Justin Gallagher

CSC 482

Packet Sniffing and Spoofing (Set 1: Using Tools to Sniff and Spoof Packets)

Packet sniffing and spoofing are two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. In this lab, we will experiment with sniffing and spoofing network packets. We will use the Scapy library to read and write packets to and from the network in python code.

Task 1.1: Sniffing Packets

In this task, we will learn how to use Scapy in order to do packet within Python programs. We are given a sample program, and I write this program into a Python file called 'sniff.py'. The first part of this task has us run 'sniff.py' with and without root privilege and compare the difference. The program will display information about the packets it sniffs. I head to the command line and change directory to the /home/seed/labs/sniffing-spoofing directory (where I saved the 'sniff.py' file) using this command: `cd labs/sniffing-spoofing`. I then change the program to be executable, using this command: `chmod a+x sniff.py`. To run the program with root privileges, I then used this command: `sudo ./sniff.py`. After running the command, I receive output in the terminal. This is a part of the output that represents one of the sniffed packets:

```
###[ Ethernet ]###
  dst       = 00:1d:71:f4:b0:00
  src       = 00:50:56:b9:85:80
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 105
     id        = 59198
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x7b32
     src       = 10.2.3.36
     dst       = 10.11.0.51
     \options   \
###[ ICMP ]###
        type      = dest-unreach
        code      = port-unreachable
        chksum    = 0x14ab
        reserved  = 0
```

```
        length    = 0
        nexthopmtu= 0
###[ IP in ICMP ]###
           version   = 4
           ihl       = 5
           tos       = 0x0
           len       = 77
           id        = 5036
           flags     = DF
           frag      = 0
           ttl       = 126
           proto     = udp
           chksum    = 0xd190
           src       = 10.11.0.51
           dst       = 10.2.3.36
           \options   \
###[ UDP in ICMP ]###
              sport     = domain
              dport     = 12764
              len       = 57
              chksum    = 0xdb0f
###[ DNS ]###
                 id        = 56897
                 qr        = 1
                 opcode    = QUERY
                 aa        = 0
                 tc        = 0
                 rd        = 1
                 ra        = 1
                 z         = 0
                 ad        = 0
                 cd        = 0
                 rcode     = ok
                 qdcount   = 1
                 ancount   = 1
                 nscount   = 0
                 arcount   = 0
                 \qd        \
                  |###[ DNS Question Record ]###
                  |  qname     = 'docs.google.com.'
                  |  qtype     = A
                  |  qclass    = IN
                 \an        \
                  |###[ DNS Resource Record ]###
                  |  rrname    = 'docs.google.com.'
                  |  type      = A
                  |  rclass    = IN
                  |  ttl       = 0
                  |  rdlen     = None
                  |  rdata     = 216.58.192.142
                 ns        = None
```

```
                ar        = None
```

I then repeat the process but this time not using the root privilege by using this command: `./sniff.py`. The result is shown below:

```
Traceback (most recent call last):
  File "./sniff.py", line 7, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1036, in
sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 907, in _run
    *arg, **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py", line 398, in
__init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
# noqa: E501
  File "/usr/lib/python3.5/socket.py", line 134, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

This shows that we are unable to sniff packets without root privilege. The second part of this task has us modifying the program in order to limit our search to certain types of packets. This is done by setting filters within our program. The first packet we need to capture is an ICMP packet. Our program is already designed to do this, and an example is shown above when we ran 'sniff.py' with root privilege. The next packet we must capture is any TCP packet that comes from a particular IP and with a destination port number 23. The following line is added to the program in order to achieve this: `pkt = sniff(filter='tcp port 23', prn=print_pkt)`. This will replace the `pkt =` line already in the code that we used for the ICMP packet, so I comment it out. I then save the program and run it in terminal. After doing so, I open another terminal and run the netcat command with my VM's IP as the destination and specified port 23: `netcat localhost 23`. After doing so, I typed 'hello world!' into the terminal and then switch back to the terminal running the sniffer program. This is some of the output I received after running:

```
###[ Ethernet ]###
  dst       = 00:50:56:b9:c6:ba
  src       = 00:50:56:b9:a4:29
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 65
     id        = 32011
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xa356
```

```
     src       = 10.2.3.45
     dst       = 10.2.3.37
     \options   \
###[ TCP ]###
        sport     = 42676
        dport     = telnet
        seq       = 981032434
        ack       = 57165189
        dataofs   = 8
        reserved  = 0
        flags     = PA
        window    = 229
        chksum    = 0x7705
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (1648424,
582121057))]
###[ Raw ]###
           load      = 'hello world!\n'
```

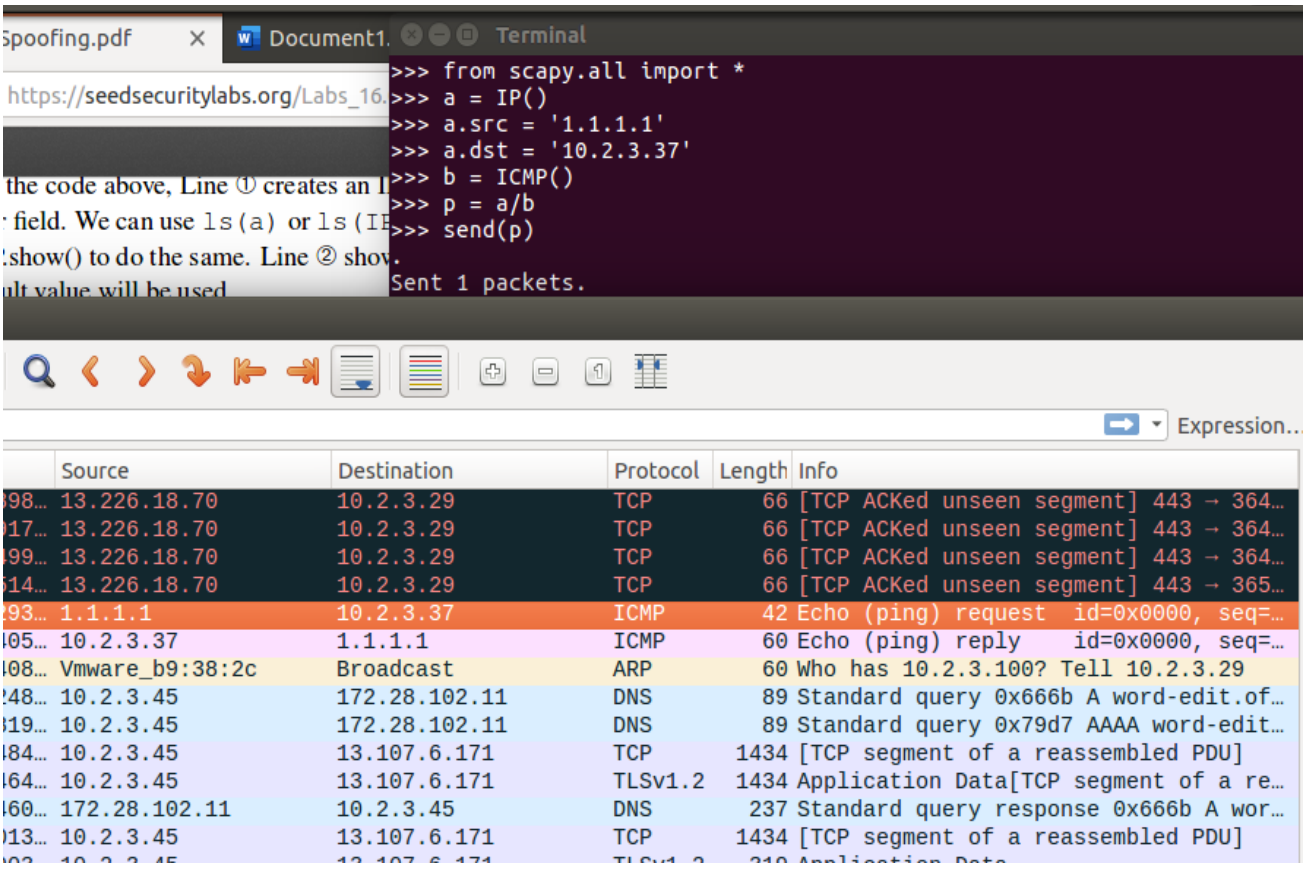
This shows that we can sniff packets on a specified port. Next, we are attempting to capture packets that comes from or to goes to a particular subnet. I will use 128.230.0.0/16. The program will need to be changed again, so I comment out the previous `pkt` = line and replace it with the following: `pkt = sniff(filter='net 128.230.0.0/16', prn=print_pkt)`. I then save the program and run it. After running the program, I open another terminal, and run the ping command with the subnet 128.230.0.0/16: `ping 128.230.0.0 16`. After running the command, I check back to the terminal running 'sniff.py' and see that packets have been received. This is the resulting output:

```
###[ Ethernet ]###
  dst       = 00:1d:71:f4:b0:00
  src       = 00:50:56:b9:a4:29
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 22809
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x537b
     src       = 10.2.3.45
     dst       = 128.230.0.0
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xc748
        id        = 0x1bef
```

```
        seq       = 0x3
###[ Raw ]###
           load      =
'\xd0\x15\x97_\xc1L\x01\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17
\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Task 1.2: Spoofing ICMP Packets

For this task, we spoof IP packets with an arbitrary source IP address. We will spoof ICMP echo request packets and send them to our VM under a spoofed source. We will use Wireshark to observe whether our request was accepted. I start by opening WireShark and begin monitoring traffic. I then use the code given in the lab to model my own program which creates an IP object, assigns it a source and destination IP, then creates a ICMP object and stacks the IP object onto the ICMP object. The new object is then sent. I typed my code and saved it in the /home/seed/labs/sniffing-spoofing directory as 'sniffspoof.py'. I made 'sniffspoof.py' an executable program and ran it with root privileges. Upon running the program, WireShark was able to capture the spoofed packet. I used the source '1.1.1.1' for this example. I will include a picture of the code and of the captured packet below:

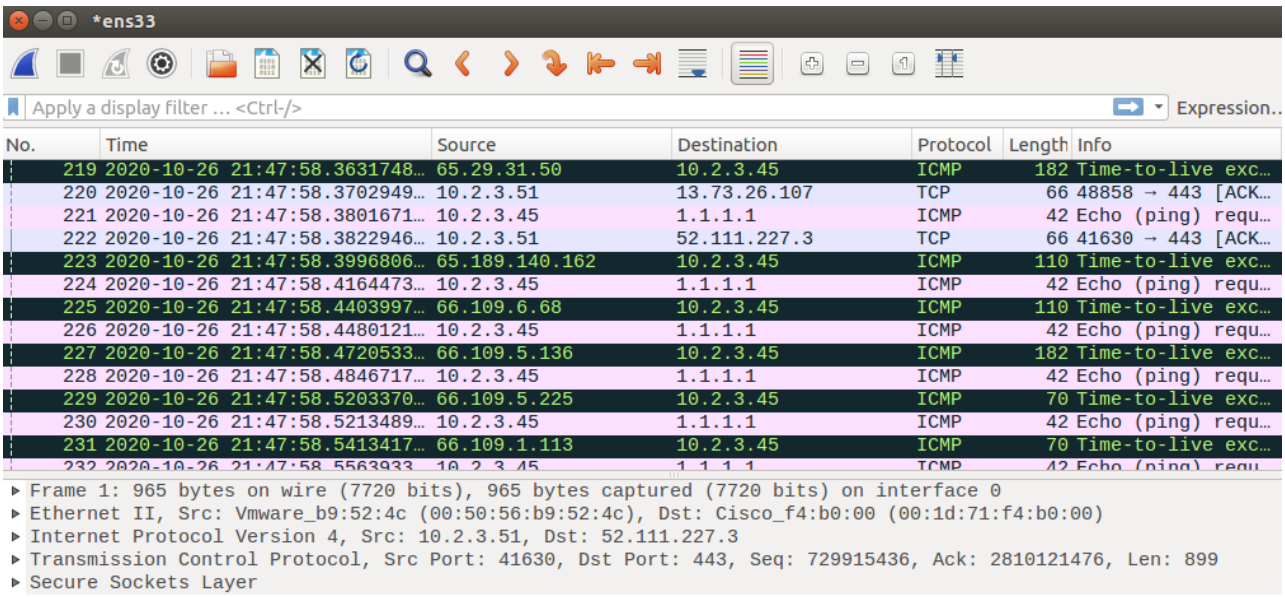


Task 1.3: Traceroute

In this task, we use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. This is basically what is implemented by the traceroute tool, but we will write our own tool. I start this by typing out the following code into a Python file called 'traceroute.py':

```
#!/usr/bin/python3
from scapy.all import *
for i in range(1,70):
    a = IP(dst='1.1.1.1',ttl=i)
    send(a/ICMP())
```

I then make the 'traceroute.py' file an executable file. I open WireShark and begin monitoring traffic. I then run the 'traceroute.py' program and keep track of the packets sent out and the IMCP errors received. I will include a screenshot of the results, but basically the program attempted to send out a packet to the destination '1.1.1.1' but the first couple of times an error message is received. These error messages reveal the IP of the first router in our route because this router drops the packet. The next error message reveals the second router, so on and so forth until the packet finally reaches its destination. The total IP's give an estimated length of the route in the chain but is only an estimate because not all packets follow the same path.



Task 1.4: Sniffing and-then Spoofing

In this task, we combine the sniffing and spoofing techniques used in the previous parts of this lab to make a program that sniffs for a certain type of packet, and then spoofs a reply from the destination address. We will use two VMs on the same LAN, the SEED and ATTACK VM's. From VM ATTACK, I start by pinging IP address '1.1.1.1'. This generates an ICMP echo request packet. If the IP address is alive, the ping program will receive an echo reply, and print out the response. The program I will write will run on the SEED VM and will monitor the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, the program immediately sends out an echo reply using the packet spoofing technique. Regardless of whether the IP address is alive or not, the ping program will always receive a reply, indicating that the IP is alive. This is the program I wrote in order to achieve this task (saved as sniffthespoof.py under the /home/seed/labs/sniffing-spoofing directory), and below will be a picture that includes the attempt to send a packet to a destination, followed by the spoofed response:

```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    a = IP()
    a.src = pkt[IP].dst
    a.dst = pkt[IP].src
    b = ICMP()
    b.type ="echo-reply"
    b.code = 0
    b.id = pkt[ICMP].id
    b.seq = pkt[ICMP].seq
    p = a/b
    send(p)
pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=print_pkt)
```