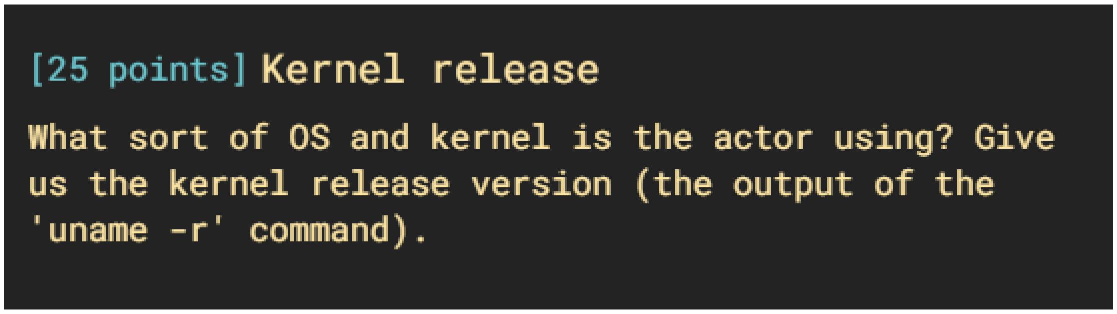
Challenge Information

A malicious actor has broken into our research centre and has stolen some important information. Help us investigate their confiscated laptop memory dump! Note: if you need to crack passwords during this challenge, all potential passwords apear in rockyou-75.txt. Note 2: the pcap is only relevant for the last subtask. Note 2: do NOT attempt to brute-force rockyou-75.txt against the flag submission - it will get your IP banned.

Flag 1:

Description: Kernel release



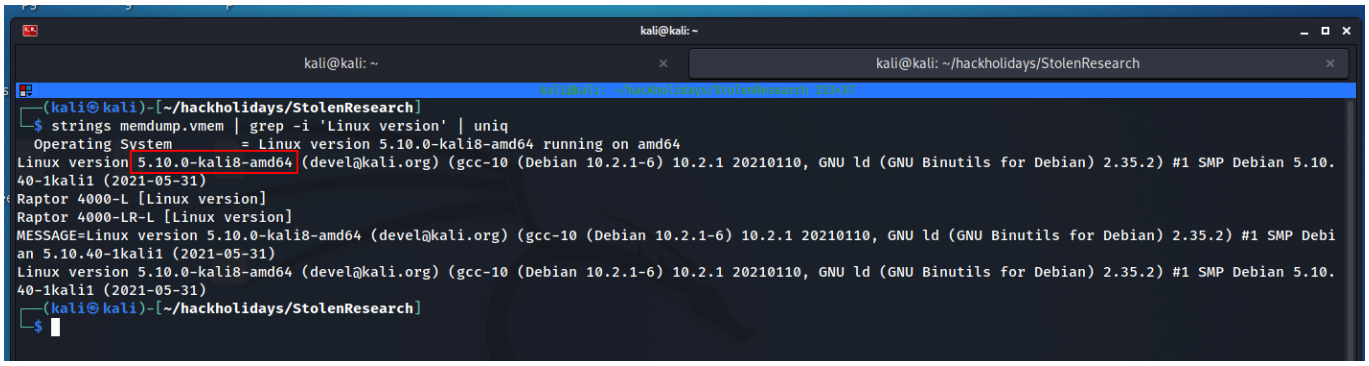
```
[25 points] Kernel release

What sort of OS and kernel is the actor using? Give
us the kernel release version (the output of the
'uname -r' command).
```

The following gives us the uname -r value of the downloaded memory file

Command→

```
strings memdump.vmem | grep -i 'Linux version' | uniq
```

Output→



```
┌──(kali㉿kali)-[~/hackholidays/StolenResearch]
└─$ strings memdump.vmem | grep -i 'Linux version' | uniq
  Operating System       = Linux version 5.10.0-kali8-amd64 running on amd64
Linux version 5.10.0-kali8-amd64 (devel@kali.org) (gcc-10 (Debian 10.2.1-6) 10.2.1 20210110, GNU ld (GNU Binutils for Debian) 2.35.2) #1 SMP Debian 5.10.
40-1kali1 (2021-05-31)
Raptor 4000-L [Linux version]
Raptor 4000-LR-L [Linux version]
MESSAGE=Linux version 5.10.0-kali8-amd64 (devel@kali.org) (gcc-10 (Debian 10.2.1-6) 10.2.1 20210110, GNU ld (GNU Binutils for Debian) 2.35.2) #1 SMP Debi
an 5.10.40-1kali1 (2021-05-31)
Linux version 5.10.0-kali8-amd64 (devel@kali.org) (gcc-10 (Debian 10.2.1-6) 10.2.1 20210110, GNU ld (GNU Binutils for Debian) 2.35.2) #1 SMP Debian 5.10.
40-1kali1 (2021-05-31)
┌──(kali㉿kali)-[~/hackholidays/StolenResearch]
└─$ 
```

Flag: 5.10.0-kali8-amd64

Setup Custom Volatility Profile

The first thing we need to is setup volatility with a custom profile:

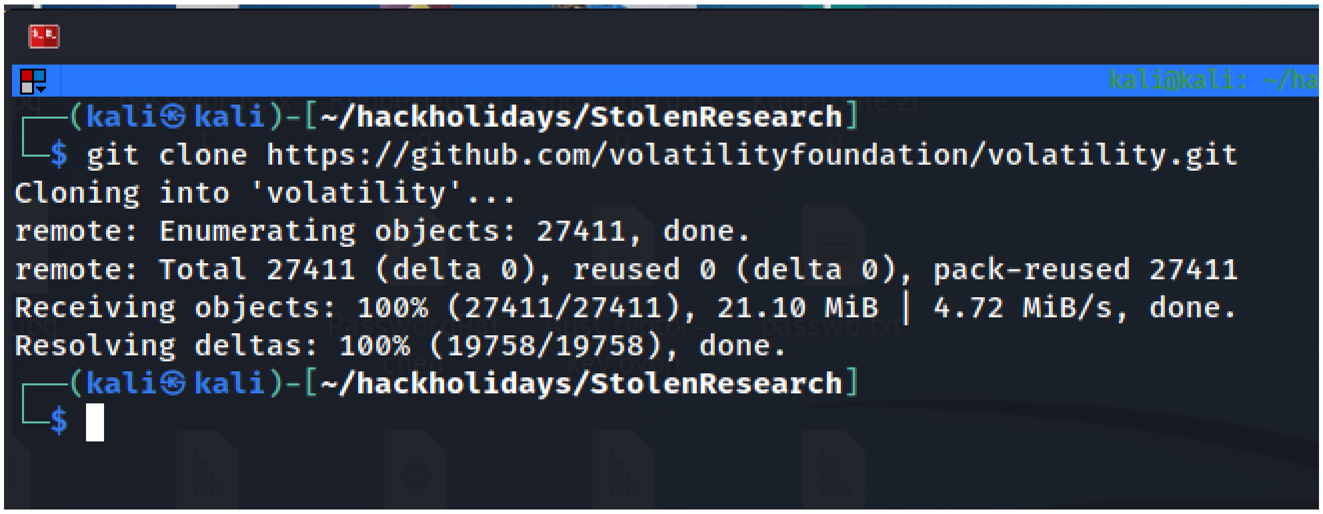
Step 1: Download volatility 2.6 from github & Install

https://github.com/volatilityfoundation/volatility

Command→

```
git clone https://github.com/volatilityfoundation/volatility.git
cd volatility/
sudo python setup.py install
```

Output→



```
┌──(kali㉿kali)-[~/hackholidays/StolenResearch]
└─$ git clone https://github.com/volatilityfoundation/volatility.git
Cloning into 'volatility'...
remote: Enumerating objects: 27411, done.
remote: Total 27411 (delta 0), reused 0 (delta 0), pack-reused 27411
Receiving objects: 100% (27411/27411), 21.10 MiB | 4.72 MiB/s, done.
Resolving deltas: 100% (19758/19758), done.
┌──(kali㉿kali)-[~/hackholidays/StolenResearch]
└─$ 
```

Step2: Setup custom volatility Profile

We need to download the correct header files that match our OS in the mem file
https://http.kali.org/kali/pool/main/l/linux/

We need to download 4 Files to build our profile

File1

linux-image-5.10.0-kali8-amd64-dbg_5.10.40-1kali1_amd64.deb
https://http.kali.org/kali/pool/main/l/linux/linux-image-5.10.0-kali8-amd64-dbg_5.10.40-1kali1_amd64.deb

File 2

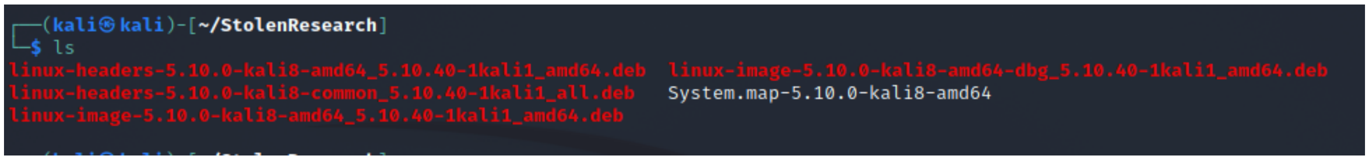
linux-headers-5.10.0-kali8-amd64_5.10.40-1kali1_amd64.deb
https://http.kali.org/kali/pool/main/l/linux/linux-headers-5.10.0-kali8-amd64_5.10.40-1kali1_amd64.deb

File 3

linux-headers-5.10.0-kali8-common_5.10.40-1kali1_all.deb
https://http.kali.org/kali/pool/main/l/linux/linux-headers-5.10.0-kali8-common_5.10.40-1kali1_all.deb

File 4

linux-image-5.10.0-kali8-amd64_5.10.40-1kali1_amd64.deb
https://http.kali.org/kali/pool/main/l/linux/linux-image-5.10.0-kali8-amd64_5.10.40-1kali1_amd64.deb



```
┌──(kali㉿kali)-[~/StolenResearch]
└─$ ls
linux-headers-5.10.0-kali8-amd64_5.10.40-1kali1_amd64.deb  linux-image-5.10.0-kali8-amd64-dbg_5.10.40-1kali1_amd64.deb
linux-headers-5.10.0-kali8-common_5.10.40-1kali1_all.deb   System.map-5.10.0-kali8-amd64
linux-image-5.10.0-kali8-amd64_5.10.40-1kali1_amd64.deb
```

We now need install our downloaded Packages

Command→

```
sudo dpkg -i *.deb
sudo reboot now
```

Create Custom Volatility Profile

Generate Profile
Command→

```
nm /usr/lib/debug/vmlinux-5.10.0-kali8-amd64 > ~/volatility/tools/linux/System.map-
5.10.0-kali8-amd64
```

Navigate to were you installed volatility , then run the following commands:

```
cd ~/volatility/tools/linux/
make
head module.dwarf
sudo zip Kali.zip module.dwarf System.map-5.10.0-kali8-amd64
```

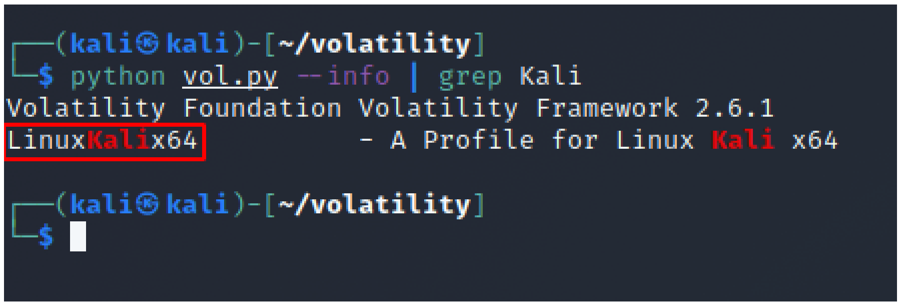
we now need to copy our new custom volatility profile (Kali.zip) into the volatility plugins folder

```
cp ~/volatility/tools/linux/Kali.zip
~/volatility/volatility/plugins/overlays/linux/Kali.zip
```

let now confirm our profile has been Loaded

```
python ~/volatility/vol.py --info | grep Kali
```

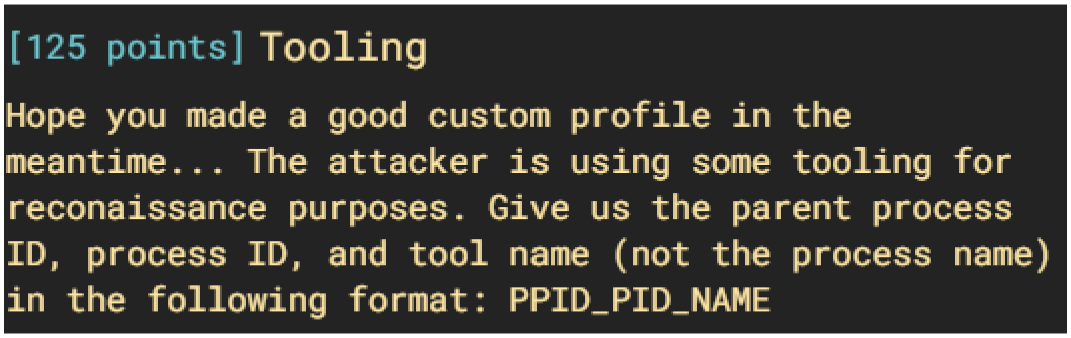
Output→



```
┌──(kali㉿kali)-[~/volatility]
└─$ python vol.py --info | grep Kali
Volatility Foundation Volatility Framework 2.6.1
LinuxKalix64          - A Profile for Linux Kali x64

┌──(kali㉿kali)-[~/volatility]
└─$ 
```

Flag 2:

Description: Tooling (We will use our Custom Profile)



[125 points] Tooling

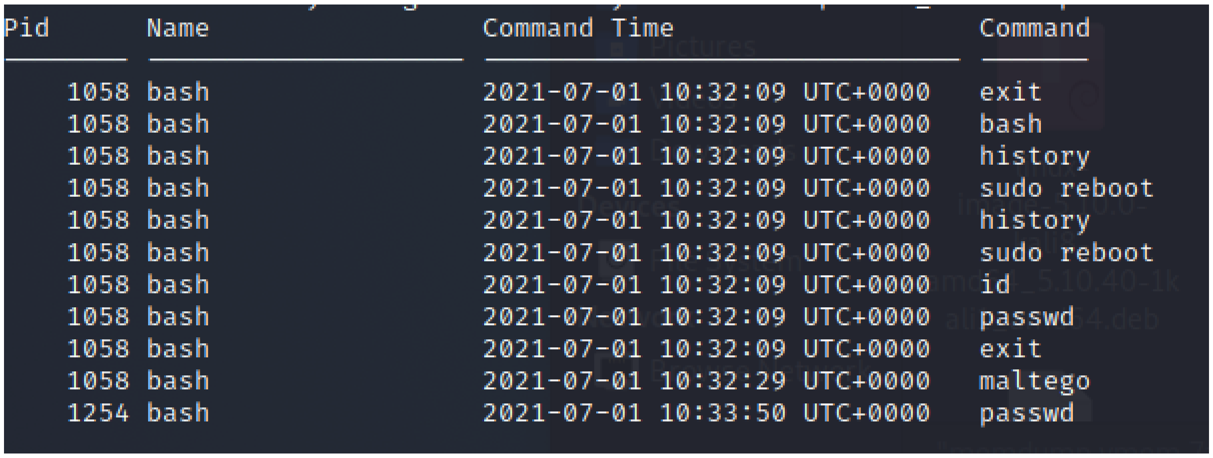
Hope you made a good custom profile in the meantime... The attacker is using some tooling for reconaissance purposes. Give us the parent process ID, process ID, and tool name (not the process name) in the following format: PPID_PID_NAME

Lets get the parent process ID, process ID, and tool name

Command→

```
python vol.py -f ~/StolenResearch/memdump.vmem --profile=LinuxKalix64 linux_bash
```

Ouput→



```
Pid      Name                 Command Time                   Command
-------- -------------------- ------------------------------ -------
    1058 bash                 2021-07-01 10:32:09 UTC+0000   exit
    1058 bash                 2021-07-01 10:32:09 UTC+0000   bash
    1058 bash                 2021-07-01 10:32:09 UTC+0000   history
    1058 bash                 2021-07-01 10:32:09 UTC+0000   sudo reboot
    1058 bash                 2021-07-01 10:32:09 UTC+0000   history
    1058 bash                 2021-07-01 10:32:09 UTC+0000   sudo reboot
    1058 bash                 2021-07-01 10:32:09 UTC+0000   id
    1058 bash                 2021-07-01 10:32:09 UTC+0000   passwd
    1058 bash                 2021-07-01 10:32:09 UTC+0000   exit
    1058 bash                 2021-07-01 10:32:29 UTC+0000   maltego
    1254 bash                 2021-07-01 10:33:50 UTC+0000   passwd
```

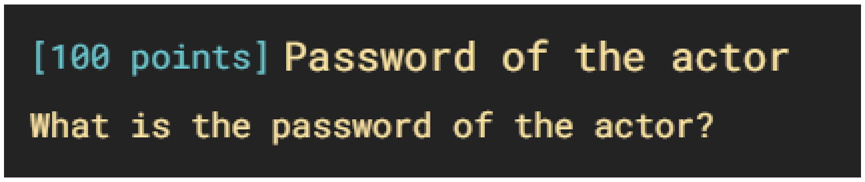
Find Process:

```
python2 vol.py -f memdump.vmem --profile=LinuxKaliProfilex64 linux_psaux | grep
maltego
```

bash history shows maltego - googling this shows that it is an OSINT tool
flag: 1082_1208_maltego

Flag 3:

Description: Password of the Actor



[100 points] Password of the actor

What is the password of the actor?

Firstly we already have the kernel type from the first task as well as the software that was used. This time we are looking for the password.

If it is available we can extract passwords through JtR if we have both the /etc/passwd and /etc/shadow files.

Using volatility we can try and recover those files. I got a number of errors every time it ran, these are just repeated and bear no impact on the result so I have removed then to keep the space down.

This works in two phases. We find the file by name, which gives us the inode. We then get the inode and save it.

For passwd

┌──(karti㉿kali-ctf)-[~/git/volatility]
└─$ sudo python vol.py -f ~/Downloads/stolen/memdump.vmem --profile=Linux5_10_0-kali8-amd64×64 linux_find_file -F "/etc/passwd"
Volatility Foundation Volatility Framework 2.6.1
*** Failed to import volatility.plugins.registry.shimcache (ImportError: No module named Crypto.Hash)
WARNING : volatility.debug : Overlay structure cpuinfo_x86 not present in vtypes
WARNING : volatility.debug : Overlay structure cpuinfo_x86 not present in vtypes
Inode Number Inode File Path

```
      2628853 0xffff931202b685e0 /etc/passwd
```

```
┌──(karti㉿kali-ctf)-[~/git/volatility]
└─$ sudo python vol.py -f ~/Downloads/stolen/memdump.vmem --profile=Linux5_10_0-kali8-
amd64x64 linux_find_file -i 0xffff931202b685e0 -O ~/Downloads/stolen/passwd
1 ×
Volatility Foundation Volatility Framework 2.6.1
*** Failed to import volatility.plugins.registry.shimcache (ImportError: No module
named Crypto.Hash)
WARNING : volatility.debug    : Overlay structure cpuinfo_x86 not present in vtypes
WARNING : volatility.debug    : Overlay structure cpuinfo_x86 not present in vtypes
```

For shadow

```
┌──(karti㉿kali-ctf)-[~/git/volatility]
└─$ sudo python vol.py -f ~/Downloads/stolen/memdump.vmem --profile=Linux5_10_0-kali8-
amd64x64 linux_find_file -F "/etc/shadow"
Volatility Foundation Volatility Framework 2.6.1
WARNING : volatility.debug    : Overlay structure cpuinfo_x86 not present in vtypes
WARNING : volatility.debug    : Overlay structure cpuinfo_x86 not present in vtypes
Inode Number                  Inode File Path
---------------- ------------------ ---------
         2621958 0xffff931202980140 /etc/shadow
```

```
┌──(karti㉿kali-ctf)-[~/git/volatility]
└─$ sudo python vol.py -f ~/Downloads/stolen/memdump.vmem --profile=Linux5_10_0-kali9-
amd64x64 linux_find_file -i 0xffff931202980140 -O ~/Downloads/stolen/shadow
Volatility Foundation Volatility Framework 2.6.1
*** Failed to import volatility.plugins.registry.shimcache (ImportError: No module
named Crypto.Hash)
WARNING : volatility.debug    : Overlay structure cpuinfo_x86 not present in vtypes
WARNING : volatility.debug    : Overlay structure cpuinfo_x86 not present in vtypes
```

Now we have both files we look to use JtR unshadow to get a hash that John can utilse.

```
┌──(karti㉿kali-ctf)-[~/git/volatility]
└─$ unshadow /Downloads/stolen/passwd /Downloads/stolen/shadow >
~/Downloads/stolen/hash.txt
```

This gives us our JtR hash.

invictus:yj9Ti6GkFortXamhKHY0bpTN.0
FLCqzsvVB1ZnfpffqSuvdLgzwLJvkmz6.aHfyoo11NB:1001:1001::/home/invictus:/bin/bash

However if you look at the hash id - y it doesn't look like most standard hashes.

Some research indicated i was known as yescrypt, and with hashcat modes in mind I had a look to see what I could do. I found this on github were I found the following details that allowed me to attempt it with the --format=crypt

```
Yescrypt is a notable algorithm:

-   Publicly known, modern algorithm, by Solar Designer ([@solardiz]
(https://github.com/solardiz))
-   PHC finalist: [https://www.password-hashing.net/](https://www.password-
hashing.net/)
-   Interest appears to be increasing in this algorithm - starting to appear in the
wild and reported in StackExchange questions, etc.
-   Demonstrating that the hash is hard/slow in hashcat could actually encourage
adoption :D
-   Discovering shortcuts/weaknesses could serve to drive improvement in the hash

Where used:

-   Supported by libxcrypt, a drop-in replacement for libcrypt.so.1:
[https://github.com/besser82/libxcrypt/](https://github.com/besser82/libxcrypt/)
-   Will soon be an option for Fedora:
[https://fedoraproject.org/wiki/Changes/yescrypt_as_default_hashing_method_for_shadow]
(https://fedoraproject.org/wiki/Changes/yescrypt_as_default_hashing_method_for_shadow)
-   Some discussion of using it in Debian, not yet clear:
[https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=978553]
(https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=978553)

Tool coverage:

-   No direct support in john-jumbo (but supported using passthrough `--format=crypt`)
-   Not supported by MDXfind

Limitations:

-   Maximum plaintext length: none
-   Max salt size: 512 bits
-   GPU- and FPGA/ASIC unfriendly, by design

Tech details:

-   Pseudocode: [https://openwall.info/wiki/yescrypt]
(https://openwall.info/wiki/yescrypt)
```

```
-   PHC full documentation: [https://www.password-
hashing.net/submissions/specs/yescrypt-v2.pdf](https://www.password-
hashing.net/submissions/specs/yescrypt-v2.pdf)
-   Source code: [https://github.com/openwall/yescrypt]
(https://github.com/openwall/yescrypt)
-   Official doc (yescrypt): [https://www.openwall.com/yescrypt/]
(https://www.openwall.com/yescrypt/)
-   Official doc (john-jumbo): [https://github.com/openwall/john/tree/bleeding-
jumbo/src/yescrypt](https://github.com/openwall/john/tree/bleeding-jumbo/src/yescrypt)
-   More details from Debian manpage
([https://manpages.debian.org/experimental/libcrypt1-dev/crypt.5.en.html]
(https://manpages.debian.org/experimental/libcrypt1-dev/crypt.5.en.html)):
```

I did try with my default john build on my Ubuntu, but I couldn't find anything. I reinstalled Jumbo John with the latest and got the password.

~/src/john/run$ ./john --format=crypt --wordlist=/home/jim/rockyou-75.txt ~/stolen_hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
security1 (invictus)
1g 0:00:00:53 DONE (2021-07-07 15:31) 0.01885g/s 505.0p/s 505.0c/s 505.0C/s
020585..pimp10
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

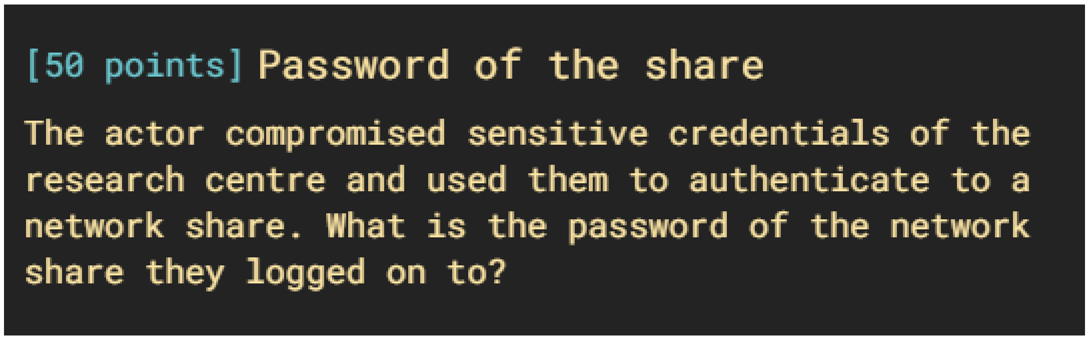
Then a check with --show

```
~/src/john/run$ ./john --show ~/stolen_hash.txt
invictus:security1:1001:1001::/home/invictus:/bin/bash
```

Flag security1

Flag 4:

Description: Password of the Share



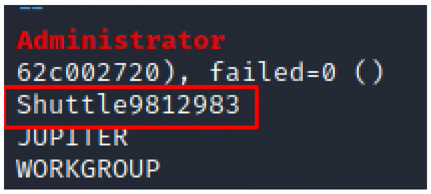
```
[50 points] Password of the share
The actor compromised sensitive credentials of the
research centre and used them to authenticate to a
network share. What is the password of the network
share they logged on to?
```

after struggling to find what i wanted with volatility (mostly looked at Linux_enumerate_files output) i went to strings:

Command→

```
strings memdump.vmem | grep -A 4 Administrator
```

Output→

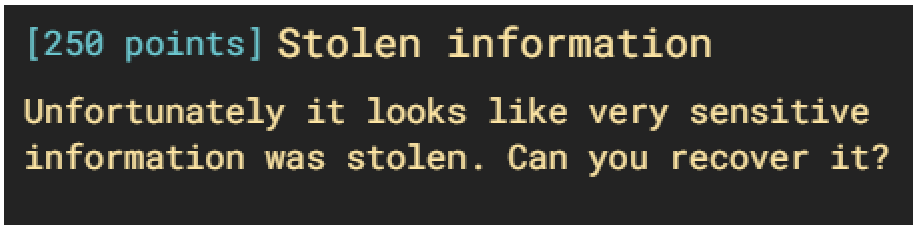


```
Administrator
62c002720), failed=0 ()
Shuttle9812983
JUPITER
WORKGROUP
```

flag: Shuttle9812983

Flag 5:

Description: Stolen Information



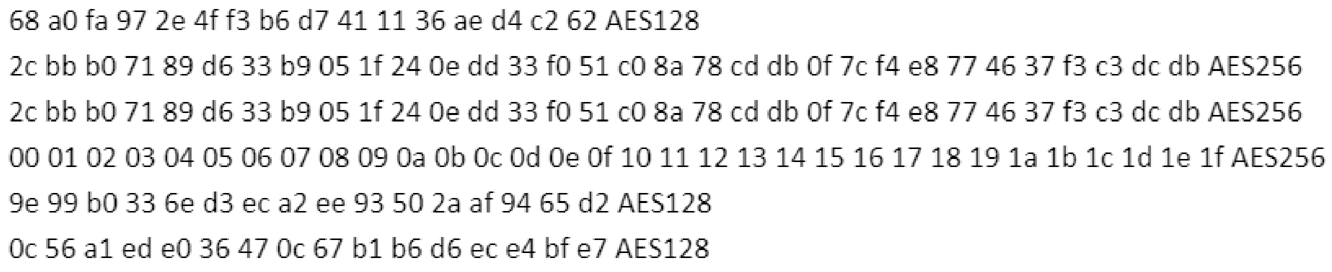
```
[250 points] Stolen information
Unfortunately it looks like very sensitive
information was stolen. Can you recover it?
```

Info from Admins gave hint that we needed to analyse the binary in memory for the file browser as they will contain the keys to decrypt the pcap. I tried endless options in volatility of dumping the maps and dumping the processes with no luck.

tried tool ./aeskeyfind with no luck

successfully extracted aes keys with kali already installed program - decided to run program on folder recursively as i had already extracted so many maps and libraries etc from the vmem and didnt know if they still might be located in thouse

bulk_extractor -o . -R ~/Downloads/stolen
bulk_extractor created aes.txt with following:



```
68 a0 fa 97 2e 4f f3 b6 d7 41 11 36 ae d4 c2 62 AES128
2c bb b0 71 89 d6 33 b9 05 1f 24 0e dd 33 f0 51 c0 8a 78 cd db 0f 7c f4 e8 77 46 37 f3 c3 dc db AES256
2c bb b0 71 89 d6 33 b9 05 1f 24 0e dd 33 f0 51 c0 8a 78 cd db 0f 7c f4 e8 77 46 37 f3 c3 dc db AES256
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f AES256
9e 99 b0 33 6e d3 ec a2 ee 93 50 2a af 94 65 d2 AES128
0c 56 a1 ed e0 36 47 0c 67 b1 b6 d6 ec e4 bf e7 AES128
```

Ignore the AES256 keys as we know they arent the correct length for our smb3 keys
lets assume the 3 AES128 are our keys in the following order; session key, signing key, serverIn Key

problem is we dont have the session Id - we know its a 8 byte hex string so i searched "Session Id" string in pcap and found fe3f6db000000000

enter all these into wireshark, edit>preference>protocols>smb2
was then able to successfully export object research.png which contained the flag

Flag: CTF{secret_research_facility}