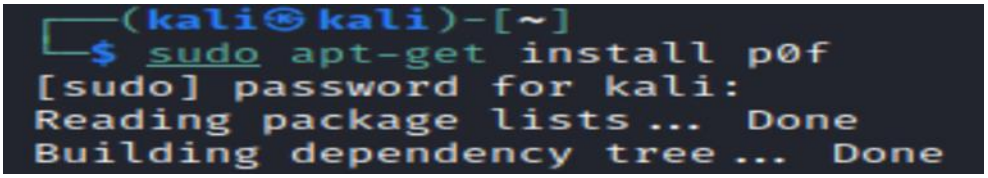
Report 12

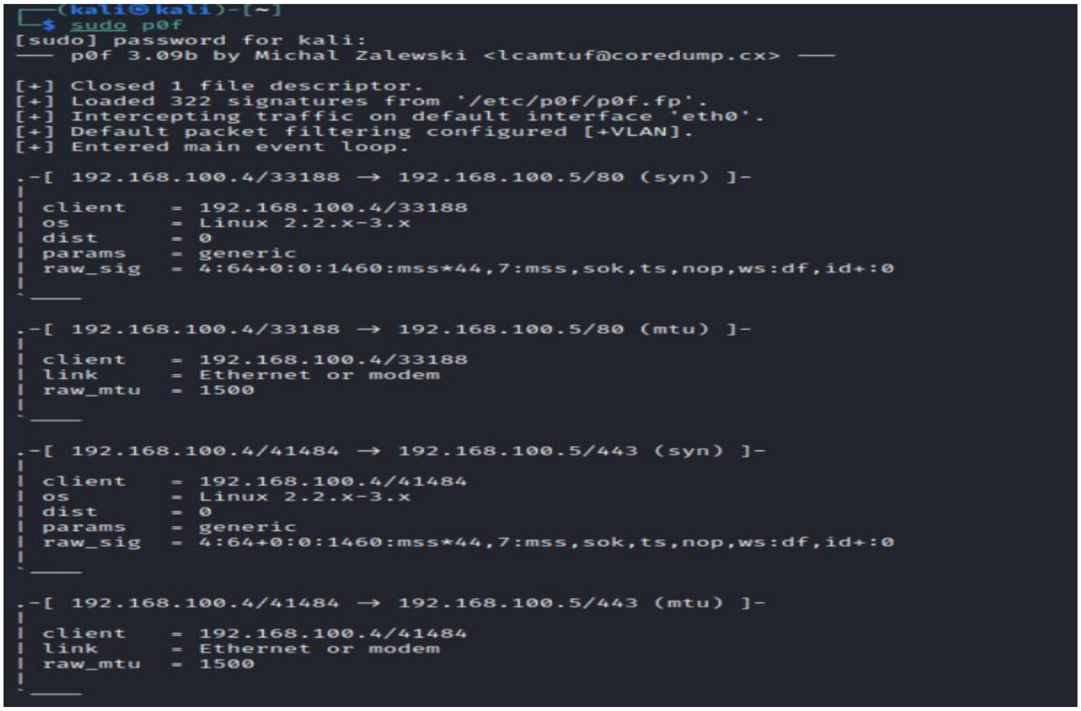
This week, we focused on using the forensic tools available in Linux. Part 1 focuses on Identifying Devices and OSs with p0f. Part 2 is on Information gathering and fingerprinting with arp-scan & Nmap. Part 3 involves Information Gathering with swap_digger. Part 4 is on Password Dumping with mimipenguin. Part 5 goes over more Linux Digital Forensic Tools and Part 6 covers the Sleuth Toolkit (STK).

The first part of the lab involved determining the operating system and other configuration properties of a remote host. To do this we can identify devices and the OS, by using the tool **p0f** in Linux. First, we need to install the tool using the command **sudo apt-get install p0f.**



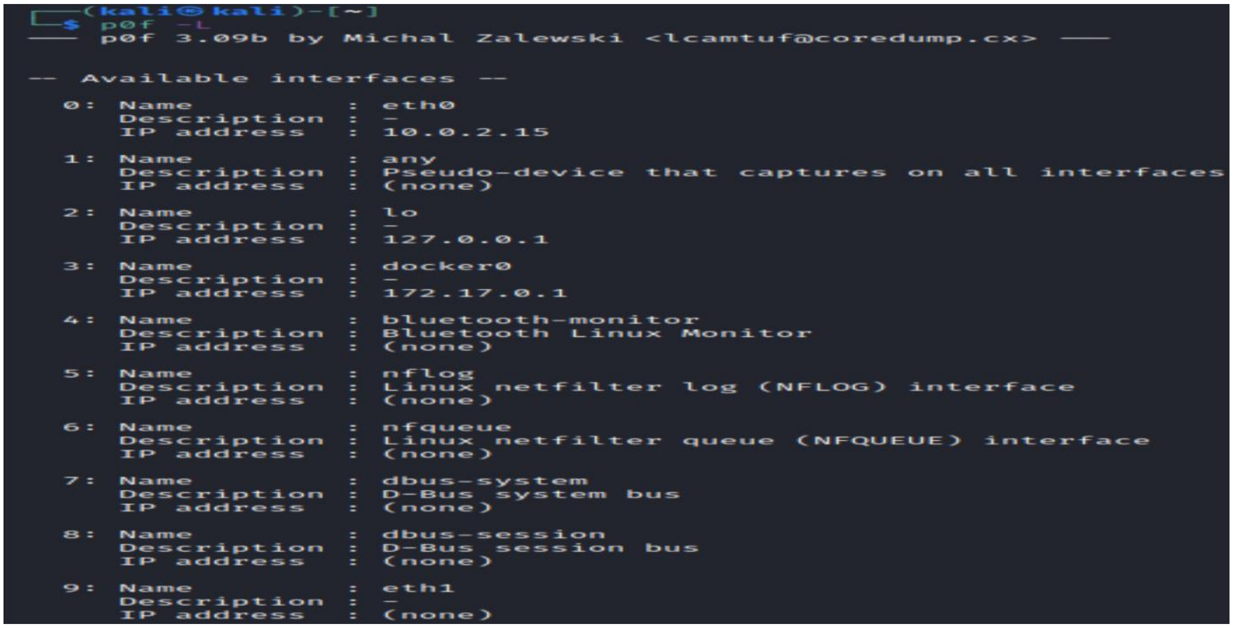
```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install p0f
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
```

Once it is installed, you can view the use **p0f** for various options. This uses a fingerprinting technique that analyzes the TCP/IP structure of the operating system host. Here is the output below.



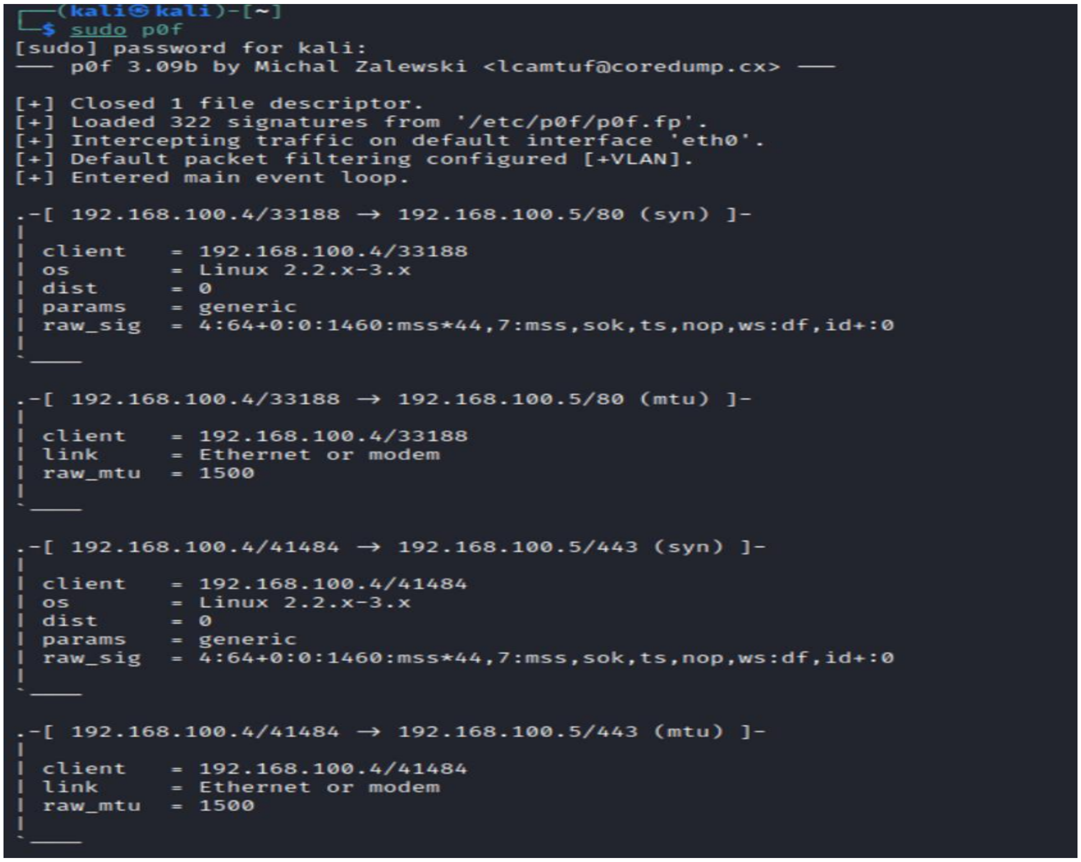
```
┌──(kali㉿kali)-[~]
└─$ sudo p0f
[sudo] password for kali:
--- p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> ---

[+] Closed 1 file descriptor.
[+] Loaded 322 signatures from '/etc/p0f/p0f.fp'.
[+] Intercepting traffic on default interface 'eth0'.
[+] Default packet filtering configured [+VLAN].
[+] Entered main event loop.

.-[ 192.168.100.4/33188 -> 192.168.100.5/80 (syn) ]-
|
| client   = 192.168.100.4/33188
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic
| raw_sig  = 4:64+0:0:1460:mss*44,7:mss,sok,ts,nop,ws:df,id+:0
|
`----

.-[ 192.168.100.4/33188 -> 192.168.100.5/80 (mtu) ]-
|
| client   = 192.168.100.4/33188
| link     = Ethernet or modem
| raw_mtu  = 1500
|
`----

.-[ 192.168.100.4/41484 -> 192.168.100.5/443 (syn) ]-
|
| client   = 192.168.100.4/41484
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic
| raw_sig  = 4:64+0:0:1460:mss*44,7:mss,sok,ts,nop,ws:df,id+:0
|
`----

.-[ 192.168.100.4/41484 -> 192.168.100.5/443 (mtu) ]-
|
| client   = 192.168.100.4/41484
| link     = Ethernet or modem
| raw_mtu  = 1500
|
`----
```

You can list all the available interfaces on your computer using **p0f -L** command.



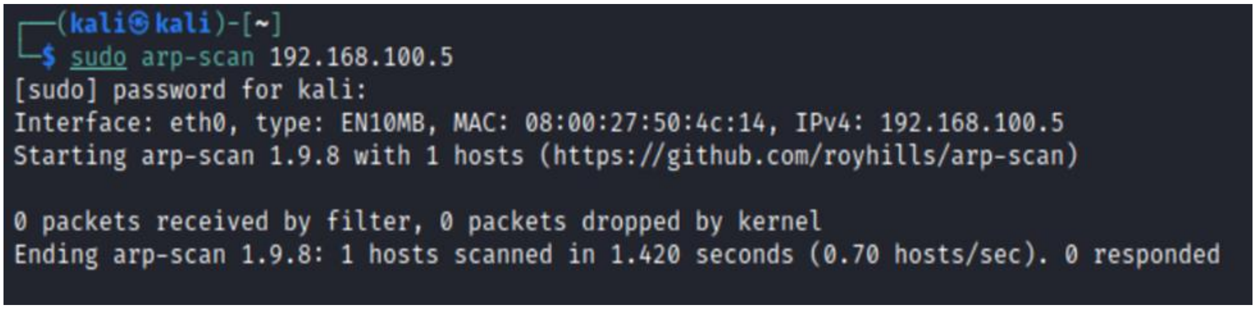
```
┌──(kali㉿kali)-[~]
└─$ p0f -L
─── p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> ───

-- Available interfaces --

  0: Name        : eth0
     Description : -
     IP address  : 10.0.2.15

  1: Name        : any
     Description : Pseudo-device that captures on all interfaces
     IP address  : (none)

  2: Name        : lo
     Description : -
     IP address  : 127.0.0.1

  3: Name        : docker0
     Description : -
     IP address  : 172.17.0.1

  4: Name        : bluetooth-monitor
     Description : Bluetooth Linux Monitor
     IP address  : (none)

  5: Name        : nflog
     Description : Linux netfilter log (NFLOG) interface
     IP address  : (none)

  6: Name        : nfqueue
     Description : Linux netfilter queue (NFQUEUE) interface
     IP address  : (none)

  7: Name        : dbus-system
     Description : D-Bus system bus
     IP address  : (none)

  8: Name        : dbus-session
     Description : D-Bus session bus
     IP address  : (none)

  9: Name        : eth1
     Description : -
     IP address  : (none)
```

Using the command **sudo p0f** without any parameters starts the process of fingerprinting process in the local machine. Note that this may take a few minutes before all of interfaces are shown. Here below the output.



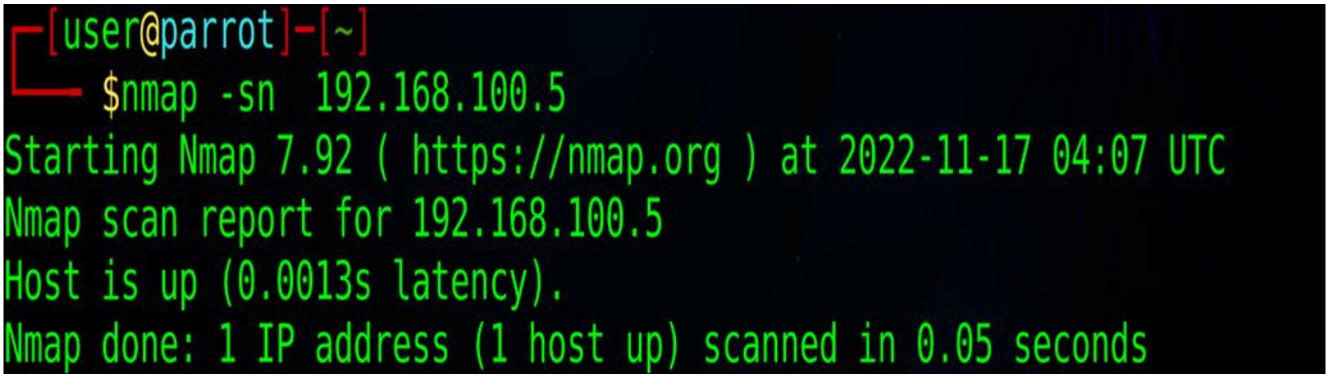
```
┌──(kali㉿kali)-[~]
└─$ sudo p0f
[sudo] password for kali:
─── p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> ───

[+] Closed 1 file descriptor.
[+] Loaded 322 signatures from '/etc/p0f/p0f.fp'.
[+] Intercepting traffic on default interface 'eth0'.
[+] Default packet filtering configured [+VLAN].
[+] Entered main event loop.

.-[ 192.168.100.4/33188 -> 192.168.100.5/80 (syn) ]-
|
| client   = 192.168.100.4/33188
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic
| raw_sig  = 4:64+0:0:1460:mss*44,7:mss,sok,ts,nop,ws:df,id+:0
|
`____

.-[ 192.168.100.4/33188 -> 192.168.100.5/80 (mtu) ]-
|
| client   = 192.168.100.4/33188
| link     = Ethernet or modem
| raw_mtu  = 1500
|
`____

.-[ 192.168.100.4/41484 -> 192.168.100.5/443 (syn) ]-
|
| client   = 192.168.100.4/41484
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic
| raw_sig  = 4:64+0:0:1460:mss*44,7:mss,sok,ts,nop,ws:df,id+:0
|
`____

.-[ 192.168.100.4/41484 -> 192.168.100.5/443 (mtu) ]-
|
| client   = 192.168.100.4/41484
| link     = Ethernet or modem
| raw_mtu  = 1500
|
`____
```

The second part of the lab covers Information gathering and fingerprinting with arp-scan & Nmap. This is used to list the ARP table content in the local network. Here below I scan my local machine using **sudo arp-scan 192.168.100.5** command. Note that you need to specify the Ip address you want to scan with. Here below is the output.



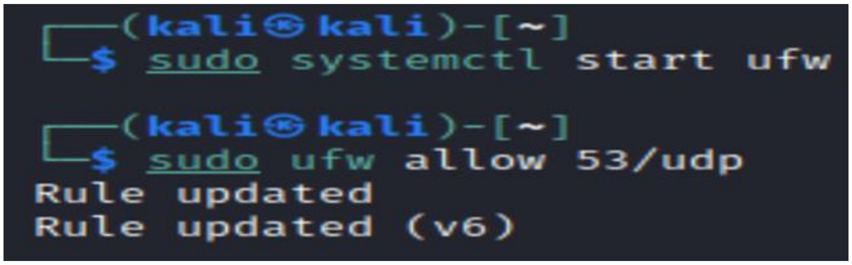
```
┌──(kali㉿kali)-[~]
└─$ sudo arp-scan 192.168.100.5
[sudo] password for kali:
Interface: eth0, type: EN10MB, MAC: 08:00:27:50:4c:14, IPv4: 192.168.100.5
Starting arp-scan 1.9.8 with 1 hosts (https://github.com/royhills/arp-scan)

0 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.8: 1 hosts scanned in 1.420 seconds (0.70 hosts/sec). 0 responded
```

I scanned my kali machine using another local machine on parrot. Here I used the command **nmap -sn 192.168.100.5**



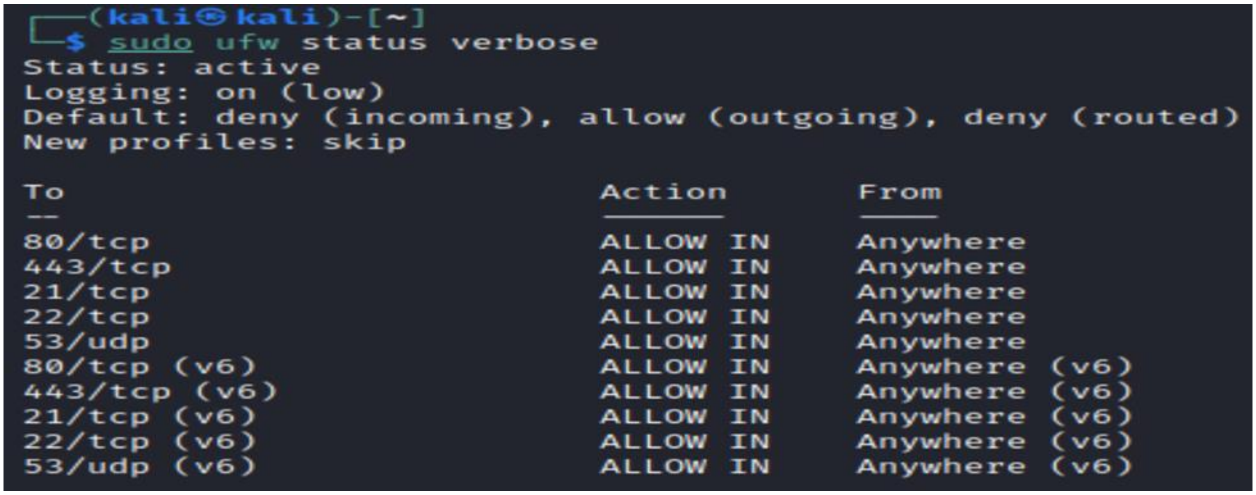
```
┌─[user@parrot]─[~]
└──── $nmap -sn  192.168.100.5
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-17 04:07 UTC
Nmap scan report for 192.168.100.5
Host is up (0.0013s latency).
Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds
```

If you want to perform a SYN scan, use **sudo nmap -sS** followed by the Ip that you want. Here below I performed a UDP port scan. This was performed in kali machine. First, I used the command **sudo systemctl start ufw** and **sudo ufw allow 53/udp** to start ufw service and allow port number 53. Here below is the output.



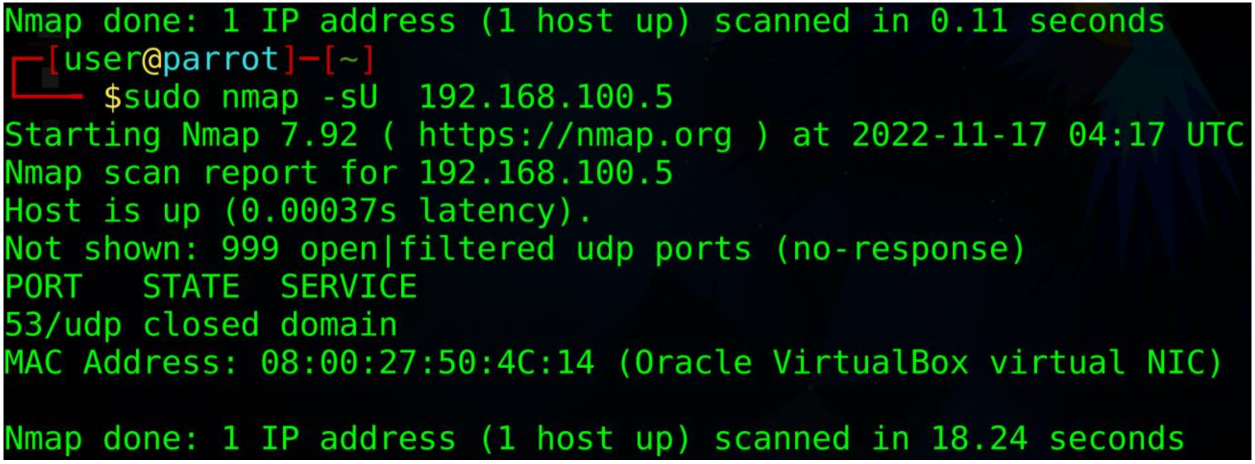
```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl start ufw

┌──(kali㉿kali)-[~]
└─$ sudo ufw allow 53/udp
Rule updated
Rule updated (v6)
```

Next, I wanted to make sure that the service was running. To do this, use the command **sudo ufw status verbose.** Here is the output below.



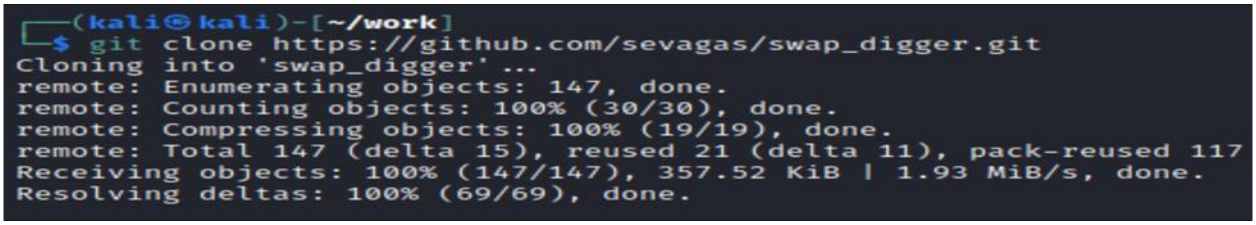
```
┌──(kali㉿kali)-[~]
└─$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
80/tcp                     ALLOW IN    Anywhere
443/tcp                    ALLOW IN    Anywhere
21/tcp                     ALLOW IN    Anywhere
22/tcp                     ALLOW IN    Anywhere
53/udp                     ALLOW IN    Anywhere
80/tcp (v6)                ALLOW IN    Anywhere (v6)
443/tcp (v6)               ALLOW IN    Anywhere (v6)
21/tcp (v6)                ALLOW IN    Anywhere (v6)
22/tcp (v6)                ALLOW IN    Anywhere (v6)
53/udp (v6)                ALLOW IN    Anywhere (v6)
```

Now that I have this set up. I used my second machine parrotOS to scan the UPD port on my kali machine. Here below I used the command **sudo nmap -sU 192.168.100.5**. You see below that I the port is closed yet I can see that it is still view the status of it on my parrotOs machine.



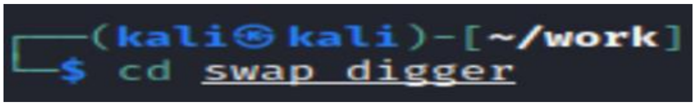
```
Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
┌─[user@parrot]─[~]
└──── $sudo nmap -sU  192.168.100.5
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-17 04:17 UTC
Nmap scan report for 192.168.100.5
Host is up (0.00037s latency).
Not shown: 999 open|filtered udp ports (no-response)
PORT   STATE  SERVICE
53/udp closed domain
MAC Address: 08:00:27:50:4C:14 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 18.24 seconds
```

The next lab covers Information Gathering using swap_digger. This tool performs analysis of the Linux swap file to retrieve system passwords, usernames, +credentials, among others. Before I installed the tool, first I created a new directory using **mkdir work** then I navigated to that directory using **cd work** so that I can install the swap_digger tool. Now, I can install the tool using the command **git clone** https://github.com/sevagas/swap_digger.git. Here below is the output.



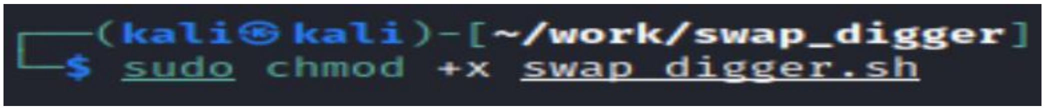
```
┌──(kali㉿kali)-[~/work]
└─$ git clone https://github.com/sevagas/swap_digger.git
Cloning into 'swap_digger'...
remote: Enumerating objects: 147, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 147 (delta 15), reused 21 (delta 11), pack-reused 117
Receiving objects: 100% (147/147), 357.52 KiB | 1.93 MiB/s, done.
Resolving deltas: 100% (69/69), done.
```

After I installed it, I used to change the directory using the command **cd swap_digger** to access that directory. Below is the output.



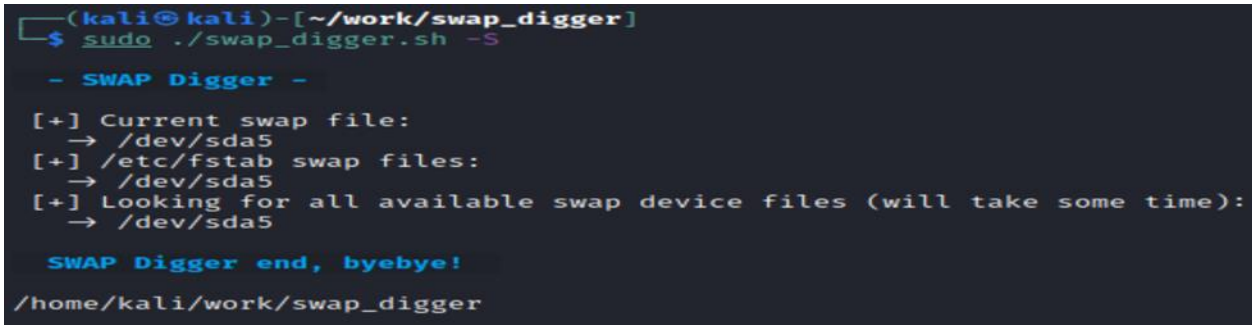
```
┌──(kali㉿kali)-[~/work]
└─$ cd swap_digger
```

Now I am in this directory, use we need to use swap_digger for the mounted drive. To do this, use the command **chmod +x swap_digger.sh**. Here below is the output. Note that no output was returned.



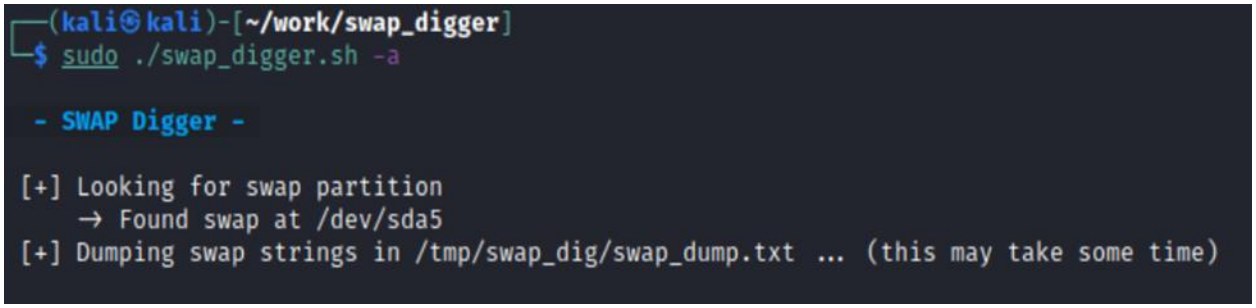
```
┌──(kali㉿kali)-[~/work/swap_digger]
└─$ sudo chmod +x swap_digger.sh
```

We can now view the swap file. To this use the **sudo ./swap_digger.sh -S** command. Below you see the swap file of sda5.



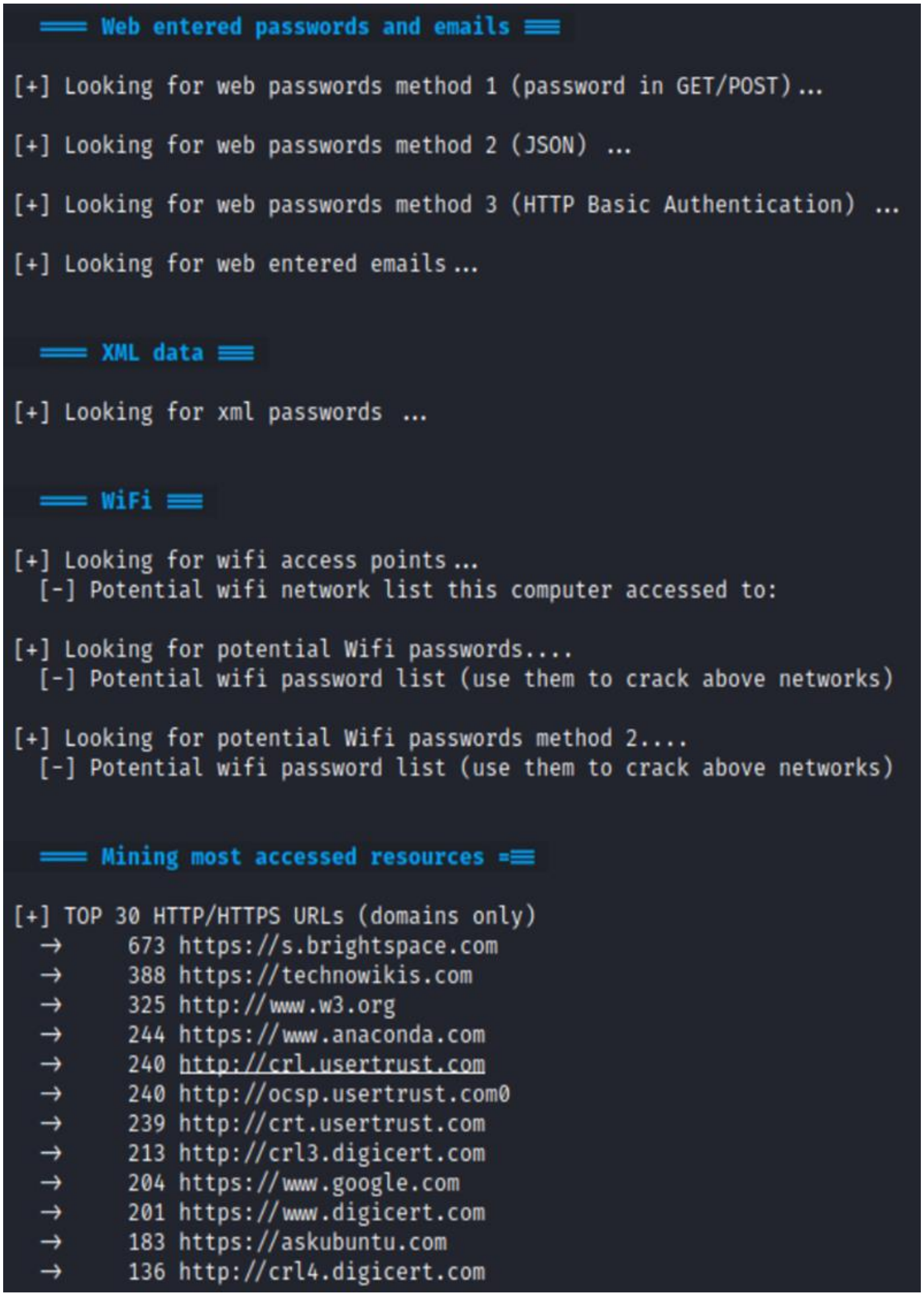
```
┌──(kali㉿kali)-[~/work/swap_digger]
└─$ sudo ./swap_digger.sh -S

  - SWAP Digger -

 [+] Current swap file:
   → /dev/sda5
 [+] /etc/fstab swap files:
   → /dev/sda5
 [+] Looking for all available swap device files (will take some time):
   → /dev/sda5

  SWAP Digger end, byebye!

/home/kali/work/swap_digger
```

If want information on application data, you can dump it by using **sudo ./swap_digger.sh -a** command. Note that this may take a few minutes to process every information. Here below are the outputs. You see that it found a swap file in sda5.



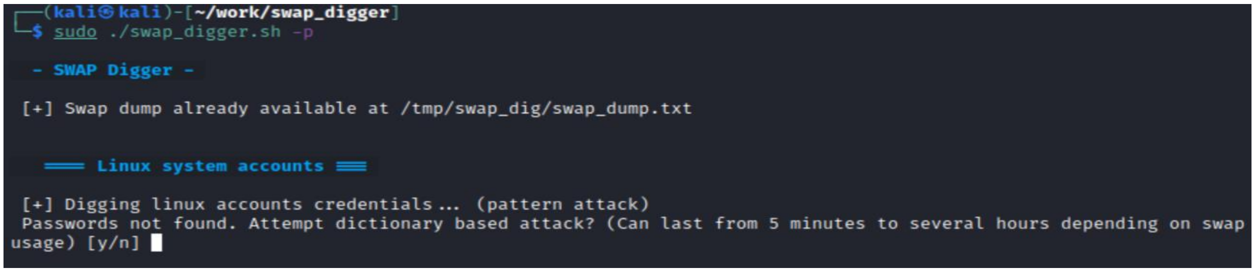
```
┌──(kali㉿kali)-[~/work/swap_digger]
└─$ sudo ./swap_digger.sh -a

  - SWAP Digger -

[+] Looking for swap partition
    → Found swap at /dev/sda5
[+] Dumping swap strings in /tmp/swap_dig/swap_dump.txt … (this may take some time)
```

Here is list of details data the swap file can contain. These include web passwords, XML data, Wi-Fi (will look for access points, potential Wi-Fi passwords and methods to crack them, Mining most accessed resources (The most visited websites) and many more. You see the output below.



```
 === Web entered passwords and emails ===

[+] Looking for web passwords method 1 (password in GET/POST)...

[+] Looking for web passwords method 2 (JSON) ...

[+] Looking for web passwords method 3 (HTTP Basic Authentication) ...

[+] Looking for web entered emails...


 === XML data ===

[+] Looking for xml passwords ...


 === WiFi ===

[+] Looking for wifi access points...
  [-] Potential wifi network list this computer accessed to:

[+] Looking for potential Wifi passwords....
  [-] Potential wifi password list (use them to crack above networks)

[+] Looking for potential Wifi passwords method 2....
  [-] Potential wifi password list (use them to crack above networks)


 === Mining most accessed resources ====

[+] TOP 30 HTTP/HTTPS URLs (domains only)
  →     673 https://s.brightspace.com
  →     388 https://technowikis.com
  →     325 http://www.w3.org
  →     244 https://www.anaconda.com
  →     240 http://crl.usertrust.com
  →     240 http://ocsp.usertrust.com0
  →     239 http://crt.usertrust.com
  →     213 http://crl3.digicert.com
  →     204 https://www.google.com
  →     201 https://www.digicert.com
  →     183 https://askubuntu.com
  →     136 http://crl4.digicert.com
```

Another option you can do as well are passwords. To this you need to use **sudo ./swap_digger.sh -p** command. Here below is the output. Note that this can take a while to execute.

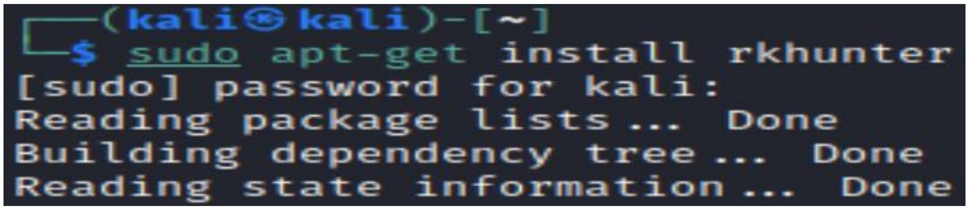


```
┌──(kali㉿kali)-[~/work/swap_digger]
└─$ sudo ./swap_digger.sh -p

  - SWAP Digger -

 [+] Swap dump already available at /tmp/swap_dig/swap_dump.txt


   ═══ Linux system accounts ═══

 [+] Digging linux accounts credentials ... (pattern attack)
 Passwords not found. Attempt dictionary based attack? (Can last from 5 minutes to several hours depending on swap
usage) [y/n] █
```

Part 4 covers Password Dumping with **mimipenguin**. I navigated to work directory and the installed the tool **mimipenguin** using the command **git clone https://github.com/huntergregal/mimipenguin.git**. Now I can run the tool using **sudo ./mimipenguin.sh**. Below is the output. Note that no passwords where dumped.

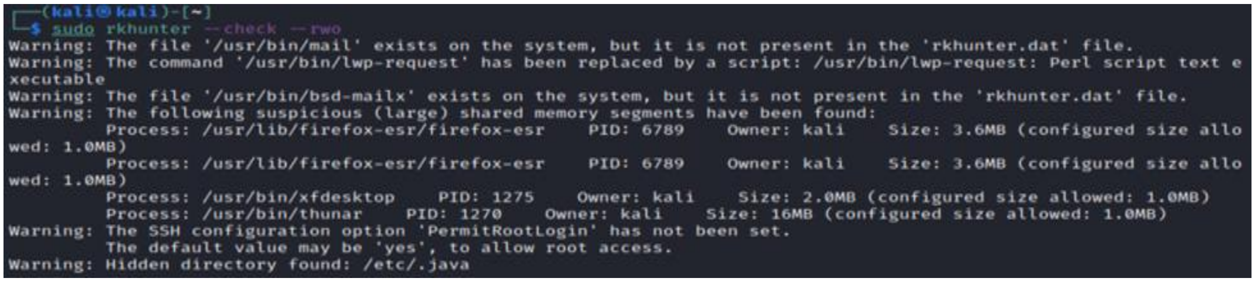
```
kali@kali [~/work] git clone https://github.com/huntergregal/mimipenguin.git
kali@kali [~/work] cd mimipenguin
kali@kali [~/work/mimipenguin] sudo ./mimipenguin.sh
     MimiPenguin Results:
```

There also other tools you can use for Linux forensics. In part 5, we going to use a utility called **rkhunter**. This tool checks if they are any suspicious files, hidden directories, and rootkits in your system. The first thing to is to install **rkhunter** by using **sudo apt-get install rkhunter** command. Here below is the output.



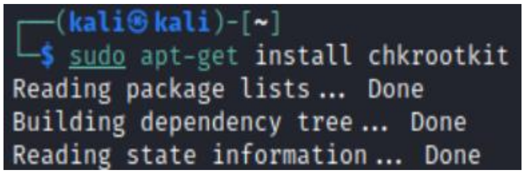
```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install rkhunter
[sudo] password for kali:
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
```

Now we can run the command **sudo rkhunter --check –rwo** to check files that in the local machine. Here below is the output. You can see below it that it found a hidden directory file called **/etc/.java**. There also other things that it found as well.



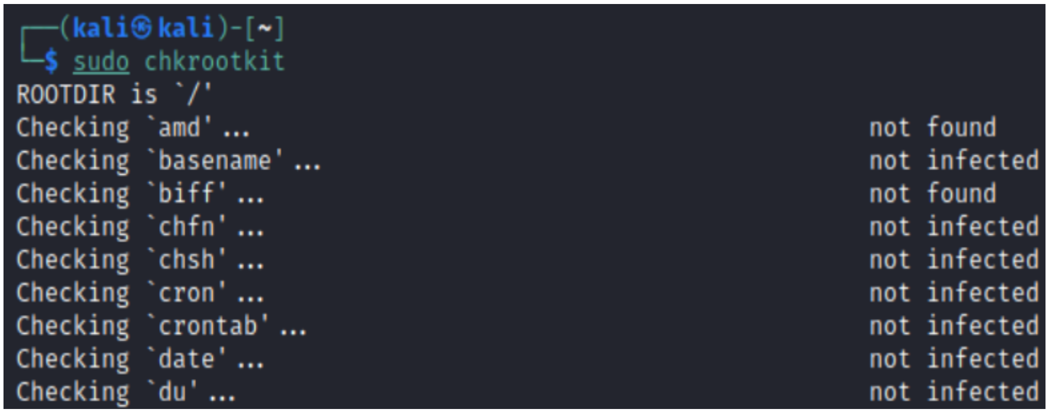
```
┌──(kali㉿kali)-[~]
└─$ sudo rkhunter --check --rwo
Warning: The file '/usr/bin/mail' exists on the system, but it is not present in the 'rkhunter.dat' file.
Warning: The command '/usr/bin/lwp-request' has been replaced by a script: /usr/bin/lwp-request: Perl script text e
xecutable
Warning: The file '/usr/bin/bsd-mailx' exists on the system, but it is not present in the 'rkhunter.dat' file.
Warning: The following suspicious (large) shared memory segments have been found:
         Process: /usr/lib/firefox-esr/firefox-esr    PID: 6789    Owner: kali    Size: 3.6MB (configured size allo
wed: 1.0MB)
         Process: /usr/lib/firefox-esr/firefox-esr    PID: 6789    Owner: kali    Size: 3.6MB (configured size allo
wed: 1.0MB)
         Process: /usr/bin/xfdesktop    PID: 1275    Owner: kali    Size: 2.0MB (configured size allowed: 1.0MB)
         Process: /usr/bin/thunar    PID: 1270    Owner: kali    Size: 16MB (configured size allowed: 1.0MB)
Warning: The SSH configuration option 'PermitRootLogin' has not been set.
         The default value may be 'yes', to allow root access.
Warning: Hidden directory found: /etc/.java
```

We also use the tool **chkrootkit** to see check the files in the local machine for signs of a rootkit as well. Using this tool will display the files and will show if they are infected or not. First, we need to install the tool using the command **sudo apt-get install chkrootkit.** Here below is the output.



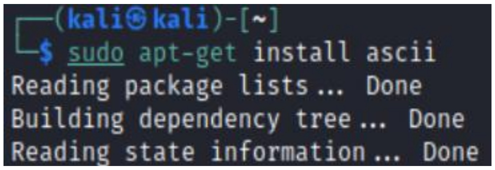
```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install chkrootkit
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Now if you want to run this tool, use **sudo chkrootkit** command. This tool looks for known "signatures" in trojan Ed system binaries. For example, some trojan Ed versions of PS have "/dev/ptyp" inside them. Here below is the output



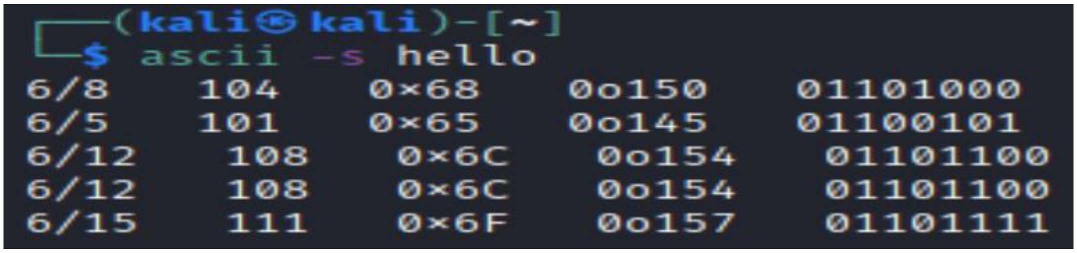
```
┌──(kali㉿kali)-[~]
└─$ sudo chkrootkit
ROOTDIR is `/'
Checking `amd'...                                          not found
Checking `basename'...                                     not infected
Checking `biff'...                                         not found
Checking `chfn'...                                         not infected
Checking `chsh'...                                         not infected
Checking `cron'...                                         not infected
Checking `crontab'...                                      not infected
Checking `date'...                                         not infected
Checking `du'...                                           not infected
```

The ascii can then be utilized to the ascii values for files. To use this, you first need to install this utility using **sudo apt-get install ascii**. Here below is the output.



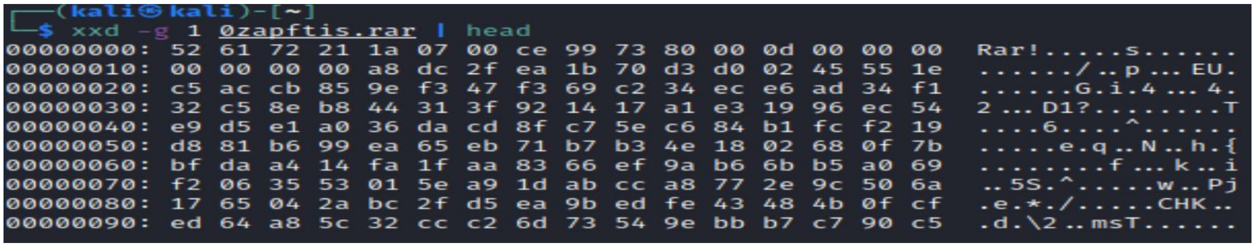
```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install ascii
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Now that we have this installed, we can show the ascii values for the input “hello.” To this use the command **ascii -s hello.** Here below is the output.



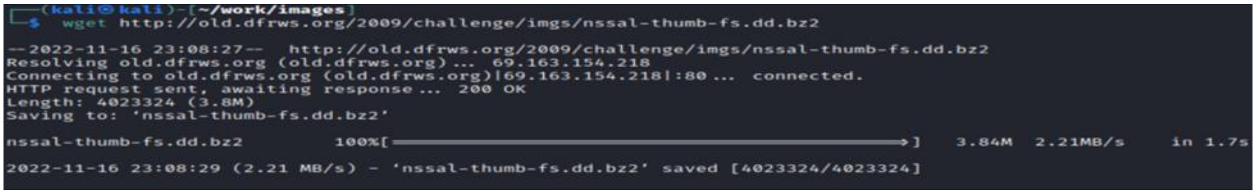
```
┌──(kali㉿kali)-[~]
└─$ ascii -s hello
6/8    104    0x68    0o150    01101000
6/5    101    0x65    0o145    01100101
6/12    108    0x6C    0o154    01101100
6/12    108    0x6C    0o154    01101100
6/15    111    0x6F    0o157    01101111
```

You can also display the file signature and content using the **xxd** parameter. Here below is a file from Volatility3 called 0zapftis.rar. Here you can see the first 7 values of the **rar** file. This is the signature which is **52 61 72 21 1a 07 01 00.** To do this used the **command xxd -g 1 0zapftis.rar | head**. Below you can see the output.



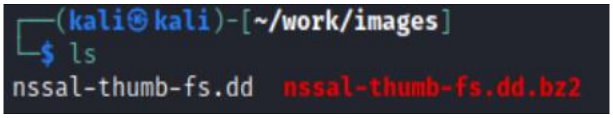
```
┌──(kali㉿kali)-[~]
└─$ xxd -g 1 0zapftis.rar | head
00000000: 52 61 72 21 1a 07 00 ce 99 73 80 00 0d 00 00 00  Rar!......s......
00000010: 00 00 00 00 a8 dc 2f ea 1b 70 d3 d0 02 45 55 1e  ....../..p...EU.
00000020: c5 ac cb 85 9e f3 47 f3 69 c2 34 ec e6 ad 34 f1  ......G.i.4...4.
00000030: 32 c5 8e b8 44 31 3f 92 14 17 a1 e3 19 96 ec 54  2...D1?........T
00000040: e9 d5 e1 a0 36 da cd 8f c7 5e c6 84 b1 fc f2 19  ....6....^......
00000050: d8 81 b6 99 ea 65 eb 71 b7 b3 4e 18 02 68 0f 7b  .....e.q..N..h.{
00000060: bf da a4 14 fa 1f aa 83 66 ef 9a b6 6b b5 a0 69  ........f...k..i
00000070: f2 06 35 53 01 5e a9 1d ab cc a8 77 2e 9c 50 6a  ..5S.^.....w..Pj
00000080: 17 65 04 2a bc 2f d5 ea 9b ed fe 43 48 4b 0f cf  .e.*./.....CHK..
00000090: ed 64 a8 5c 32 cc c2 6d 73 54 9e bb b7 c7 90 c5  .d.\2..msT......
```

The last part of this lab involves using Sleuth and its various forensics tools available for analyzing. To make the process cleaner, I created a new directory using the command **mkdir images** in work directory to save all the forensic analysis in this file. Sleuth already comes pre-installed using Kali Linux, if is not installed used the command **sudo apt install sleuthkit**. Now we can download the image file we are going to analyze. Use the command **wget** **http://old.dfrws.org/2009/challenge/imgs/nssal-thumb-fs.dd.bz2** Here below is the output.



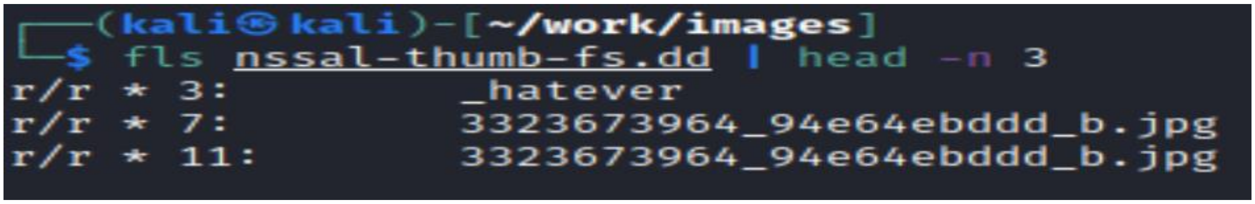
```
┌──(kali㉿kali)-[~/work/images]
└─$ wget http://old.dfrws.org/2009/challenge/imgs/nssal-thumb-fs.dd.bz2

--2022-11-16 23:08:27--  http://old.dfrws.org/2009/challenge/imgs/nssal-thumb-fs.dd.bz2
Resolving old.dfrws.org (old.dfrws.org)... 69.163.154.218
Connecting to old.dfrws.org (old.dfrws.org)|69.163.154.218|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4023324 (3.8M)
Saving to: 'nssal-thumb-fs.dd.bz2'

nssal-thumb-fs.dd.bz2        100%[==================================================>]   3.84M  2.21MB/s    in 1.7s

2022-11-16 23:08:29 (2.21 MB/s) - 'nssal-thumb-fs.dd.bz2' saved [4023324/4023324]
```

First thing to do involves uncompressing the image file. To do this use the command **bzip2 -dk nssal-thumb-fs.dd.bz2**. We can then view the contents of this using the command **ls.** You can see below that the file was compressed successfully. It is the one that is called **nssal-thumb-fs.dd**



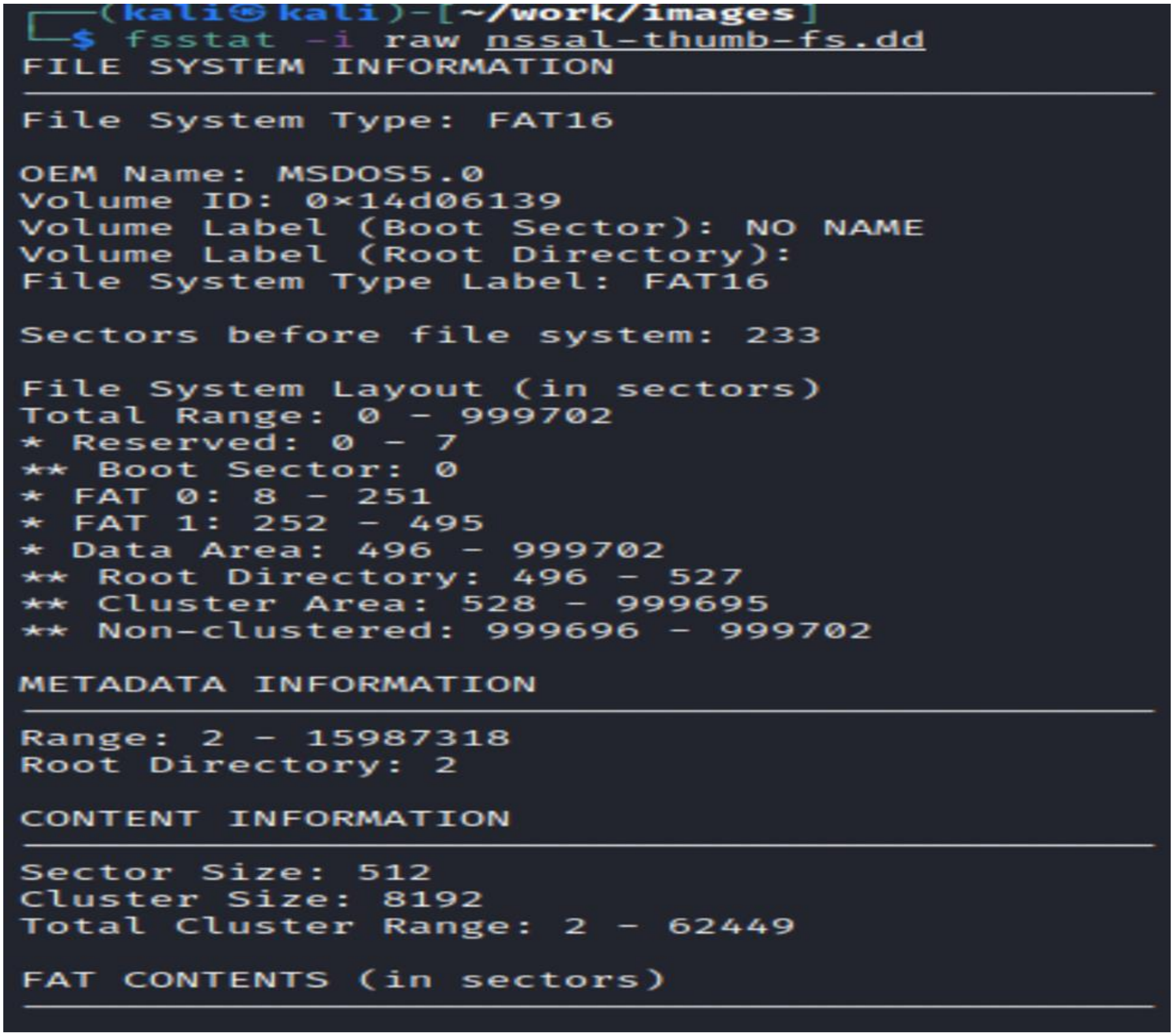
```
┌──(kali㉿kali)-[~/work/images]
└─$ ls
nssal-thumb-fs.dd  nssal-thumb-fs.dd.bz2
```

The first command to go over is **fls**. This parameter lists the files and directories of this file. To do this used the command **fls nssal-thumb-fs.dd | head -n 3**. Here below is the output.



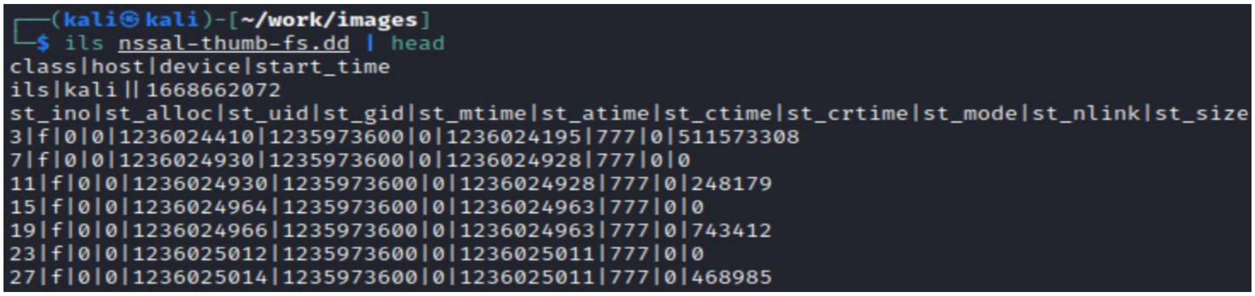
```
┌──(kali㉿kali)-[~/work/images]
└─$ fls nssal-thumb-fs.dd | head -n 3
r/r * 3:        _hatever
r/r * 7:        3323673964_94e64ebddd_b.jpg
r/r * 11:       3323673964_94e64ebddd_b.jpg
```

The second command **fsstat.** This parameter will display general details of the file system. You can see below the File system layout in sectors, the metadata, and the filesystem it using which is FAT16.



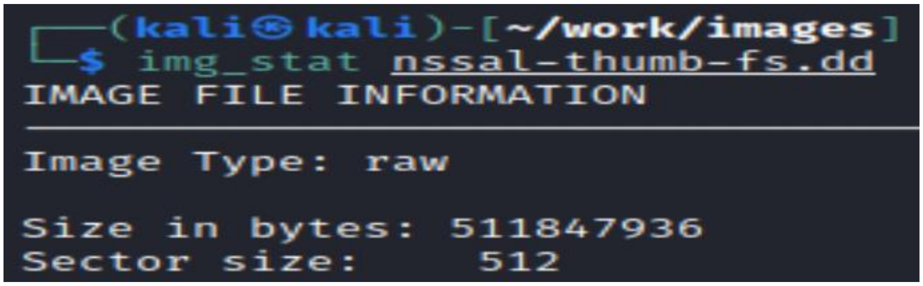
```
┌──(kali㉿kali)-[~/work/images]
└─$ fsstat -i raw nssal-thumb-fs.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: MSDOS5.0
Volume ID: 0×14d06139
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory):
File System Type Label: FAT16

Sectors before file system: 233

File System Layout (in sectors)
Total Range: 0 - 999702
* Reserved: 0 - 7
** Boot Sector: 0
* FAT 0: 8 - 251
* FAT 1: 252 - 495
* Data Area: 496 - 999702
** Root Directory: 496 - 527
** Cluster Area: 528 - 999695
** Non-clustered: 999696 - 999702

METADATA INFORMATION
--------------------------------------------
Range: 2 - 15987318
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 8192
Total Cluster Range: 2 - 62449

FAT CONTENTS (in sectors)
--------------------------------------------
```

The third command is **ils**. This parameter will list inode information. Here below I list information using the command **ils nssal-thumb-fs.dd | head.** You can see below the info of the inodes.



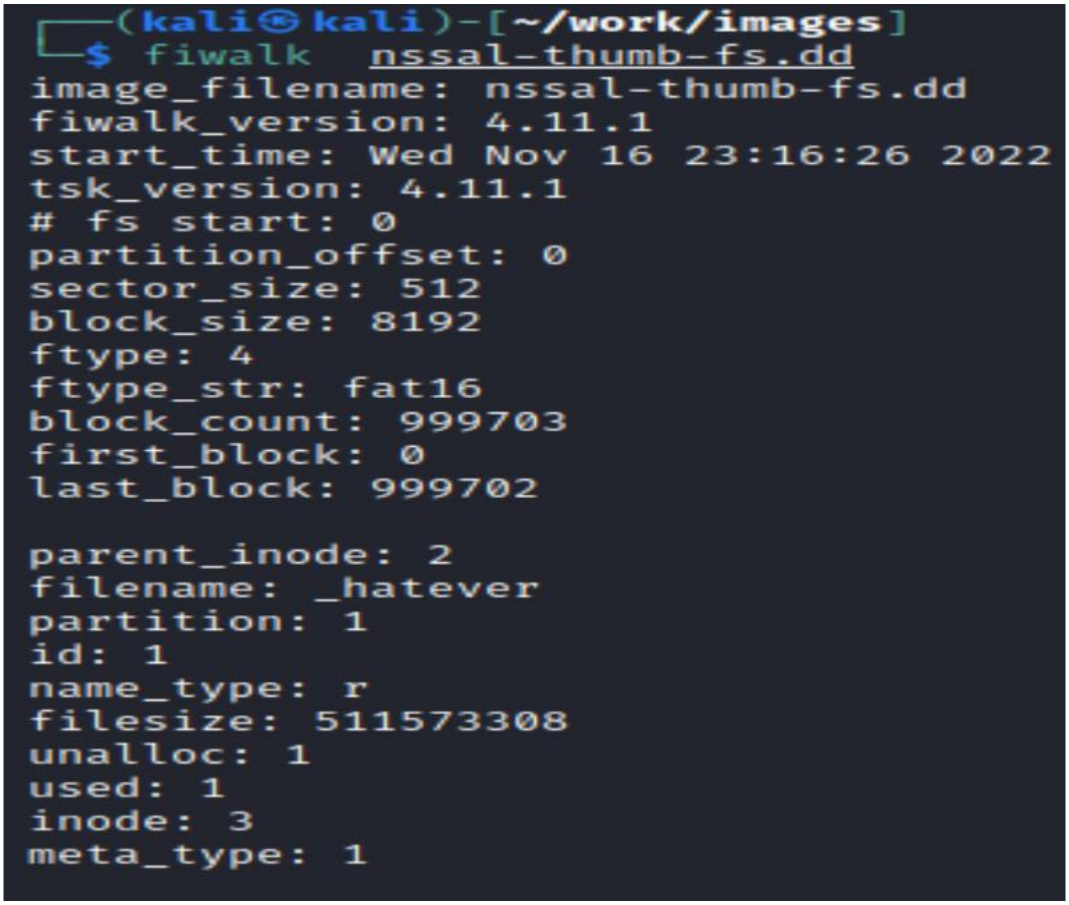
```
┌──(kali㉿kali)-[~/work/images]
└─$ ils nssal-thumb-fs.dd | head
class|host|device|start_time
ils|kali||1668662072
st_ino|st_alloc|st_uid|st_gid|st_mtime|st_atime|st_ctime|st_crtime|st_mode|st_nlink|st_size
3|f|0|0|1236024410|1235973600|0|1236024195|777|0|511573308
7|f|0|0|1236024930|1235973600|0|1236024928|777|0|0
11|f|0|0|1236024930|1235973600|0|1236024928|777|0|248179
15|f|0|0|1236024964|1235973600|0|1236024963|777|0|0
19|f|0|0|1236024966|1235973600|0|1236024963|777|0|743412
23|f|0|0|1236025012|1235973600|0|1236025011|777|0|0
27|f|0|0|1236025014|1235973600|0|1236025011|777|0|468985
```

The fourth command is **img_stat**. This parameter displays the details of the image file. It lists the image type, the size in bytes and the sector size. This can give us a lot of information to see what type of image this is. To view this, use the command **img_stat nssal-thumb-fs.dd**. Here below is the output.



```
┌──(kali㉿kali)-[~/work/images]
└─$ img_stat nssal-thumb-fs.dd
IMAGE FILE INFORMATION
--------------------------------------------
Image Type: raw

Size in bytes: 511847936
Sector size:    512
```

The firth command is **fiwalk**. This parameter prints the filesystem details**.** To do this, use the command **fiwalk nssal-thumb-fs.dd**. Here you can see information such as the block count, parent_inode and the last_block.



```
┌──(kali㉿kali)-[~/work/images]
└─$ fiwalk  nssal-thumb-fs.dd
image_filename: nssal-thumb-fs.dd
fiwalk_version: 4.11.1
start_time: Wed Nov 16 23:16:26 2022
tsk_version: 4.11.1
# fs start: 0
partition_offset: 0
sector_size: 512
block_size: 8192
ftype: 4
ftype_str: fat16
block_count: 999703
first_block: 0
last_block: 999702

parent_inode: 2
filename: _hatever
partition: 1
id: 1
name_type: r
filesize: 511573308
unalloc: 1
used: 1
inode: 3
meta_type: 1
```