CYBR371

LAB 2

Olivia Fletcher
300534281
fletcholiv

PART 1 - Sniffing, Spoofing and ARP Poisoning

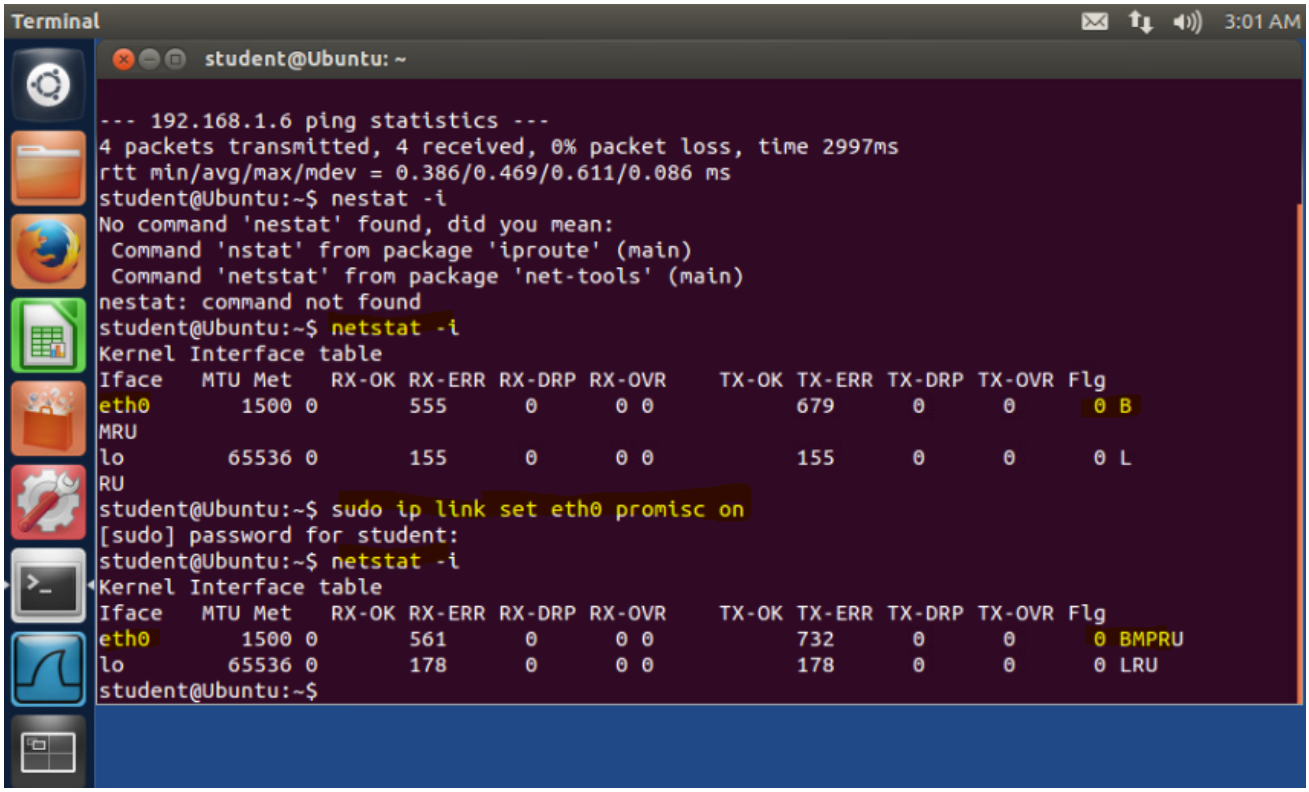
Investigating ARP poisoning and Capturing Network Traffic

Investigating ARP Poisoning & Capturing Network Traffic Labs - Completed

Question 1 [1 mark] - Explain the utility of the following settings/commands used in these labs. Provide clear, concise answers with one example scenario highlighting their significance.

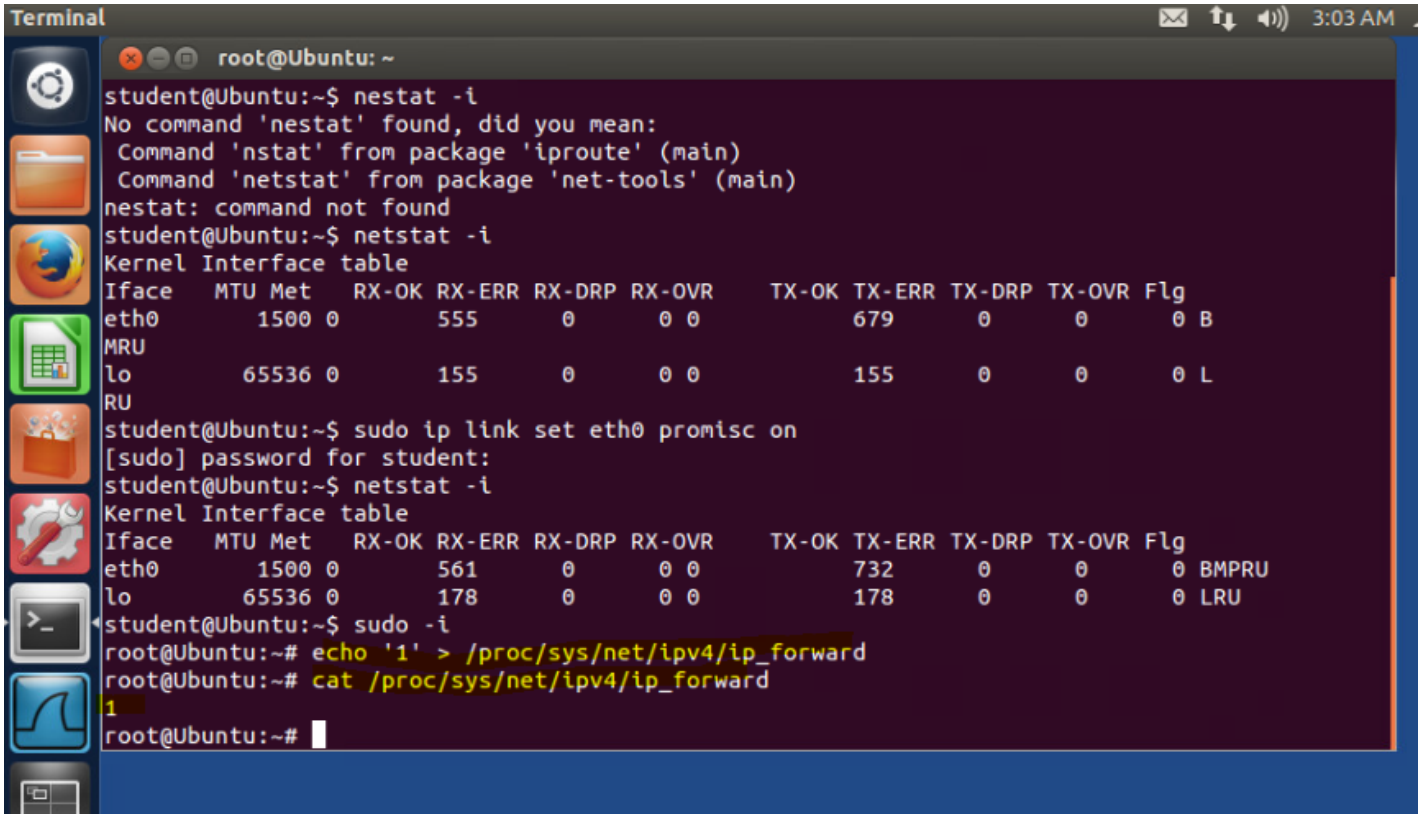
- a. Promiscuous mode
    - A type of networking that works in an operational mode which is the ability to be able to access all network data packets and can also be viewed by all network adapters in this same mode.
    - We used Promiscuous mode in wireshark in the ARP Spoofing lab, which shows all network traffic through the current device.
    - In the lab using the;
    - `student@Ubuntu:~$ netstat -i`
    - command we checked protocols in use and changed eth0 promiscuous mode on using
    - `student@Ubuntu:~$ sudo ip link set eth0 promisc on`

`Screenshot next page showing turing promisc mode on`



- b. IP forward field
  - IP forwarding is the process which determines which path a packet can be sent, this process is used to keep unwanted traffic off of networks.
  - In the ARP Spoofing lab we used IP forwarding so we could deceive the gateway device.
  - `student@Ubuntu:~$ sudo nano /proc/sys/net/ipv4/ip_forward`
  - Logging into root we had to then change the ip_forwards field to contain 1 instead of 0 to allow access, we did this using the following lines;
  - `root@Ubuntu:~# echo '1' > /proc/sys/net/ipv4/ip_forward`
  - `root@Ubuntu:!# cat /proc/sys/net/ipv4/ip_forward`

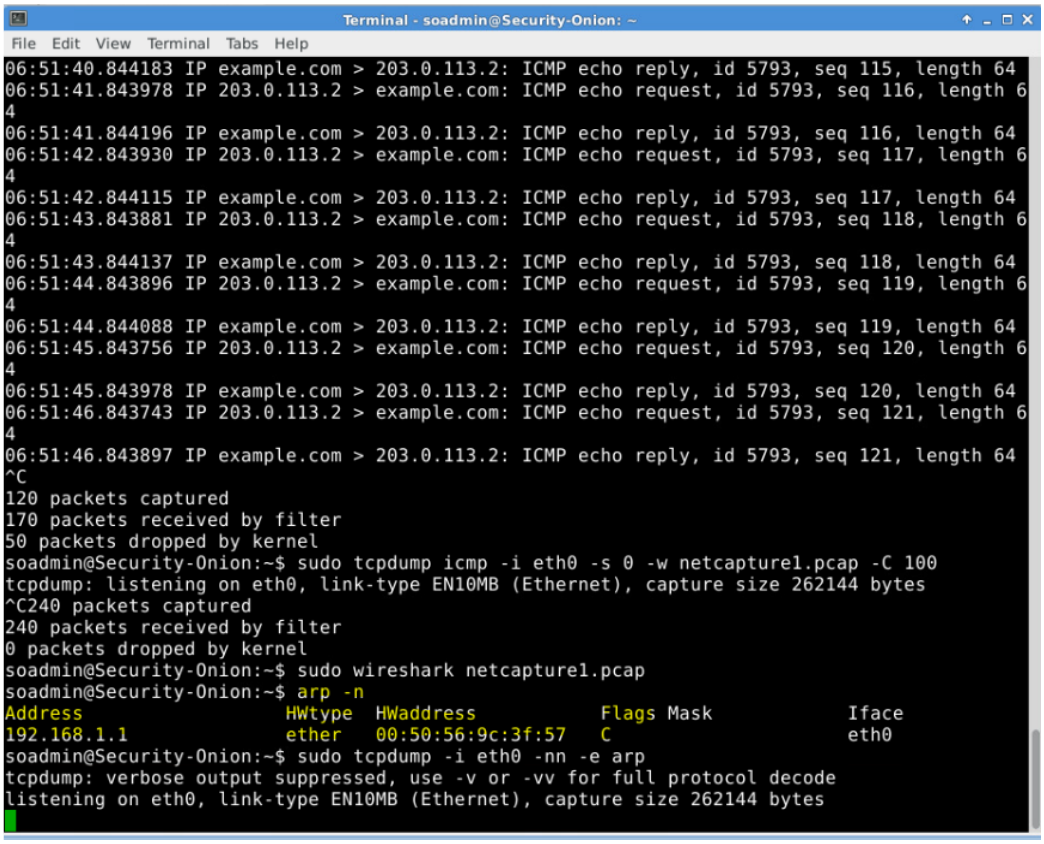
Screenshot next page showing the ip_forward field

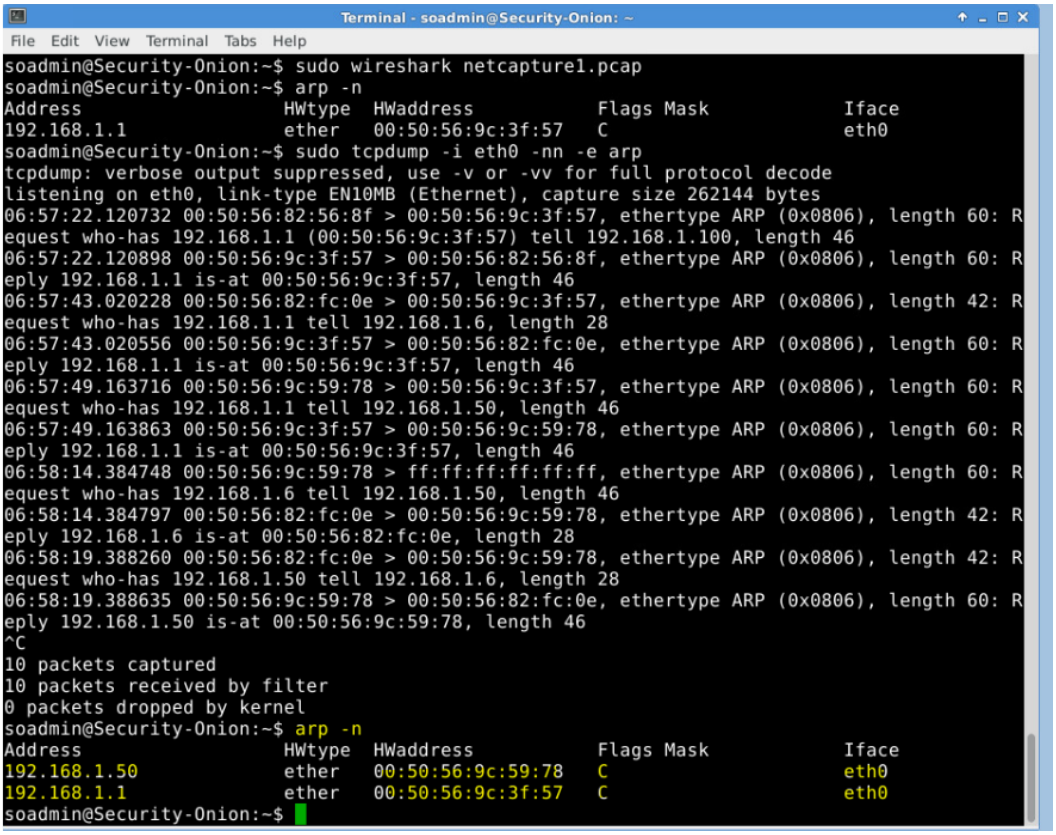


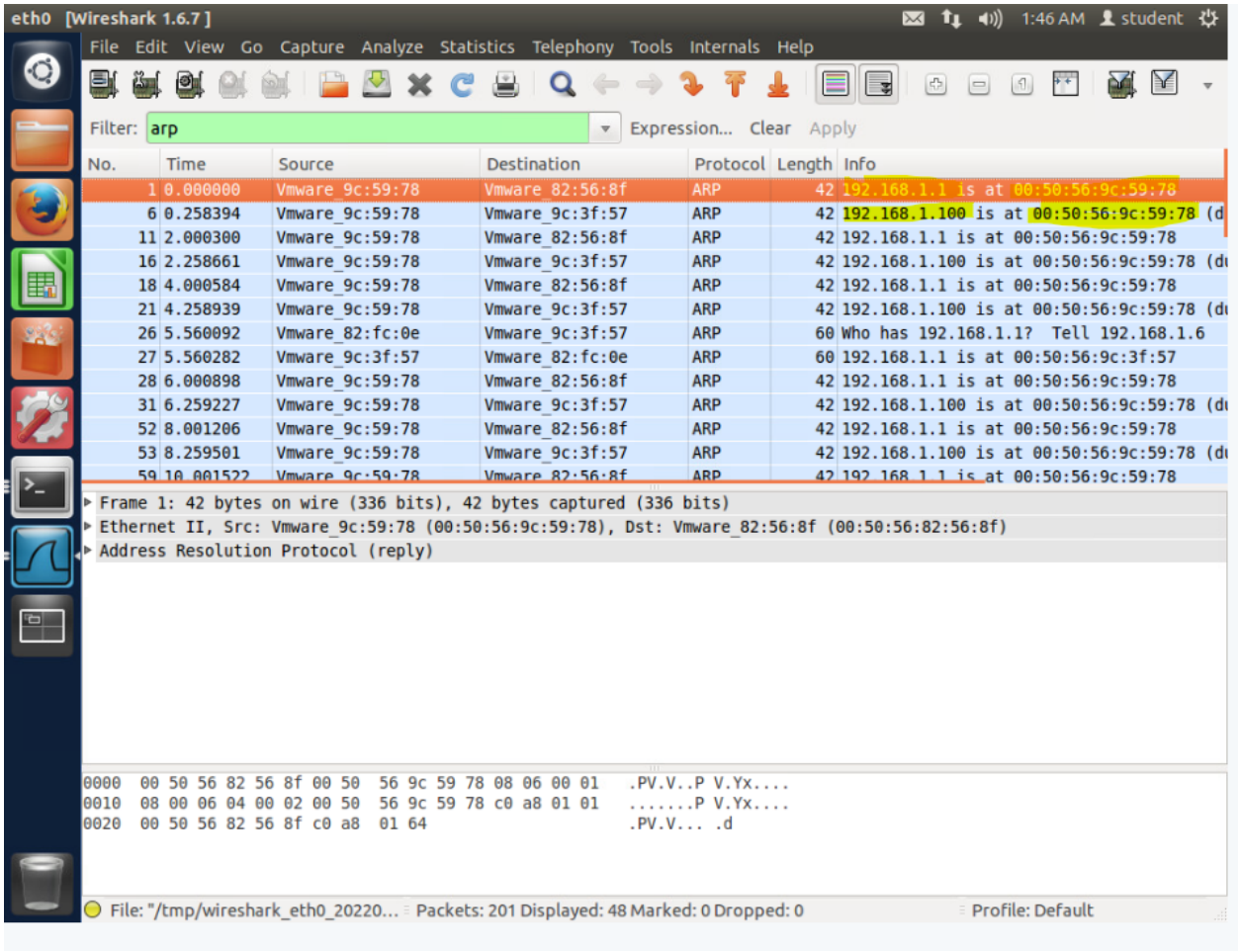
c. arpwatch

- Arpwatch is a type of monitoring which checks all the links by periodically sending ARP packets to the destination/s devices.
- We used this in the lab as we attempted to spoof using; and viewed the ARP data in wireshark
- `[student@Ubuntu:~]$ sudo arpspoof -i eth0 -t 192.168.1.100 192.168.1.1`

Below are a few screenshots showing from when we used arp to monitor the network and when we used wireshark to show that we have 2 ips coming from the same MAC address

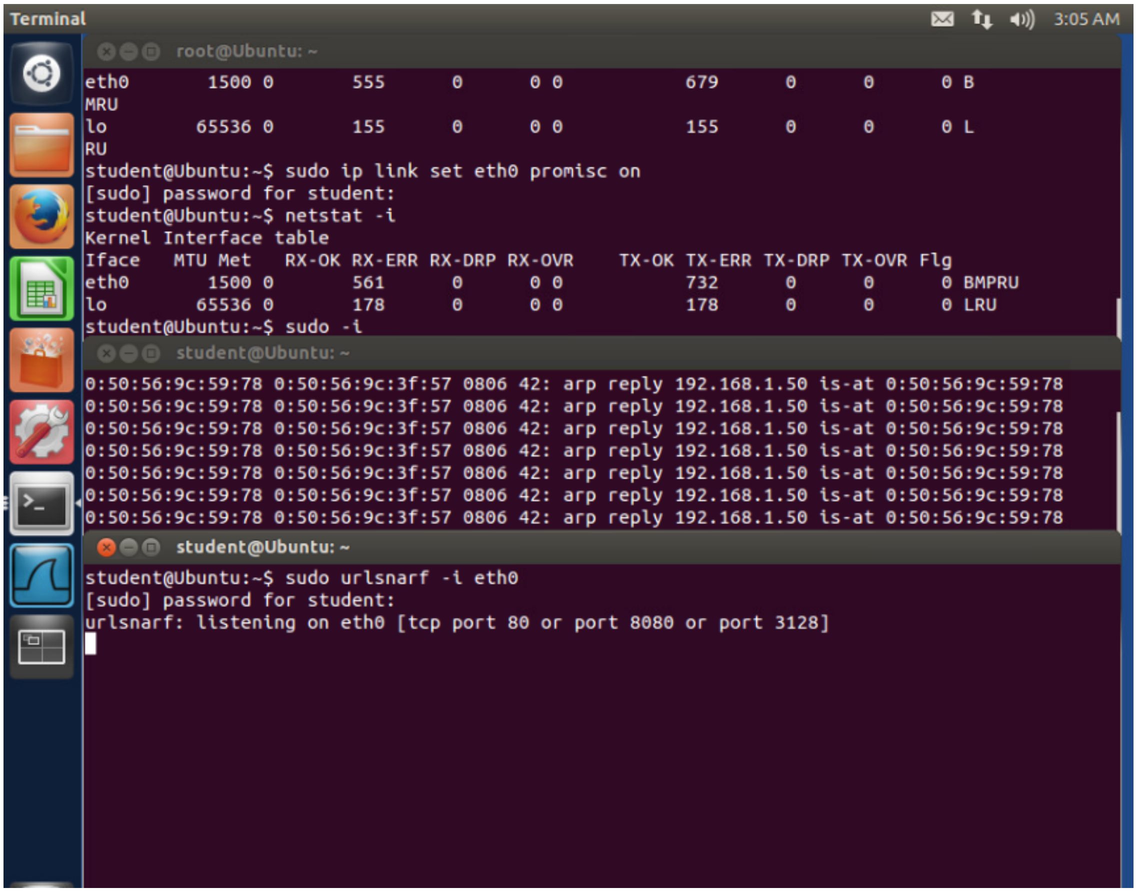

Terminal - soadmin@Security-Onion: ~
File Edit View Terminal Tabs Help
06:51:40.844183 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 115, length 64
06:51:41.843978 IP 203.0.113.2 > example.com: ICMP echo request, id 5793, seq 116, length 6
4
06:51:41.844196 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 116, length 64
06:51:42.843930 IP 203.0.113.2 > example.com: ICMP echo request, id 5793, seq 117, length 6
4
06:51:42.844115 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 117, length 64
06:51:43.843881 IP 203.0.113.2 > example.com: ICMP echo request, id 5793, seq 118, length 6
4
06:51:43.844137 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 118, length 64
06:51:44.843896 IP 203.0.113.2 > example.com: ICMP echo request, id 5793, seq 119, length 6
4
06:51:44.844088 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 119, length 64
06:51:45.843756 IP 203.0.113.2 > example.com: ICMP echo request, id 5793, seq 120, length 6
4
06:51:45.843978 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 120, length 64
06:51:46.843743 IP 203.0.113.2 > example.com: ICMP echo request, id 5793, seq 121, length 6
4
06:51:46.843897 IP example.com > 203.0.113.2: ICMP echo reply, id 5793, seq 121, length 64
^C
120 packets captured
170 packets received by filter
50 packets dropped by kernel
soadmin@Security-Onion:~$ sudo tcpdump icmp -i eth0 -s 0 -w netcapture1.pcap -C 100
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C240 packets captured
240 packets received by filter
0 packets dropped by kernel
soadmin@Security-Onion:~$ sudo wireshark netcapture1.pcap
soadmin@Security-Onion:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
192.168.1.1 ether 00:50:56:9c:3f:57 C eth0
soadmin@Security-Onion:~$ sudo tcpdump -i eth0 -nn -e arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

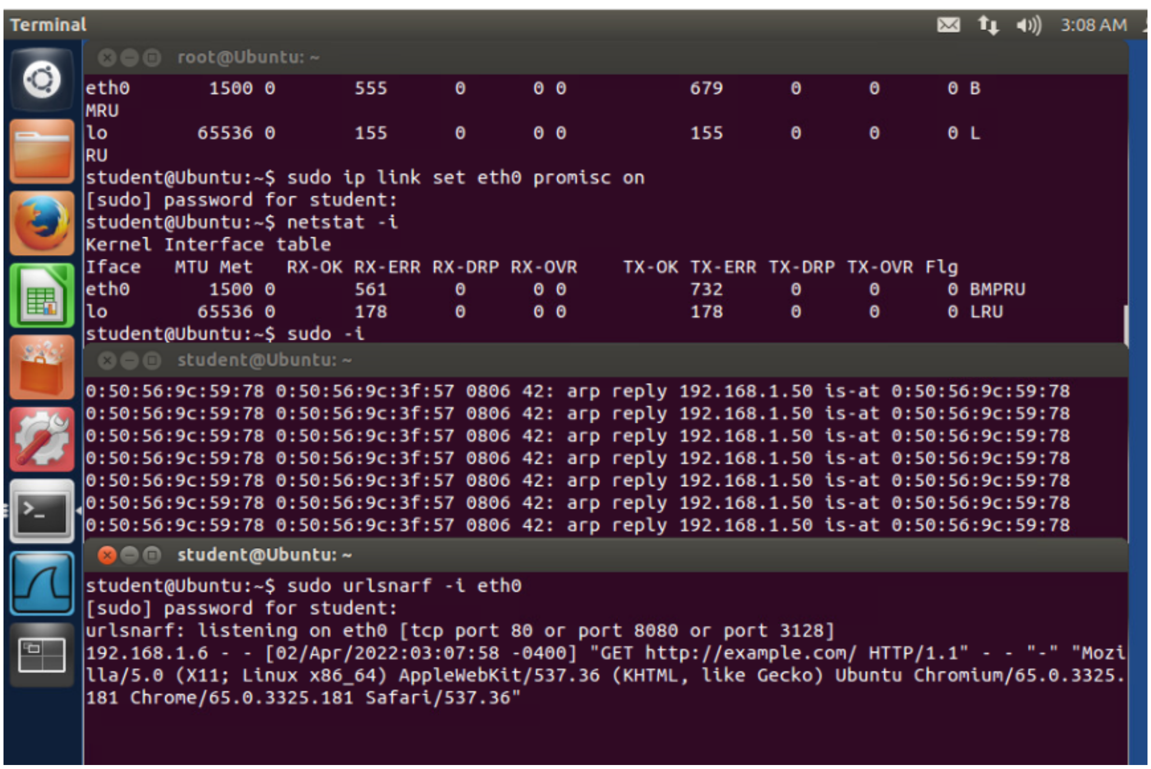

Terminal - soadmin@Security-Onion: ~
File Edit View Terminal Tabs Help
soadmin@Security-Onion:~$ sudo wireshark netcapture1.pcap
soadmin@Security-Onion:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
192.168.1.1 ether 00:50:56:9c:3f:57 C eth0
soadmin@Security-Onion:~$ sudo tcpdump -i eth0 -nn -e arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
06:57:22.120732 00:50:56:82:56:8f > 00:50:56:9c:3f:57, ethertype ARP (0x0806), length 60: R
equest who-has 192.168.1.1 (00:50:56:9c:3f:57) tell 192.168.1.100, length 46
06:57:22.120898 00:50:56:9c:3f:57 > 00:50:56:82:56:8f, ethertype ARP (0x0806), length 60: R
eply 192.168.1.1 is-at 00:50:56:9c:3f:57, length 46
06:57:43.020228 00:50:56:82:fc:0e > 00:50:56:9c:3f:57, ethertype ARP (0x0806), length 42: R
equest who-has 192.168.1.1 tell 192.168.1.6, length 28
06:57:43.020556 00:50:56:9c:3f:57 > 00:50:56:82:fc:0e, ethertype ARP (0x0806), length 60: R
eply 192.168.1.1 is-at 00:50:56:9c:3f:57, length 46
06:57:49.163716 00:50:56:9c:59:78 > 00:50:56:9c:3f:57, ethertype ARP (0x0806), length 60: R
equest who-has 192.168.1.1 tell 192.168.1.50, length 46
06:57:49.163863 00:50:56:9c:3f:57 > 00:50:56:9c:59:78, ethertype ARP (0x0806), length 60: R
eply 192.168.1.1 is-at 00:50:56:9c:3f:57, length 46
06:58:14.384748 00:50:56:9c:59:78 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 60: R
equest who-has 192.168.1.6 tell 192.168.1.50, length 46
06:58:14.384797 00:50:56:82:fc:0e > 00:50:56:9c:59:78, ethertype ARP (0x0806), length 42: R
eply 192.168.1.6 is-at 00:50:56:82:fc:0e, length 28
06:58:19.388260 00:50:56:82:fc:0e > 00:50:56:9c:59:78, ethertype ARP (0x0806), length 42: R
equest who-has 192.168.1.50 tell 192.168.1.6, length 28
06:58:19.388635 00:50:56:9c:59:78 > 00:50:56:82:fc:0e, ethertype ARP (0x0806), length 60: R
eply 192.168.1.50 is-at 00:50:56:9c:59:78, length 46
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
soadmin@Security-Onion:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
192.168.1.50 ether 00:50:56:9c:59:78 C eth0
192.168.1.1 ether 00:50:56:9c:3f:57 C eth0
soadmin@Security-Onion:~$



d. urlsnarf

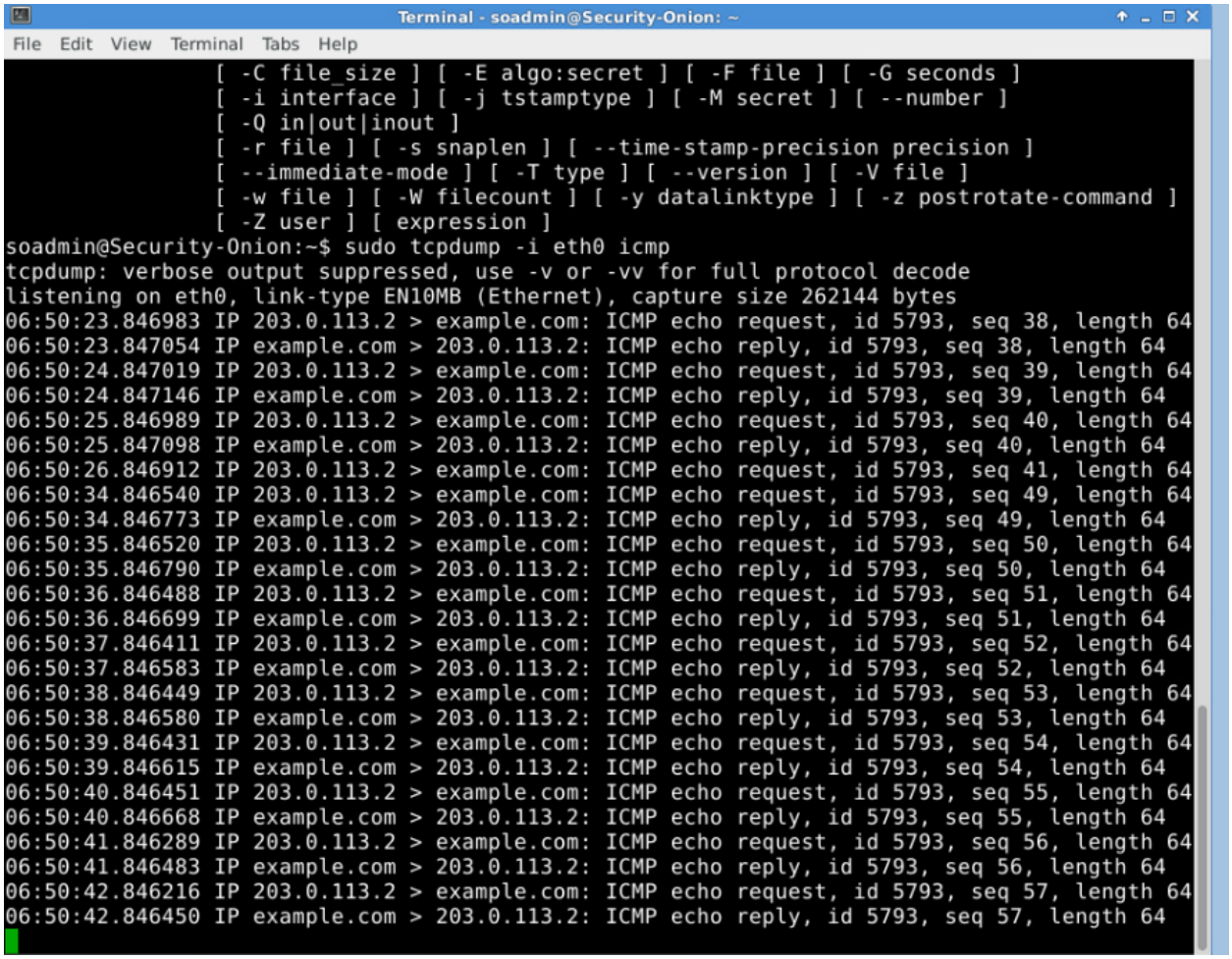
- This is used to sniff HTTP requests from live and offline (.pcap files) network traffic. The urlsnarf checks which websites were visited by which user on an accessed network.
- In the lab we used a few different terminals to test urlsnarf, first we logged into root on ubuntu and run this command to spoof the hosts MAC on the switch
- `root@Ubuntu:!# arpspoof –i eth0 –t 192.168.1.50 192.168.1.1`
- Then in a new terminal without root priv we spoofed the MAC on the host and left it running in the background
- `student@Ubuntu:~$ sudo arpspoof –i eth0 –t 192.168.1.1 192.168.1.50`
- Then we starting to sniff the HTTP requests using urlsnarf on a new terminal
- `student@Ubuntu:~$ sudo urlsnarf –i eth0`
- Switching to the secOnion we opened a new browser and loaded a webpage
- Going back to the terminal sniffing using urlsnarf we get the data sent back to us

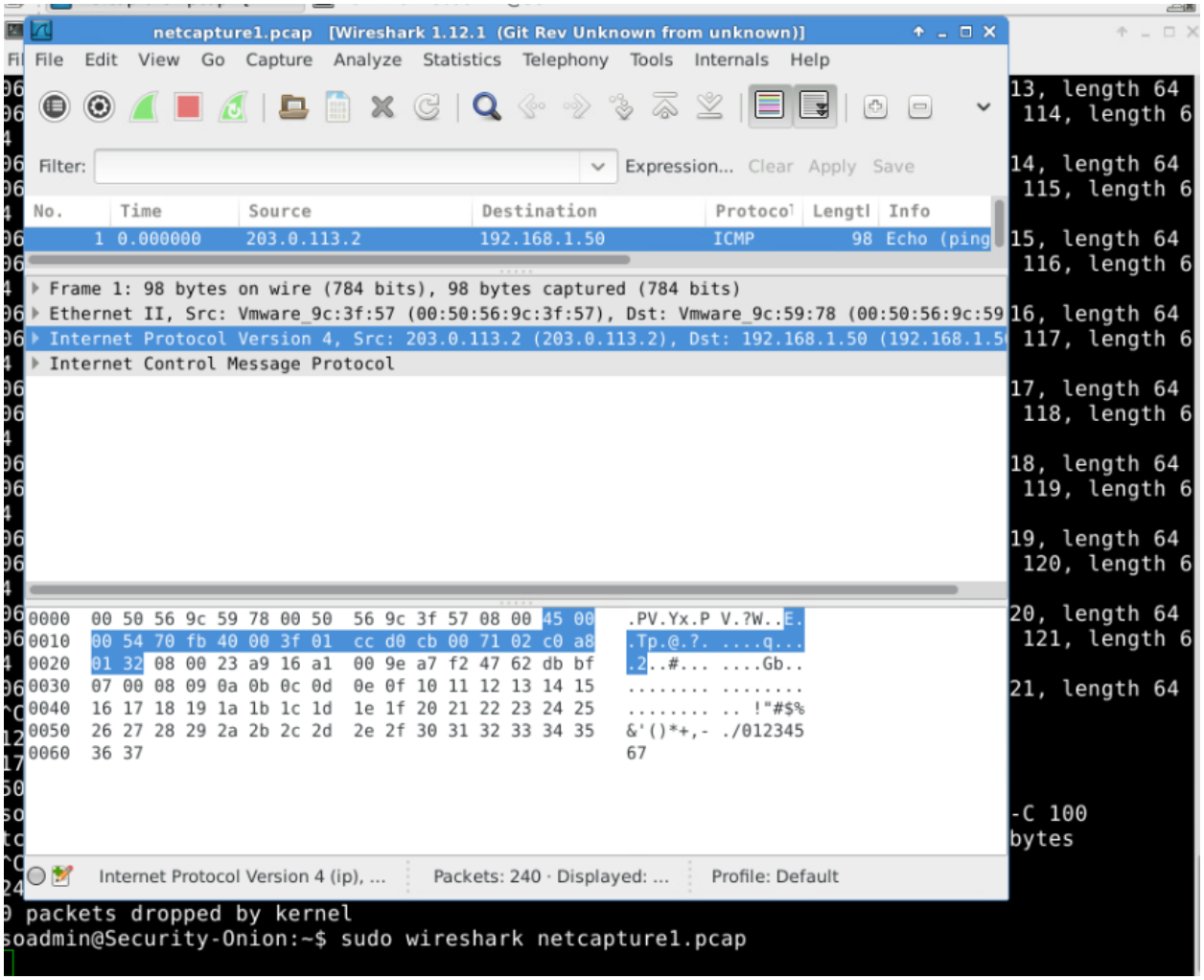

Terminal
3:05 AM
root@Ubuntu: ~
eth0   1500 0   555   0   0 0   679   0   0   0 B
MRU
lo   65536 0   155   0   0 0   155   0   0   0 L
RU
student@Ubuntu:~$ sudo ip link set eth0 promisc on
[sudo] password for student:
student@Ubuntu:~$ netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500 0   561   0   0 0   732   0   0   0 BMPRU
lo   65536 0   178   0   0 0   178   0   0   0 LRU
student@Ubuntu:~$ sudo -i
student@Ubuntu: ~
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
student@Ubuntu: ~
student@Ubuntu:~$ sudo urlsnarf -i eth0
[sudo] password for student:
urlsnarf: listening on eth0 [tcp port 80 or port 8080 or port 3128]


Terminal
3:08 AM
root@Ubuntu: ~
eth0   1500 0   555   0   0 0   679   0   0   0 B
MRU
lo   65536 0   155   0   0 0   155   0   0   0 L
RU
student@Ubuntu:~$ sudo ip link set eth0 promisc on
[sudo] password for student:
student@Ubuntu:~$ netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500 0   561   0   0 0   732   0   0   0 BMPRU
lo   65536 0   178   0   0 0   178   0   0   0 LRU
student@Ubuntu:~$ sudo -i
student@Ubuntu: ~
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
0:50:56:9c:59:78 0:50:56:9c:3f:57 0806 42: arp reply 192.168.1.50 is-at 0:50:56:9c:59:78
student@Ubuntu: ~
student@Ubuntu:~$ sudo urlsnarf -i eth0
[sudo] password for student:
urlsnarf: listening on eth0 [tcp port 80 or port 8080 or port 3128]
192.168.1.6 - - [02/Apr/2022:03:07:58 -0400] "GET http://example.com/ HTTP/1.1" - - "-" "Mozi
lla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/65.0.3325.
181 Chrome/65.0.3325.181 Safari/537.36"

e. tcpdump

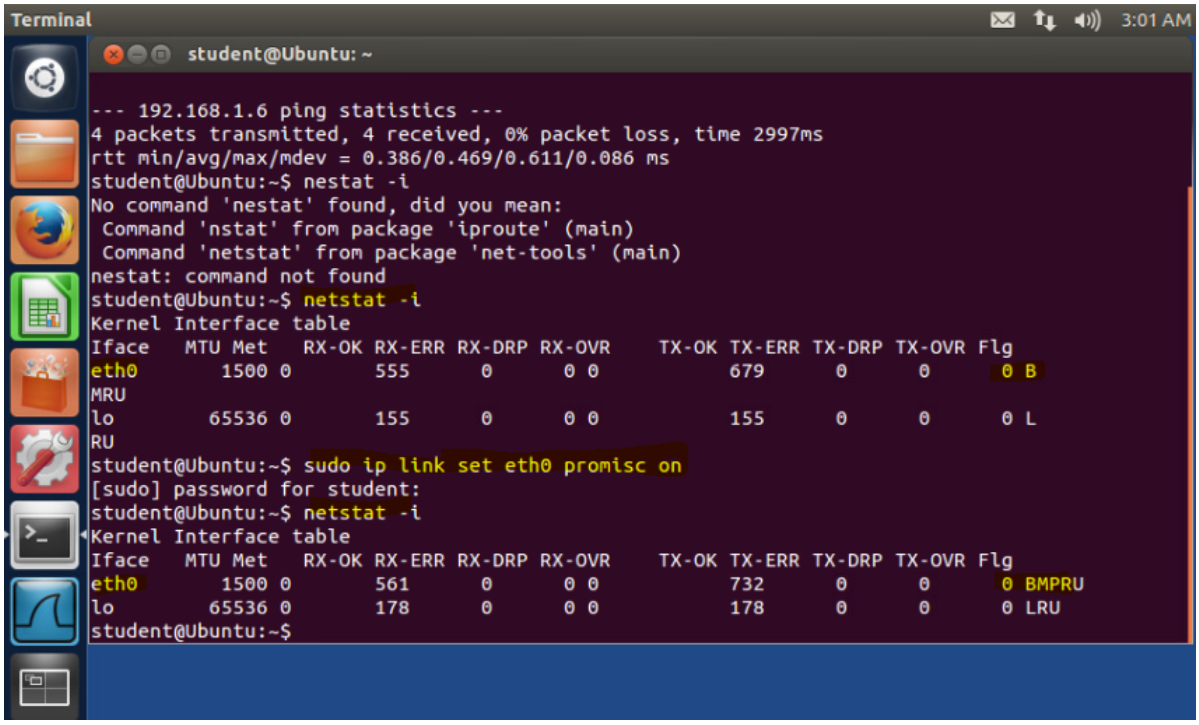
- The tcpdump analyzes networking traffic by intercepting and displays the packets that are being created or received on the device.
- We run using tcpdump in the lab on the secOnion machine and then save the network capture
- `soadmin@Security-Onion:~$ sudo tcpdump icmp –i eth0 –s 0 –w netcapture1.pcap –C 100`
- `soadmin@Security-Onion:~$ sudo wireshark netcapture1.pcap`

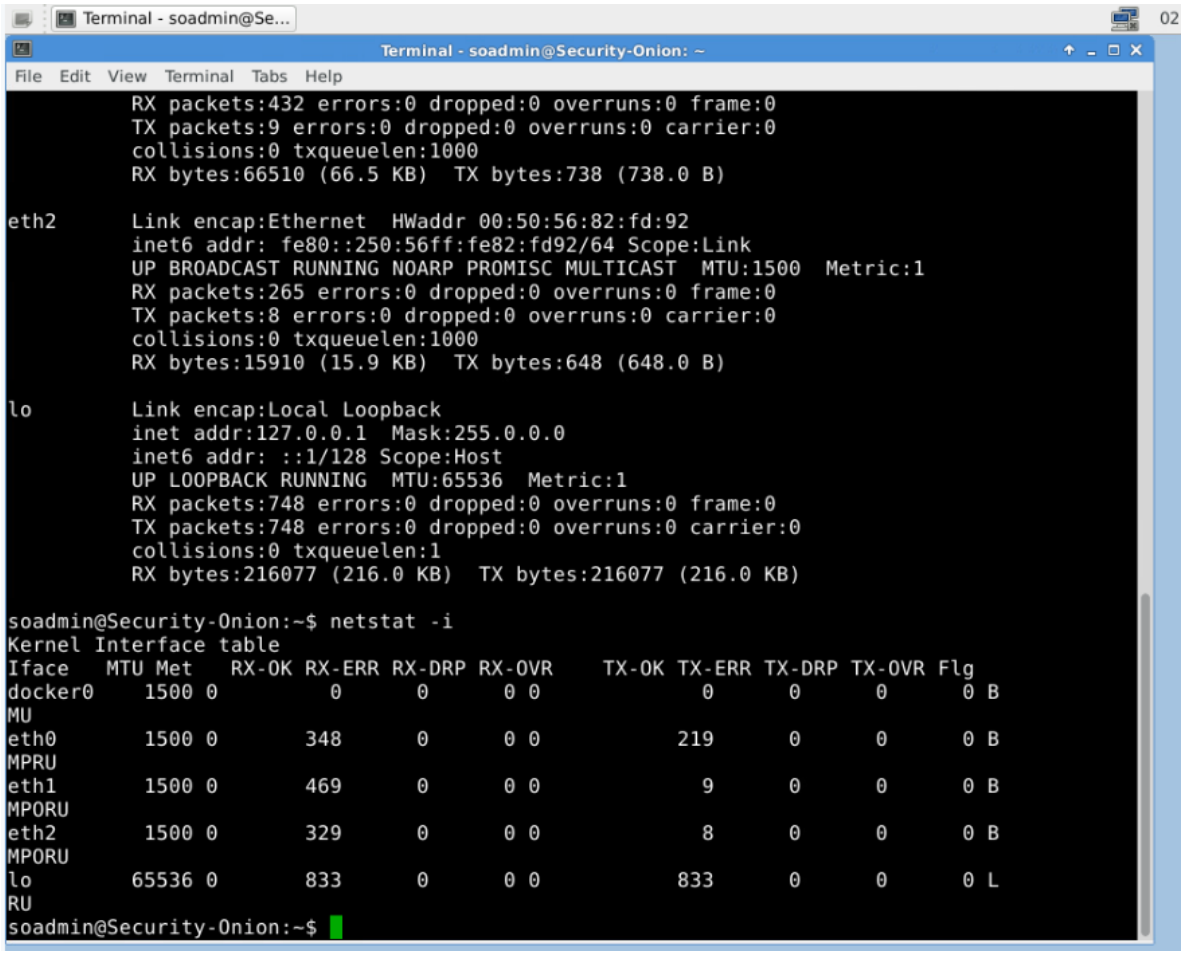




f. netstat

- Network Statistics which is used as a troubleshooter and for configuration. Netstat is also used as a monitoring tool for incoming/outgoing connections, routing tables, port listening and usage stats.
- Using the command;
- `student@Ubuntu:~$ netstat -i`
- I used it a couple times to check if changed we made have worked or to identify which flags are configured for which interface


Terminal
3:01 AM
student@Ubuntu: ~
--- 192.168.1.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.386/0.469/0.611/0.086 ms
student@Ubuntu:~$ nestat -i
No command 'nestat' found, did you mean:
Command 'nstat' from package 'iproute' (main)
Command 'netstat' from package 'net-tools' (main)
nestat: command not found
student@Ubuntu:~$ netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500 0 555 0 0 0 679 0 0 0 BMRU
lo 65536 0 155 0 0 0 155 0 0 0 LRU
student@Ubuntu:~$ sudo ip link set eth0 promisc on
[sudo] password for student:
student@Ubuntu:~$ netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500 0 561 0 0 0 732 0 0 0 BMPRU
lo 65536 0 178 0 0 0 178 0 0 0 LRU
student@Ubuntu:~$


Terminal - soadmin@Se...
02
Terminal - soadmin@Security-Onion: ~
File Edit View Terminal Tabs Help
RX packets:432 errors:0 dropped:0 overruns:0 frame:0
TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:66510 (66.5 KB) TX bytes:738 (738.0 B)
eth2 Link encap:Ethernet HWaddr 00:50:56:82:fd:92
inet6 addr: fe80::250:56ff:fe82:fd92/64 Scope:Link
UP BROADCAST RUNNING NOARP PROMISC MULTICAST MTU:1500 Metric:1
RX packets:265 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:15910 (15.9 KB) TX bytes:648 (648.0 B)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:748 errors:0 dropped:0 overruns:0 frame:0
TX packets:748 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:216077 (216.0 KB) TX bytes:216077 (216.0 KB)
soadmin@Security-Onion:~$ netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
docker0 1500 0 0 0 0 0 0 0 0 0 BMU
eth0 1500 0 348 0 0 0 219 0 0 0 BMPRU
eth1 1500 0 469 0 0 0 9 0 0 0 BMPORU
eth2 1500 0 329 0 0 0 8 0 0 0 BMPORU
lo 65536 0 833 0 0 0 833 0 0 0 LRU
soadmin@Security-Onion:~$

PART 2

Packet Crafting with Scapy

Question 1. [0.5 marks] Explain the utility of the following settings/commands used in this lab

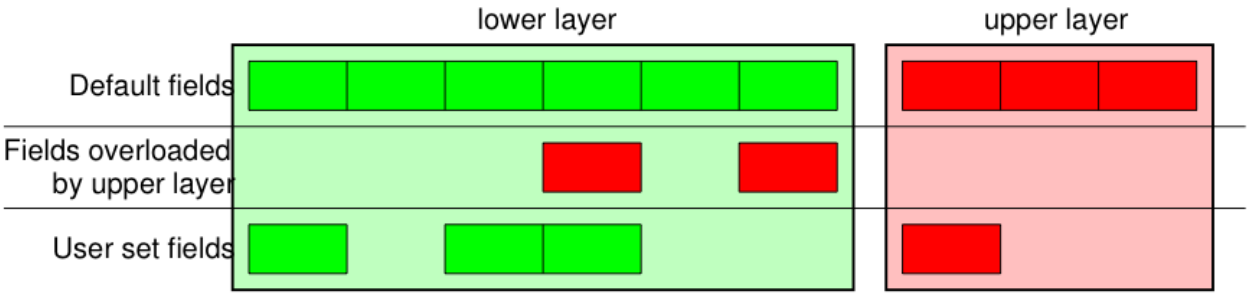
- a. TTL
  - Time to Live, we used this because its usage is for every time a machine receives an IP packet, it decreases the TTL by whatever value you have set and passes it on, this ensures that packets don't get into infinite loops.
  - Screenshots below example taken from lab

```
>>> ip=IP(ttl=10)
>>>
```

```
<IP  ttl=10 |>
>>>
```

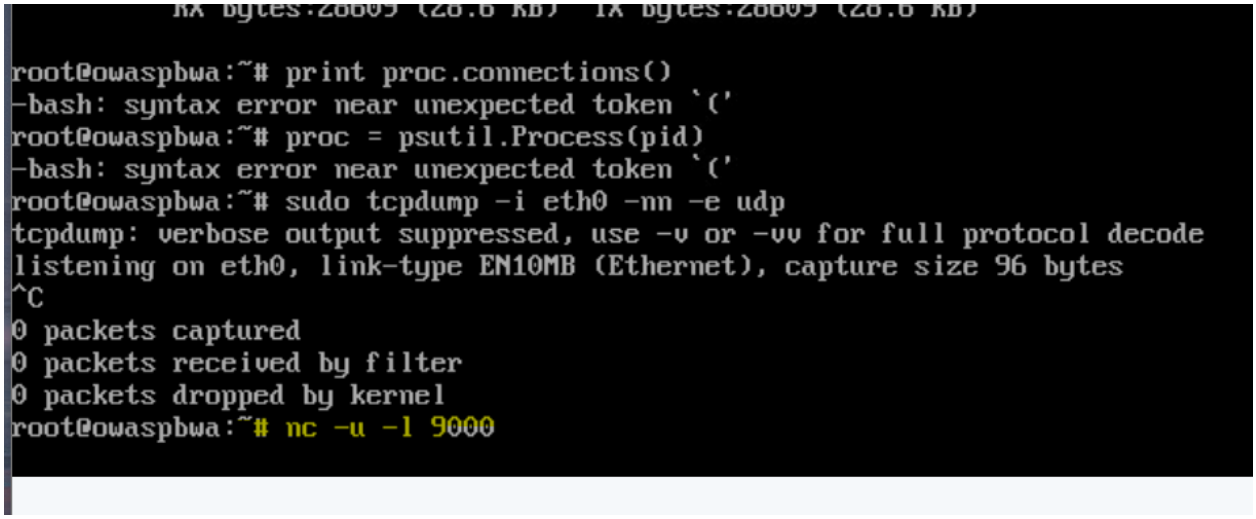
- b. / Operator (Provide an example)
  - The "/" slash operator is used to differentiate between two or more layers, so when using this the lower layer can contain one or however more default fields overloaded depending on the contents of the upper layer.
  - For example below further down in this lab I sent a udp packet and after setting the ip destination the packet is being sent to using the "/" operator I set another command in creating the source and destination ports and again when setting the payload contents to "hello". This separates the commands while in association with one another.
  - Documentation; https://scapy.readthedocs.io/en/latest/usage.html

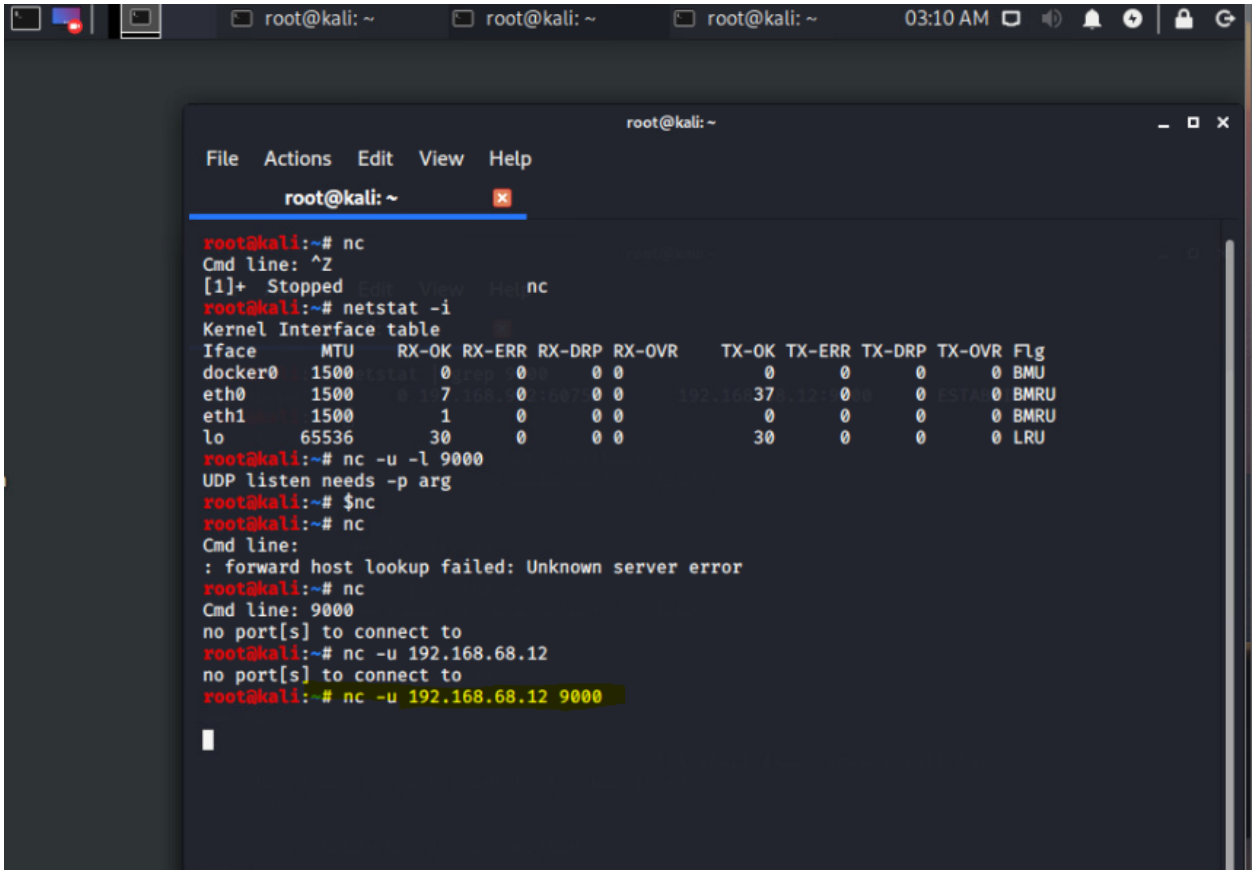
```
                    ccaacs
                                    using IPython 5.8.0
>>> get_if_addr(conf. iface)
'192.168.9.2'
>>> send(IP(dst="192.168.68.12")/UDP(sport=9000,dport=9090)/Raw(load="hello"))
.
Sent 1 packets.
>>>
```

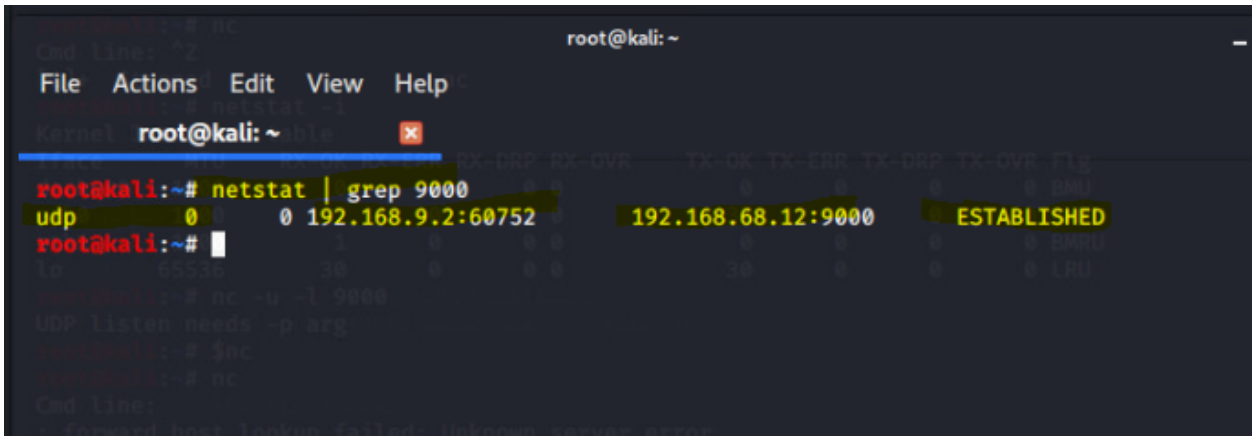


Question 2. [2 marks] Write the commands in Scapy for the following

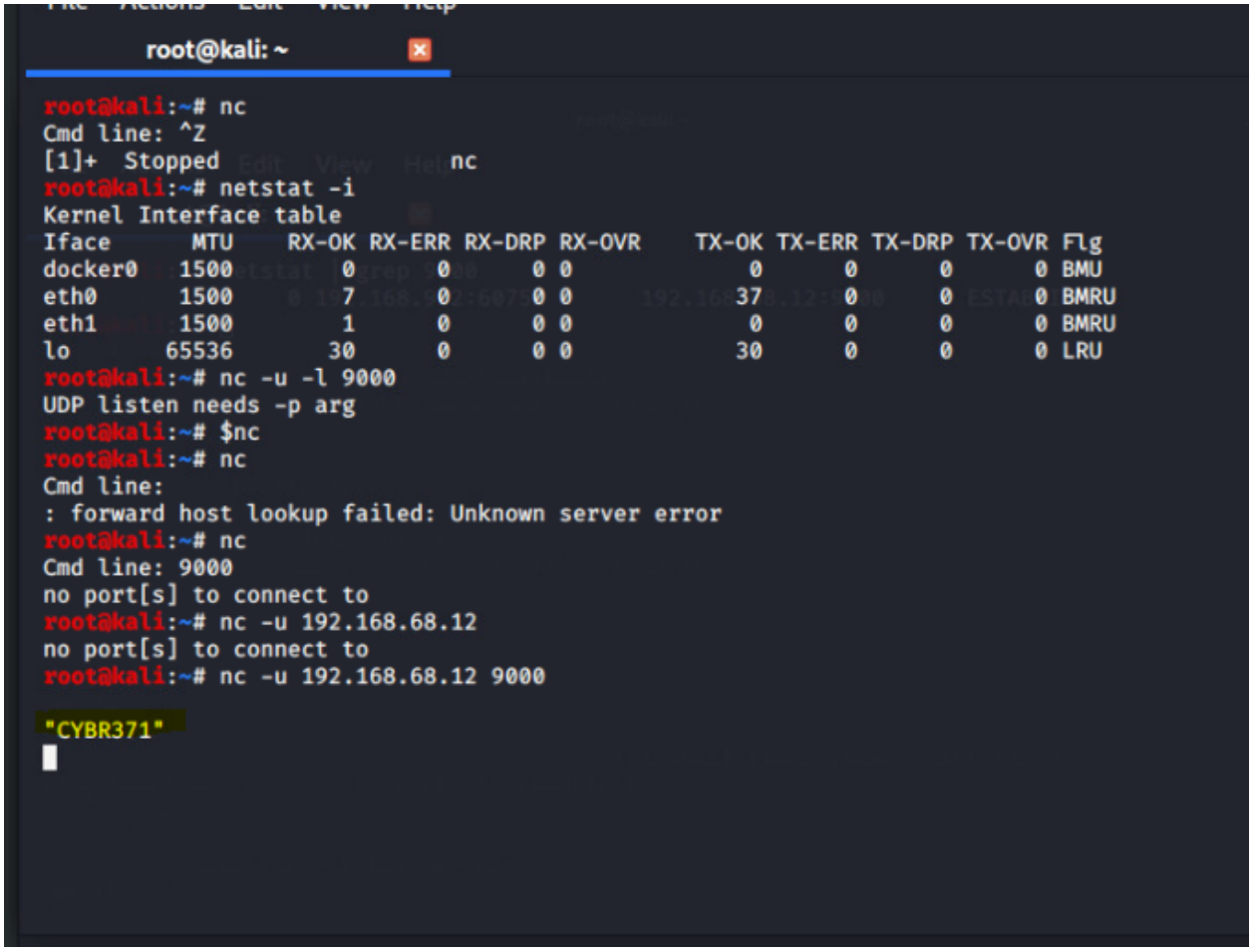
- a. - Create and send a UDP packet with the payload “CYBR371” with source port of 9000 from the Kali host, and to OWASP BWA host on destination port of 9090. Take a screenshot of the tcpdump capture on the destination, confirming your packet was delivered.
    - I did this a couple different ways with different results;
    - Using this website as reference https://linuxhint.com/send_receive_udp_packets_linux_cli/
    - OWASP IP: 192.168.68.12
    - also;
    - From the scapy documentation we can send a UDP packet using;
    - send(IP(dst=”192.168.68.12”)/UDP(dport=9000)/Raw(load=”CYBR371”))



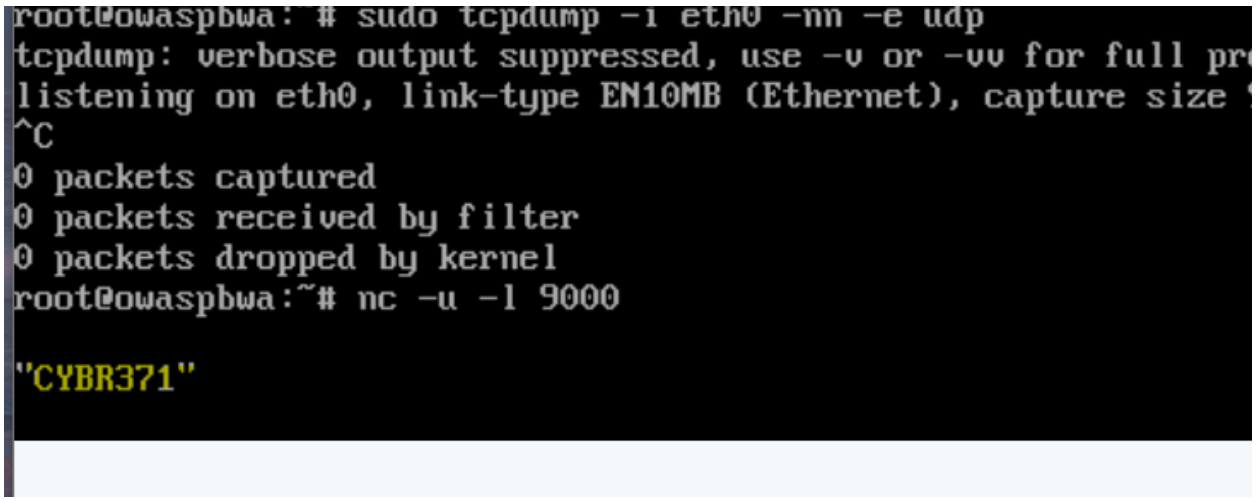




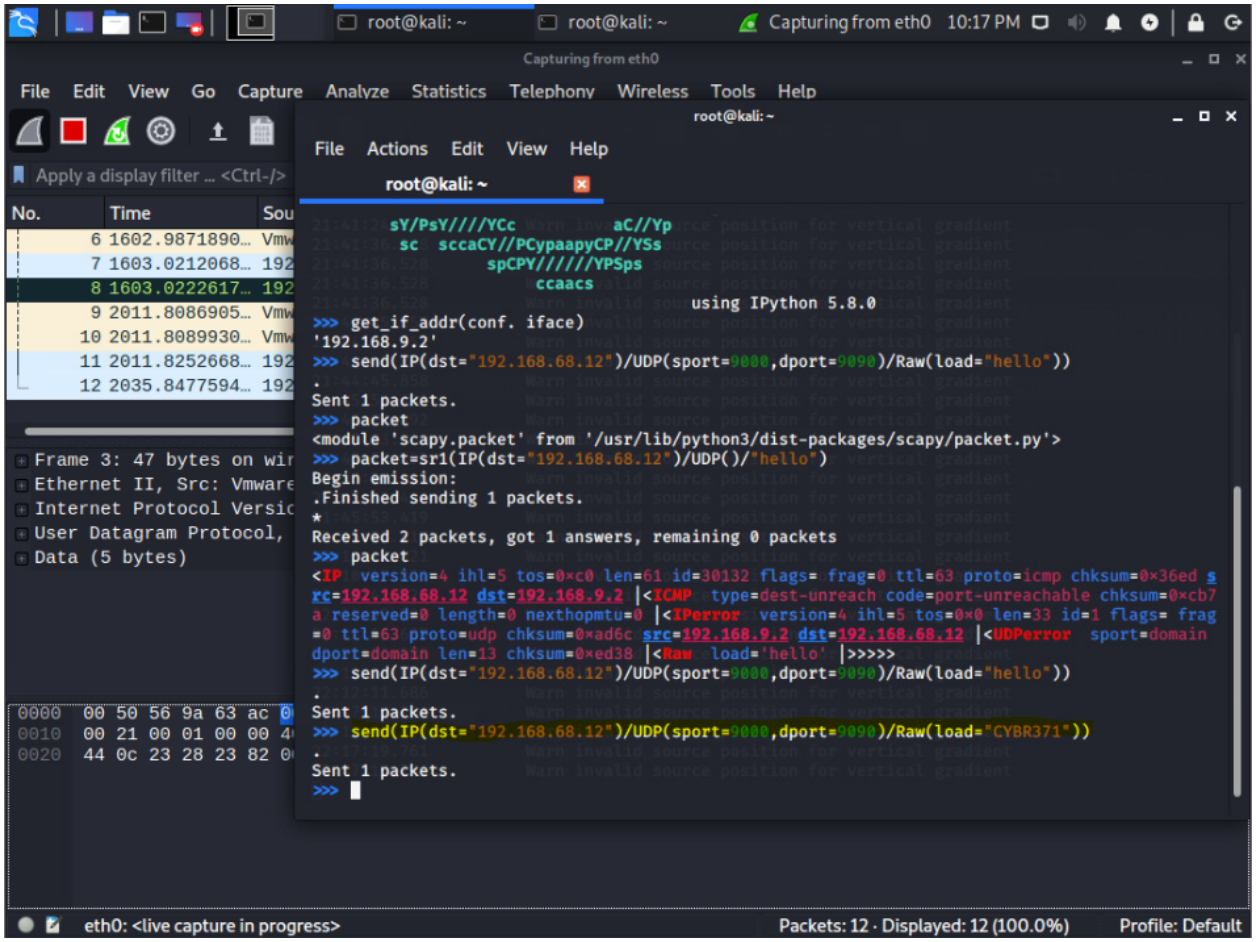
Sending the payload “CYBR371” from the host Kali machine on port 9000;

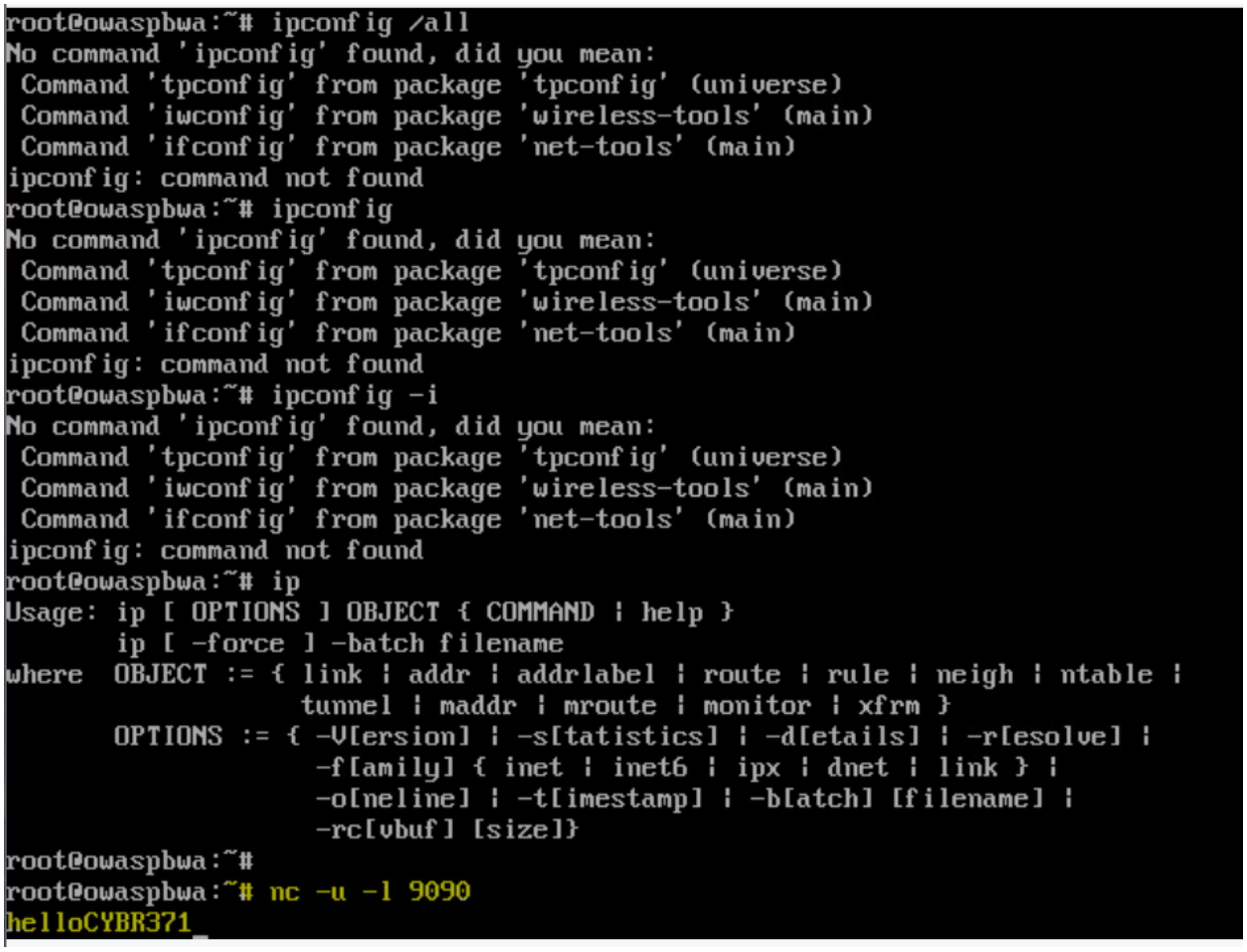


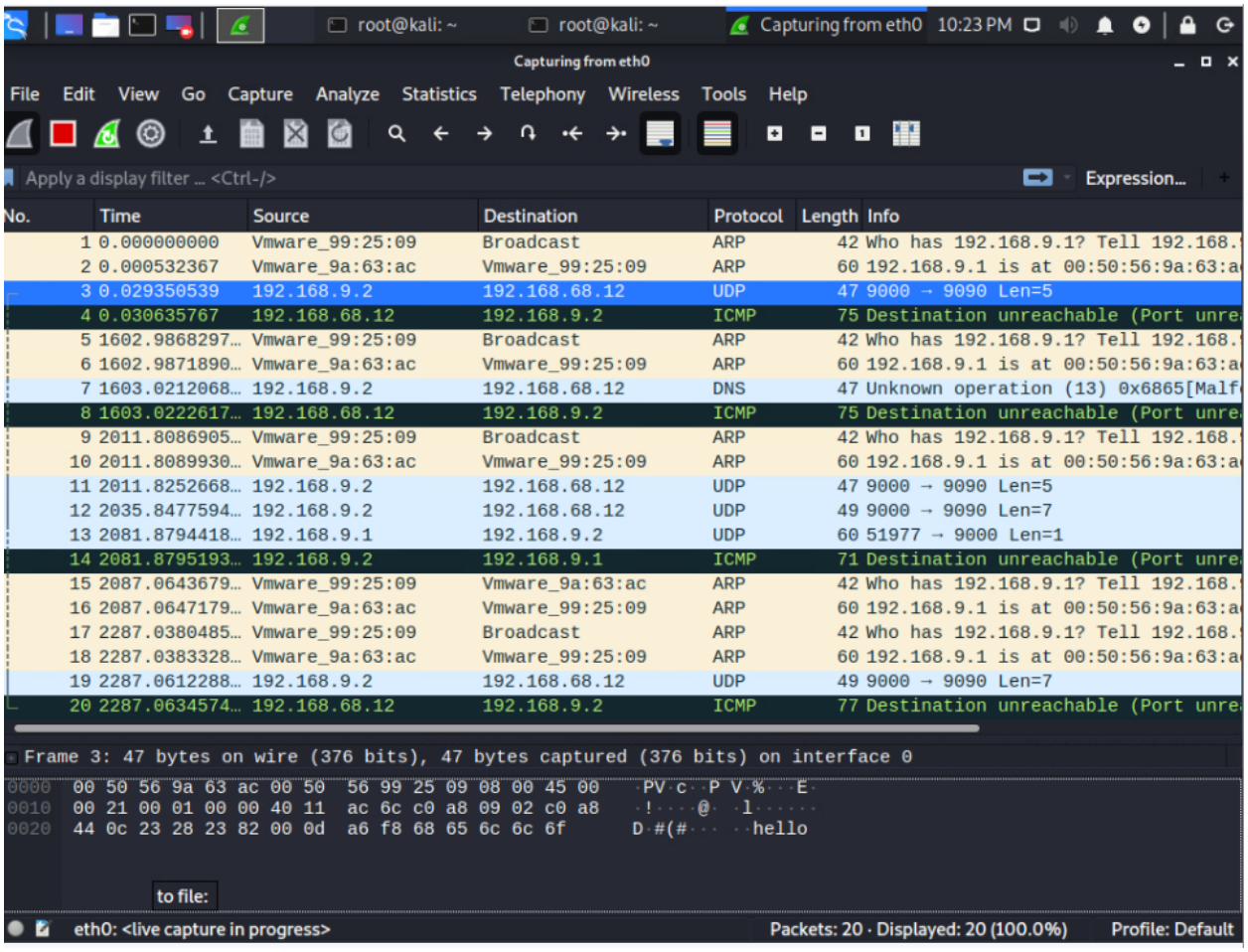
Receiving the payload “CYBR371” on the OWASP BWA machine



- Using documentation I also did this:


root@kali: ~
root@kali: ~
Capturing from eth0
10:17 PM
Capturing from eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Sou
6 1602.9871890… Vmw
7 1603.0212068… 192
8 1603.0222617… 192
9 2011.8086905… Vmw
10 2011.8089930… Vmw
11 2011.8252668… 192
12 2035.8477594… 192
Frame 3: 47 bytes on wir
Ethernet II, Src: Vmware
Internet Protocol Versio
User Datagram Protocol,
Data (5 bytes)
0000 00 50 56 9a 63 ac 0
0010 00 21 00 01 00 00 4
0020 44 0c 23 28 23 82 0
root@kali: ~
File Actions Edit View Help
root@kali: ~
sY/PsY////YCc aC//Yp
sc sccaCY//PCypaapyCP//YSs
spCPY//////YPSps
ccaacs
using IPython 5.8.0
>>> get_if_addr(conf. iface)
'192.168.9.2'
>>> send(IP(dst="192.168.68.12")/UDP(sport=9000,dport=9090)/Raw(load="hello"))
.
Sent 1 packets.
>>> packet
<module 'scapy.packet' from '/usr/lib/python3/dist-packages/scapy/packet.py'>
>>> packet=sr1(IP(dst="192.168.68.12")/UDP()/"hello")
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
>>> packet
<IP version=4 ihl=5 tos=0x0 len=61 id=30132 flags= frag=0 ttl=63 proto=icmp chksum=0x36ed src=192.168.68.12 dst=192.168.9.2 |<ICMP type=dest-unreach code=port-unreachable chksum=0xcb7a reserved=0 length=0 nexthopmtu=0 |<IPerror version=4 ihl=5 tos=0x0 len=33 id=1 flags= frag=0 ttl=63 proto=udp chksum=0xad6c src=192.168.9.2 dst=192.168.68.12 |<UDPerror sport=domain dport=domain len=13 chksum=0xed38 |<Raw load='hello' |>>>>>
>>> send(IP(dst="192.168.68.12")/UDP(sport=9000,dport=9090)/Raw(load="hello"))
.
Sent 1 packets.
>>> send(IP(dst="192.168.68.12")/UDP(sport=9000,dport=9090)/Raw(load="CYBR371"))
.
Sent 1 packets.
>>>
eth0: <live capture in progress>
Packets: 12 · Displayed: 12 (100.0%)
Profile: Default


root@owaspbwa:~# ipconfig /all
No command 'ipconfig' found, did you mean:
Command 'tpconfig' from package 'tpconfig' (universe)
Command 'iwconfig' from package 'wireless-tools' (main)
Command 'ifconfig' from package 'net-tools' (main)
ipconfig: command not found
root@owaspbwa:~# ipconfig
No command 'ipconfig' found, did you mean:
Command 'tpconfig' from package 'tpconfig' (universe)
Command 'iwconfig' from package 'wireless-tools' (main)
Command 'ifconfig' from package 'net-tools' (main)
ipconfig: command not found
root@owaspbwa:~# ipconfig -i
No command 'ipconfig' found, did you mean:
Command 'tpconfig' from package 'tpconfig' (universe)
Command 'iwconfig' from package 'wireless-tools' (main)
Command 'ifconfig' from package 'net-tools' (main)
ipconfig: command not found
root@owaspbwa:~# ip
Usage: ip [OPTIONS] OBJECT { COMMAND | help }
ip [-force] -batch filename
where OBJECT := { link | addr | addrlabel | route | rule | neigh | ntable |
tunnel | maddr | mroute | monitor | xfrm }
OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
-f[amily] { inet | inet6 | ipx | dnet | link } |
-o[neline] | -t[imestamp] | -b[atch] [filename] |
-rc[vbuf] [size]}
root@owaspbwa:~#
root@owaspbwa:~# nc -u -l 9090
helloCYBR371



- b. - Send an ICMP packet from the host (Kali) to the destination (OWASP BWA). Take a screenshot of the tcpdump capture on the destination confirming your ICMP packet was delivered.
    - Using the lab as an example I started a netcapture using tcpdump icmp
    - I then went to the Kali machine and pinged OWASP BWA's IP
    - Using the following command I am able to view the netcapture.pcap data
    - Sudo tcpdump -ttttnnr netcapture1.pcap
    - Reference taken from
    - https://serverfault.com/questions/38626/how-can-i-read-pcap-files-in-a-friendly-format

```
root@owaspbwa:~# sudo tcpdump icmp -i eth0 -s 0 -w netcapture1.pcap -C 100
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 byte
s
^C10 packets captured
10 packets received by filter
0 packets dropped by kernel
root@owaspbwa:~#
```

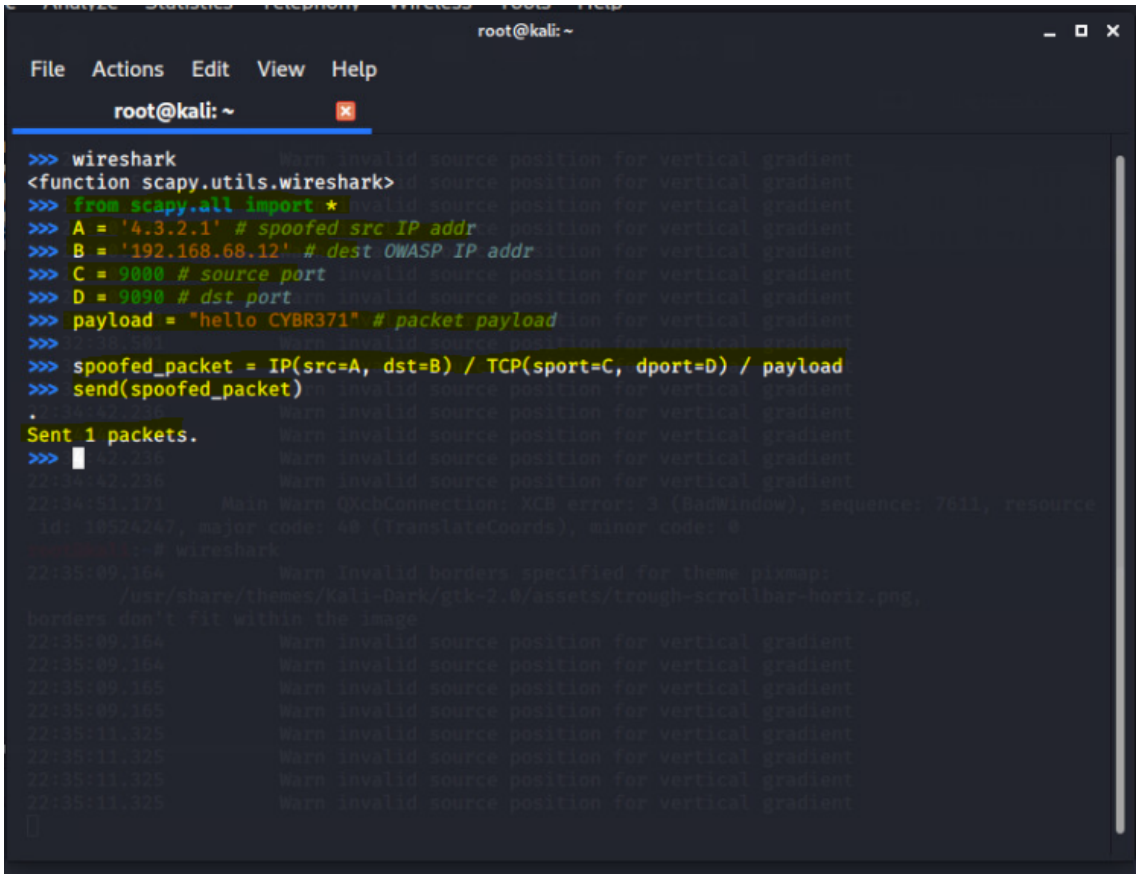
```
root@kali:~# ping -c4 192.168.68.12
PING 192.168.68.12 (192.168.68.12) 56(84) bytes of data.
64 bytes from 192.168.68.12: icmp_seq=1 ttl=63 time=4.34 ms
64 bytes from 192.168.68.12: icmp_seq=2 ttl=63 time=0.746 ms
64 bytes from 192.168.68.12: icmp_seq=3 ttl=63 time=0.695 ms
64 bytes from 192.168.68.12: icmp_seq=4 ttl=63 time=0.620 ms

--- 192.168.68.12 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3053ms
rtt min/avg/max/mdev = 0.620/1.600/4.340/1.582 ms
root@kali:~#
```

```
z♦LIb;♦ !"#$%&'()*+,-./01234567♦LIbc♦                    ♦-
                                    bbPV♦♦XPV♦ET@♦♦♦D
z♦LIb;♦ !"#$%&'()*+,-./01234567♦LIb♦                  ♦ ♦-
                                   bPV♦♦PV♦♦ET ♦@?L¦♦  ♦D
z♦LIb♦0 !"#$%&'()*+,-./01234567♦LIb♦                    ♦♦
                                   bPV♦♦XPV♦ET@♦♦♦D
z♦LIb♦0 !"#$%&'()*+,-./01234567root@owaspbwa:~#     ♦   ♦♦
root@owaspbwa:~# sudo tcpdump -ttttnnr netcapture1.pcap
reading from file netcapture1.pcap, link-type EN10MB (Ethernet)
2022-04-03 03:20:37.496275 IP 192.168.68.12 > 192.168.9.2: ICMP 192.168.68.12 ud
p port 9000 unreachable, length 37
2022-04-03 03:21:00.147260 IP 192.168.68.12 > 192.168.9.2: ICMP 192.168.68.12 ud
p port 9000 unreachable, length 44
2022-04-03 03:30:00.740729 IP 192.168.9.2 > 192.168.68.12: ICMP echo request, id
 3450, seq 1, length 64
2022-04-03 03:30:00.744208 IP 192.168.68.12 > 192.168.9.2: ICMP echo reply, id 3
450, seq 1, length 64
2022-04-03 03:30:01.742213 IP 192.168.9.2 > 192.168.68.12: ICMP echo request, id
 3450, seq 2, length 64
2022-04-03 03:30:01.742241 IP 192.168.68.12 > 192.168.9.2: ICMP echo reply, id 3
450, seq 2, length 64
2022-04-03 03:30:02.769611 IP 192.168.9.2 > 192.168.68.12: ICMP echo request, id
 3450, seq 3, length 64
2022-04-03 03:30:02.769635 IP 192.168.68.12 > 192.168.9.2: ICMP echo reply, id 3
450, seq 3, length 64
2022-04-03 03:30:03.793510 IP 192.168.9.2 > 192.168.68.12: ICMP echo request, id
 3450, seq 4, length 64
2022-04-03 03:30:03.793534 IP 192.168.68.12 > 192.168.9.2: ICMP echo reply, id 3
450, seq 4, length 64
root@owaspbwa:~#
```

- c. - Sniff UDP traffic on all the interfaces on the host machine (i.e., Kali)
    - Sniff the traffic on eth0

- d. - Send a spoofed IP packet (with forged source IP address) from the host (Kali) to the destination machine (OWASP BWA) with forged source IP address of 4.3.2.1. Take a screenshot of the tcpdump capture on the destination proving your message was delivered. Why was the message not dropped in transmissions even through the source IP address does not exist?
    - Used stack overflow as reference; https://stackoverflow.com/questions/27448905/send-packet-and-change-its-source-ip

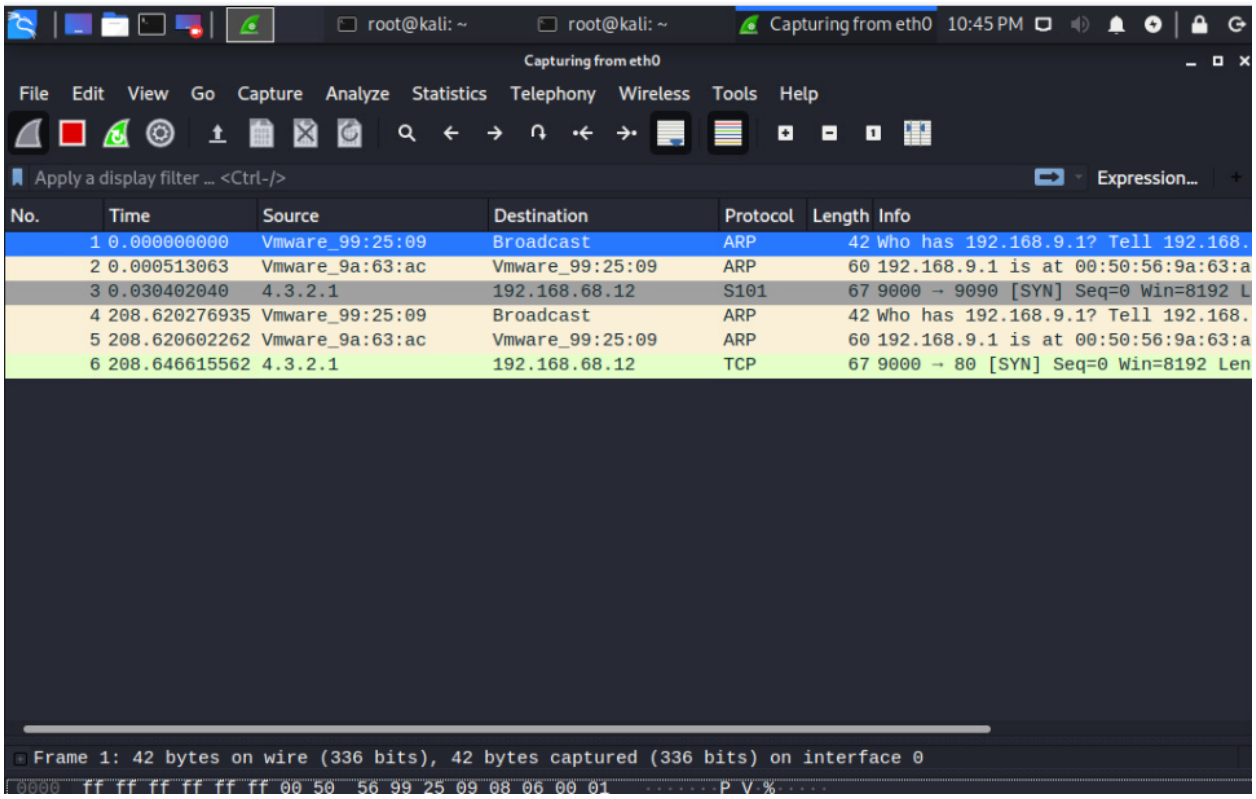


```
root@owaspbwa:~# sudo tcpdump icmp -i eth0 -s 0 -w netcapture2.pcap -C 100
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 byte
s

^C1 packets captured
1 packets received by filter
0 packets dropped by kernel
root@owaspbwa:~# _
```

```
root@owaspbwa:~# sudo tcpdump icmp -i eth0 -s 0 -w netcapture2.pcap -C 100
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 byte
s

^C1 packets captured
1 packets received by filter
0 packets dropped by kernel
root@owaspbwa:~# sudo tcpdump -ttttnnr netcapture2.pcap
reading from file netcapture2.pcap, link-type EN10MB (Ethernet)
2022-04-06 22:42:04.438729 IP 192.168.68.254 > 192.168.68.12: ICMP host 4.3.2.1
unreachable, length 48
root@owaspbwa:~#
```


root@kali: ~
root@kali: ~
Capturing from eth0
10:45 PM
Capturing from eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 0.000000000 Vmware_99:25:09 Broadcast ARP 42 Who has 192.168.9.1? Tell 192.168.
2 0.000513063 Vmware_9a:63:ac Vmware_99:25:09 ARP 60 192.168.9.1 is at 00:50:56:9a:63:a
3 0.030402040 4.3.2.1 192.168.68.12 S101 67 9000 → 9090 [SYN] Seq=0 Win=8192 L
4 208.620276935 Vmware_99:25:09 Broadcast ARP 42 Who has 192.168.9.1? Tell 192.168.
5 208.620602262 Vmware_9a:63:ac Vmware_99:25:09 ARP 60 192.168.9.1 is at 00:50:56:9a:63:a
6 208.646615562 4.3.2.1 192.168.68.12 TCP 67 9000 → 80 [SYN] Seq=0 Win=8192 Len
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
0000 ff ff ff ff ff ff 00 50 56 99 25 09 08 06 00 01 ....P V.%.....