pkt= Ether()/IP()/UDP()/"Hello World!" #creating packet with layers

1. pkt.haslayer(UDP) #works with TCP and Raw

2. The sniffer program Wireshark is a free open-source network analyzer through capturing packets and displaying along with analyzing them for important detailed information. On the defensive, Wireshark is good for monitoring network traffic, troubleshooting issues, and detecting suspicious activities like unauthorized access or malware. Though on the offensive, Wireshark can be used for gathering information about a network and identify vulnerabilities if gained certain permissions.

   The sniffer program shown on page 392 shows the captured header packet using the .summary() to get the packet layers and the .show() function to get where it's going and where it came from which we can in fact see moving down in the IP information shown.

3. The spoofing program in 19.2 creates a fake source IP with a real destination. Next in 2 the program creates a spoofed ICMP request as it is a default echo request. It stacks the IP and ICMP objects creating the packet object and sends it out onto the network. Once the packet is sent, if we turn on Wireshark, we should be able to see the packet since scapy's send() function gives the packet to the kernel in a special socket called the raw socket.

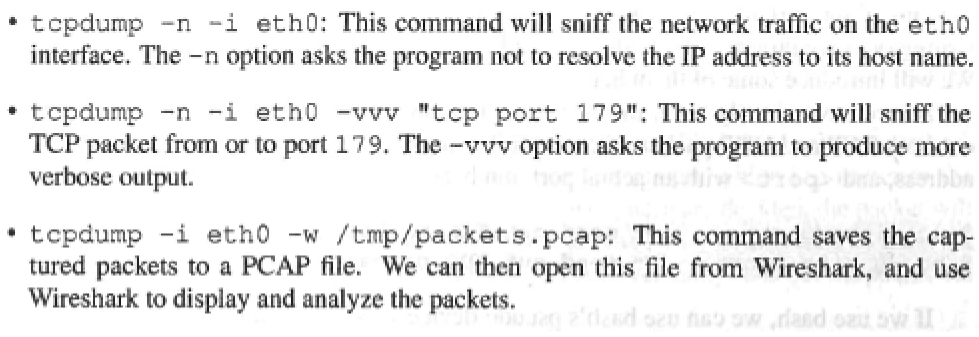
4. We use a YAML file called docker-compose.yml to configure our containers. Then, with a single command, we can create and start all the containers from the configuration.

5. The command "tail -f /dev/null" is a common practice to prevent the container or process from being terminated. This is often meant for testing or debugging.

Notes

- Telnet program sends out TCP packets telnet <ip>
  Ping program sends out ICMP packets ping <ip>

- Many attack and defense tools are built on top of packet sniffing and spoofing. Packet sniffer is a computer program or hardware that can log traffic passing over a computer network



- `tcpdump -n -i eth0`: This command will sniff the network traffic on the `eth0` interface. The `-n` option asks the program not to resolve the IP address to its host name.
- `tcpdump -n -i eth0 -vvv "tcp port 179"`: This command will sniff the TCP packet from or to port `179`. The `-vvv` option asks the program to produce more verbose output.
- `tcpdump -i eth0 -w /tmp/packets.pcap`: This command saves the captured packets to a PCAP file. We can then open this file from Wireshark, and use Wireshark to display and analyze the packets.

- Packet spoofing is when someone fakes the source address or details of a network packet to hide their identity or trick systems from critical information. I know it's often used in attacks like DDoS to overwhelm a target or to bypass security rules. To prevent it, you should use firewalls, filters, and other secure communication methods.

- Spoofing ICMP packets

```
#!/usr/bin/python3
from scapy.all import *
print(“Sending spoofed icmp packet..”)
ip = IP(stc=”1.2.3.4”, dst=”93.184.216.34”) # creates IP object
icmp = ICMP() #creates ICMP object
pkt = ip/icmp #stack 2 header objects to form a new object
pkt.show() #print out values of each header field
send(pkt,verbose=0) #send out packet and turn on Wireshark
```

- Spoofing UDP packets

```
#!/usr/bin/python3
from scapy.all import *
print(“Sending spoofed UDP packet..”)
```

```
ip = IP(src=”1.2.3.4”, dst=”10.0.2.69”) # creates IP object layer
Udp = UDP(sport=888, dport=9090) #UDP layer
data= “Hello UDP!\n” #payload
pkt = ip/udp/data #construct the complete packet
pkt.show() #print out values of each header field
send(pkt,verbose=0) #send out packet and turn on Wireshark
```

We create a UDP packet with an IP header, UDP header, and a "Hello UDP!" payload, then send it using send(). Run nc -lnuv 9090 on 10.0.2.69 to see the message.

- Sniffing and then spoofing

```
#!/usr/bin/python3
from scapy.all import *
def spoof_pkt(pkt):
        If ICMP in pkt and pkt[ICMP].type == 8:
                print(“Original Packet...”)
                print(“Source IP: ”, pkt[IP].src)
                print(“Destination IP: ”, pkt[ip].dst)

                ip = IP(src=pkt[IP].dst, dst = pkt[IP].src, ihl = pkt[IP].ihl)
                icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
                data= pkt[Raw].load
                Newpkt = ip/icmp/data

                print(“Original Packet...”)
                print(“Source IP: ”, newpkt[IP].src)
                print(“Destination IP: ”, newpkt[ip].dst)
                send(newpkt, verbose= 0)
pkt= sniff(filter=’icmp and src host 10.0.2.69’, prn = spoof_pkt)
```

- send(): Sends packets at Layer 3.
- sendp(): Sends packets at Layer 2.
- sr(): Sends packets at Layer 3 and receives answers.
- srp(): Sends packets at Layer 2 and receives answers.
- sr1(): Sends packets at Layer 3 and waits for the first answer.
- srp1(): Sends packets at Layer 2 and waits for the first answer.

- Privileged Mode: Allows a container to access all devices on the host and perform operations that require elevated privileges, like modifying system settings. It gives the container almost the same access as the host machine.

  Host Mode: Configures the container to use the host’s network stack directly, meaning it shares the host's IP address and network interfaces, without any isolation.

  Network Mode: Defines how containers communicate with each other and the outside world, such as using bridge, host, or overlay networks, to manage network connectivity.

  Shared Folder: A directory that is shared between the host and the container, allowing files to be accessed or modified by both the host system and the container.

- When we run a sniffer program inside a container, we can only sniff the packets going in and out of this container. But if a container uses the host mode in its network setup, it can sniff other containers packets