CSE 433S HW/Lab1 - Packet Sniffing and Spoofing

Zihan Chen

September 2024

Contents

1 Topic 1

1.1 ifconfig -a

ifconfig displays the status of the currently active interfaces.

If a single interface argument is given, it displays the status of the given interface only.

ens160: the net can broadcast and accept multicast, it also in running. It can accept max length of packet is 1500. It has a ip 192.168.92.132 and netmask means the internal ip area is in 192.168.92.0-192.168.92.255.

lo: local loopback network, with ip 127.0.0.1 means the computer itself, and can accept a 65536 size packet.

```
[09/12/24]seed@VM:Steven$ ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.92.132  netmask 255.255.255.0  broadcast 192.168.92.255
        inet6 fe80::99b4:96c0:ff4d:3dd4  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:50:5e:ff  txqueuelen 1000  (Ethernet)
        RX packets 240203  bytes 320869686 (320.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43412  bytes 2931338 (2.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 44  memory 0x3fe00000-3fe20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2398  bytes 203818 (203.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2398  bytes 203818 (203.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

If a single -a argument is given, it displays the status of all interfaces, even those that are down.

Here it displays three interfaces: docker0, ens160, and lo. docker0 is a bridge interface created by Docker. ens160 is the main interface of the VM. lo is the loopback interface.

```
[09/12/24]seed@VM:Steven$ ifconfig -a
docker0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 02:42:22:56:13:27  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.92.132  netmask 255.255.255.0  broadcast 192.168.92.255
        inet6 fe80::99b4:96c0:ff4d:3dd4  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:50:5e:ff  txqueuelen 1000  (Ethernet)
        RX packets 240208  bytes 320870453 (320.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43417  bytes 2931789 (2.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 44  memory 0x3fe00000-3fe20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2398  bytes 203818 (203.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2398  bytes 203818 (203.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Otherwise, it configures an interface.

1.2 ip addr

ip addr can print out similar information as ifconfig -a. It will display the status of all interfaces, even those that are down.

```
IP - COMMAND SYNTAX
    OBJECT
        address
                - protocol (IP or IPv6) address on a device.

        addrlabel
                - label configuration for protocol address selection.
```

You can find that the output of ip addr is more concise than ifconfig -a.

```
[09/12/24]seed@VM:Steven$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group 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: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:50:5e:ff brd ff:ff:ff:ff:ff:ff
    altname enp2s0
    inet 192.168.92.132/24 brd 192.168.92.255 scope global dynamic noprefixroute ens160
       valid_lft 1359sec preferred_lft 1359sec
    inet6 fe80::99b4:96c0:ff4d:3dd4/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:22:56:13:27 brd ff:ff:ff:ff:ff:ff
```

1.3 ping -c

ping is used to send ICMP ECHO_REQUEST packets to network hosts.

-c is used to specify the number of packets to send.

```
[09/12/24]seed@VM:Steven$ ping -c 5 google.com
PING google.com (142.250.191.110) 56(84) bytes of data.
64 bytes from ord38s28-in-f14.1e100.net (142.250.191.110): icmp_seq=1 ttl=128 time=18.1 ms
64 bytes from ord38s28-in-f14.1e100.net (142.250.191.110): icmp_seq=2 ttl=128 time=21.7 ms
64 bytes from ord38s28-in-f14.1e100.net (142.250.191.110): icmp_seq=3 ttl=128 time=18.0 ms
64 bytes from ord38s28-in-f14.1e100.net (142.250.191.110): icmp_seq=4 ttl=128 time=20.2 ms
64 bytes from ord38s28-in-f14.1e100.net (142.250.191.110): icmp_seq=5 ttl=128 time=22.9 ms

--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 17.957/20.165/22.862/1.943 ms
```

```
[09/12/24]seed@VM:Steven$ ping -c 5 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.119 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.106 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.115 ms

--- localhost ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4111ms
rtt min/avg/max/mdev = 0.100/0.109/0.119/0.006 ms
```

1.4 tcpdump

tcpdump is a common packet analyzer that runs under the command line.

We can capture ip packets but actually TCP packets are captured. These packets are between 2 VMs, actually, they are from HW1's program.

```
21:17:29.247289 IP 192.168.92.133.39366 > seed-virtual-machine.10888: Flags [P.], seq 1:18, ack 1,
win 502, options [nop,nop,TS val 1917907702 ecr 2104636992], length 17
21:17:29.247417 IP seed-virtual-machine.10888 > 192.168.92.133.39366: Flags [.], ack 18, win 509, o
ptions [nop,nop,TS val 2104642943 ecr 1917907702], length 0
21:17:29.247646 IP seed-virtual-machine.10888 > 192.168.92.133.39366: Flags [P.], seq 1:31, ack 18,
 win 509, options [nop,nop,TS val 2104642943 ecr 1917907702], length 30
21:17:29.247786 IP seed-virtual-machine.10888 > 192.168.92.133.39366: Flags [F.], seq 31, ack 18, w
in 509, options [nop,nop,TS val 2104642943 ecr 1917907702], length 0
21:17:29.248371 IP 192.168.92.133.39366 > seed-virtual-machine.10888: Flags [.], ack 31, win 502, o
ptions [nop,nop,TS val 1917907703 ecr 2104642943], length 0
21:17:29.248447 IP 192.168.92.133.39366 > seed-virtual-machine.10888: Flags [F.], seq 18, ack 32, w
in 502, options [nop,nop,TS val 1917907703 ecr 2104642943], length 0
21:17:29.248466 IP seed-virtual-machine.10888 > 192.168.92.133.39366: Flags [.], ack 19, win 509, o
ptions [nop,nop,TS val 2104642944 ecr 1917907703], length 0
21:17:33.389076 IP 192.168.92.1.57621 > 192.168.92.255.57621: UDP, length 44
21:17:33.458830 IP seed-virtual-machine.52280 > _gateway.domain: 11641+ PTR? 1.92.168.192.in-addr.a
rpa. (43)
```

We can capture arp packets. Arp packets ask who are the gateway, someones reply an address for it.

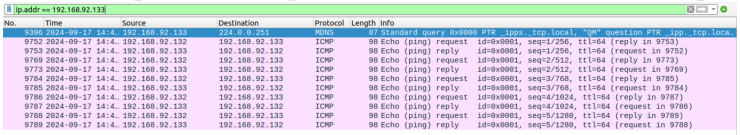
```
21:17:22.382324 IP seed-virtual-machine.33107 > _gateway.domain: 38756+ PTR? 6.1.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
21:17:22.456776 ARP, Request who-has seed-virtual-machine tell _gateway, length 46
21:17:22.456820 ARP, Reply seed-virtual-machine is-at 00:0c:29:50:5e:ff (oui Unknown), length 28
21:17:22.456986 IP is-content-cache-2.ps5.canonical.com.http > seed-virtual-machine.36072: Flags [F
.], seq 1:190, ack 88, win 64240, length 189: HTTP: HTTP/1.1 204 No Content
21:17:22.457133 IP seed-virtual-machine.36072 > is-content-cache-2.ps5.canonical.com.http: Flags [F
.], seq 88, ack 191, win 64050, length 0
```

2 Topic 2

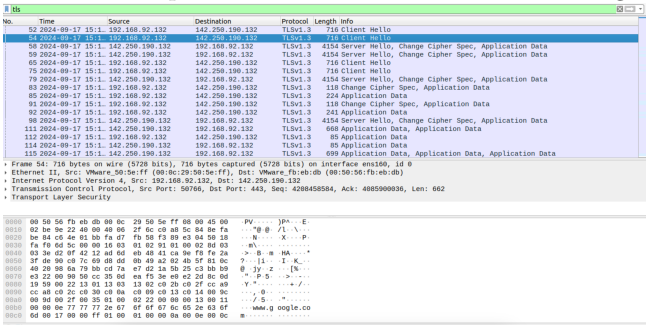
Open Wireshark, record ens160 interface, search in website and ping 192.168.92.133. In fact, I do the task for many times, so the packets may be not in the same capture.

First, Use filter "ip.addr==192.168.92.133". We can find the ip packets between two VMs. These packets are from the ping command. We can easily find that there is a request and a reply of each ping.

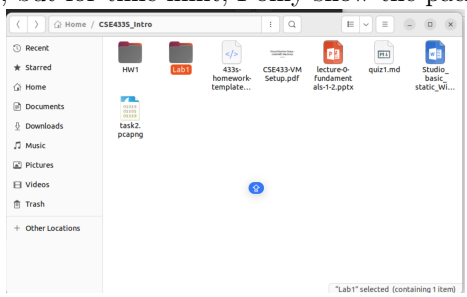
For each request, source ip is 192.168.92.132 and destination ip is 192.168.92.133



Second, ues filter "tls", we can find the packets of searching in website. However, I don't apply a certificate, so the packets are not encrypted. But we can find that the packets can show the website name is google.com.



Store the packets in pcap format. There are many other packets in the pcap file, but for time limit, I only show the packets of ping and search in website.



3 Topic 3

Install scapy and write the sample code as the lab pdf and then run it.

```
[09/17/24]seed@VM:Steven$ sudo python3 mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

Make it runnable by ‘chmod a+x mycode.py‘.

```
[09/17/24]seed@VM:Steven$ chmod a+x mycode.py
[09/17/24]seed@VM:Steven$ sudo ./mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

3.1 Task1.1

The code use sniff the capture the packet, and use filter to filter the icmp packets. Then it will print the packet.

I use another VM to ping the VM, and the code can capture the packets. Its ip shows that the ping is from 192.168.92.133 to 192.168.92.132

```
def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp', prn=print_pkt)
```

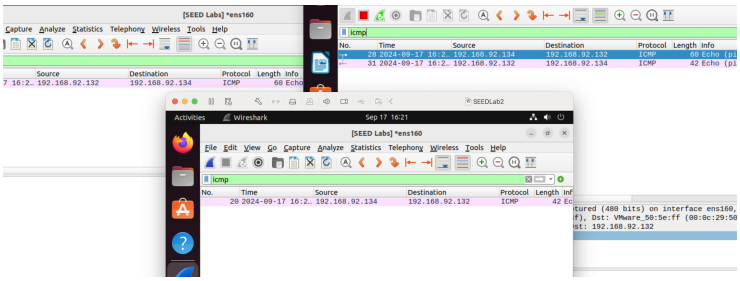
```
[09/17/24]seed@VM:Steven$ chmod a+x sniffer.py
[09/17/24]seed@VM:Steven$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 00:0c:29:50:5e:ff
  src       = 00:0c:29:d2:d7:8f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 20485
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xb049
     src       = 192.168.92.133
     dst       = 192.168.92.132
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x2bc5
        id        = 0x1
        seq       = 0x1
        unused    = ''
###[ Raw ]###
           load      = '[\\xe9\\xe9f\x00\x00\x00\x00\\xc0\x15\x08\
x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1
```

If we don't run it in root mode, the code will show the error message. It has an Operation not permitted error. This is because the code needs to capture the packets, and it needs root permission to do that.

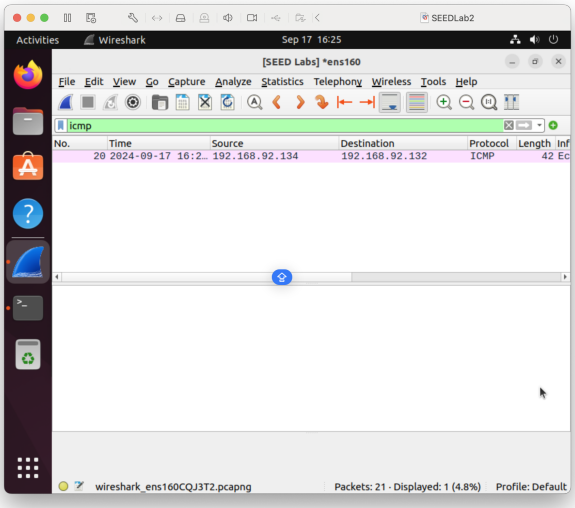
```
[09/17/24]seed@VM:Steven$ ./sniffer.py
Traceback (most recent call last):
  File "/home/seed/CSE433S_Intro/Lab1/./sniffer.py", line 9, in <m
odule>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py"
, line 1311, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py"
, line 1171, in _run
    sniff_sockets[_RL2(iface)(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.10/dist-packages/scapy/arch/linux.p
y", line 484, in __init__
    self.ins = socket.socket(
  File "/usr/lib/python3.10/socket.py", line 232, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

3.2 Task1.2

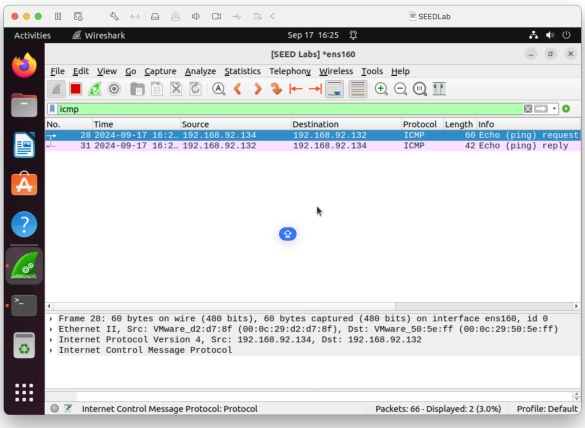
Make 3 vm open wireshark, and filter icmp packets. The VM2 send a spoof packet to VM1, and set the source ip to VM3's ip. VM1 will recieve the packet and reply to VM3.



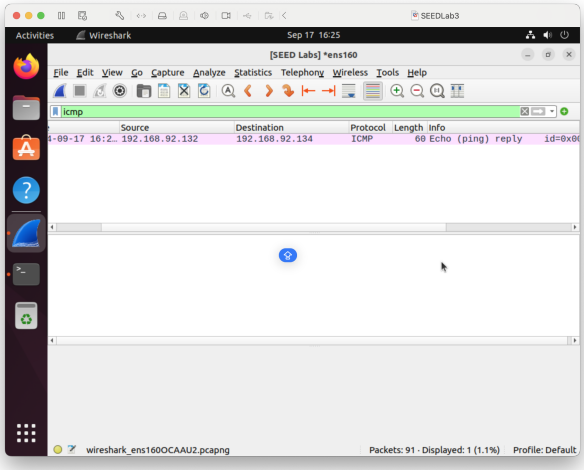
We find in VM2, there is only one packet, which is the spoof packet. The source ip is 192.168.92.134 and destination ip is 192.168.92.132



And in VM1, there are two packets, one is the spoof packet, and the other is the reply packet.



ANd in VM3, there is only one packet, which is the reply packet.



CODE IS HERE:

```
#!/usr/bin/python3

from scapy.all import *

a = IP()
a.src = '192.168.92.134'
a.dst = '192.168.92.132'
b = ICMP()
p = a/b
send(p)
```

3.3 Task2

Write a sniff by c pcap library. According to the sample code relating in the lab pdf file, I write the following code and Add packet26-33 to print the source and destination ip.

```
#include <pcap.h>
#include <stdio.h>

/* This function will be invoked by pcap for each captured packet.
We can process each packet inside the function.
*/
void got_packet(u_char *args, const struct pcap_pkthdr *header,
const u_char *packet)
{
    printf("Got a packet\n");
```

```
    // Print it source and destination IP address
    printf("Source IP: %d.%d.%d.%d\n", packet[26],
        packet[27], packet[28], packet[29]);
    printf("Destination IP: %d.%d.%d.%d\n", packet[30],
        packet[31], packet[32], packet[33]);
}
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "ip proto icmp";
    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with name eth3
    // Students needs to change "eth3" to the name
    // found on their own machines (using ifconfig).
    handle = pcap_open_live("ens160", BUFSIZ, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);

    pcap_setfilter(handle, &fp);

    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);

    pcap_close(handle); //Close the handle
    return 0;
}
```

The result is as follows: We can find when we use a ping, the source and destination ip shows in the snapshot.