Parocyber x Cisco Ethical Hacking Course Module 3 Lab Assignment

Background

Packet crafting plays a vital role in bypassing IDS, IPS, and firewall controls because these systems are configured to allow only specific types of traffic. In this lab, the goal is to understand the fundamentals of creating custom packets using a tool called **Scapy**.

About Scapy

Scapy is a Python-based program used to send, sniff, dissect, and forge network packets. Its capabilities allow users to build tools that can probe, scan, or attack networks.

In simple terms, Scapy is a powerful interactive packet manipulation framework. It can craft or decode packets across many protocols, transmit them over the network, capture traffic, analyze responses, and automate many networking tasks. Scapy can perform activities such as scanning, tracerouting, probing, testing, attacks, and network discovery. It can also replace utilities such as hping, arpspoof, arp-sk, arping, p0f, and even parts of Nmap, tcpdump, and tshark.

Lab — Packet Crafting with Scapy

Objectives

In this lab, you will use Scapy to create custom network packets for reconnaissance on a target system. The tasks covered include:

- Part 1: Investigate the Scapy Tool
- Part 2: Use Scapy to Sniff Network Traffic
- Part 3: Create and Send an ICMP Packet
- Part 4: Create and Send TCP SYN Packets

Scenario

Penetration testers and ethical hackers frequently craft packets to discover or exploit vulnerabilities in client environments. Previously, Nmap was used to scan and analyze a device on the local network. In this lab, you will extend your reconnaissance by crafting and sending custom ICMP and TCP packets to the same system.

Required Resources

- Kali Linux VM
- Internet access

Part 1: Investigate the Scapy Tool

Scapy was originally written by Philippe Biondi and functions as a multi-purpose interactive packet manipulation tool. In this section, you will load Scapy and explore its features. Remember: tools like Scapy should only be used on systems you own or have explicit permission to test.

Step 1: Review Scapy Documentation

Scapy can be used interactively from the Python shell or imported into Python scripts using the `python-scapy` module. Its documentation is available at:
https://scapy.readthedocs.io/en/latest/introduction.html

Steps:

1. Start the Kali VM and log in.
2. Open a web browser and review the Introduction page from the official Scapy documentation.

Step 2: Use Scapy in Interactive Mode

1. Packet crafting and sending require root privileges. Obtain them using:

```
sudo su
```

2. Start Scapy by entering:

```
scapy
```

3. At the Scapy prompt (>>>), use:

```
ls()
```

to list the available protocols and formats. The list is extensive and spans multiple screens.

You can inspect modifiable fields for any protocol by using:

```
ls(<protocol>)
```

Step 3: Examine IPv4 Packet Header Fields

Before creating packets, it is essential to understand the structure of an IPv4 header. Important fields include:

- **Version** — Identifies the packet as IPv4.
- **Differentiated Services (DS)** — Formerly ToS; sets packet priority.
- **TTL (Time to Live)** — Limits the packet's lifetime across routers.
- **Protocol** — Indicates the upper-layer protocol (ICMP = 1, TCP = 6, UDP = 17).
- **Header Checksum** — Detects corruption in the header.

- **Source IPv4 Address** — 32-bit source address.
- **Destination IPv4 Address** — 32-bit destination address.

Part 2: Use Scapy to Sniff Network Traffic

Scapy can capture and display network traffic similar to tcpdump or tshark.

Step 1: Use the `sniff()` Function

1. Start a capture by typing:

```
sniff()
```

2. Open a second terminal and run:

```
ping -c 5 www.cisco.com
```

3. Return to the Scapy window and press **CTRL+C**.
   You should see something like:

```
<Sniffed: TCP:525 UDP:42 ICMP:10 Other:845>
```

4. View the summary of captured packets:

```
a = _
a.summary()
```

To filter and limit captured packets:

```
sniff(iface="br-internal", filter="icmp", count=10)
```

After capture, use:

```
a = _
a.nsummary()
```

to view numbered packet summaries.

To inspect a specific packet:

```
a[2]
```

To save captures to a `.pcap` file:

```
wrpcap("capture1.pcap", a)
```

Part 3: Create and Send an ICMP Packet

ICMP is used for control and diagnostic messages. Two of the most common types are echo-request and echo-reply.

Step 1: Craft and Send an ICMP Packet

1. Start a packet capture:

```
sniff(iface="eth0")
```

2. In another terminal, switch to root and open Scapy:

```
sudo su
scapy
```

3. Send a custom ICMP packet with a payload:

```
send(IP(dst="IP")/ICMP()/"This is a test")
```

4. Return to the sniffing terminal and press **CTRL+C**.
   You should see an ICMP packet captured.

Part 4: Create and Send a TCP SYN Packet

Step 1: Capture and Send SYN Packets

1. Start a packet capture on the internal interface.
2. In another terminal, create and send a TCP SYN packet:

```
send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
```

Scapy sends one TCP SYN packet to port **445** of the target host.

Conclusion

This lab provided an introductory, hands-on experience with packet crafting using Scapy. Scapy is a powerful and flexible tool, and building a strong foundation in its capabilities is essential to leveraging its full potential in penetration testing, network research, and security assessments. Hopefully, this exercise has strengthened your understanding of packet structures, network reconnaissance, and custom packet generation.

Appendix

```
>>> ls()
AD_AND_OR  : None
AD_KDCIssued : None
AH         : AH
AKMSuite   : AKM suite
ARP        : ARP
ASN1P_INTEGER : None
ASN1P_OID  : None
ASN1P_PRIVSEQ : None
ASN1_Packet : None
ATT_Error_Response : Error Response
ATT_Exchange_MTU_Request : Exchange MTU Request
ATT_Exchange_MTU_Response : Exchange MTU Response
ATT_Execute_Write_Request : Execute Write Request
ATT_Execute_Write_Response : Execute Write Response
ATT_Find_By_Type_Value_Request : Find By Type Value Request
ATT_Find_By_Type_Value_Response : Find By Type Value Response
ATT_Find_Information_Request : Find Information Request
ATT_Find_Information_Response : Find Information Response
ATT_Handle : ATT Short Handle
ATT_Handle_UUID128 : ATT Handle (UUID 128)
ATT_Handle_Value_Indication : Handle Value Indication
ATT_Handle_Value_Notification : Handle Value Notification
ATT_Handle_Variable : None
ATT_Hdr    : ATT header
ATT_Prepare_Write_Request : Prepare Write Request
ATT_Prepare_Write_Response : Prepare Write Response
```

```
TIP: You may use explore() to navigate through all layers using a clear GUI
>>> ls(TCP)
sport      : ShortEnumField                = ('20')
dport      : ShortEnumField                = ('80')
seq        : IntField                      = ('0')
ack        : IntField                      = ('0')
dataofs    : BitField  (4 bits)            = ('None')
reserved   : BitField  (3 bits)            = ('0')
flags      : FlagsField                    = ('<Flag 2 (S)>')
window     : ShortField                    = ('8192')
chksum     : XShortField                   = ('None')
urgptr     : ShortField                    = ('0')
options    : TCPOptionsField               = ("b''")
>>>
```

```
>>> ls(DNS)
length     : ShortField (Cond)             = ('None')
id         : ShortField                    = ('0')
qr         : BitField  (1 bit)             = ('0')
opcode     : BitEnumField                  = ('0')
aa         : BitField  (1 bit)             = ('0')
tc         : BitField  (1 bit)             = ('0')
rd         : BitField  (1 bit)             = ('1')
ra         : BitField  (1 bit)             = ('0')
z          : BitField  (1 bit)             = ('0')
ad         : BitField  (1 bit)             = ('0')
cd         : BitField  (1 bit)             = ('0')
rcode      : BitEnumField                  = ('0')
qdcount    : FieldLenField                 = ('None')
ancount    : FieldLenField                 = ('None')
nscount    : FieldLenField                 = ('None')
arcount    : FieldLenField                 = ('None')
qd         : _DNSPacketListField           = ('[\x1b[0m<\x1b[0m\x1b[31m\x1b[1mDNSQR\x1b[0m  \x1b[0m|\x1b[0m\x1b[0m>\x1b[0m]')
an         : _DNSPacketListField           = ('[]')
ns         : _DNSPacketListField           = ('[]')
ar         : _DNSPacketListField           = ('[]')
>>>
```

```
>>> explore("dns")
Packets contained in scapy.layers.dns:
Class                 |Name
----------------------+----------------------------------
DNS                   |DNS
DNSCompressedPacket   |
DNSQR                 |DNS Question Record
DNSRR                 |DNS Resource Record
DNSRRDLV              |DNS DLV Resource Record
DNSRRDNSKEY           |DNS DNSKEY Resource Record
DNSRRDS               |DNS DS Resource Record
DNSRRHINFO            |DNS HINFO Resource Record
DNSRRHTTPS            |DNS HTTPS Resource Record
DNSRRMX               |DNS MX Resource Record
DNSRRNAPTR            |DNS NAPTR Resource Record
DNSRRNSEC             |DNS NSEC Resource Record
DNSRRNSEC3            |DNS NSEC3 Resource Record
DNSRRNSEC3PARAM       |DNS NSEC3PARAM Resource Record
DNSRROPT              |DNS OPT Resource Record
```

```
>>> sniff()
```

```
┌──(kali㉿kali)-[~]
└─$ ping -c 5 www.cisco.com
PING e2867.dsca.akamaiedge.net (23.208.168.98) 56(84) bytes of data.
64 bytes from a23-208-168-98.deploy.static.akamaitechnologies.com (23.208.168.98): icmp_seq=1 ttl=49 time=94.5 ms
64 bytes from a23-208-168-98.deploy.static.akamaitechnologies.com (23.208.168.98): icmp_seq=2 ttl=49 time=94.5 ms
64 bytes from a23-208-168-98.deploy.static.akamaitechnologies.com (23.208.168.98): icmp_seq=3 ttl=49 time=95.2 ms
64 bytes from a23-208-168-98.deploy.static.akamaitechnologies.com (23.208.168.98): icmp_seq=4 ttl=49 time=95.6 ms
64 bytes from a23-208-168-98.deploy.static.akamaitechnologies.com (23.208.168.98): icmp_seq=5 ttl=49 time=95.0 ms

--- e2867.dsca.akamaiedge.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 94.468/94.959/95.596/0.424 ms
```

```
>>> sniff()
^C<Sniffed: TCP:524 UDP:42 ICMP:10 Other:845>
>>>
```

```
>>> a=_
>>> a.summary
<bound method _PacketList.summary of <Sniffed: TCP:524 UDP:42 ICMP:10 Other:845>>
>>>
```

```
┌──(kali㉿kali)-[~]
└─$ ping -c 10  www.google.com
PING www.google.com (192.178.24.132) 56(84) bytes of data.
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=1 ttl=113 time=34.4 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=2 ttl=113 time=34.3 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=3 ttl=113 time=35.5 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=4 ttl=113 time=35.7 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=5 ttl=113 time=34.4 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=6 ttl=113 time=34.1 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=7 ttl=113 time=35.2 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=8 ttl=113 time=34.8 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=9 ttl=113 time=35.0 ms
64 bytes from mct04s01-in-f4.1e100.net (192.178.24.132): icmp_seq=10 ttl=113 time=34.6 ms

--- www.google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9018ms
rtt min/avg/max/mdev = 34.106/34.794/35.657/0.505 ms
```

```
>>> sniff(iface="eth0", filter="tcp", count=10)
... :
<Sniffed: TCP:10 UDP:0 ICMP:0 Other:0>
>>>
```

```
>>> a=_
>>> a.nsummary()
0000 Ether / IP / TCP 192.168.100.7:56041 > 103.86.38.67:https FA / Padding
0001 Ether / IP / TCP 192.168.100.7:56042 > 103.86.38.67:https FA / Padding
0002 Ether / IP / TCP 103.86.38.67:https > 192.168.100.7:56041 PA / Raw
0003 Ether / IP / TCP 192.168.100.7:56041 > 103.86.38.67:https RA / Padding
0004 Ether / IP / TCP 103.86.38.67:https > 192.168.100.7:56041 FA / Padding
0005 Ether / IP / TCP 103.86.38.67:https > 192.168.100.7:56042 PA / Raw
0006 Ether / IP / TCP 192.168.100.7:56042 > 103.86.38.67:https RA / Padding
0007 Ether / IP / TCP 103.86.38.67:https > 192.168.100.7:56042 FA / Padding
0008 Ether / IP / TCP 192.168.100.7:55449 > 170.72.239.201:https PA / Raw
0009 Ether / IP / TCP 170.72.239.201:https > 192.168.100.7:55449 PA / Raw
>>>
```

```
>>> a[2]
<Ether  dst=a4:bb:6d:16:03:c2 src=18:dd:e7:42:12:0a type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=64 id=57064 flags=DF
 frag=0 ttl=59 proto=tcp chksum=0xae86 src=103.86.38.67 dst=192.168.100.7 |<TCP  sport=https dport=56041 seq=4015241
125 ack=3277315144 dataofs=5 reserved=0 flags=PA window=501 chksum=0x80ff urgptr=0 |<Raw  load=b"\x17\x03\x03\x00\x1
3\x03\xd8\x90\x95\xa4\x0b\xe6\x828\xa6\x1f\xcdp\xa2/'\x96\xf0\x87" |>>>>
>>>
```

```
>>> wrpcap("capture1.pcap",a)
>>>
```

```
┌──(kali㉿kali)-[~]
└─$ ls *.pcap
capture.pcap
```

```
>>> sniff(iface="eth0")
^C<Sniffed: TCP:13 UDP:2 ICMP:1 Other:38>
```

```
                                    using IPython 8.30.0
>>> send(IP(dst="192.168.100.69")/ICMP()/"This is a test")
WARNING: MAC address to reach destination not found. Using broadcast.
.
```

```
>>> a[29]
<Ether  dst=ff:ff:ff:ff:ff:ff src=00:0c:29:af:2b:0a type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=42 id=1 flags= frag=
0 ttl=64 proto=icmp chksum=0x30f7 src=192.168.100.69 dst=192.168.100.69 |<ICMP  type=echo-request code=0 chksum=0x5d
a0 id=0x0 seq=0x0 unused=b'' |<Raw  load=b'This is a test' |>>>>
>>>
```

```
.
Sent 1 packets.
>>> send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
.
Sent 1 packets.
```

```
>>> sniff()
^C<Sniffed: TCP:4 UDP:5 ICMP:0 Other:21>
>>>
```

```
>>> send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
.
Sent 1 packets.
```

```
>>> sniff()
^C<Sniffed: TCP:1 UDP:6 ICMP:0 Other:23>
>>>
```