hash, Salted and / or Iterated
 4700 | sha1(md5($pass))                                 | Raw Hash, Salted and / or Iterated
 4900 | sha1($salt.$pass.$salt)                          | Raw Hash, Salted and / or Iterated
14400 | sha1(CX)                                         | Raw Hash, Salted and / or Iterated
```

1410 | sha256($pass.$salt) | Raw Hash, Salted and / or Iterated
1420 | sha256($salt.$pass) | Raw Hash, Salted and / or Iterated
1430 | sha256(unicode($pass).$salt) | Raw Hash, Salted and / or Iterated
1440 | sha256($salt.unicode($pass)) | Raw Hash, Salted and / or Iterated
1710 | sha512($pass.$salt) | Raw Hash, Salted and / or Iterated
1720 | sha512($salt.$pass) | Raw Hash, Salted and / or Iterated
1730 | sha512(unicode($pass).$salt) | Raw Hash, Salted and / or Iterated
1740 | sha512($salt.unicode($pass)) | Raw Hash, Salted and / or Iterated
50 | HMAC-MD5 (key = $pass) | Raw Hash, Authenticated
60 | HMAC-MD5 (key = $salt) | Raw Hash, Authenticated
150 | HMAC-SHA1 (key = $pass) | Raw Hash, Authenticated
160 | HMAC-SHA1 (key = $salt) | Raw Hash, Authenticated
1450 | HMAC-SHA256 (key = $pass) | Raw Hash, Authenticated
1460 | HMAC-SHA256 (key = $salt) | Raw Hash, Authenticated
1750 | HMAC-SHA512 (key = $pass) | Raw Hash, Authenticated
1760 | HMAC-SHA512 (key = $salt) | Raw Hash, Authenticated
14000 | DES (PT = $salt, key = $pass) | Raw Cipher, Known-Plaintext attack
14100 | 3DES (PT = $salt, key = $pass) | Raw Cipher, Known-Plaintext attack
400 | phpass | Generic KDF
8900 | scrypt | Generic KDF
11900 | PBKDF2-HMAC-MD5 | Generic KDF
12000 | PBKDF2-HMAC-SHA1 | Generic KDF
10900 | PBKDF2-HMAC-SHA256 | Generic KDF
12100 | PBKDF2-HMAC-SHA512 | Generic KDF
23 | Skype | Network protocols
2500 | WPA/WPA2 | Network protocols
4800 | iSCSI CHAP authentication, MD5(Chap) | Network protocols
5300 | IKE-PSK MD5 | Network protocols
5400 | IKE-PSK SHA1 | Network protocols
5500 | NetNTLMv1 | Network protocols
5500 | NetNTLMv1 + ESS | Network protocols
5600 | NetNTLMv2 | Network protocols
7300 | IPMI2 RAKP HMAC-SHA1 | Network protocols
7500 | Kerberos 5 AS-REQ Pre-Auth etype 23 | Network protocols
8300 | DNSSEC (NSEC3) | Network protocols
10200 | Cram MD5 | Network protocols
11100 | PostgreSQL CRAM (MD5) | Network protocols
11200 | MySQL CRAM (SHA1) | Network protocols
11400 | SIP digest authentication (MD5) | Network protocols
13100 | Kerberos 5 TGS-REP etype 23 | Network protocols
121 | SMF (Simple Machines Forum) | Forums, CMS, E-Commerce, Frameworks
400 | phpBB3 | Forums, CMS, E-Commerce, Frameworks
2611 | vBulletin < v3.8.5 | Forums, CMS, E-Commerce, Frameworks
2711 | vBulletin > v3.8.5 | Forums, CMS, E-Commerce, Frameworks

2811 | MyBB | Forums, CMS, E-Commerce, Frameworks
2811 | IPB (Invison Power Board) | Forums, CMS, E-Commerce, Frameworks
8400 | WBB3 (Woltlab Burning Board) | Forums, CMS, E-Commerce, Frameworks
11 | Joomla < 2.5.18 | Forums, CMS, E-Commerce, Frameworks
400 | Joomla > 2.5.18 | Forums, CMS, E-Commerce, Frameworks
400 | Wordpress | Forums, CMS, E-Commerce, Frameworks
2612 | PHPS | Forums, CMS, E-Commerce, Frameworks
7900 | Drupal7 | Forums, CMS, E-Commerce, Frameworks
21 | osCommerce | Forums, CMS, E-Commerce, Frameworks
21 | xt:Commerce | Forums, CMS, E-Commerce, Frameworks
11000 | PrestaShop | Forums, CMS, E-Commerce, Frameworks
124 | Django (SHA-1) | Forums, CMS, E-Commerce, Frameworks
10000 | Django (PBKDF2-SHA256) | Forums, CMS, E-Commerce, Frameworks
3711 | Mediawiki B type | Forums, CMS, E-Commerce, Frameworks
7600 | Redmine | Forums, CMS, E-Commerce, Frameworks
13900 | OpenCart | Forums, CMS, E-Commerce, Frameworks
12 | PostgreSQL | Database Server
131 | MSSQL(2000) | Database Server
132 | MSSQL(2005) | Database Server
1731 | MSSQL(2012) | Database Server
1731 | MSSQL(2014) | Database Server
200 | MySQL323 | Database Server
300 | MySQL4.1/MySQL5 | Database Server
3100 | Oracle H: Type (Oracle 7+) | Database Server
112 | Oracle S: Type (Oracle 11+) | Database Server
12300 | Oracle T: Type (Oracle 12+) | Database Server
8000 | Sybase ASE | Database Server
141 | EPiServer 6.x < v4 | HTTP, SMTP, LDAP Server
1441 | EPiServer 6.x > v4 | HTTP, SMTP, LDAP Server
1600 | Apache $apr1$ | HTTP, SMTP, LDAP Server
12600 | ColdFusion 10+ | HTTP, SMTP, LDAP Server
1421 | hMailServer | HTTP, SMTP, LDAP Server
101 | nsldap, SHA-1(Base64), Netscape LDAP SHA | HTTP, SMTP, LDAP Server
111 | nsldaps, SSHA-1(Base64), Netscape LDAP SSHA | HTTP, SMTP, LDAP Server
1711 | SSHA-512(Base64), LDAP {SSHA512} | HTTP, SMTP, LDAP Server
11500 | CRC32 | Checksums
3000 | LM | Operating-Systems
1000 | NTLM | Operating-Systems
1100 | Domain Cached Credentials (DCC), MS Cache | Operating-Systems
2100 | Domain Cached Credentials 2 (DCC2), MS Cache 2 | Operating-Systems
12800 | MS-AzureSync PBKDF2-HMAC-SHA256 | Operating-Systems
1500 | descrypt, DES(Unix), Traditional DES | Operating-Systems
12400 | BSDiCrypt, Extended DES | Operating-Systems
500 | md5crypt 1, MD5(Unix) | Operating-Systems

3200 | bcrypt $2*$, Blowfish(Unix) | Operating-Systems
7400 | sha256crypt 5, SHA256(Unix) | Operating-Systems
1800 | sha512crypt 6, SHA512(Unix) | Operating-Systems
122 | OSX v10.4, OSX v10.5, OSX v10.6 | Operating-Systems
1722 | OSX v10.7 | Operating-Systems
7100 | OSX v10.8, OSX v10.9, OSX v10.10 | Operating-Systems
6300 | AIX {smd5} | Operating-Systems
6700 | AIX {ssha1} | Operating-Systems
6400 | AIX {ssha256} | Operating-Systems
6500 | AIX {ssha512} | Operating-Systems
2400 | Cisco-PIX | Operating-Systems
2410 | Cisco-ASA | Operating-Systems
500 | Cisco-IOS 1 | Operating-Systems
5700 | Cisco-IOS 4 | Operating-Systems
9200 | Cisco-IOS 8 | Operating-Systems
9300 | Cisco-IOS 9 | Operating-Systems
22 | Juniper Netscreen/SSG (ScreenOS) | Operating-Systems
501 | Juniper IVE | Operating-Systems
5800 | Android PIN | Operating-Systems
13800 | Windows 8+ phone PIN/Password | Operating-Systems
8100 | Citrix Netscaler | Operating-Systems
8500 | RACF | Operating-Systems
7200 | GRUB 2 | Operating-Systems
9900 | Radmin2 | Operating-Systems
125 | ArubaOS | Operating-Systems
7700 | SAP CODVN B (BCODE) | Enterprise Application Software (EAS)
7800 | SAP CODVN F/G (PASSCODE) | Enterprise Application Software (EAS)
10300 | SAP CODVN H (PWDSALTEDHASH) iSSHA-1 | Enterprise Application Software (EAS)
8600 | Lotus Notes/Domino 5 | Enterprise Application Software (EAS)
8700 | Lotus Notes/Domino 6 | Enterprise Application Software (EAS)
9100 | Lotus Notes/Domino 8 | Enterprise Application Software (EAS)
133 | PeopleSoft | Enterprise Application Software (EAS)
13500 | PeopleSoft Token | Enterprise Application Software (EAS)
11600 | 7-Zip | Archives
12500 | RAR3-hp | Archives
13000 | RAR5 | Archives
13200 | AxCrypt | Archives
13300 | AxCrypt in memory SHA1 | Archives
13600 | WinZip | Archives
62XY | TrueCrypt | Full-Disk encryptions (FDE)
X | 1 = PBKDF2-HMAC-RipeMD160 | Full-Disk encryptions (FDE)
X | 2 = PBKDF2-HMAC-SHA512 | Full-Disk encryptions (FDE)
X | 3 = PBKDF2-HMAC-Whirlpool | Full-Disk encryptions (FDE)

```
  X  | 4 = PBKDF2-HMAC-RipeMD160 + boot-mode          | Full-Disk encryptions (FDE)
  Y | 1 = XTS  512 bit pure AES                    | Full-Disk encryptions (FDE)
  Y | 1 = XTS  512 bit pure Serpent                | Full-Disk encryptions (FDE)
  Y | 1 = XTS  512 bit pure Twofish                | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit pure AES                    | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit pure Serpent                | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit pure Twofish                | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit cascaded AES-Twofish        | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit cascaded Serpent-AES        | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit cascaded Twofish-Serpent    | Full-Disk encryptions (FDE)
  Y | 3 = XTS 1536 bit all                         | Full-Disk encryptions (FDE)
 8800 | Android FDE < v4.3                         | Full-Disk encryptions (FDE)
12900 | Android FDE (Samsung DEK)                  | Full-Disk encryptions (FDE)
12200 | eCryptfs                                   | Full-Disk encryptions (FDE)
137XY | VeraCrypt                                  | Full-Disk encryptions (FDE)
  X  | 1 = PBKDF2-HMAC-RipeMD160                  | Full-Disk encryptions (FDE)
  X  | 2 = PBKDF2-HMAC-SHA512                     | Full-Disk encryptions (FDE)
  X  | 3 = PBKDF2-HMAC-Whirlpool                  | Full-Disk encryptions (FDE)
  X  | 4 = PBKDF2-HMAC-RipeMD160 + boot-mode      | Full-Disk encryptions (FDE)
  X  | 5 = PBKDF2-HMAC-SHA256                     | Full-Disk encryptions (FDE)
  X  | 6 = PBKDF2-HMAC-SHA256 + boot-mode         | Full-Disk encryptions (FDE)
  Y | 1 = XTS  512 bit pure AES                    | Full-Disk encryptions (FDE)
  Y | 1 = XTS  512 bit pure Serpent                | Full-Disk encryptions (FDE)
  Y | 1 = XTS  512 bit pure Twofish                | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit pure AES                    | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit pure Serpent                | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit pure Twofish                | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit cascaded AES-Twofish        | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit cascaded Serpent-AES        | Full-Disk encryptions (FDE)
  Y | 2 = XTS 1024 bit cascaded Twofish-Serpent    | Full-Disk encryptions (FDE)
  Y | 3 = XTS 1536 bit all                         | Full-Disk encryptions (FDE)
 9700 | MS Office <= 2003 $0|$1, MD5 + RC4          | Documents
 9710 | MS Office <= 2003 $0|$1, MD5 + RC4, collider #1  | Documents
 9720 | MS Office <= 2003 $0|$1, MD5 + RC4, collider #2  | Documents
 9800 | MS Office <= 2003 $3|$4, SHA1 + RC4          | Documents
 9810 | MS Office <= 2003 $3|$4, SHA1 + RC4, collider #1 | Documents
 9820 | MS Office <= 2003 $3|$4, SHA1 + RC4, collider #2 | Documents
 9400 | MS Office 2007                             | Documents
 9500 | MS Office 2010                             | Documents
 9600 | MS Office 2013                             | Documents
10400 | PDF 1.1 - 1.3 (Acrobat 2 - 4)              | Documents
10410 | PDF 1.1 - 1.3 (Acrobat 2 - 4), collider #1     | Documents
10420 | PDF 1.1 - 1.3 (Acrobat 2 - 4), collider #2     | Documents
10500 | PDF 1.4 - 1.6 (Acrobat 5 - 8)              | Documents
```

10600 | PDF 1.7 Level 3 (Acrobat 9) | Documents
10700 | PDF 1.7 Level 8 (Acrobat 10 - 11) | Documents
9000 | Password Safe v2 | Password Managers
5200 | Password Safe v3 | Password Managers
6800 | Lastpass + Lastpass sniffed | Password Managers
6600 | 1Password, agilekeychain | Password Managers
8200 | 1Password, cloudkeychain | Password Managers
11300 | Bitcoin/Litecoin wallet.dat | Password Managers
12700 | Blockchain, My Wallet | Password Managers
13400 | Keepass 1 (AES/Twofish) and Keepass 2 (AES) | Password Managers
99999 | Plaintext | Plaintext