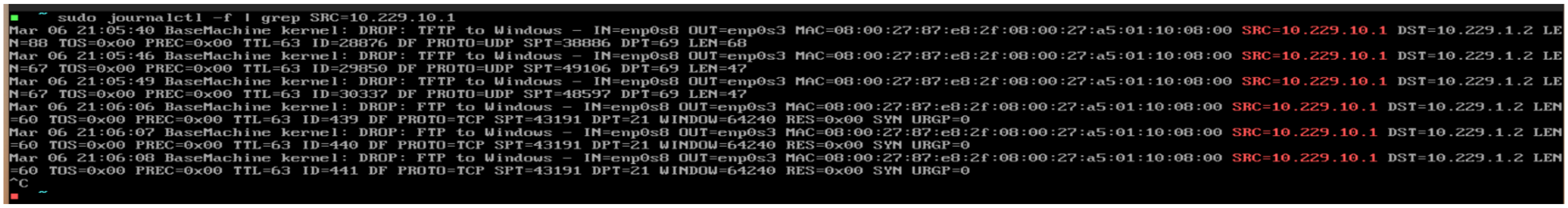
PART 1

This is how we checked our rules worked:

“Allow any host from any network to access the tftp and ftp service of your Windows host EXCEPT for hosts from the network 10.229.10.0/24”



```
~ sudo journalctl -f | grep SRC=10.229.10.1
Mar 06 21:05:40 BaseMachine kernel: DROP: TFTP to Windows - IN=enp0s8 OUT=enp0s3 MAC=08:00:27:87:e8:2f:08:00:27:a5:01:10:08:00 SRC=10.229.10.1 DST=10.229.1.2 LE
N=88 TOS=0x00 PREC=0x00 TTL=63 ID=28876 DF PROTO=UDP SPT=38886 DPT=69 LEN=68
Mar 06 21:05:46 BaseMachine kernel: DROP: TFTP to Windows - IN=enp0s8 OUT=enp0s3 MAC=08:00:27:87:e8:2f:08:00:27:a5:01:10:08:00 SRC=10.229.10.1 DST=10.229.1.2 LE
N=67 TOS=0x00 PREC=0x00 TTL=63 ID=29850 DF PROTO=UDP SPT=49106 DPT=69 LEN=47
Mar 06 21:05:49 BaseMachine kernel: DROP: TFTP to Windows - IN=enp0s8 OUT=enp0s3 MAC=08:00:27:87:e8:2f:08:00:27:a5:01:10:08:00 SRC=10.229.10.1 DST=10.229.1.2 LE
N=67 TOS=0x00 PREC=0x00 TTL=63 ID=30337 DF PROTO=UDP SPT=48597 DPT=69 LEN=47
Mar 06 21:06:06 BaseMachine kernel: DROP: FTP to Windows - IN=enp0s8 OUT=enp0s3 MAC=08:00:27:87:e8:2f:08:00:27:a5:01:10:08:00 SRC=10.229.10.1 DST=10.229.1.2 LEN
=60 TOS=0x00 PREC=0x00 TTL=63 ID=439 DF PROTO=TCP SPT=43191 DPT=21 WINDOW=64240 RES=0x00 SYN URGP=0
Mar 06 21:06:07 BaseMachine kernel: DROP: FTP to Windows - IN=enp0s8 OUT=enp0s3 MAC=08:00:27:87:e8:2f:08:00:27:a5:01:10:08:00 SRC=10.229.10.1 DST=10.229.1.2 LEN
=60 TOS=0x00 PREC=0x00 TTL=63 ID=440 DF PROTO=TCP SPT=43191 DPT=21 WINDOW=64240 RES=0x00 SYN URGP=0
Mar 06 21:06:08 BaseMachine kernel: DROP: FTP to Windows - IN=enp0s8 OUT=enp0s3 MAC=08:00:27:87:e8:2f:08:00:27:a5:01:10:08:00 SRC=10.229.10.1 DST=10.229.1.2 LEN
=60 TOS=0x00 PREC=0x00 TTL=63 ID=441 DF PROTO=TCP SPT=43191 DPT=21 WINDOW=64240 RES=0x00 SYN URGP=0
^C
~
```

All the dropped packets are logged. Following was the result of running command

“Sudo journalctl -f | grep 10.229.10.1”

We can see that any UDP port 69 (TFTP) from 10.229.10.1 to 10.229.1.2 and TCP port 21 (FTP) are being dropped. It was further verified that not other TFTP and FTP is dropped using command

sudo journalctl -f | awk '/DST=10.229.1.2/ && /DPT=(21|69)/'

Running it for some time, showed no other packets other than the one coming from 10.229.10.1 were being dropped

“Allow any host from any network to access the tftp and ftp service of your Linux host EXCEPT for hosts from the network 10.229.11.0/24”

```
~ sudo cp /home/shared/create_iptables.sh /root/
~ sudo /root/create_iptables.sh
Setting up firewall rules...
net.ipv4.ip_forward = 1
Firewall rules applied successfully!
~ sudo journalctl -f | grep SRC=10.229.11.1
Mar 06 20:57:13 BaseMachine kernel: DROP: TFTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=88 TOS
=0x00 PREC=0x00 TTL=64 ID=4420 DF PROTO=UDP SPT=53903 DPT=69 LEN=68
Mar 06 20:57:16 BaseMachine kernel: DROP: TFTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=88 TOS
=0x00 PREC=0x00 TTL=64 ID=4916 DF PROTO=UDP SPT=59670 DPT=69 LEN=68
Mar 06 20:57:26 BaseMachine kernel: DROP: TFTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=67 TOS
=0x00 PREC=0x00 TTL=64 ID=7691 DF PROTO=UDP SPT=59893 DPT=69 LEN=47
Mar 06 20:57:29 BaseMachine kernel: DROP: TFTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=67 TOS
=0x00 PREC=0x00 TTL=64 ID=7798 DF PROTO=UDP SPT=33007 DPT=69 LEN=47
Mar 06 20:57:40 BaseMachine kernel: DROP: FTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=60 TOS=
0x00 PREC=0x00 TTL=64 ID=22633 DF PROTO=TCP SPT=50607 DPT=21 WINDOW=64240 RES=0x00 SYN URGP=0
Mar 06 20:57:41 BaseMachine kernel: DROP: FTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=60 TOS=
0x00 PREC=0x00 TTL=64 ID=22634 DF PROTO=TCP SPT=50607 DPT=21 WINDOW=64240 RES=0x00 SYN URGP=0
Mar 06 20:57:42 BaseMachine kernel: DROP: FTP to Linux - IN=enp0s8 OUT= MAC=08:00:27:87:e8:2f:08:00:27:aa:a1:a1:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=60 TOS=
0x00 PREC=0x00 TTL=64 ID=22635 DF PROTO=TCP SPT=50607 DPT=21 WINDOW=64240 RES=0x00 SYN URGP=0
^C
```

Using *“Sudo journalctl -f | grep 10.229.11.1”*, we can see that any UDP port 69 (TFTP) from 10.229.11.1 to 10.229.1.1 and TCP port 21 (FTP) are being dropped.

It was further verified that not other TFTP and FTP is dropped using command *sudo journalctl -f | awk '/DST=10.229.1.1/ && /DPT=(21|69)/'*

Running it for some time, showed no other packets other than the one coming from 10.229.11.1 were being dropped

“Allow any host from any network to connect to the ssh service port on any of your group’s hosts as well as allow ICMP echo messages (pings) from any host from any network.”

Running *“dmesg | grep DPT=22”*, shows no output implying that ssh requests from Other Linux 1 and Other Linux 2 are notting getting dropped.

```
     ssh [-Q query_option]
 333-assign2 git:(part-1) ■ dmesg | grep DPT=22
 333-assign2 git:(part-1) ■
```

The same is true for ping requests from Other Linux 1 and Other Linux 2.

```
 333-assign2 git:(part-1) ■ dmesg | grep ICMP
 333-assign2 git:(part-1) ■
```

Similarly, ssh and pinging from Our Linux to Our Windows

```
■  333-assign2 git:(part-1) ■ ssh -v rom@10.229.1.2
OpenSSH_9.9p1, OpenSSL 3.4.0 22 Oct 2024
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Reading configuration data /etc/ssh/ssh_config.d/20-systemd-ssh-proxy.conf
debug1: Connecting to 10.229.1.2 [10.229.1.2] port 22.
^C
■  333-assign2 git:(part-1) ■ dmesg | grep DST=22
■  333-assign2 git:(part-1) ■
```

```
■  333-assign2 git:(part-1) ■ ping 10.229.1.2
PING 10.229.1.2 (10.229.1.2) 56(84) bytes of data.
^C
--- 10.229.1.2 ping statistics ---
13 packets transmitted, 0 received, 100% packet loss, time 12155ms

■  333-assign2 git:(part-1) ■ dmesg | grep ICMP
■  333-assign2 git:(part-1) ■
```

Finally, ssh and ping requests from Our Windows produce no dropped packets.

```
C:\Users\Administrator>ping 10.229.1.1

Pinging 10.229.1.1 with 32 bytes of data:
Reply from 10.229.1.1: bytes=32 time<1ms TTL=64
Reply from 10.229.1.1: bytes=32 time<1ms TTL=64
Reply from 10.229.1.1: bytes=32 time<1ms TTL=64
Reply from 10.229.1.1: bytes=32 time<1ms TTL=64

Ping statistics for 10.229.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Administrator>
```

```
■  333-assign2 git:(part-1) ■ dmesg | grep ICMP
■  333-assign2 git:(part-1) ■ dmesg | grep ICMP
■  333-assign2 git:(part-1) ■
```

"If none of the above rules apply, the default is to refuse all other inbound traffic (that is, unless the inbound traffic is caused by your own group's permitted outbound traffic)."

Since we know what type of requests are being sent each minute by Other Linux 1 and Other Linux 2 and the only type of request that is not handled by the above rules are http requests, however, we should still see in the logs that http requests are dropped.

Running the command *"dmesg | grep DPT=80"*, we see that this is the case. The http requests coming from Other Linux 1 and Other Linux 2 are indeed getting dropped by the later rules of our script.

```
[13160.293195] Dropped: IN=enp0s8 OUT= MAC=08:00:27:77:1f:6c:08:00:27:51:49:c1:08:00 SRC=10.229.10.1 DST=10.229.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=30884 DF
 PROTO=TCP SPT=36269 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13161.306828] Dropped: IN=enp0s8 OUT= MAC=08:00:27:77:1f:6c:08:00:27:51:49:c1:08:00 SRC=10.229.10.1 DST=10.229.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=30885 DF
 PROTO=TCP SPT=36269 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13163.288261] Dropped: IN=enp0s8 OUT=enp0s3 MAC=08:00:27:77:1f:6c:08:00:27:51:49:c1:08:00 SRC=10.229.10.1 DST=10.229.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=41
908 DF PROTO=TCP SPT=59405 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13164.293248] Dropped: IN=enp0s8 OUT=enp0s3 MAC=08:00:27:77:1f:6c:08:00:27:51:49:c1:08:00 SRC=10.229.10.1 DST=10.229.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=41
909 DF PROTO=TCP SPT=59405 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13165.306728] Dropped: IN=enp0s8 OUT=enp0s3 MAC=08:00:27:77:1f:6c:08:00:27:51:49:c1:08:00 SRC=10.229.10.1 DST=10.229.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=41
910 DF PROTO=TCP SPT=59405 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13173.432487] Dropped: IN=enp0s8 OUT= MAC=08:00:27:77:1f:6c:08:00:27:b3:1f:e5:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=48732 DF
 PROTO=TCP SPT=59123 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13174.444940] Dropped: IN=enp0s8 OUT= MAC=08:00:27:77:1f:6c:08:00:27:b3:1f:e5:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=48733 DF
 PROTO=TCP SPT=59123 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13175.458279] Dropped: IN=enp0s8 OUT= MAC=08:00:27:77:1f:6c:08:00:27:b3:1f:e5:08:00 SRC=10.229.11.1 DST=10.229.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=48734 DF
 PROTO=TCP SPT=59123 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13177.437509] Dropped: IN=enp0s8 OUT=enp0s3 MAC=08:00:27:77:1f:6c:08:00:27:b3:1f:e5:08:00 SRC=10.229.11.1 DST=10.229.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=54
59 DF PROTO=TCP SPT=50747 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13178.444943] Dropped: IN=enp0s8 OUT=enp0s3 MAC=08:00:27:77:1f:6c:08:00:27:b3:1f:e5:08:00 SRC=10.229.11.1 DST=10.229.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=54
60 DF PROTO=TCP SPT=50747 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
[13179.458351] Dropped: IN=enp0s8 OUT=enp0s3 MAC=08:00:27:77:1f:6c:08:00:27:b3:1f:e5:08:00 SRC=10.229.11.1 DST=10.229.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=54
61 DF PROTO=TCP SPT=50747 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0
```

In addition, we made an http request from Our Windows and 10.229.11.1 responded by sending a packet to port 49781 which was not filtered by our firewall as required.

```
ssign2 git:(part-1) ■ tcpdump -n -l -i enp0s3
 verbose output suppressed, use -v[v]... for full protocol decode
g on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
.558448 ARP, Request who-has 10.229.1.2 tell 10.229.1.1, length 28
.559735 ARP, Reply 10.229.1.2 is-at 08:00:27:3a:c3:8d, length 46
.439307 IP 10.229.11.1.60513 > 10.229.1.2.21: Flags [S], seq 3882042378, win 64240, options [mss 1460,sackOK,TS val 2441435441 ecr

.460979 IP 10.229.11.1.60513 > 10.229.1.2.21: Flags [S], seq 3882042378, win 64240, options [mss 1460,sackOK,TS val 2441436462 ecr

.474737 IP 10.229.11.1.60513 > 10.229.1.2.21: Flags [S], seq 3882042378, win 64240, options [mss 1460,sackOK,TS val 2441437475 ecr

.459710 IP 10.229.11.1.37893 > 10.229.1.2.22: Flags [S], seq 1074423956, win 64240, options [mss 1460,sackOK,TS val 2441440459 ecr

.476729 IP 10.229.11.1.37893 > 10.229.1.2.22: Flags [S], seq 1074423956, win 64240, options [mss 1460,sackOK,TS val 2441441475 ecr

.490909 IP 10.229.11.1.37893 > 10.229.1.2.22: Flags [S], seq 1074423956, win 64240, options [mss 1460,sackOK,TS val 2441442489 ecr

.848719 IP 10.229.10.1 > 10.229.1.2: ICMP echo request, id 598, seq 1, length 64
.866443 IP 10.229.1.2.49781 > 10.229.11.1.80: Flags [SEW], seq 957592708, win 65535, options [mss 1460,nop,wscale 3,nop,nop,sackOK]
.867381 IP 10.229.11.1.80 > 10.229.1.2.49781: Flags [R.], seq 0, ack 957592709, win 0, length 0
.378450 IP 10.229.1.2.49781 > 10.229.11.1.80: Flags [S], seq 957592708, win 65535, options [mss 1460,nop,wscale 3,nop,nop,sackOK],
.379610 IP 10.229.11.1.80 > 10.229.1.2.49781: Flags [R.], seq 0, ack 1, win 0, length 0
.894058 IP 10.229.1.2.49781 > 10.229.11.1.80: Flags [S], seq 957592708, win 65535, options [mss 1460,nop,nop,sackOK], length 0
.895140 IP 10.229.11.1.80 > 10.229.1.2.49781: Flags [R.], seq 0, ack 1, win 0, length 0
```

In contrast, we made an http request from Our Windows to 10.229.11.1 but, this time, without the commands that accepted established connections and there is no response from 10.229.11.1.80 meaning the packet was dropped.

```
■ 333-assign2 git:(part-1) ■ tcpdump -n -l -i enp0s3
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
06:21:20.534021 IP 10.229.11.1.56650 > 10.229.1.2.69: TFTP, length 39, RRQ "" octet tsize 0 blksize 512 timeout 1
06:21:21.518201 ARP, Request who-has 10.229.1.2 tell 10.229.1.1, length 28
06:21:21.518617 ARP, Reply 10.229.1.2 is-at 08:00:27:3a:c3:8d, length 46
06:21:23.541356 IP 10.229.11.1.48600 > 10.229.1.2.69: TFTP, length 39, RRQ "" octet tsize 0 blksize 512 timeout 1
06:21:31.605724 IP 10.229.11.1 > 10.229.1.2: ICMP echo request, id 599, seq 1, length 64
06:21:33.283876 IP 10.229.1.2.49785 > 10.229.11.1.80: Flags [SEW], seq 3071712754, win 65535, options [mss 1460,nop,wscale
06:21:34.297284 IP 10.229.1.2.49785 > 10.229.11.1.80: Flags [SEW], seq 3071712754, win 65535, options [mss 1460,nop,wscale
06:21:35.615343 IP 10.229.11.1.54819 > 10.229.1.2.21: Flags [S], seq 2197693932, win 64240, options [mss 1460,sackOK,TS val
th 0
06:21:36.313257 IP 10.229.1.2.49785 > 10.229.11.1.80: Flags [S], seq 3071712754, win 65535, options [mss 1460,nop,nop,sackO
06:21:36.618004 IP 10.229.11.1.54819 > 10.229.1.2.21: Flags [S], seq 2197693932, win 64240, options [mss 1460,sackOK,TS val
th 0
06:21:37.631871 IP 10.229.11.1.54819 > 10.229.1.2.21: Flags [S], seq 2197693932, win 64240, options [mss 1460,sackOK,TS val
th 0
06:21:37.814161 ARP, Request who-has 10.229.1.1 (08:00:27:00:f4:95) tell 10.229.1.2, length 46
06:21:37.814218 ARP, Reply 10.229.1.1 is-at 08:00:27:00:f4:95, length 28
^C
```

“Prohibit your Windows host from accessing any services provided by the hosts on 10.229.10.0/24. Note that we only want to block Windows from connecting to services in 10.229.10.0/24 but the reverse (connections from 10.229.10.0/24 to services on your Windows host should still be possible (if not subjected to any other restriction).”

After making an ftp request to 10.229.10.1 in Our Windows, the following log shows that we are dropping the request.

```
■ 333-assign2 git:(part-1) ■ dmesg | grep DST=10.229.10.1
[ 3517.222567] Dropped: IN=enp0s3 OUT=enp0s8 MAC=08:00:27:00:f4:95:08:00:27:3a:c3:8d:08:00 SRC=10.229.1.2 DST=10.229.10.1 I
9064 DF PROTO=TCP SPT=49726 DPT=21 WINDOW=8192 RES=0x00 CWR ECE SYN URGP=0
[ 3520.231637] Dropped: IN=enp0s3 OUT=enp0s8 MAC=08:00:27:00:f4:95:08:00:27:3a:c3:8d:08:00 SRC=10.229.1.2 DST=10.229.10.1 I
9065 DF PROTO=TCP SPT=49726 DPT=21 WINDOW=8192 RES=0x00 CWR ECE SYN URGP=0
[ 3526.247405] Dropped: IN=enp0s3 OUT=enp0s8 MAC=08:00:27:00:f4:95:08:00:27:3a:c3:8d:08:00 SRC=10.229.1.2 DST=10.229.10.1 I
9066 DF PROTO=TCP SPT=49726 DPT=21 WINDOW=8192 RES=0x00 SYN URGP=0
```

On the other hand, Our Windows is still getting ping and ssh requests from 10.229.10.1.

```
■ 333-assign2 git:(part-1) ■ tcpdump -n -l -i enp0s3
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
04:46:12.208998 IP 10.229.11.1.44829 > 10.229.1.2.22: Flags [S], seq 2018764632, win 64240, options [mss 1460,s
th 0
04:46:13.222233 IP 10.229.11.1.44829 > 10.229.1.2.22: Flags [S], seq 2018764632, win 64240, options [mss 1460,s
th 0
04:46:19.406120 IP 10.229.10.1 > 10.229.1.2: ICMP echo request, id 280, seq 1, length 64
04:46:20.426354 IP 10.229.10.1 > 10.229.1.2: ICMP echo request, id 280, seq 2, length 64
04:46:29.292054 IP 10.229.11.1.41286 > 10.229.1.2.69: TFTP, length 60, WRQ "other_linux_tftp.txt" octet tsize 3
04:46:30.482069 IP 10.229.10.1.35803 > 10.229.1.2.22: Flags [S], seq 2843022663, win 64240, options [mss 1460,s
th 0
04:46:31.497925 IP 10.229.10.1.35803 > 10.229.1.2.22: Flags [S], seq 2843022663, win 64240, options [mss 1460,s
th 0
04:46:32.296172 IP 10.229.11.1.39708 > 10.229.1.2.69: TFTP, length 60, WRQ "other_linux_tftp.txt" octet tsize 3
04:46:32.512215 IP 10.229.10.1.35803 > 10.229.1.2.22: Flags [S], seq 2843022663, win 64240, options [mss 1460,s
th 0
```

“Add rules to log any violations of the above restriction.”

We are logging everything that is being dropped.

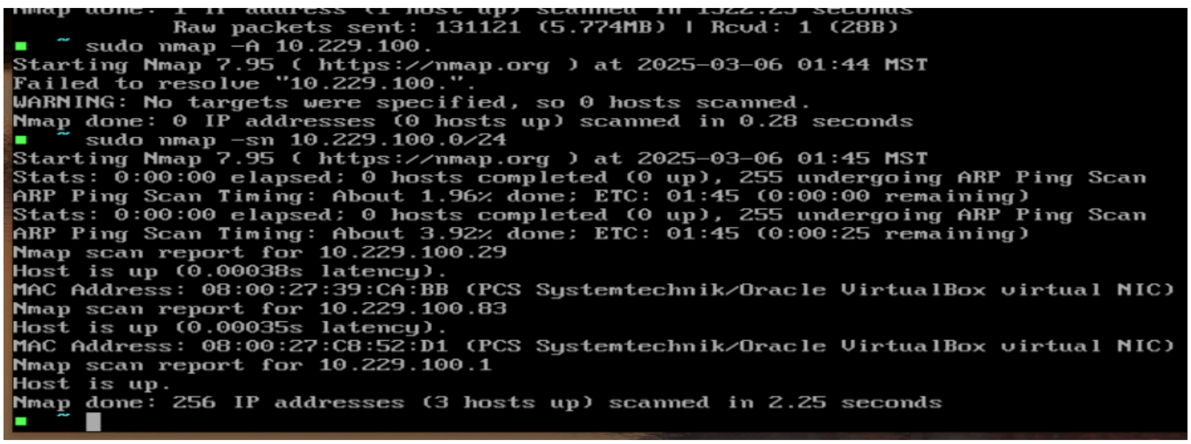
Since outgoing requests are logged after firewall rules are already applied, we used the requests from Other Linux 1 and Other Linux 2 to Our Windows to see if our logs are accurate. For example, if we see that in the logs we are dropping http and we do not see a packet with destination port 80 when running tcpdump -n -l -i enp0s3, it means our log for that rule is working.

PART 2

STEP #1

1. Determine and report, by using tools such as nmap, etc, what are the IP addresses of the victim hosts connected to the backbone

Command ran: sudo nmap -sn 10.229.100.0/24
Nmap (–sn): Detects live hosts on the network.
Found two live hosts of victims:
10.229.100.83 and **10.229.100.29**



2. Determine and report, using the same/similar tools, what are the services running on each of the victim hosts connected to the backbone.

First used nmap -p –top-ports 1000 10.229.100.29
Used nmap -p- 10.229.100.29

used nmap -p –top-ports 1000 10.229.100.83
Used nmap -p- 10.229.100.83

Then used nmap -p- 10.229.100.29
Nmap -p- 10.229.100.83

But everything is being filtered by the firewalls so nmap cannot be used to see which services are being used.

Then, we used ettercap with the command
sudo ettercap -T -i enp0s8 -M arp:remote /10.229.100.83// /10.229.100.29// -w captured_traffic.pcap to listen to the traffic between two victims
The captured traffic is in the file [captured_traffic.pcap]

VICTIM HOSTS SERVICE REPORT

We could only find the communication that was happening between 2 ports. Following was the communication between the 2 victims.

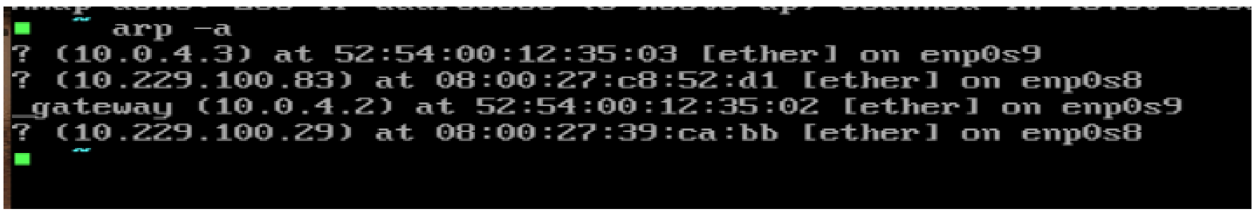
Port	Protocol	Service	Command	Evidence
21	TCP	FTP	tcpdump -r captured_traffic.pcap port 21	`FTP: USER anonymous`, `FTP: 150 Opening BINARY mode data connection`, `FTP: 226 Transfer complete`
22	TCP	SSH	tcpdump -r captured_traffic.pcap port 22	`IP 10.229.100.29.60462 > 10.229.100.83.ssh: Flags [P.]`, `IP 10.229.100.83.ssh > 10.229.100.29.60462: Flags [P.]`
80		HTTP	tcpdump -r captured_traffic.pcap port 80	`22:15:01.058940 IP 10.229.100.29.http > 10.229.100.83.59336: Flags [S.], seq 2402435255, ack 1308477828, win 65160, options [mss 1460,sackOK,TS val 2552284571 ecr 3880024405,nop,wscale 7], length 0`

Additionally, there was communication between 2 hosts in higher ports suggesting Passive FTP. We could not find any more communication between 2 hosts. Since all the port scans were being filtered, we could not find the specific ports that were open in one of them.

Ettercap was used to capture traffic between the two victims, so we can only detect services that were actively communicating between them. If a service was running but was never used between the two hosts during the capture, it would not appear in the captured file.
A port scan (e.g., using nmap) could detect open ports even if they are not actively being used, but since everything was being filtered, we could only confirm services that had actual network activity between the two hosts during the capture.In summary, if a service was not actively used between the two hosts while capturing traffic, it would remain undetected.

STEP #2

1. Determine the victim hosts



Victims are:
10.229.100.83 at mac address 08:00:27:c8:52:d1
10.229.100.29 at mac address 08:00:27:39:ca:bb

The captured traffic is in the file captured_traffic.pcap

the ARP activity for the period just before and just after you performed ARP poisoning:

Arp.txt was extracted using the command tcpdump -r captured_traffic.pcap arp >arp.txt

Analysing the arp.txt we find,
Before poisoning:
22:14:44.220484 ARP, Request who-has 10.229.100.83 tell BaseMachine, length 28
22:14:44.221134 ARP, Reply 10.229.100.83 is-at 08:00:27:c8:52:d1 (oui Unknown), length 46
22:14:44.230940 ARP, Request who-has 10.229.100.29 tell BaseMachine, length 28

Only one reply for the Who-has request

But after poisoning, we see 2 different MAC-addresses claiming to have the IP:
22:14:44.231505 ARP, Reply 10.229.100.29 is-at 08:00:27:39:ca:bb (oui Unknown), length 46
22:14:45.243163 ARP, Reply 10.229.100.29 is-at 08:00:27:87:e8:2f (oui Unknown), length 28

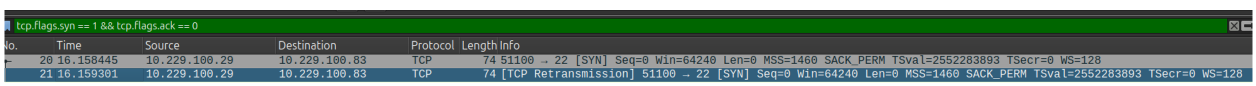
22:14:45.243177 ARP, Reply 10.229.100.83 is-at 08:00:27:87:e8:2f (oui Unknown), length 28
22:14:46.253450 ARP, Reply 10.229.100.29 is-at 08:00:27:87:e8:2f (oui Unknown), length 28
22:14:46.253496 ARP, Reply 10.229.100.83 is-at 08:00:27:87:e8:2f (oui Unknown), length 28

This shows duplicate IPs. This means that the ARP poisoning attack was successful—it caused a duplicate IP conflict, redirecting traffic to the attacker's MAC.
Longer arp.txt is attached with the assignment.

2. Determine what connections are initiated (including which host is the client and which one is the server for the connections), determine what is the service(s) to which the connections are established, and,

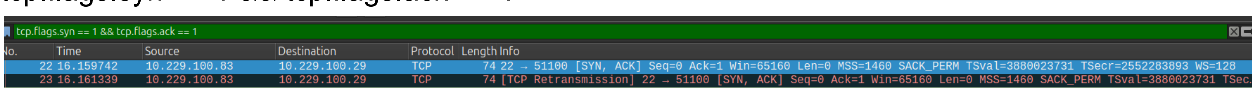
From the answer in part 1, we found out there were ssh, ftp and http being initiated
We got the captured_traffic.pcap file from listening to the 2 victims.
Imported the file to our machine to analyze the file in wireshark

SSH:
Used filter: tcp.flags.syn == 1 && tcp.flags.ack == 0 && tcp.dstport == 22
This was used to find the client, the one sending the syn to server
Client: 10.229.100.29 because it is the source for SYN and the destination is 10.229.100.83
So **Server** is 10.229.100.83


tcp.flags.syn == 1 && tcp.flags.ack == 0

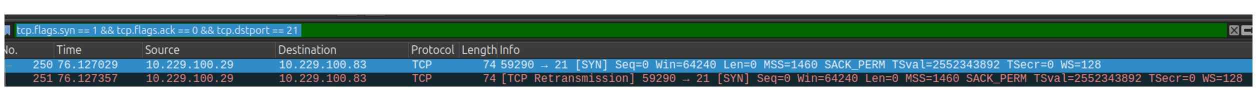
No.	Time	Source	Destination	Protocol	Length	Info
20	16.158445	10.229.100.29	10.229.100.83	TCP	74	51100 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2552283893 TSecr=0 WS=128
21	16.159301	10.229.100.29	10.229.100.83	TCP	74	[TCP Retransmission] 51100 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2552283893 TSecr=0 WS=128

10.229.100.83 being server is further verified by using the filter
tcp.flags.syn == 1 && tcp.flags.ack == 1


tcp.flags.syn == 1 && tcp.flags.ack == 1

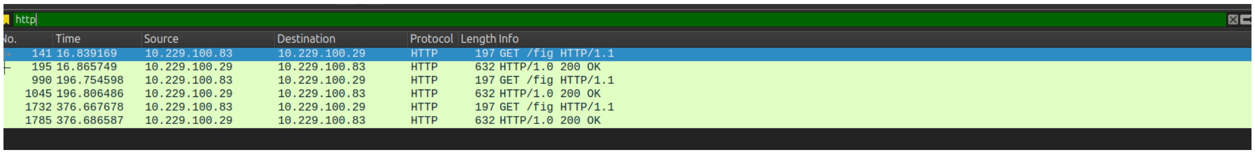
No.	Time	Source	Destination	Protocol	Length	Info
22	16.159742	10.229.100.83	10.229.100.29	TCP	74	22 → 51100 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3880023731 TSecr=2552283893 WS=128
23	16.161339	10.229.100.83	10.229.100.29	TCP	74	[TCP Retransmission] 22 → 51100 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3880023731 TSec

FTP:
Used filter: tcp.flags.syn == 1 && tcp.flags.ack == 0 && tcp.dstport == 21


tcp.flags.syn == 1 && tcp.flags.ack == 0 && tcp.dstport == 21

No.	Time	Source	Destination	Protocol	Length	Info
250	76.127029	10.229.100.29	10.229.100.83	TCP	74	59290 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2552343892 TSecr=0 WS=128
251	76.127357	10.229.100.29	10.229.100.83	TCP	74	[TCP Retransmission] 59290 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2552343892 TSecr=0 WS=128

We can see that SYN being sent from 10.229.100.29 to 10.229.100.83 to port 21(FTP)
So **client** is 10.229.100.29
Server is 10.229.100.83
HTTP:



http

No.	Time	Source	Destination	Protocol	Length	Info
141	16.839169	10.229.100.83	10.229.100.29	HTTP	197	GET /fig HTTP/1.1
195	16.865749	10.229.100.29	10.229.100.83	HTTP	632	HTTP/1.0 200 OK
990	196.754598	10.229.100.83	10.229.100.29	HTTP	197	GET /fig HTTP/1.1
1045	196.806486	10.229.100.29	10.229.100.83	HTTP	632	HTTP/1.0 200 OK
1732	376.667678	10.229.100.83	10.229.100.29	HTTP	197	GET /fig HTTP/1.1
1785	376.686587	10.229.100.29	10.229.100.83	HTTP	632	HTTP/1.0 200 OK

Filter: http
10.229.100.83 sends a get request and 10.229.100.29 responds with 200 ok
So the **client is 10.229.100.83**
The server is 10.229.100.29

Files found:
Tools used: wireshark to construct the file
From the http connection, we got the fig file.

From the ftp connection, we got the bin file bin
To analyze the bin file, we extracted the human readable strings in the file strings_file_bin.txt

Summary of `bin` Analysis

File bin in the terminal returned:

```
bin: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked,
BuildID[sha1]=dbe11c3cf0cafa4af2ade5bc39d5dfc981e0489e, not stripped
```

The extracted file, `file.bin`, is a 64-bit ELF binary for Linux, dynamically linked to the GNU C Library (glibc). It contains references to authentication mechanisms, password management, and system process control. Its key findings include:

- References to **password authentication (`pam_sm_authenticate`, `wrong password, return PAM_AUTH_ERR`)**
- **Interactions with `/etc/passwd` and PAM security files**, indicating possible credential management
- **Use of system calls like `fork`, `execve`, and `waitpid`**, suggesting process control capabilities

No files could be extracted from the SSH connection as the session was encrypted. Without access to private keys, it is not possible to decrypt the transferred data.

25	16.162419	10.229.100.29	10.229.100.83	SSHv2	87	Client: Protocol (SSH-2.0-OpenSSH_9.9)
30	16.172521	10.229.100.83	10.229.100.29	SSHv2	87	Server: Protocol (SSH-2.0-OpenSSH_9.9)
34	16.174990	10.229.100.29	10.229.100.83	SSHv2	186	Client: Key Exchange Init
37	16.177467	10.229.100.83	10.229.100.29	SSHv2	1234	Server: Key Exchange Init
44	16.224899	10.229.100.29	10.229.100.83	SSHv2	1274	Client: Diffie-Hellman Key Exchange Init
46	16.232534	10.229.100.83	10.229.100.29	SSHv2	1514	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=200)
47	16.232535	10.229.100.83	10.229.100.29	SSHv2	150	Server: Encrypted packet (len=84)
52	16.249176	10.229.100.29	10.229.100.83	SSHv2	150	Client: New Keys, Encrypted packet (len=68)
56	16.296850	10.229.100.29	10.229.100.83	SSHv2	110	Client: Encrypted packet (len=44)
59	16.302002	10.229.100.83	10.229.100.29	SSHv2	110	Server: Encrypted packet (len=44)
62	16.304813	10.229.100.29	10.229.100.83	SSHv2	126	Client: Encrypted packet (len=60)
64	16.310835	10.229.100.83	10.229.100.29	SSHv2	330	Server: Encrypted packet (len=264)
66	16.313934	10.229.100.29	10.229.100.83	SSHv2	566	Client: Encrypted packet (len=500)
68	16.319061	10.229.100.83	10.229.100.29	SSHv2	526	Server: Encrypted packet (len=460)
70	16.330339	10.229.100.29	10.229.100.83	SSHv2	1054	Client: Encrypted packet (len=988)
72	16.343748	10.229.100.83	10.229.100.29	SSHv2	94	Server: Encrypted packet (len=28)
74	16.346383	10.229.100.29	10.229.100.83	SSHv2	178	Client: Encrypted packet (len=112)
76	16.367048	10.229.100.83	10.229.100.29	SSHv2	818	Server: Encrypted packet (len=752)
80	16.412117	10.229.100.83	10.229.100.29	SSHv2	138	Server: Encrypted packet (len=72)
83	16.419358	10.229.100.29	10.229.100.83	SSHv2	118	Client: Encrypted packet (len=52)
86	16.427494	10.229.100.83	10.229.100.29	SSHv2	138	Server: Encrypted packet (len=72)
88	16.430340	10.229.100.29	10.229.100.83	SSHv2	102	Client: Encrypted packet (len=36)
90	16.433622	10.229.100.83	10.229.100.29	SSHv2	246	Server: Encrypted packet (len=180)
92	16.437088	10.229.100.29	10.229.100.83	SSHv2	118	Client: Encrypted packet (len=52)
94	16.440012	10.229.100.83	10.229.100.29	SSHv2	118	Server: Encrypted packet (len=52)
96	16.441515	10.229.100.29	10.229.100.83	SSHv2	118	Client: Encrypted packet (len=52)
98	16.443938	10.229.100.83	10.229.100.29	SSHv2	118	Server: Encrypted packet (len=52)
100	16.445962	10.229.100.29	10.229.100.83	SSHv2	126	Client: Encrypted packet (len=60)
102	16.448033	10.229.100.83	10.229.100.29	SSHv2	110	Server: Encrypted packet (len=44)
104	16.449936	10.229.100.29	10.229.100.83	SSHv2	558	Client: Encrypted packet (len=492)
106	16.454262	10.229.100.83	10.229.100.29	SSHv2	118	Server: Encrypted packet (len=52)
108	16.460592	10.229.100.29	10.229.100.83	SSHv2	110	Client: Encrypted packet (len=44)
110	16.463407	10.229.100.83	10.229.100.29	SSHv2	102	Server: Encrypted packet (len=36)
112	16.467099	10.229.100.29	10.229.100.83	SSHv2	102	Client: Encrypted packet (len=36)
114	16.470134	10.229.100.83	10.229.100.29	SSHv2	102	Server: Encrypted packet (len=36)
116	16.473574	10.229.100.29	10.229.100.83	SSHv2	102	Client: Encrypted packet (len=36)
118	16.477811	10.229.100.83	10.229.100.29	SSHv2	110	Server: Encrypted packet (len=44)
119	16.477813	10.229.100.83	10.229.100.29	SSHv2	138	Server: Encrypted packet (len=72)
123	16.480697	10.229.100.29	10.229.100.83	SSHv2	102	Client: Encrypted packet (len=36)
124	16.480698	10.229.100.29	10.229.100.83	SSHv2	118	Client: Encrypted packet (len=52)

This is the traffic we got from the communication between the client and server.

Server is 10.229.100.29 and client is 10.229.100.83. We can see the initial connection being established and the key exchanges. Then we see encrypted packets of len 44-64 which suggests short commands and brief responses. These likely represent keyboard input and terminal feedback during an interactive session.

After that, the server sends a bigger packet of size 264 and then the client sends an even bigger packet of size 500. This pattern suggests the client might be requesting a file and then sending back an edited version. The same pattern happens again with the server sending a packet of size 460 and the client responding with a packet of size 988.

Then we see the client sending smaller size packets suggesting some commands being issued. The server responds with larger packets which might indicate file transfers using scp or the server sending file contents requested by the client.

The overall pattern indicates an interactive session involving not just commands and responses but also bidirectional file operations where both systems are exchanging data of varying sizes.

Time Interval between packets:
The connection is sending the same data again and again in certain intervals.

For FTP: time between same data is 76.137043 to 196.078305
So interval is 196.078305 - 76.137043 =119.941 ≈ 120 sec (i.e 2 min)

For HTTP: time interval between 2 same get request: 16.839160 to 196.754598
Interval is 196.754598 - 16.839160 =179.915 ≈ 180 sec (i.e 3 minutes)

For SSH: time interval 2 same request is 16.162419 to 76.140382
Interval is 76.140382 - 16.162419 =59.978 ≈ 60 sec (ie 1 minute)