Lab5 Scapy

Lab Learning Objectives

- Use Tcpdump to sniff and analyze packets
- Use Scapy to craft packets in scenarios that are highly useful in penetration testing
- Analyze packets captured for the traceroute command

Lab Setup

In this lab, you will use Kali Linux, Ubuntu Linux, Metasploitable 2 Linux and Windows 10.

For the first part, you'll use Scapy to send a packet and use Tcpdump to sniff. Then you'll learn how to use Scapy for various cases. At the end of this lab, you'll write simple python script using Scapy.

Lab Instructions

1. packets. Bring up terminal at Kali Linux and switch the user from kali to root.

$ sudo su -

Run Tcpdump. We will use it to sniff the packets we send and receive.

 # tcpdump -nn host *Kali IP-Address* and host *Ubuntu IP_Address*

Bring up another terminal at Kali Linux Machine. We will construct a packet and send it to the Ubuntu Linux. Start the Scapy by typing

 # scapy

If you have error, "frozen importlib._bootstrap…," try reinstalling matplotlib using the website.

(https://matplotlib.org/stable/users/installing/index.html)

: git clone git@github.com:matplotlib/matplotlib.git and "python -m pip install ." at installed directory. In case you have another error, please solve the installation error on your own.

Next, let's construct a packet using the default settings of TCP and a payload of "Hi Ubuntu!!" (**Don't' change this payload**). After that, send the packet.

>>> packet= IP(dst="*Ubuntu IP-Address*")/TCP()/"Hi Ubuntu!!"

>>> sr(packet)

2. Let's first examine the output from the Tcpdump. Form the output, we can find out that our Kali Linux

```
┌──(root㉿kali)-[~]
└─# tcpdump -nn host 192.168.84.130 and host 192.168.84.131
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:54:16.984200 ARP, Request who-has 192.168.84.131 tell 192.168.84.130, length 28
16:54:16.985008 ARP, Reply 192.168.84.131 is-at 00:0c:29:1d:8d:f4, length 46
16:54:17.011174 IP 192.168.84.130.20 > 192.168.84.131.80: Flags [S], seq 0:5, win 8192, length 5: HTTP
16:54:17.011834 IP 192.168.84.131.80 > 192.168.84.130.20: Flags [S.], seq 2047220368, ack 1, win 5840, o
ptions [mss 1460], length 0
16:54:17.011894 ARP, Request who-has 192.168.84.131 tell 192.168.84.130, length 28
16:54:17.012625 ARP, Reply 192.168.84.131 is-at 00:0c:29:1d:8d:f4, length 46
16:54:17.012638 IP 192.168.84.130.20 > 192.168.84.131.80: Flags [R], seq 1, win 0, length 0
^C
7 packets captured
7 packets received by filter
0 packets dropped by kernel
```

machine broadcasted an ARP request to find out the MAC address (00:0C:29:1D:8D:F4) of the Ubuntu Linux machine. Next, The Kali Linux machine sent a TCP SYN packet to the Ubuntu Linux machine at port 80. The Ubuntu Linux machine sent us a response. From the response, we can tell whether port 80 is open or closed. After receiving the response from Ubuntu, the Kali Linux machine sent a TCP RST packet to the Ubuntu Linux machine. Because sr() only scans port, Kali Linux machine sent a RST to abort the connection effort instead of sending an ACK to finish the three way handshake.

3. Let's review the packets we received. From the output, it turned out that we received a total of 2 packets (initially) of which one packet is the response from the Ubuntu Linux machine. What is the other packet? If you look at the Tcpdump output carefully in step 2, we received extra ARP reply packet.

We can get rid of those unwanted ARP packets by modifying the Tcmdump filter by running

 # tcpdump -nn host *Kali IP-Address* and host *Ubuntu IP_Address* and not arp

```
>>> packet= IP(dst="192.168.84.131")/TCP()/"Hello"
>>> sr(packet)
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
(<Results: TCP:1 UDP:0 ICMP:0 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> ans,unans=sr(packet)
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> ans[0]
QueryAnswer(query=<IP  frag=0 proto=tcp dst=192.168.84.131 |<TCP  |<Raw  load='Hello' |>>>, answer=<IP
version=4 ihl=5 tos=0x0 len=44 id=0 flags=DF frag=0 ttl=64 proto=tcp chksum=0x1076 src=192.168.84.131 ds
t=192.168.84.130 |<TCP  sport=http dport=ftp_data seq=4181113733 ack=1 dataofs=6 reserved=0 flags=SA win
dow=5840 chksum=0xa1ce urgptr=0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>)
>>> ans[0][0]
<IP  frag=0 proto=tcp dst=192.168.84.131 |<TCP  |<Raw  load='Hello' |>>>
>>>
```

Let's examine the received packet in more details. From the output, we can see that the response is stored in a parenthesis with two parts (**Results** and **Unanswered**) which are separated by a comma. It is clear that we received one TCP packet from the Results part and there is no unanswered packets. Let's dive in to review the detailed information stored in the received packet.

 # ans,unans =_

Ans stores all packets from the Results and unans stores all packets from the Unanswered part. Let's look at the first packet stored in the Results part by running

 # ans[0]

It displays the packet Kali sent to Ubuntu as well as the packet received from Ubuntu, separated by a comma. If you just want to display the packet Kali sent to Ubuntu, type

 # ans[0][0]

If we just want to display the packet received from Ubuntu, we should type ans[0][1]. Obviously, there is plenty of information we can look at a fine grained level.

4. We will simulate the ping command next. Press CTRL-C in the terminal running Tcpdump to stop it. Run the following command

 # tcpdump -nnX icmp

(-nn: Don't convert protocol and port numbers etc. to names either / -X: print the data of each packet (minus its link level header) in hex and ASCII)

Next we will ping our Windows 10 machine by using Scapy. In the terminal running Scapy, type (**don't change the message**),

>>> packet=IP(dst="Windows 10 IP_Address")/ICMP()/"Hi Windows!"

>>> response=srloop(packet, count=3)

```
>>> packet=IP(dst="192.168.84.181")/ICMP()/"Hi Windows!"
>>> response=srloop(packet, count=3)
RECV 1:
RECV 1: IP / ICMP 192.168.84.181 > 192.168.84.157 echo-reply 0 / Raw / Padding
RECV 1:

Sent 3 packets, received 3 packets. 100.0% hits.
>>>
```

5. Let's examine the output from the Tcpdump. We can see that the Kali Linux machine sent three ICMP echo requ st packets and received three ICMP echo reply packets. This indicates that the Windows 10 machine is online. How do we know Scapy sent out the ICMP echo request instead of other ICMP type? Run

>>> ls(ICMP)

```
>>> ls(ICMP)
type        : ByteEnumField                      = ('8')
code        : MultiEnumField (Depends on 8)      = ('0')
chksum      : XShortField                        = ('None')
id          : XShortField (Cond)                 = ('0')
seq         : XShortField (Cond)                 = ('0')
ts_ori      : ICMPTimeStampField (Cond)          = ('23629586')
ts_rx       : ICMPTimeStampField (Cond)          = ('23629586')
ts_tx       : ICMPTimeStampField (Cond)          = ('23629586')
gw          : IPField (Cond)                     = ("'0.0.0.0'")
ptr         : ByteField (Cond)                   = ('0')
reserved    : ByteField (Cond)                   = ('0')
length      : ByteField (Cond)                   = ('0')
addr_mask   : IPField (Cond)                     = ("'0.0.0.0'")
nexthopmtu  : ShortField (Cond)                  = ('0')
unused      : MultipleTypeField (ShortField, IntField, StrFixedLenField) = ("b''")
>>>
```

We can see that ICMP uses a default type of 8 which is the echo request.

In addition, we can see that the payload “Hi Windows” was sent from our Kali Linux machine to the target Windows 10. We also see that the ping response (from Windows 10 to our Kali Linux machine) includes the Hello string coming back. ICMP Echo Request is indeed an echo.

```
02:04:25.085503 IP 192.168.84.181 > 192.168.84.157: ICMP echo reply, id 0, seq 0, length 19
        0x0000:  4500 0027 e24a 0000 8001 2de8 c0a8 54b5  E..'.J....-...T.
        0x0010:  c0a8 549d 0000 30ee 0000 0000 4869 2057  ..T...0.....Hi.W
        0x0020:  696e 646f 7773 2100 0000 0000 0000       indows!.......
02:04:26.088982 IP 192.168.84.157 > 192.168.84.181: ICMP echo request, id 0, seq 0, length 19
        0x0000:  4500 0027 0001 0000 4001 5032 c0a8 549d  E..'....@.P2..T.
        0x0010:  c0a8 54b5 0800 28ee 0000 0000 4869 2057  ..T...(.....Hi.W
        0x0020:  696e 646f 7773 21                        indows!
02:04:26.090276 IP 192.168.84.181 > 192.168.84.157: ICMP echo reply, id 0, seq 0, length 19
        0x0000:  4500 0027 e24b 0000 8001 2de7 c0a8 54b5  E..'.K....-...T.
        0x0010:  c0a8 549d 0000 30ee 0000 0000 4869 2057  ..T...0.....Hi.W
        0x0020:  696e 646f 7773 2100 0000 0000 0000       indows!.......
```

6. If you’d like to dive into more details about the received packets, type

>>> response[0]

>>> response[0][0]

>>> response[0][0][0]

7. Next, we will use Scapy to simulate a Denial of Service called Land Attack (old-school attack). If you send a machine a spoofed TCP SYN packet to an open port with the source IP address set to the same value as the destination IP address and with the source port the same value as the destination port, the target system’s CPU would spin up to 100% and in some cases even crash. Press CTRL-C in the terminal running Tcpdump to stop it. Run the following command

 # tcpdump -nn tcp and host Windows 10 IP_Address

Move to the terminal running Scapy and type

>>> packet=IP(src="Windows 10 IP", dst="Windows 10 IP")/TCP(sport=445,dport=445)

>>> send(packet,count=3)

From the Tcpdump output, we can see three land-style attack packets.

```
┌──(root㉿kali)-[~]
└─# tcpdump -nn tcp and host 192.168.84.141
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
18:06:21.328648 IP 192.168.84.141.445 > 192.168.84.141.445: Flags [S], seq 0, win 8192, length 0
18:06:21.330495 IP 192.168.84.141.445 > 192.168.84.141.445: Flags [S], seq 0, win 8192, length 0
18:06:21.332144 IP 192.168.84.141.445 > 192.168.84.141.445: Flags [S], seq 0, win 8192, length 0
```

8. Next, we will use Scapy as a **port scanner** to scan our Ubuntu Linux machine. Press CTRL-C in the terminal running Tcpdump to stop it. Run the following command

 # tcpdump -nn tcp and host *Ubuntu IP_Address*

Move to the terminal running Scapy and type (sending packets from port 20 to port 30)

>>> packet=IP(dst="*Ubuntu IP_Address*")/TCP(dport=(20,30))

>>> sr(packet)

```
>>> packet=IP(dst="192.168.84.131")/TCP(dport=(20,30))
>>> sr(packet)
Begin emission:
Finished sending 11 packets.
.**********
Received 12 packets, got 11 answers, remaining 0 packets
(<Results: TCP:11 UDP:0 ICMP:0 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>>
```

By looking at the output at the Tcpdump terminal, can you tell which ports are open and which ports are closed? (find out responses, [R.] or [S.])

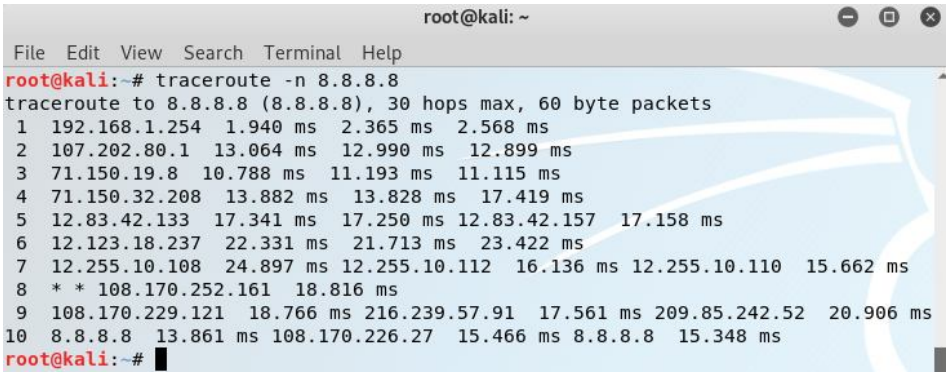
```
┌──(root㉿kali)-[~]
└─# tcpdump -nn tcp and host 192.168.84.131
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
18:10:25.615265 IP 192.168.84.130.20 > 192.168.84.131.20: Flags [S], seq 0, win 8192, length 0
18:10:25.616391 IP 192.168.84.131.20 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.617698 IP 192.168.84.130.20 > 192.168.84.131.21: Flags [S], seq 0, win 8192, length 0
18:10:25.618917 IP 192.168.84.131.21 > 192.168.84.130.20: Flags [S.], seq 663931827, ack 1, win 5840, options [mss 1460], length 0
18:10:25.618961 IP 192.168.84.130.20 > 192.168.84.131.21: Flags [R], seq 1, win 0, length 0
18:10:25.620140 IP 192.168.84.130.20 > 192.168.84.131.22: Flags [S], seq 0, win 8192, length 0
18:10:25.621227 IP 192.168.84.131.22 > 192.168.84.130.20: Flags [S.], seq 670902193, ack 1, win 5840, options [mss 1460], length 0
18:10:25.621262 IP 192.168.84.130.20 > 192.168.84.131.22: Flags [R], seq 1, win 0, length 0
18:10:25.622445 IP 192.168.84.130.20 > 192.168.84.131.23: Flags [S], seq 0, win 8192, length 0
18:10:25.623478 IP 192.168.84.131.23 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.624604 IP 192.168.84.130.20 > 192.168.84.131.24: Flags [S], seq 0, win 8192, length 0
18:10:25.625736 IP 192.168.84.131.24 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.626668 IP 192.168.84.130.20 > 192.168.84.131.25: Flags [S], seq 0, win 8192, length 0
18:10:25.627757 IP 192.168.84.131.25 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.628628 IP 192.168.84.130.20 > 192.168.84.131.26: Flags [S], seq 0, win 8192, length 0
18:10:25.629685 IP 192.168.84.131.26 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.630711 IP 192.168.84.130.20 > 192.168.84.131.27: Flags [S], seq 0, win 8192, length 0
18:10:25.631772 IP 192.168.84.131.27 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.632668 IP 192.168.84.130.20 > 192.168.84.131.28: Flags [S], seq 0, win 8192, length 0
18:10:25.633686 IP 192.168.84.131.28 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.635475 IP 192.168.84.130.20 > 192.168.84.131.29: Flags [S], seq 0, win 8192, length 0
18:10:25.636522 IP 192.168.84.131.29 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
18:10:25.637503 IP 192.168.84.130.20 > 192.168.84.131.30: Flags [S], seq 0, win 8192, length 0
18:10:25.638461 IP 192.168.84.131.30 > 192.168.84.130.20: Flags [R.], seq 0, ack 1, win 0, length 0
```

9. Next, we will use Scapy to **examine some captured packets**. Press CTRL-C in the terminal running Tcpdump to stop it. Run the following command and hit enter

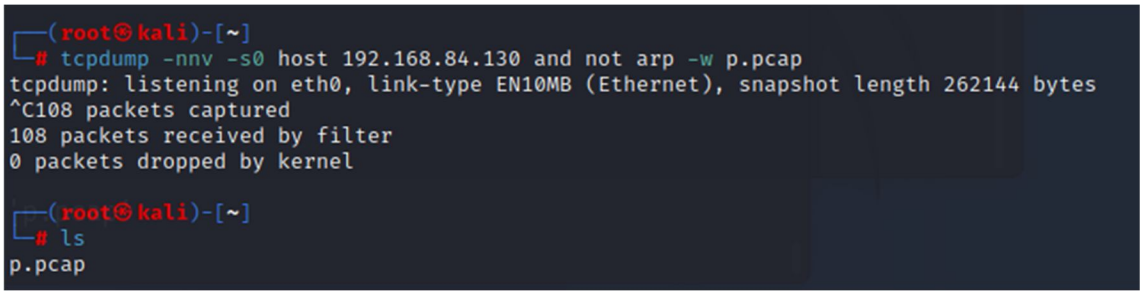
 # tcpdump -nnv -s0 host *Kali IP_Address* and not arp -w p.pcap

Bring up another terminal at Kali Linux machine. We will do a traceroute to Google's name server at 8.8.8.8.

 # traceroute -n 8.8.8.8 (*you might get only *s => that's ok)



At the terminal running Tcpdump, it says Tcpdump has sniffed some packets. Press CTRL-C to write the packets to the packet capture file p.pcap.



Now move to the terminal running Scapy. We will read the pcap file we just sniffed.

>>> packets=rdpcap("p.pcap") (*in case you cannot read the file, restart scapy from root)

>>> packets[0]

We can see that the transport layer protocol used by the traceroute command is tcp. Can you see the initial TTL value? What is the initial destination port number?

10. Finally, we will use scapy's sniff() function to implement a simple sniffer. First review the sniff function's syntax by typing

>>> help(sniff) (*if you want to end help, press 'q')

We will use the **filter** and **prn** arguments provided by the sniff function. The **filter** argument uses the Berkeley Packet Filter (BPF) syntax which is the same as the one used by tcpdump. The **prn** argument specifies a callback function which will be invoked when scapy sniffs the packets. Next, let's write a

python script to sniff and print the captured ICMP packets. On your Kali Linux, change the directory to /tmp by typing

 # cd /tmp

Then use gedit to create and open a file sniff.py

$ sudo gedit sniff.py

Type the following code to the file opened in gedit, make sure to make the correct indentation inside the print_pkt function definition.

```
#!/usr/bin/python3

from scapy.all import *

print("SNIFFING PACKETS.........")
def print_pkt(pkt):
   print("Source IP:", pkt[IP].src)
   print("Destination IP:", pkt[IP].dst)
   print("Protocol:", pkt[IP].proto)
   print("\n")

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

Run the program by typing

 # python3 sniff.py

Now switch to the Windows 10 machine and bring up a terminal. Ping the Google DNS server 8.8.8.8 by typing

C:\> ping 8.8.8.8

From your Kali Linux machine terminal, you should find that your little sniffer has captured the ICMP echo request packets sent by the Windows 10 machine as well as the ICMP echo response packets from 8.8.8.8.

Lab Report

- please include your name and 700# at the beginning of your report
- please upload your report to the Blackboard by the due date
- You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed
- only word or pdf format is acceptable
- you must show all the necessary commands associated with each task in order to receive credits
- your screenshots size must be appropriate to provide the visible details

Provide a report which includes the following items.

1. Write a python script to spoof and send a UDP packet with a payload of string “Hello World”. Set the source IP address to whatever the IP address you like. The destination IP address to be your Ubuntu’s IP address. You can also choose whatever the source and destination ports you’d like to use. Run the python script on your Kali Linux. On the Ubuntu Linux, set up a server by using
   # nc -luvp 1234
   Here we assume that the destination port you used in your script is 1234. Provide the screen shot showing the message displayed on the Ubuntu server. If we run the sever using nc -lvp 1234. What is the result? Explain the reason why you have the new result? Please provide screenshots showing your **python script** as well as the **Ubuntu terminal** showing the message you received.

   First 2 lines of the python code should be these;

   #!usr/bin/env python
   from scapy.all import *

2. Use Scapy to construct packets to scan Metasploitable Linux TCP ports 10-100 (don’t change port range). Provide screenshots showing the commands which you use to construct, send and receive the packets. List all the open ports. Please explain how do you determine a port is open based on the packets received from Metasploitable 2 Linux.