Packet Sniffing and Spoofing Lab

Setup

```
[12/04/21]seed@VM:~/.../Labsetup$ dockps
e930c059475f  hostB-10.9.0.6
886de03a8437  seed-attacker
9bbe68622d33  hostA-10.9.0.5
```

In this snippet, we display all containers, including the hosts and the attacker.

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ 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   \
```

In this snippet, we verify we can use Scapy.

Task 1.1A

```
[12/04/21]seed@VM:~/.../Labsetup$ cat sniffer.py
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(iface="br-3af7b9237c20", filter="icmp", prn=print_pkt)
```

In the snippet above, we see the sniffing program we wrote in Python. This program would allow us to sniff ICMP packets on the br-3af7b9237c20 interface.

```
[12/04/21]seed@VM:~/.../Labsetup$ chmod a+x sniffer.py
[12/04/21]seed@VM:~/.../Labsetup$ sudo ./sniffer.py
```

In this snippet, we run our sniffing program with root privileges.

```
root@e930c059475f:/# ping google.com
PING google.com (216.58.197.238) 56(84) bytes of data.
64 bytes from nrt13s49-in-f238.1e100.net (216.58.197.238): icmp_seq
=1 ttl=114 time=102 ms
64 bytes from nrt13s49-in-f238.1e100.net (216.58.197.238): icmp_seq
=2 ttl=114 time=127 ms
64 bytes from nrt13s49-in-f238.1e100.net (216.58.197.238): icmp_seq
=3 ttl=114 time=251 ms
64 bytes from nrt13s49-in-f238.1e100.net (216.58.197.238): icmp_seq
=4 ttl=114 time=72.0 ms
```

In this snippet, we access one of the hosts, host B, and ping google to generate ICMP traffic within the br-3af7b9237c20 interface.

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 02:42:03:30:c2:a8
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 48247
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xd5f9
     src       = 10.9.0.6
     dst       = 216.58.197.238
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x2ceb
        id        = 0x2f
        seq       = 0x1
###[ Raw ]###
           load      = '\x8f\x92\xaba\x00\x00\x00\x00\xc2\x1d\x0f\x
00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\
x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:03:30:c2:a8
```

In this snippet, we demonstrate our program can capture ICMP packets on the br-3af7b9237c20 interface.

```
[12/04/21]seed@VM:~/.../Labsetup$ su seed
Password:
[12/04/21]seed@VM:~/.../Labsetup$ ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 5, in <module>
    pkt = sniff( filter="icmp", prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py",
line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py",
line 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py"
, line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, soc
ket.htons(type))  # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

In this snippet, we notice we cannot use our sniffing program without root privileges.

Task 1.1B

```
[12/04/21]seed@VM:~/.../Labsetup$ cat sniffer.py
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(filter="icmp", prn=print_pkt)
```

In this snippet, we attempt to capture only ICMP packets.

```
[12/04/21]seed@VM:~/.../Labsetup$ chmod a+x sniffer.py
[12/04/21]seed@VM:~/.../Labsetup$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:0d:55:9f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 56162
     flags     = DF
     frag      = 0
     ttl       = 63
     proto     = icmp
     chksum    = 0xb616
     src       = 10.0.2.7
     dst       = 216.58.197.238
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x554b
        id        = 0x30
        seq       = 0x1
###[ Raw ]###
           load      = '\x9e\x93\xaba\x00\x00\x00\x00\x8b\xbb\x0e\x
00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\
x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 08:00:27:0d:55:9f
  src       = 52:54:00:12:35:00
```

In this snippet, we capture ICMP packets. For this, we pinged google.com from host B once more.

```
[12/04/21]seed@VM:~/.../Labsetup$ cat sniffer.py
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(filter="tcp and src host 10.0.2.123 and dst port 23", p
rn=print_pkt)
```

In this snippet, we change our sniffing program to capture TCP packets from 10.0.2.123 arriving through port 23.

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> from scapy.all import *
>>> ip = IP()
>>> ip.src = "10.0.2.123"
>>> ip.dst = "10.0.2.1"
>>> tcp = TCP()
>>> tcp.dport = 23
>>> send(ip/tcp)
.
Sent 1 packets.
```

In this snippet, we generate a TCP packet with 10.0.2.123 as source, 10.0.2.1 as destination, and 23 as destination port.

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:0d:55:9f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x6254
     src       = 10.0.2.123
     dst       = 10.0.2.1
     \options   \
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
        flags     = S
        window    = 8192
        chksum    = 0x773c
        urgptr    = 0
        options   = []
```

In this snippet, we capture TCP packets from 10.0.2.123 arriving through port 23.

```
[12/04/21]seed@VM:~/.../Labsetup$ cat sniffer.py
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(filter="net 128.230.0.0/16", prn=print_pkt)
```

In this snippet, we modify our sniffing programs to sniff on net 218.230.0.0/16.

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> from scapy.all import *
>>> ip = IP()
>>> ip.src = "10.0.2.11"
>>> ip.dst = "128.230.0.1"
>>> tcp = TCP()
>>> tcp.dport = 23
>>> send(ip/tcp)
.
Sent 1 packets.
```

In this snippet, we generate a TCP packet with 10.0.2.11 as source, 128.230.0.1 as destination, and 23 as destination port.

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:0d:55:9f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xeddd
     src       = 10.0.2.11
     dst       = 128.230.0.1
     \options   \
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
        flags     = S
        window    = 8192
        chksum    = 0x2c6
        urgptr    = 0
        options   = []
```

In this snippet, we capture TCP packets from 10.0.2.11 to 128.230.0.1, arriving through port 23.

Task 1.2

```
[12/04/21]seed@VM:~/.../Labsetup$ sudo python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> from scapy.all import *
>>> ip = IP()
>>> ip.src = "10.0.2.3"
>>> ip.dst = "128.230.0.1"
>>> icmp = ICMP()
>>> send(ip/icmp)
.
Sent 1 packets.
```

In this snippet, we create an ICMP packet from 10.0.2.3 to 128.230.0.1.

No.	Time	Source	Destination	Protocol	Length	In
1	2021-12-04 11:59:50.815469896	10.0.2.3	128.230.0.1	ICMP	44	E

In this snippet, we demonstrated that we could spoof any ICMP echo request packet with an arbitrary source IP address.

Task 1.3

```
[12/04/21]seed@VM:~/.../Labsetup$ cat tracer.py
#!/usr/bin/python3
from scapy.all import *
import sys

a = IP()
a.dst = "8.8.8.8"
a.ttl = 1
b = ICMP()
send(a/b)
```

In this snippet, we see the program we will be using to contact 8.8.8.8. We will be changing the TTL parameter until we receive a response.

→	3	2021-12-04 12:31:00.966397899	10.0.2.7	8.8.8.8	ICMP	44 E
←	4	2021-12-04 12:31:01.129253589	8.8.8.8	10.0.2.7	ICMP	62 E

In the above snippet, we see we sent an ICMP request to 8.8.8.8.

8.8.8.8	ICMP	44	Echo (ping) request	id=0x0000, seq=0/0, ttl=33 (reply in 4)
10.0.2.7	ICMP	62	Echo (ping) reply	id=0x0000, seq=0/0, ttl=114 (request in …

In this snippet, we see 8.8.8.8 replied to our request when TTL = 33.

Task 1.4

```
[12/04/21]seed@VM:~/.../Labsetup$ cat sniff_n_spoof.py
#!/usr/bin/python3
from scapy.all import *

def print_pkt(pkt):
        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
        send(newpkt, verbose = 0)
        print ("Sent spoofed packet\n")

pkt = sniff(filter="icmp[icmptype]==icmp-echo", prn=print_pkt)
```

In this snippet, we see our sniff and spoof program.

```
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5112ms
```

In this snippet, we notice we cannot ping 1.2.3.4.

```
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=36.3 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=23.2 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=20.8 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=25.7 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=26.7 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=30.2 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=24.3 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=31.7 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=68.9 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=26.0 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=18.8 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=23.8 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=17.2 ms
^C
--- 1.2.3.4 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12071ms
rtt min/avg/max/mdev = 17.221/28.734/68.852/12.607 ms
```

In this snippet, we notice we can now ping 1.2.3.4 after running our program.

```
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Host Unreachable
From 10.9.0.1 icmp_seq=2 Destination Host Unreachable
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
From 10.9.0.1 icmp_seq=10 Destination Host Unreachable
From 10.9.0.1 icmp_seq=11 Destination Host Unreachable
From 10.9.0.1 icmp_seq=12 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
15 packets transmitted, 0 received, +12 errors, 100% packet loss, t
ime 14331ms
pipe 3
```

In this snippet, we notice that we cannot ping local address 10.9.0.99. This happens even after activating our program. This is because our program works with ICMP packets, and pinging a local address involves ARP packets.

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=55.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=160 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=55.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=99.8 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=114 time=119 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=114 time=141 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=114 time=53.4 ms
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6014ms
rtt min/avg/max/mdev = 53.412/97.781/160.165/40.934 ms
```

In this snippet, we see we can ping 8.8.8.8 without using our program.

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=25.5 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=35.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=28.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=35.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=42.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=113 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=21.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=208 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=114 time=36.2 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, +5 duplicates, 0% packet loss, t
ime 4061ms
```

In this snippet, we activate our program and try to ping 8.8.8.8. We see the DUP! keyword, meaning there are duplicate ICMP packets. This is because host 8.8.8.8 is real and is responding to our ICMP request, while our program does the same.