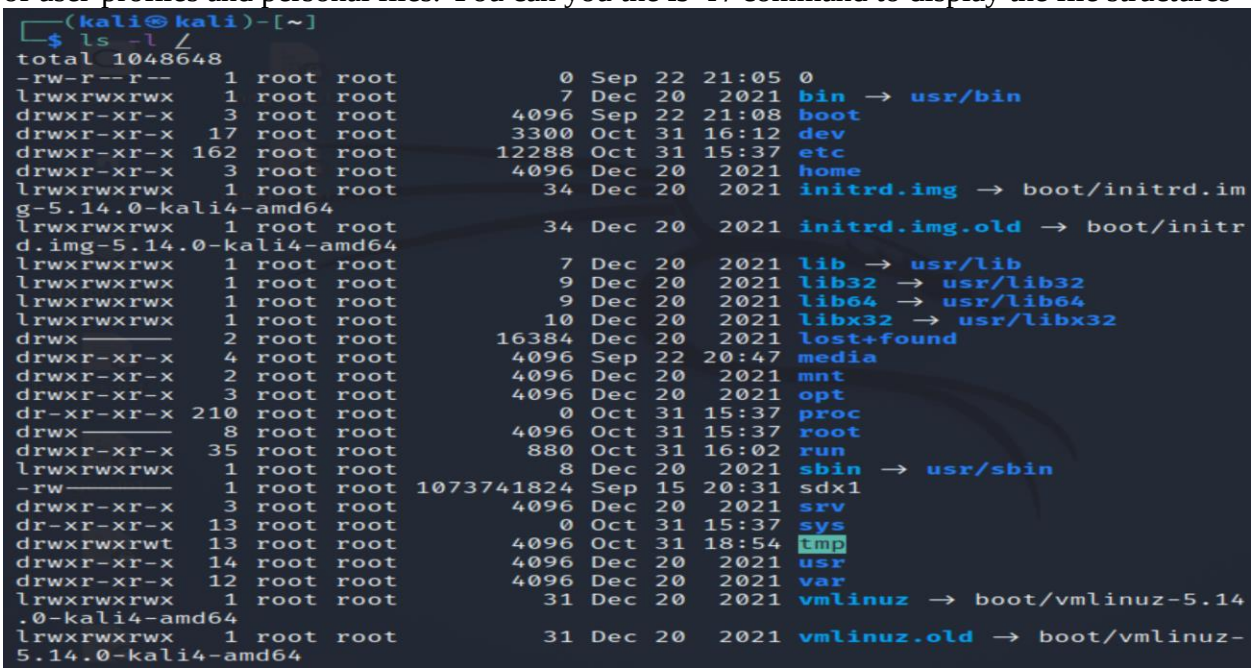
Report 10

For this week's lab, we covered the first part of Linux Forensics. This lab focused on Volatile Data Collection. Part 1 was on Linux File Structure and Important Files. Part consisted of Collecting Basic Volatile Information. The part involved the Linux Firewall, SSH Service, and Port Scanning using Nmap.

In the first part of the lab, we discussed the importance of file structure in Linux. These files include **/media/** which is the mount folder for removable media and **/home/** which has the data of user-profiles and personal files. You can you the **ls -l /** command to display the file structures



Kali also stores vital information in other files as well. These include **/etc/ssh/sshd_config** which is the SSH server configuration and the **/proc/crypto** which contains a List of ciphers, hashing algorithms and authentication algorithms supported by the kernel.

To display these files, you can use the **cat** command. To display the uptime, you need to use the **cat /proc/uptime.** Here below are the command and the output.

To display the swaps, you need to use the **cat /proc/swaps** command. Here below is the output.

```
┌──(kali㉿kali)-[~]
└─$ cat /proc/swaps
Filename                                Ty
pe                Size                  Used      P
riority
/dev/sda5                               pa
rtition 998396            0             -2
```

The **/etc/resolv.conf** file contains information that is read by the resolver routines the first time they are invoked by a process. To display the contents, use the **cat /etc/resolv.conf**. Below is the output

```
┌──(kali㉿kali)-[~]
└─$ cat /etc/resolv.conf
# Generated by NetworkManager
search attlocal.net
nameserver 192.168.1.254
```

The **cat /proc/version** command displays the specifics about the version of Linux kernel used in your distribution and confirms the version of a GCC compiler used to build it. Also, include the date and time when it was built. You can see the out of my Linux version below.

```
┌──(kali㉿kali)-[~]
└─$ cat /proc/version
Linux version 5.14.0-kali4-amd64 (devel@kali.org) (gcc
-10 (Debian 10.3.0-12) 10.3.0, GNU ld (GNU Binutils fo
r Debian) 2.37) #1 SMP Debian 5.14.16-1kali1 (2021-11-
05)
```

The part 2 of involved collecting basic volatile information. One of the first tasks you should do when you are investigating a system is to identify the hostname. To do this use the **hostname** command. Below you can see the name of my Linux machine.

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

If you want to display the current date, you need to use the **date** command. Below is the output.

```
┌──(kali㉿kali)-[~]
└─$ date
Mon Oct 31 07:19:09 PM CDT 2022
```

The **cat /etc/timezone** command displays the time zone. If you want to switch the timezone, you need to get the list of time zones using the **timedatectl list-timezones** command. This displays a list of time zones. Once you select the desired timezone you can use **the sudo timedatectl set-timezone** "desired timezone". Below you can see the timezone that is on my system.

```
┌──(kali㉿kali)-[~]
└─$ cat /etc/timezone
America/Chicago
```

The **cat /proc/uptime** command displays the server uptime in seconds. The first number in the command output is the total number of seconds the server is up.

```
┌──(kali㉿kali)-[~]
└─$ uptime
 19:22:46 up  3:45,  1 user,  load average: 0.03, 0.06, 0.07
```

To show all the IP addresses of the interfaces and related info that is on your Linux machine, use the **ip addr** command. Here below I have 4 network interfaces in my Linux machine.

```
┌──(kali㉿kali)-[~]
└─$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN gro
up default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast st
ate UP group default qlen 1000
    link/ether 08:00:27:50:4c:14 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute
 eth0
       valid_lft 72809sec preferred_lft 72809sec
    inet6 fe80::a00:27ff:fe50:4c14/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group def
ault qlen 1000
    link/ether 08:00:27:c8:08:2e brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group default
    link/ether 02:42:cc:30:08:96 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

If you want to display the link layer information you can use the **ip link show.** This **will** fetch the characteristics of the link layer devices currently available. Any networking device which has a driver loaded can be classified as an available device.

```
┌──(kali㉿kali)-[~]
└─$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mod
e DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast st
ate UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:50:4c:14 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFA
ULT group default qlen 1000
    link/ether 08:00:27:c8:08:2e brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN mode DEFAULT group default
    link/ether 02:42:cc:30:08:96 brd ff:ff:ff:ff:ff:ff
```

You can also specify the interface that you want to want to show. In my case below, I used the command Ip link show eth1 to display that specific interface.

```
┌──(kali㉿kali)-[~]
└─$ ip link show eth1
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFA
ULT group default qlen 1000
    link/ether 08:00:27:c8:08:2e brd ff:ff:ff:ff:ff:ff
```

If you want to display the statistics for the specific interface, you can use the **ip -s link show dev eth0.** Below is an example displaying the eth0 interface statistics.

```
┌──(kali㉿kali)-[~]
└─$ ip -s link show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast st
ate UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:50:4c:14 brd ff:ff:ff:ff:ff:ff
    RX:  bytes packets errors dropped  missed   mcast
           590       1      0       0       0       0
    TX:  bytes packets errors dropped carrier collsns
          1762      22      0       0       0       0
```

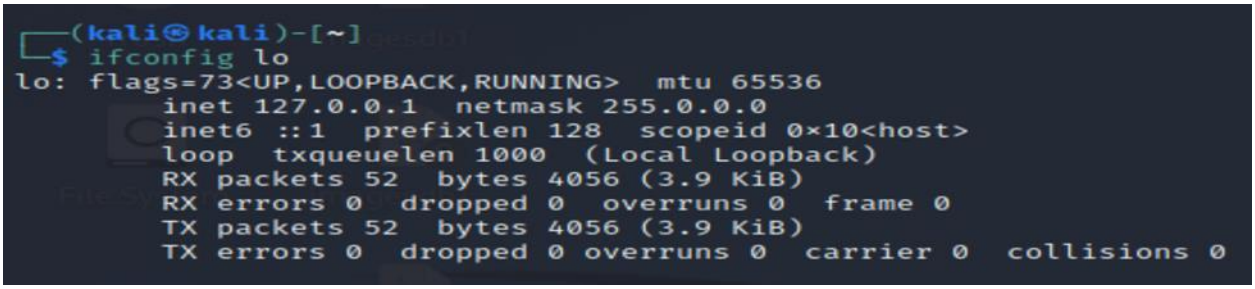
Another command to collect network information is by using the **ip route** command. With this command, you can show the routing table you can use this to manipulate entries in the kernel routing tables. To display the routing tables in your system you need to use **ip route** command. Here below are the route tables for my system.

```
┌──(kali㉿kali)-[~]
└─$ ip route
default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15 m
etric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.
15 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 17
2.17.0.1 linkdown
```

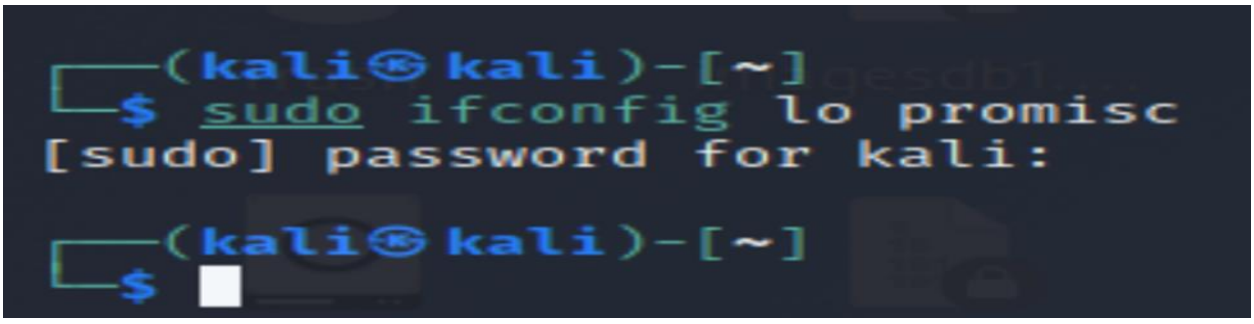
The command **ip neigh** can manipulate neighbor objects that establish bindings between protocol addresses and link layer addresses for hosts sharing the same link. To display the neighbor entries, use the **ip neigh** command. Below is the output for my system.

It is also important to display the socket statistics as well. Here are some commands that you can use in conjunction with **ss** command. The **-a** shows all sockets, **-e** shows detailed socket information, **-o** shows the timer information, **-n** refers to addresses that don't resolve, and the **-p Show** the process using the socket. Here below I used **the ss -a | head** command to show the info on all the sockets.

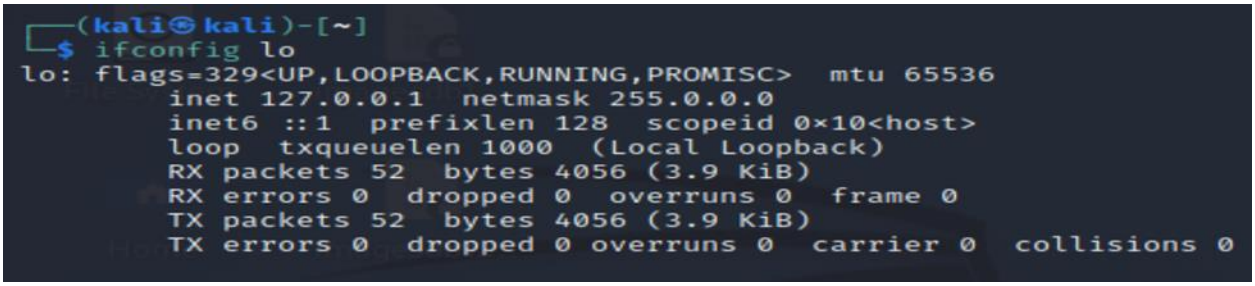
The **ifconfig lo** command shows the loopback interface. This is a special network interface that the system uses to communicate with itself.



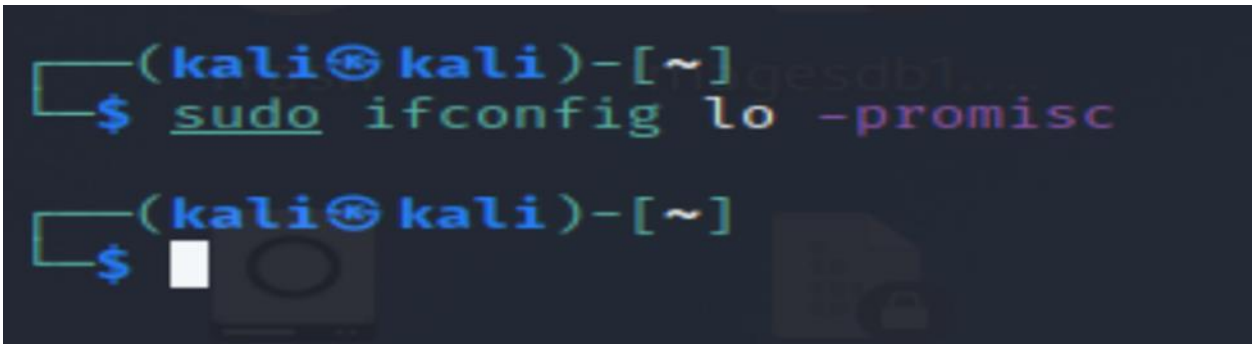
With that in mind, you can enable the loopback to run promiscuously. If promiscuous mode is enabled, all packets on the network will be received by the interface. To do this you need to use the command **sudo ifconfig lo promisc.** Note after writing the command no output is shown**.**



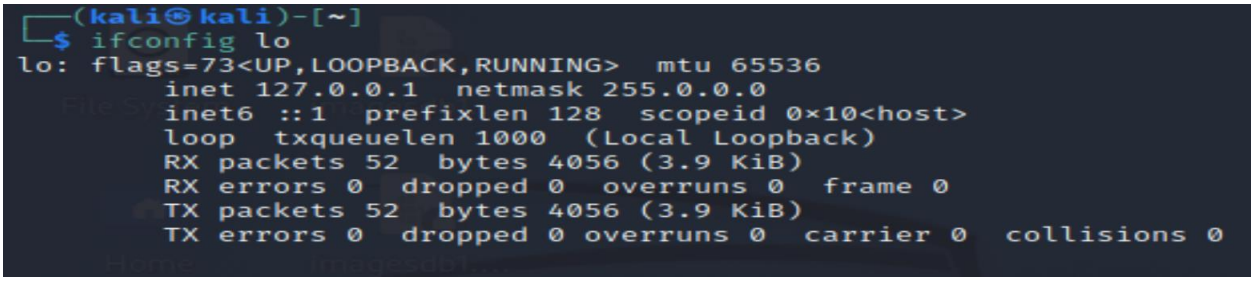
We can then verify our change by using the previous command **ifconfig lo.** Here below is the output. Note that the loopback is “RUNNING PROMISC”



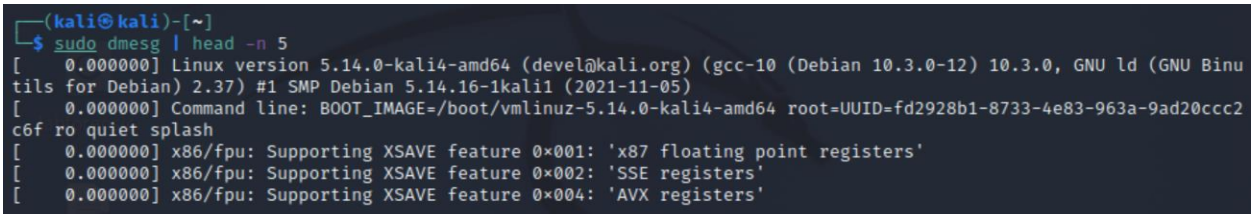
When you want to stop promiscuous mode, you need to use the **sudo ifconfig lo -promisc command.** No output is displayed.



To verify the changes, you need to use **ifconfig lo** command again You can below that we disabled promiscuous mode.



The **dmesg** command displays kernel-related messages retrieved from the kernel ring buffer. The ring buffer stores information about hardware, device driver initialization, and messages from kernel modules that take place during system startup. Forensic investigators can track actions performed on the machine. You can use the **sudo dmesg | head -n 5** commands to display the messages. Note that the head command prints the first lines (10 lines by default). If you in conjunction with the **-n 5**, it will print the first 5 lines. This is the output below.



Here below I display only one line using the **sudo dmesg | head -n 1** command.

```
┌──(kali㉿kali)-[~]
└─$ sudo dmesg | head -n 1
[    0.000000] Linux version 5.14.0-kali4-amd64 (devel@kali.org) (gcc-10 (Debian 10.3.0-12) 10.3.0, GNU ld (GNU Binu
tils for Debian) 2.37) #1 SMP Debian 5.14.16-1kali1 (2021-11-05)
```

You can specify what you want to display as well. Such as when we enabled and disabled promiscuous mode for the device lo. If you want to display the output where the lines contain "device lo" you need to use the **sudo dmesg -T | grep "device lo.** You can see below the output.

```
┌──(kali㉿kali)-[~]
└─$ sudo dmesg | grep "device lo"
[14512.798276] device lo entered promiscuous mode
[14656.455003] device lo left promiscuous mode
```

The **lsof** command stands for **LiSt Open Files** and shows open files and which process uses them. Since Linux sees every object as a file, such as devices, and directories. Unidentifiable files that are open prevent users from modifying them. To display the opened file, use the **sudo lsof | head -n** command. Here below is the output showing the first 4 lines.

```
┌──(kali㉿kali)-[~]
└─$ sudo lsof | head -n 4
[sudo] password for kali:
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND     PID   TID TASKCMD               USER   FD      TYPE             DEVICE SIZE/OFF       NODE NAME
systemd       1                             root  cwd       DIR                8,1    36864          2 /
systemd       1                             root  rtd       DIR                8,1    36864          2 /
systemd       1                             root  txt       REG                8,1  1817072    1052746 /usr/lib/systemd/sy
stemd
```

To see the list of files that were opened by a network connection using the **sudo lsof -i** command. Here is the output below.

```
┌──(kali㉿kali)-[~]
└─$ sudo lsof -i
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
NetworkMa  513 root   26u  IPv4 140974      0t0  UDP 10.0.2.15:bootpc→10.0.2.2:bootps
container  701 root   13u  IPv4  14923      0t0  TCP localhost:36359 (LISTEN)
```

The **sudo lsof -u kali | wc -**l command will display and counts the number of files opened. Note that you need to specify the user. In my system, the username is *kali*.

```
┌──(kali㉿kali)-[~]
└─$ sudo lsof -u kali | wc -l
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
4443
```

The command **sudo lsof -u kali | head -n** will display the first 5 results. The **-ui** option prints all files opened by everyone except a specific user. Here below is the output.

```
┌──(kali㉿kali)-[~]
└─$ sudo lsof -u kali | head -n 5
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND     PID USER   FD      TYPE             DEVICE SIZE/OFF       NODE NAME
systemd    1069 kali  cwd       DIR                8,1    36864          2 /
systemd    1069 kali  rtd       DIR                8,1    36864          2 /
systemd    1069 kali  txt       REG                8,1  1817072    1052746 /usr/lib/systemd/systemd
systemd    1069 kali  mem       REG                8,1   161864    1048642 /usr/lib/x86_64-linux-gnu/libgpg-error.so
.0.33.0
```

If you want to display files that were opened by a particular process, you can specify the process. In this example, I used the **sudo lsof -c ssh** the process that is used in SSH. Note that some of the types of files were DIR (Directory File), REF (Regular File), and CHR (Special Character File). Here below is the e output.

```
┌──(kali㉿kali)-[~]
└─$ sudo lsof -c ssh
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND    PID USER   FD   TYPE             DEVICE SIZE/OFF    NODE NAME
ssh-agent 1181 kali  cwd    DIR               8,1    36864       2 /
ssh-agent 1181 kali  rtd    DIR               8,1    36864       2 /
ssh-agent 1181 kali  txt    REG               8,1   370824 1059926 /usr/bin/ssh-agent
ssh-agent 1181 kali  mem    REG               8,1    14408 1063299 /usr/lib/x86_64-linux-gnu/libpthread.so.0
ssh-agent 1181 kali  mem    REG               8,1    14408 1063281 /usr/lib/x86_64-linux-gnu/libdl.so.2
ssh-agent 1181 kali  mem    REG               8,1  2049032 1063278 /usr/lib/x86_64-linux-gnu/libc.so.6
ssh-agent 1181 kali  mem    REG               8,1  3081088 1068105 /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
ssh-agent 1181 kali  mem    REG               8,1   206640 1058384 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
ssh-agent 1181 kali    0u   CHR               1,3      0t0       4 /dev/null
ssh-agent 1181 kali    1u   CHR               1,3      0t0       4 /dev/null
ssh-agent 1181 kali    2u   CHR               1,3      0t0       4 /dev/null
ssh-agent 1181 kali    3u  unix 0x000000006566cfdd      0t0   17997 /tmp/ssh-XXXXXXmjSZZh/agent.1097 type=STREAM (LI
STEN)
```

The **mount -l** command list all the file systems that are mounted and the corresponding directories. Here below is the list of them in my system.

```
┌──(kali㉿kali)-[~]
└─$ mount -l
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=972148k,nr_inodes=243037,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=202952k,mode=755,inode64)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro) [labforensic]
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
none on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=29,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,p
ipe_ino=12192)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
sunrpc on /run/rpc_pipefs type rpc_pipefs (rw,relatime)
none on /run/credentials/systemd-sysusers.service type ramfs (ro,nosuid,nodev,noexec,relatime,mode=700)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=202952k,nr_inodes=50738,mode=700,uid=1000,gid=1000
,inode64)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
```

The **df** command information about total space and available space on a file system.
The *FileSystem* parameter specifies the name of the device on which the file system resides, the directory on which the file system is mounted, or the relative path name of a file system. Below is the output for my system.

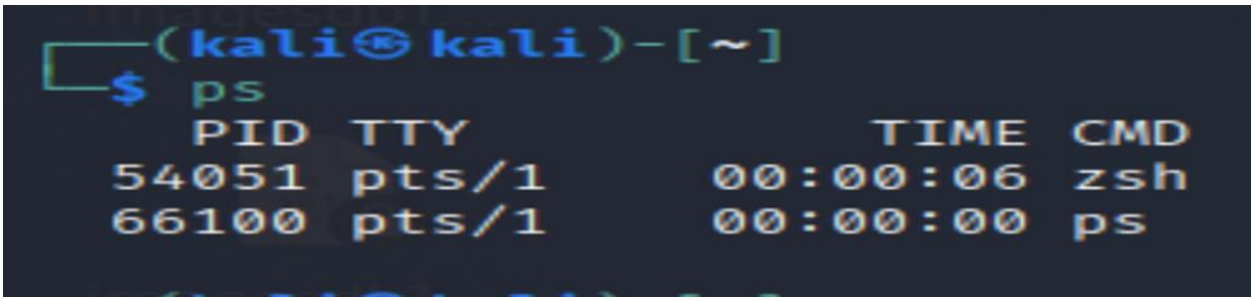
```
┌──(kali㉿kali)-[~]
└─$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
udev              972148         0    972148   0% /dev
tmpfs             202952       984    201968   1% /run
/dev/sda1       81000912  23950004  52890296  32% /
tmpfs            1014760         0   1014760   0% /dev/shm
tmpfs               5120         0      5120   0% /run/lock
tmpfs             202952        68    202884   1% /run/user/1000
```

To display the modules that were loaded into the kernel use the **lsmod | head -n 6** command.

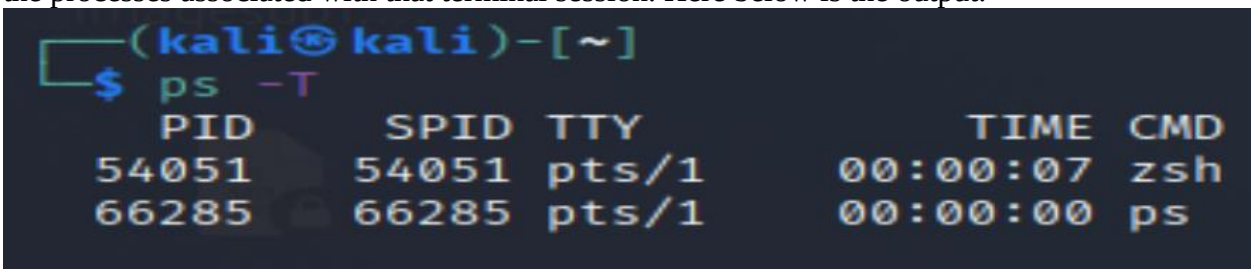
```
┌──(kali㉿kali)-[~]
└─$ lsmod | head -n 6
Module                  Size  Used by
mptcp_diag             16384  0
tcp_diag               16384  0
udp_diag               16384  0
raw_diag               16384  0
inet_diag              28672  4 tcp_diag,mptcp_diag,raw_diag,udp_diag
```

The **modinfo** command is used to display information about a Linux Kernel module. This command extracts the information from the Linux kernel modules given on the command line. The **vsock** protocol allows for socket communication between a virtual machine and its host. We can use the **modinfo vsock** command to display the information of **vsock** module. Here below is the output. The **insmod and rmmode** can be used to insert or remove modules.

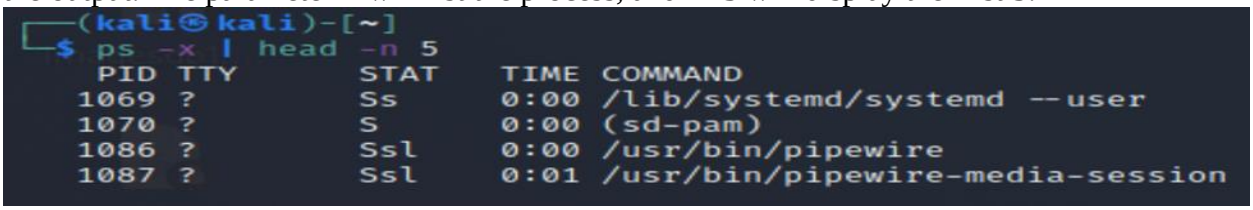
The **ps** command is used to get information on a process. It shows the *PID* number (Unique Process ID) and the Terminal type user is logged in to (TTY). TIME is the amount of *CPU* in minutes and seconds a process has been running. *CMD* is the name of the command that launched the process. Here below is the output.



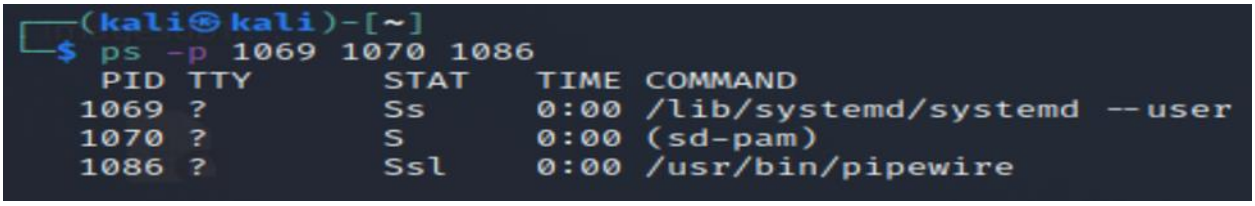
To view, all the running processes use the **ps -e | head** command. The **ps -T** command will view the processes associated with that terminal session. Here below is the output.



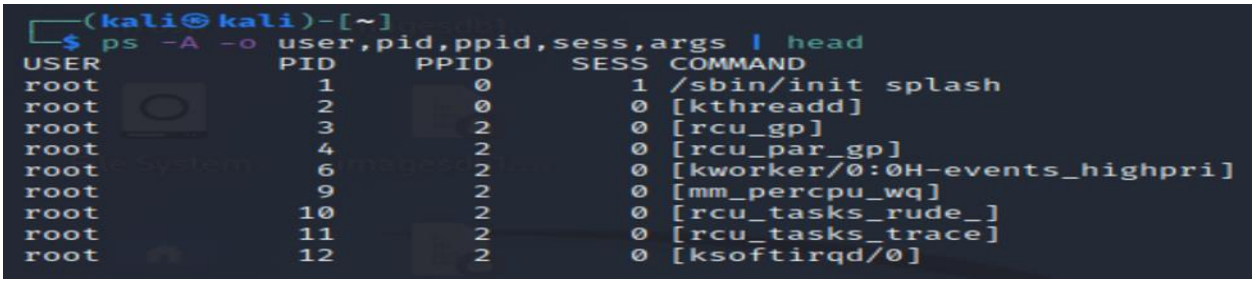
If you want to view the process selection by list, use the **ps -x | head -n 5** command**.** Below is the output. The parameter**-x** will list the process, and **-n 5** will display the first 5.



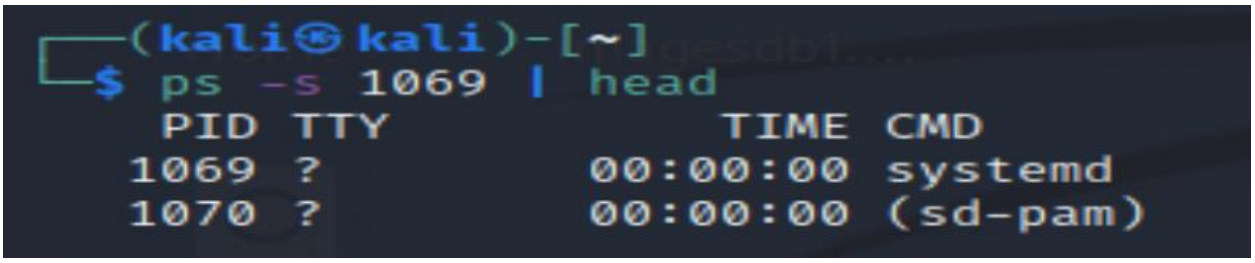
Now if you want to list by process ID use the **ps – p** followed by your desired process ID numbers. In this case, I used the PID numbers for the previous command output. These were PID 1069,1070 and 1086. Below is the output.



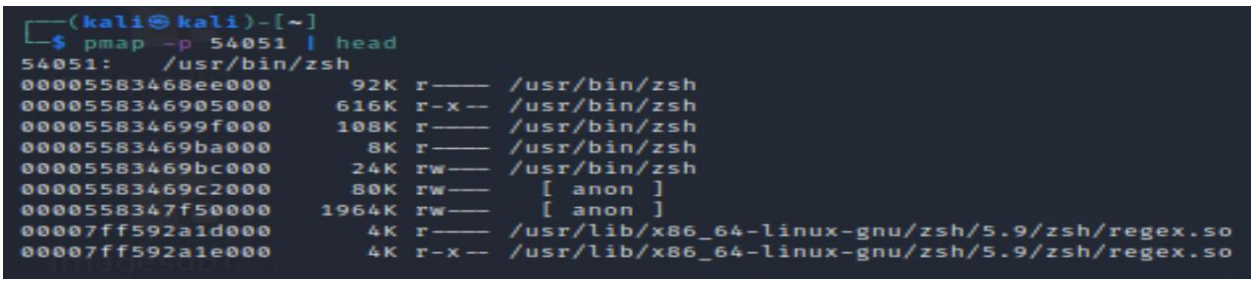
To list the processes with specific parameters you can use the list of the parameters you want to use. The **-o** parameter will show it in a user-oriented format. The **-A** parameter will View all processes except both session leaders and processes not associated with a terminal. Here below is used the **ps -A -o user,pid,ppid,sess,args | head** command as an example below.



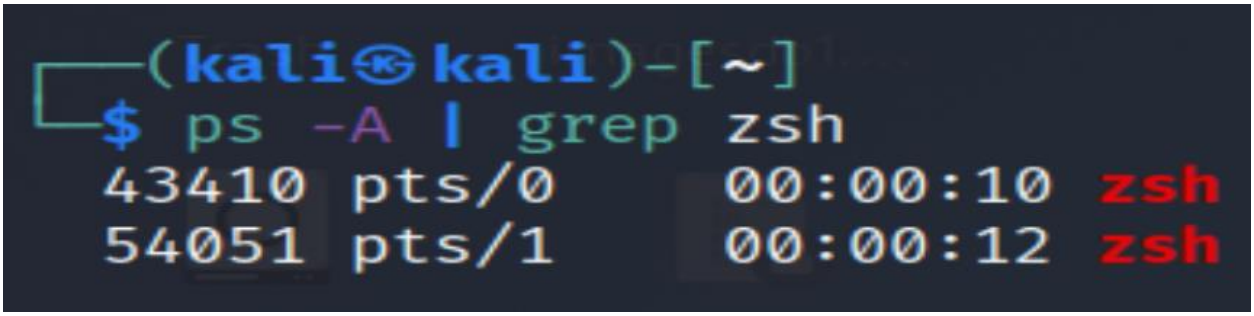
The **-s** parameter will sort the process. The output will be a list of processes attributed to a particular session. Here below is an example using the command **ps -s 1069 | head.**



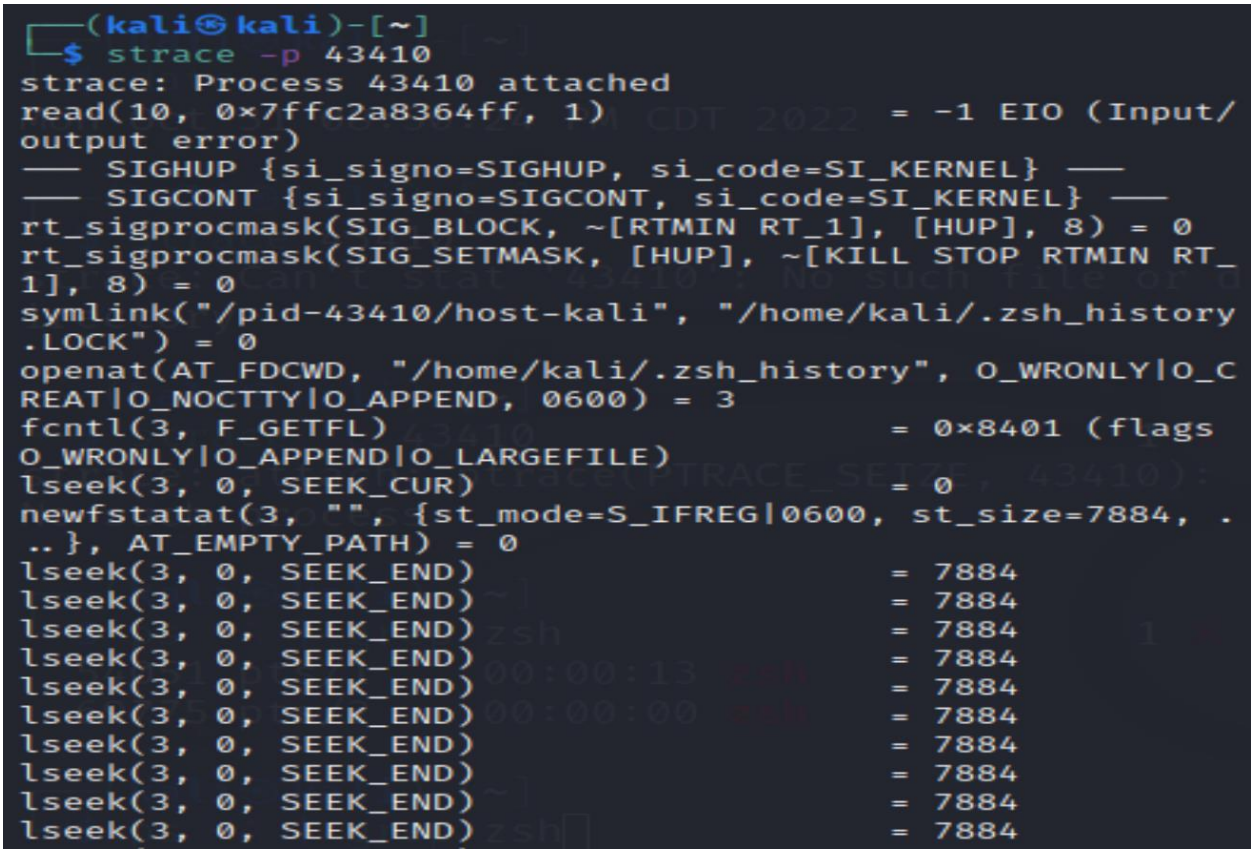
The **pmap** command is utilized to get reports of the memory map of a process or process. Below I use the command **pmap -p 54051 | head** to get the process information on the terminal session.



The **strace** command intercepts and records the system calls which are called by a process and the signals which are received by a process. In this example, we will use this command to find the zsh (Z shell) and then trace the system calls from another terminal and display it. Will first use the command **ps -A | grep zsh** to display the PID number Below is the output.



Next by opening another terminal, we will write the command **strace -p 43410** to start to trace its system calls using this terminal. Then I used the command **hostname** on the first terminal. The system call was printed in the second terminal. You can see below the output.



The last part of the lab involved the Linux Firewall, SSH Service, and Port Scanning using Nmap. It is important to be aware that open port scanning without authorization can be considered illegal as this is a method attackers use for the reconnaissance phase of an attack.

The command **nmap** is used for port scanning. We will scan **scanme.nmap.org**. The **-s** parameter scans and The **-T** parameter refers to TCP. Listening on open ports is an indication of an application. Ports are filtered if they are intercepted by firewalls, filters, or midway devices, and **nmap** cannot determine whether they are open or closed. If **nmap** finds the port accessible but there is no application listening on it, it is closed. Here below is the output.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sT scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 20
22-10-31 20:55 CDT
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.094s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 996 filtered tcp ports (no-response)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
9929/tcp  open  nping-echo
31337/tcp open  Elite
```

Before using for **ufw** firewall service for the next step, the first task you need to do is to upgrade and install packages. The first command is **sudo apt-get upgrade**, then once it is upgrading, use the following command to install the **ufw** firewall which is **sudo apt-get install ufw.** Now we can display the status using **sudo systemctl status ufw** command. Note that the status should be inactive because we have not started it yet. Here is below the output.

```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl status ufw
○ ufw.service - Uncomplicated firewall
     Loaded: loaded (/lib/systemd/system/ufw.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Thu 2022-11-03 14:15:28 CDT; 3s ago
       Docs: man:ufw(8)
    Process: 332 ExecStart=/lib/ufw/ufw-init start quiet (code=exited, status=0/SUCCESS)
    Process: 96420 ExecStop=/lib/ufw/ufw-init stop (code=exited, status=0/SUCCESS)
   Main PID: 332 (code=exited, status=0/SUCCESS)
        CPU: 309ms
```

Now if we want to start the firewall service, you need to use **sudo systemctl start ufw** command. Below is the output. Note that nothing is displayed.

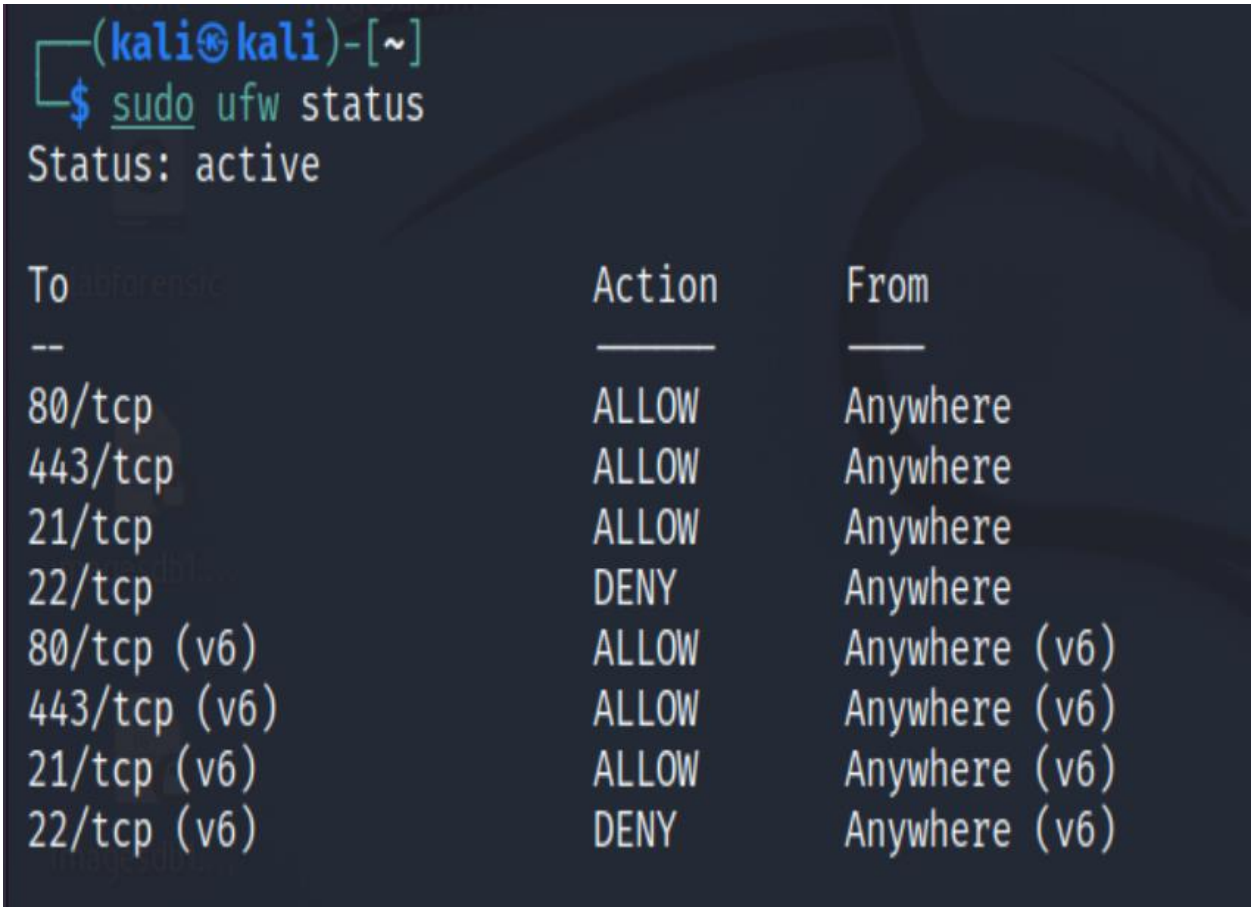
```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl start ufw
```

To check if the **ufw** service is running use **sudo systemctl status ufw.** Here below is output. You can see that it is “active”.

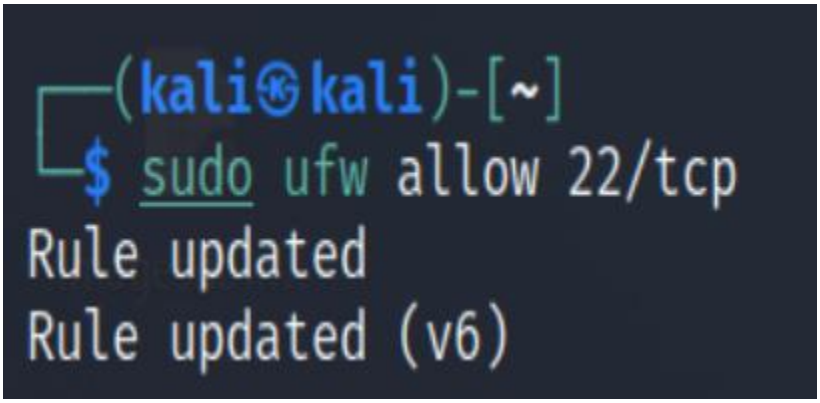
```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl status ufw
● ufw.service - Uncomplicated firewall
     Loaded: loaded (/lib/systemd/system/ufw.service; enabled; vendor preset: enabled)
     Active: active (exited) since Mon 2022-10-31 15:37:16 CDT; 2 days ago
       Docs: man:ufw(8)
    Process: 332 ExecStart=/lib/ufw/ufw-init start quiet (code=exited, status=0/SUCCESS)
   Main PID: 332 (code=exited, status=0/SUCCESS)
        CPU: 81ms

Oct 31 15:37:16 kali systemd[1]: Finished Uncomplicated firewall.
```

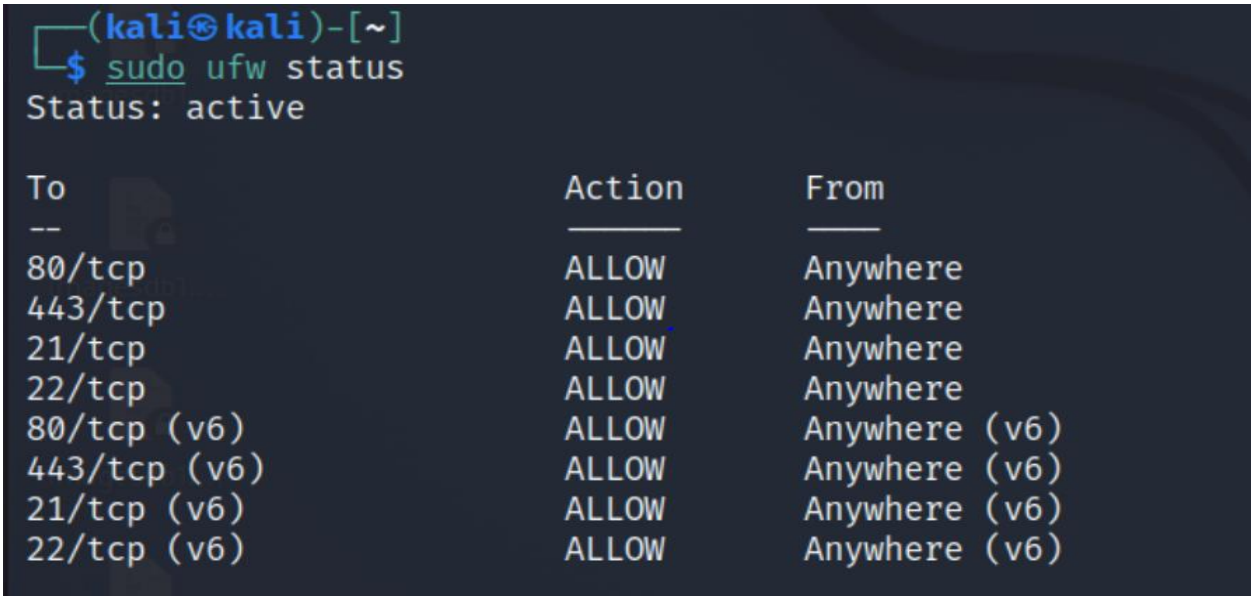
You can display the status of the firewall by using **sudo ufw status**. Below is the output. There are a couple of port numbers that allowed such as port 80 and 443. Note that port 22 is denied.



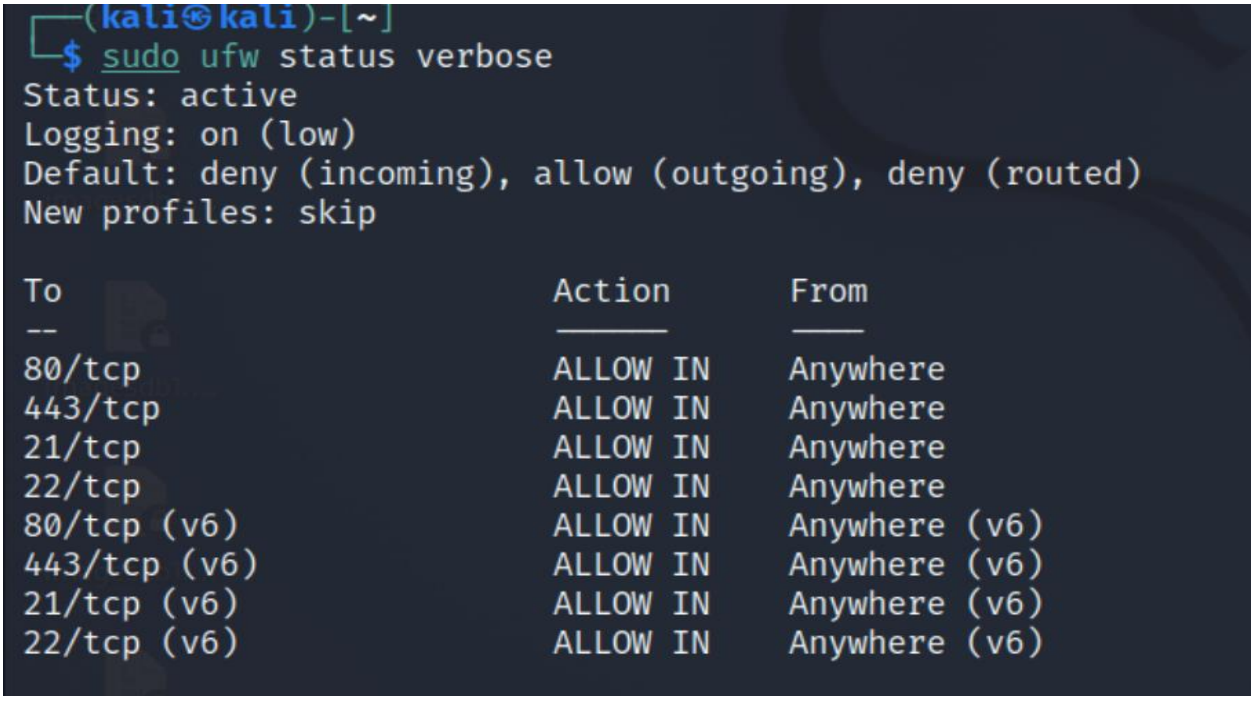
If you want to allow port 22 (SSH service) you need to use the command **sudo ufw allow 22/tcp.** SSH uses TCP. Below is the output. Note the rules are updated.



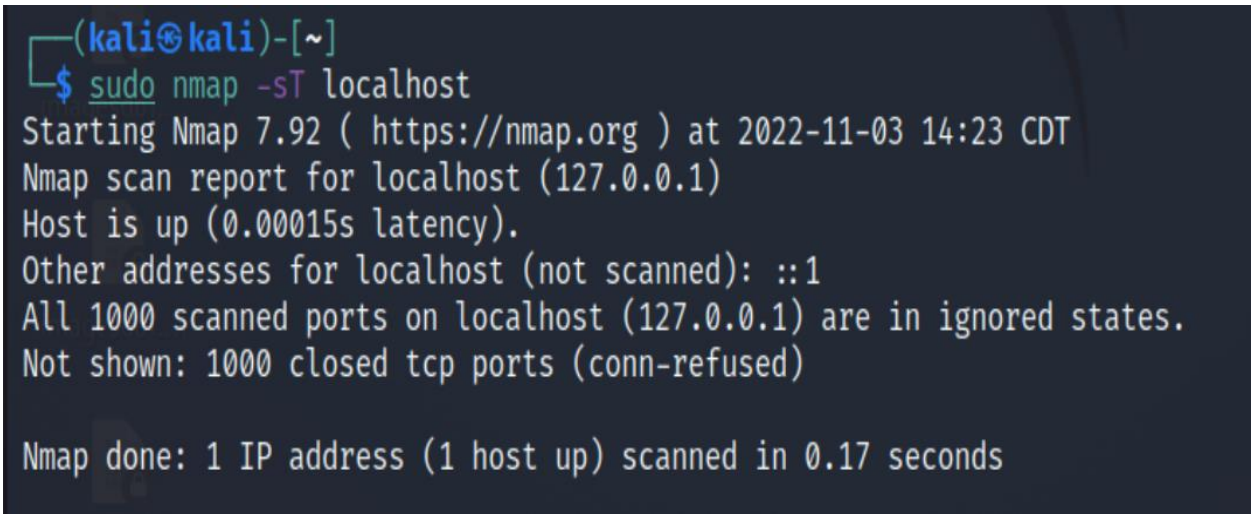
Now when we run previous command **sudo ufw status** to check the status. You will see that now that port is allowed. Here is the output. The (v6) are reserved for version ip6 protocol.



You can get a more detailed look at the firewall status using **sudo ufw status verbose** command. it shows the logging intensity. If set to high, the act of logging all network monitoring itself can hamper the performance of your server. By default, logging is set to low. Any incoming traffic is denied, it will allow outgoing traffic and deny routed traffic. Here below is the output.



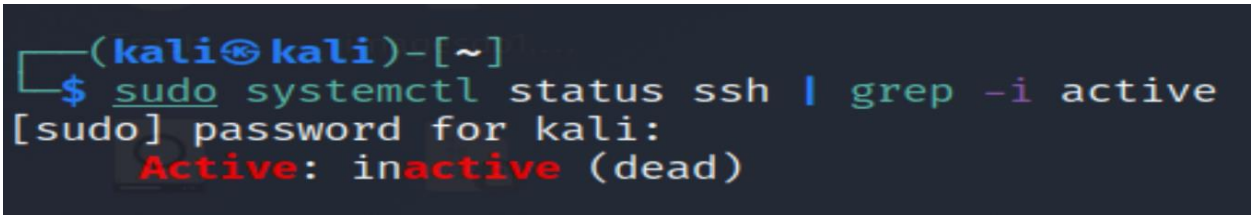
Here, I used **sudo nmap -sT localhost** command to get info of the local host. Here is the output.



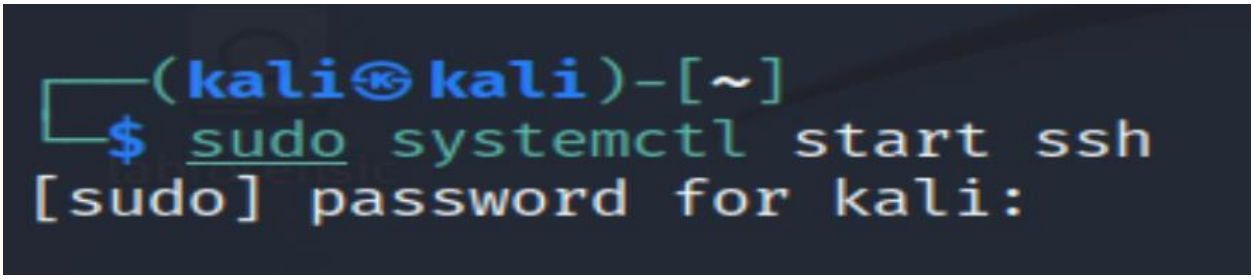
I ran the command **sudo nmap -sT 127.0.0.1** command on my Ubuntu machine to check the status of the ports. Note that the port 22 (ssh) is closed.

```
Host is up (0.0012s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT   STATE   SERVICE
22/tcp closed  ssh
```

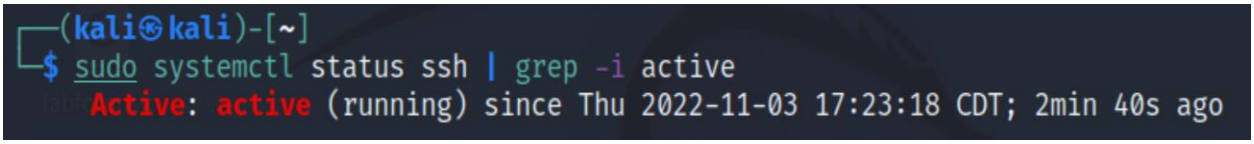
SSH is a protocol that is used to manage remote systems. To use this, we first must install this by using **sudo apt-get install openssh-server.** The next this is to display the status. To do that, you need to use **sudo systemctl status ssh | grep -I active** command**.** You can see below that the service is inactive.



To start the service, use **sudo systemctl start ssh** command**.** Below is the output. Note that nothing is displayed.



Now, to check if ssh is running use **sudo systemctl status ssh | grep -i active** command. You can see below that shows the time and date the service was started.



Here I used **sudo nmap -sT 127.0.0.1** command on my Ubuntu machine. Note that port 22 ssh is now open.

```
Host is up (0.0012s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT   STATE   SERVICE
22/tcp open    ssh
```

Now we can establish communication using SSH. To do this, use **ssh kali@** command followed by the Ip. In my case, used **127.0.0.1.**

Now that I am connected, I wrote the command on my Ubuntu machine to create a text file. Here is the command and details of the file which is **echo Test from the Ubuntu host > test.txt.**

I opened the terminal in my kali machine and used **ls test.txt** command to find the file I created in the ubuntu machine.