

Barrett Bobilin

1/31/25

Florida State University

CIS 4360

CIS 4360 – Lab 1

Task Set 1.1:

The first task was to test a very simple sniffing script to make sure we were able to see the packets traversing the network as seen here:

3.1 Task 1.1: Sniffing Packets

Wireshark is the most popular sniffing tool, and it is easy to use. We will use it throughout the entire lab. However, it is difficult to use Wireshark as a building block to construct other tools. We will use Scapy for that purpose. The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs. A sample code is provided in the following:

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-c93733e9f913', filter='icmp', prn=print_pkt)
```

The code above will sniff the packets on the `br-c93733e9f913` interface. Please read the instruction in the lab setup section regarding how to get the interface name. If we want to sniff on multiple interfaces, we can put all the interfaces in a list, and assign it to `iface`. See the following example:

```
iface=['br-c93733e9f913', 'enp0s3']
```

To accomplish this, I ran “ipconfig” to show the interface name and IPs. The setup page (not shown here) tells us to find the network with IP 10.9.0.1 as highlighted in the left terminal. The right terminal contains our script using Scapy to sniff the ping packets:

```

[01/29/25]seed@VM: ~/Desktop$ ifconfig
br-1bae5aae065e: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:fdff:fe63:177d prefixlen 64 scopeid 0x200<link>
    ether 02:42:fd:63:17:7d txqueuelen 0 (Ethernet)
    RX packets 17 bytes 812 (812.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 47 bytes 5848 (5.8 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions

vbr-3876afalbaaa: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.20.0.1 netmask 255.255.255.0 broadcast 172.20.0.255
    ether 02:42:52:3c:bb:59 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions

GNU nano 4.8 mycode.py
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-1bae5aae065e', filter='icmp', prn=print)
  
```

Next, I ran a ping from Host A (10.9.0.5) to Host B (10.9.0.6), while using tcpdump on Host B to see the incoming ping. On the left is Host A running ping, on the right is Host B running tcpdump, and on the bottom is our attacker (seed-attacker) running the Scapy script to sniff the packets.

```

root@072d94b68e8d:~/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.221 ms
^C
--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.221/0.221/0.221/0.000 ms
root@072d94b68e8d:~/#

root@23f9dc01f061:~/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:47:54.045439 IP hostA-10.9.0.5.net-10.9.0.0 > 23f9dc01f061: ICMP echo request, id 6, seq 1, length 64
00:47:54.045512 IP 23f9dc01f061 > hostA-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 6, seq 1, length 64
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@23f9dc01f061:~/#

root@VM:~/# mycode.py
##[ Ethernet ]##
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
##[ IP ]##
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 21950
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = 0xd0ce
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
##[ ICMP ]##
  type     = echo-request
  code     = 0
  checksum = 0x7741
  
```

The rest of the sniffed packet is shown here:

```

###[ ICMP ]###
    type      = echo-request
    code       = 0
    chksum     = 0x7241
    id         = 0x6
    seq        = 0x1
###[ Raw ]###
    load       = '\xcc\x9a\x00\x00\x00\x00\xf1\xb0\x00\x00\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:09:00:05
    src        = 02:42:0a:09:00:06
    type       = IPv4
###[ IP ]###
    version    = 4
    ihl        = 5
    tos        = 0x0
    len        = 84
    id         = 41336
    flags      =
    frag       = 0
    ttl        = 64
    proto      = icmp
    chksum     = 0xc514
    src        = 10.9.0.6
    dst        = 10.9.0.5
    \options   \
###[ ICMP ]###
    type      = echo-reply
    code       = 0
    chksum     = 0x7a41
    id         = 0x6
    seq        = 0x1
###[ Raw ]###
    load       = '\xcc\x9a\x00\x00\x00\x00\xf1\xb0\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'

root@VM:/# █

```

Task 1.1A

For this task, we need to switch our user to “seed” and re-run the sniffing program using reduced permissions. In the picture you can see I switched to the “seed” user and checked the file permissions to ensure that the “seed” user is able to execute the file. Despite this, I am still unable to run the sniffing program. The reason for this is because (I think) only root has permission to create raw sockets. (Changed terminal to dark because white hurt my eyes)

```

root@VM:/# su seed
seed@VM:/$ ls -l mycode.py
-rwxr-xr-x 1 root root 152 Jan 30 00:37 mycode.py
seed@VM:/$ mycode.py
Traceback (most recent call last):
  File "./mycode.py", line 8, in <module>
    pkt = sniff(iface='br-lbae5aae065e', 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, socket.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
seed@VM:/$ █

```

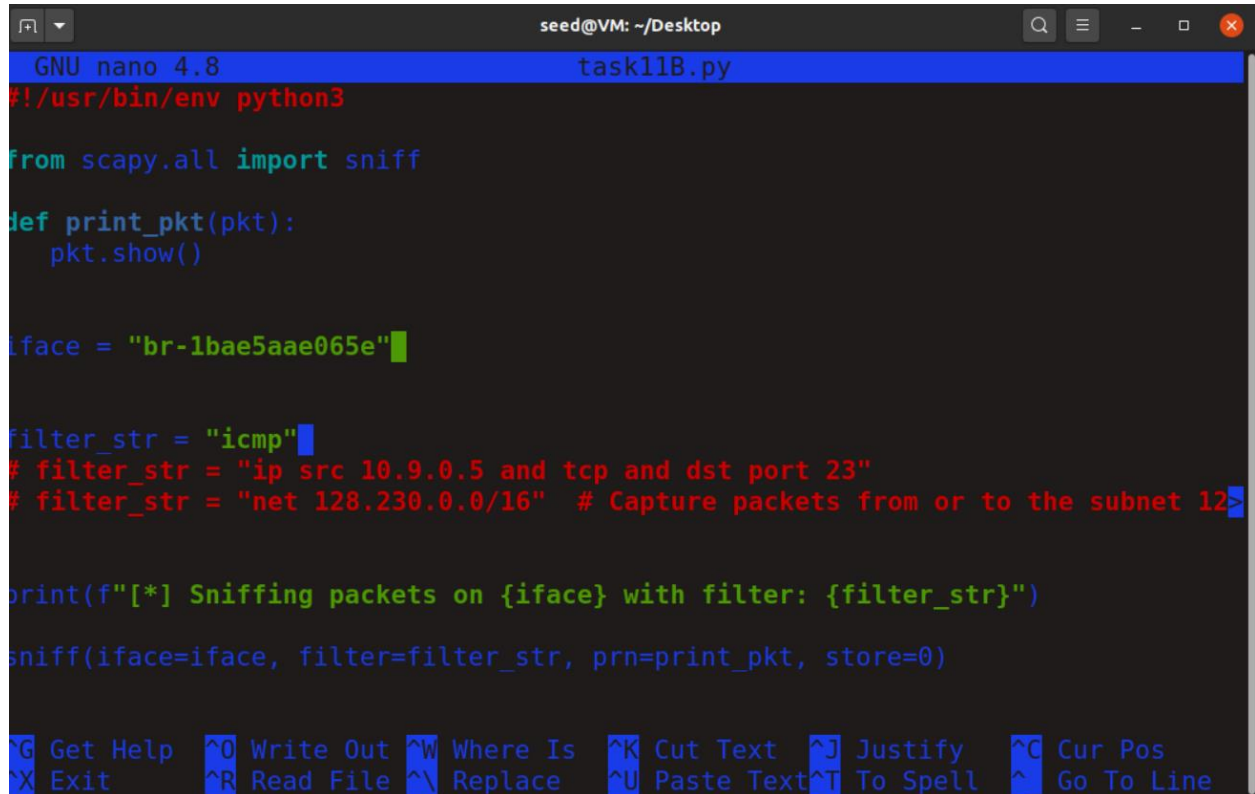
Task 1.1B

In this task we have 3 different objectives.

- Capture a packet only via ICMP

- Capture any TCP packet that comes from a particular IP (in our case, HostA-10.9.0.5 and with a destination port number 23
- Capture packets coming from or going to a particular subnet, 128.230.0.0/16

Here is the code used for the 3 tasks listed above:



```

GNU nano 4.8 task11B.py
#!/usr/bin/env python3

from scapy.all import sniff

def print_pkt(pkt):
    pkt.show()

iface = "br-1bae5aae065e"

filter_str = "icmp"
# filter_str = "ip src 10.9.0.5 and tcp and dst port 23"
# filter_str = "net 128.230.0.0/16" # Capture packets from or to the subnet 128.230.0.0/16

print(f"[*] Sniffing packets on {iface} with filter: {filter_str}")

sniff(iface=iface, filter=filter_str, prn=print_pkt, store=0)

```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
 ^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

Here is the ICMP captured code. The bottom terminal is running the sniffer, the top left terminal is running ping, and the top right terminal is running tcpdump:

```
root@072d94b68e8d:~# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data:
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.153 ms
^C
--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.153/0.153/0.153/0.000 ms
root@072d94b68e8d:~#

root@023f9dc01f061:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
03:12:21.197629 IP hostA-10.9.0.5.net-10.9.0.0 > 23f9dc01f061: ICMP echo request,
10.9, seq 1, length 64
03:12:21.197670 IP 23f9dc01f061 > hostA-10.9.0.5.net-10.9.0.0: ICMP echo reply, 10
9, seq 1, length 64
03:12:26.358827 ARP, Request who-has hostA-10.9.0.5.net-10.9.0.0 tell 23f9dc01f061
, length 28
03:12:26.359883 ARP, Request who-has 23f9dc01f061 tell hostA-10.9.0.5.net-10.9.0.0
, length 28
03:12:26.359106 ARP, Reply 23f9dc01f061 is-at 02:42:0a:09:00:06 (oui Unknown), len
gth 28
03:12:26.359118 ARP, Reply hostA-10.9.0.5.net-10.9.0.0 is-at 02:42:0a:09:00:05 (ou
i Unknown), length 28

root@VM:~# task118.py
[*] Sniffing packets on br-lbae3aee065e with filter: icmp
##[ Ethernet ]##
dst      = 02:42:0a:09:00:06
src      = 02:42:0a:09:00:05
type     = IPv4
##[ IP ]##
version  = 4
ihl      = 5
tos      = 0x0
len      = 64
id       = 28924
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xb598
src      = 10.9.0.5
dst      = 10.9.0.6
options  \
##[ ICMP ]##
type     = echo-request
code     = 0
chksum   = 0x7178
id       = 0x0
seq      = 0x1
```

Here is the TCP port 23 captured code. The bottom terminal is running the sniffer, the top left terminal is running telnet which connet via port 23, and the top right terminal is running tcpdump:

```
File Machine View Input Devices Help
Activities Terminal Jan 30 21:56

root@072d94b68e8d:~# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^['.
Ubuntu 20.04.1 LTS
23f9dc01f061 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the

root@VM:~# task118.py
[*] Sniffing packets on br-lbae3aee065e with filter: ip src 10.9.0.5 and tcp and dst port 23
##[ Ethernet ]##
src      = 02:42:0a:09:00:06
dst      = 02:42:0a:09:00:05
type     = IPv4
##[ IP ]##
version  = 4
ihl      = 5
tos      = 0x10
len      = 60
id       = 65478
flags    = DF
frag     = 0
ttl      = 64
proto    = tcp
chksum   = 0x26c9
src      = 10.9.0.5
dst      = 10.9.0.6
options  \
##[ TCP ]##
src      = 49346
dest     = telnet
seq      = 923315623
ack      = 0
dataoffs = 10

root@023f9dc01f061:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
02:54:06.278632 IP hostA-10.9.0.5.net-10.9.0.0.49346 > 23f9dc01f061:telnet: Flags
[SY], seq 923315623, win 64240, options [mss 1460,sackOK,TS val 2449263596 ecr 0,no
p,wscale 7], length 0
02:54:06.278740 IP 23f9dc01f061:telnet > hostA-10.9.0.5.net-10.9.0.0.49346: Flags
[SA], seq 542467882, ack 923315624, win 65160, options [mss 1460,sackOK,TS val 111
9649289 ecr 2449263596,nop,wscale 7], length 0
02:54:06.278632 IP hostA-10.9.0.5.net-10.9.0.0.49346 > 23f9dc01f061:telnet: Flags
[SY], seq 1, win 502, options [nop,nop,TS val 2449263597 ecr 1119649289], length 0
02:54:06.302222 IP hostA-10.9.0.5.net-10.9.0.0.49346 > 23f9dc01f061:telnet: Flags
[P], seq 1:25, ack 1, win 502, options [nop,nop,TS val 2449263620 ecr 1119649289]
, length 24 [telnet DO SUPPRESS GO AHEAD, WILL TERMINAL TYPE, WILL NAW, WILL TSPE
ED, WILL LFLOC, WILL LINEMODE, WILL NEW-ENVIRON, DO STATUS [telnet]
02:54:06.302247 IP 23f9dc01f061:telnet > hostA-10.9.0.5.net-10.9.0.0.49346: Flags
[I], ack 25, win 509, options [nop,nop,TS val 1119649312 ecr 2449263620], length 0
02:54:06.317582 IP 23f9dc01f061:telnet > hostA-10.9.0.5.net-10.9.0.0.49346: Flags
[P], seq 1:13, ack 25, win 509, options [nop,nop,TS val 1119649327 ecr 2449263620
], length 12 [telnet DO TERMINAL TYPE, DO TSPEED, DO NEW-ENVIRON [te
```

Here is the subnet captured code. The bottom terminal is running the sniffer, and the top terminal is running ping to the subnet in question, 128.230.0.0.


```
seed@VM: ~/Desktop
root@072d94b68e8d:/# ping 128.230.0.0
PING 128.230.0.0 (128.230.0.0) 56(84) bytes of data.
^C
--- 128.230.0.0 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1026ms

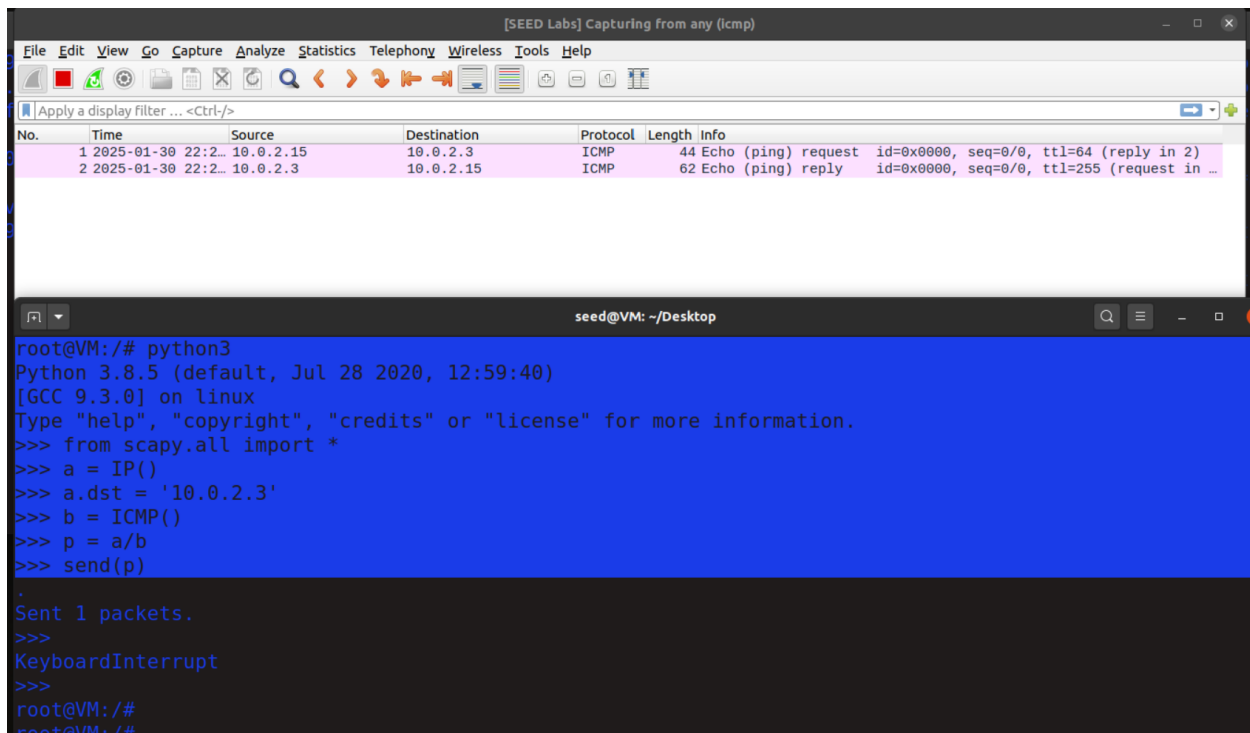
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#
root@072d94b68e8d:/#

seed@VM: ~/Desktop
root@VM:/# task11B.py
*) Sniffing packets on br-1bae5aae065e with filter: net 128.230.0.0/16
##[ Ethernet ]###
  dst      = 02:42:c9:b7:87:1b
  src      = 02:42:0a:09:00:05
  type     = IPv4
##[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 31562
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x346b
  src      = 10.9.0.5
  dst      = 128.230.0.0
  \options \
##[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x61b2
  id       = 0x8
  seq      = 0x1
```

Task 1.2

Task 1.2 asks us to do the following: spoof an ICMP echo request packet with an arbitrary source IP address. Here I used the interactive python command, “python3” to run the code 1 line at a

time, and using Wireshark on the host VM, I was able to capture the packet that had the spoofed IP address. I chose 10.0.2.3 as the random IP address, as you can see on the bottom terminal, and on the top, Wireshark has captured the spoofed ICMP packet and its reply.



Task 1.3

Task 1.3 asks us to implement traceroute to see how packet switching occurs. Specifically: “The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination”. Unfortunately, I fell into the common problem of only being able to see the send and reply, and not being able to see the routers along the way. However, I ran the interactive python script twice, once with a TTL of 1, and once with a TTL of 30, which are both listed in the top Wireshark program (+1 send a receive test). I was able to perform a proper traceroute by switching my network settings from NAT to Bridged, which allowed me to run traceroute correctly.

The first picture below shows the script with only the first and last hop, the second picture is the full traceroute with all subsequent routers.

[SEED Labs] any (icmp)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-01-30 22:4...	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (reply in 2)
2	2025-01-30 22:4...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=255 (request in ...)
3	2025-01-30 22:5...	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (reply in 4)
4	2025-01-30 22:5...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=255 (request in ...)
5	2025-01-30 23:0...	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=30 (reply in 6)
6	2025-01-30 23:0...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=255 (request in ...)

Frame 1: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface any, id 0

- Linux cooked capture
- Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8
 - 0100 ... = Version: 4
 - ... 0101 = Header Length: 20 bytes (5)
 - Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 - Total Length: 28
 - Identification: 0x0001 (1)
 - Flags: 0x0000
 - Fragment offset: 0
 - Time to live: 1
 - Protocol: ICMP (1)

```

root@VM: ~/Desktop
root@VM:~# 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 information.
>>> from scapy.all import *
>>> a = IP()
>>> a.dst = '8.8.8.8'
>>> a.ttl = 1
>>> b = ICMP()
>>> send(a/b)
.
Sent 1 packets.
>>>
root@VM:~# traceroute
bash: traceroute: command not found
root@VM:~# tracert
bash: tracert: command not found
root@VM:~# 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 information.
>>> from scapy.all import *
>>> a = IP()
>>> a.dst = '8.8.8.8'
>>> a.ttl = 30
>>> b = ICMP()
>>> send(a/b)

```

ed: 6 (100.0%) Profile: Default

[SEED Labs] Capturing from any (icmp)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

No.	Time	Source	Destination	Protocol	Length	Info
28	2025-01-31 11:0...	108.59.28.49	10.132.102.114	ICMP	104	Time-to-live exceeded (Time to live exceeded in transit)
29	2025-01-31 11:0...	108.59.31.154	10.132.102.114	ICMP	108	Time-to-live exceeded (Time to live exceeded in transit)
30	2025-01-31 11:0...	108.59.31.202	10.132.102.114	ICMP	112	Time-to-live exceeded (Time to live exceeded in transit)
31	2025-01-31 11:0...	108.59.31.202	10.132.102.114	ICMP	112	Time-to-live exceeded (Time to live exceeded in transit)
32	2025-01-31 11:0...	108.59.31.202	10.132.102.114	ICMP	112	Time-to-live exceeded (Time to live exceeded in transit)
33	2025-01-31 11:0...	108.59.28.49	10.132.102.114	ICMP	104	Time-to-live exceeded (Time to live exceeded in transit)
34	2025-01-31 11:0...	8.8.8.8	10.132.102.114	ICMP	72	Destination unreachable (Port unreachable)
35	2025-01-31 11:0...	8.8.8.8	10.132.102.114	ICMP	72	Destination unreachable (Port unreachable)
36	2025-01-31 11:0...	8.8.8.8	10.132.102.114	ICMP	72	Destination unreachable (Port unreachable)
37	2025-01-31 11:0...	8.8.8.8	10.132.102.114	ICMP	72	Destination unreachable (Port unreachable)

Frame 34: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface any, id 0

- Linux cooked capture
- Internet Protocol Version 4, Src: 8.8.8.8, Dst: 10.132.102.114
 - 0100 ... = Version: 4
 - ... 0101 = Header Length: 20 bytes (5)
 - Differentiated Services Field: 0x80 (DSCP: CS4, ECN: Not-ECT)
 - Total Length: 56
 - Identification: 0x0000 (0)
 - Flags: 0x0000
 - Fragment offset: 0
 - Time to live: 112
 - Protocol: ICMP (1)

```

01/31/25]seed@VM:~/Desktop$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1 be9975-243.a9k.net.fsu.edu (10.132.0.1) 13.158 ms 12.564 ms 12.249 ms
 2 rsb-p.hu-0-0-1.net.fsu.edu (10.128.253.70) 11.970 ms 11.674 ms 11.386 ms
 3 rsb-pel.hu-0-0-1-0.net.fsu.edu (10.128.253.81) 11.084 ms msb-pel.hu-0-0-1-1.net.fsu.e
  (10.128.253.145) 11.006 ms 10.703 ms
 4 10.128.242.59 (10.128.242.59) 10.138 ms 10.017 ms 9.732 ms
 5 1rb100.rsb-br.net.fsu.edu (128.186.248.96) 9.436 ms 9.159 ms 8.727 ms
 6 tlh-flrcore-asr9010-1-te0203-1903.net.flrnet.org (108.59.27.240) 12.610 ms 11.833 ms
 11.389 ms
 7 tpa-flrcore-asr9010-1-hu0600-1.net.flrnet.org (108.59.31.156) 15.883 ms 16.406 ms 1
 .066 ms
 8 108.59.31.204 (108.59.31.204) 14.920 ms 14.590 ms 15.460 ms
 9 mla-flrcore-asr9010-1-hu0600-1.net.flrnet.org (108.59.31.154) 14.748 ms 19.304 ms 1
 .930 ms
 0 108.59.31.202 (108.59.31.202) 17.206 ms 16.929 ms 16.672 ms
 1 flrnetcp-flrcore-108-59-28-49.rtr.net.flrnet.org (108.59.28.49) 16.253 ms 15.681 ms
 13.420 ms
 2 * * *
 3 dns.google (8.8.8.8) 15.077 ms 13.174 ms 13.528 ms
01/31/25]seed@VM:~/Desktop$

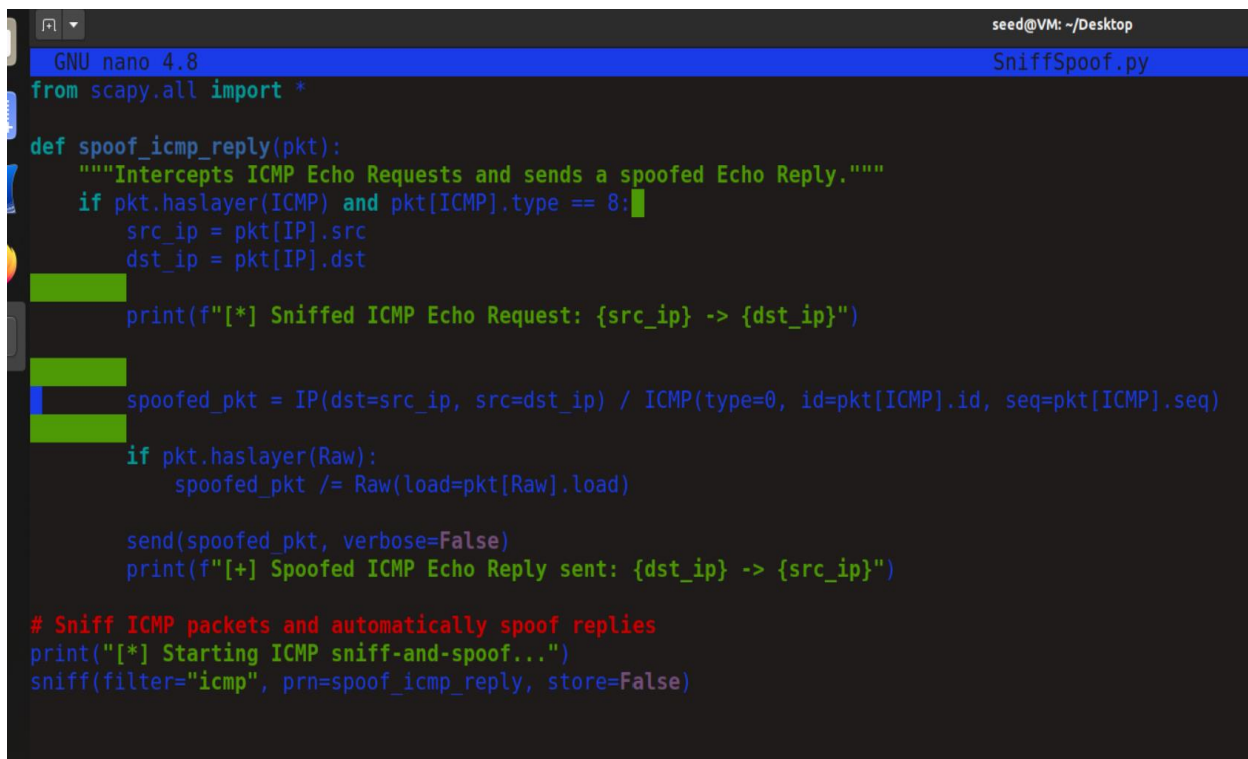
```

Default

Task 1.4

This task asks us to send a ping 3 separate IP addresses from our host VM. 1 non-existing host on the internet, 1 non-existing host on the LAN, and one real host on the internet. We will then run a script to sniff the sent packet's destination addresses, and send a reply from our attacker with a spoofed source address, matching the destination address of the host's sent packet.

Here is the code used to intercept and spoof the reply packet. Notice how "src_ip" is used as the destination Ip and vice-versa in line 12 (spoofed_pkt = ...)



```
seed@VM: ~/Desktop
GNU nano 4.8 SniffSpoof.py
from scapy.all import *

def spoof_icmp_reply(pkt):
    """Intercepts ICMP Echo Requests and sends a spoofed Echo Reply."""
    if pkt.haslayer(ICMP) and pkt[ICMP].type == 8:
        src_ip = pkt[IP].src
        dst_ip = pkt[IP].dst

        print(f"[*] Sniffed ICMP Echo Request: {src_ip} -> {dst_ip}")

        spoofed_pkt = IP(dst=src_ip, src=dst_ip) / ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)

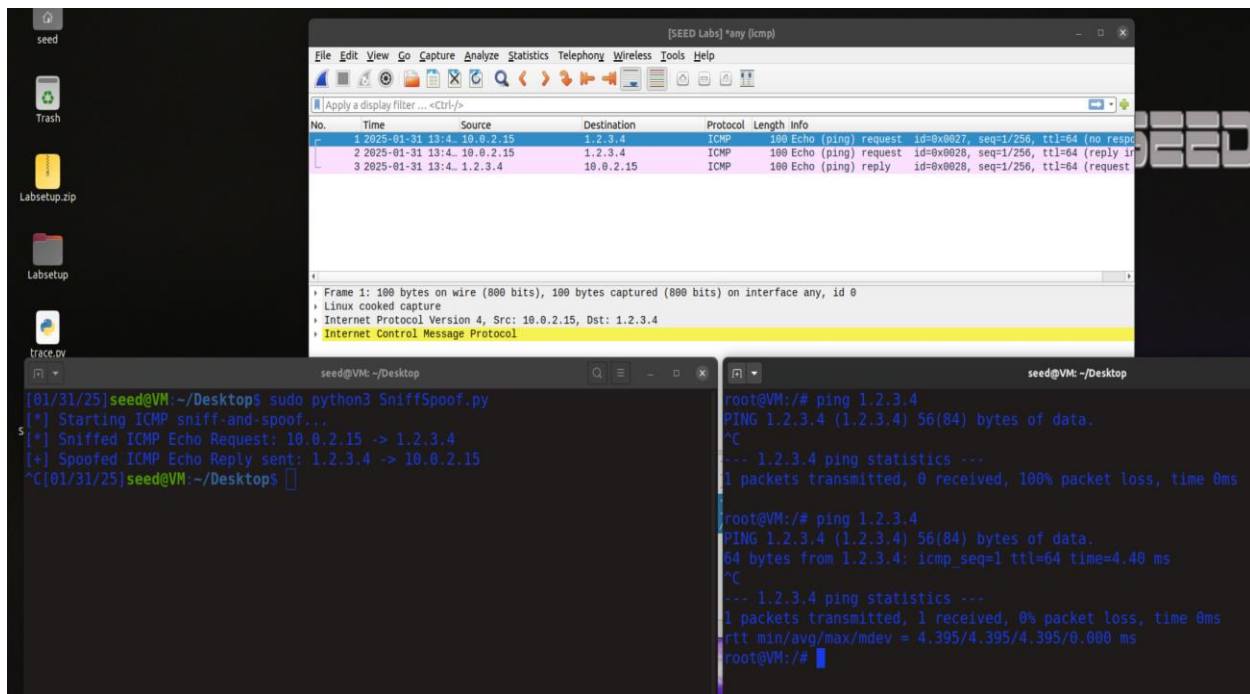
        if pkt.haslayer(Raw):
            spoofed_pkt /= Raw(load=pkt[Raw].load)

        send(spoofed_pkt, verbose=False)
        print(f"[+] Spoofed ICMP Echo Reply sent: {dst_ip} -> {src_ip}")

# Sniff ICMP packets and automatically spoof replies
print("[*] Starting ICMP sniff-and-spoof...")
sniff(filter="icmp", prn=spoof_icmp_reply, store=False)
```

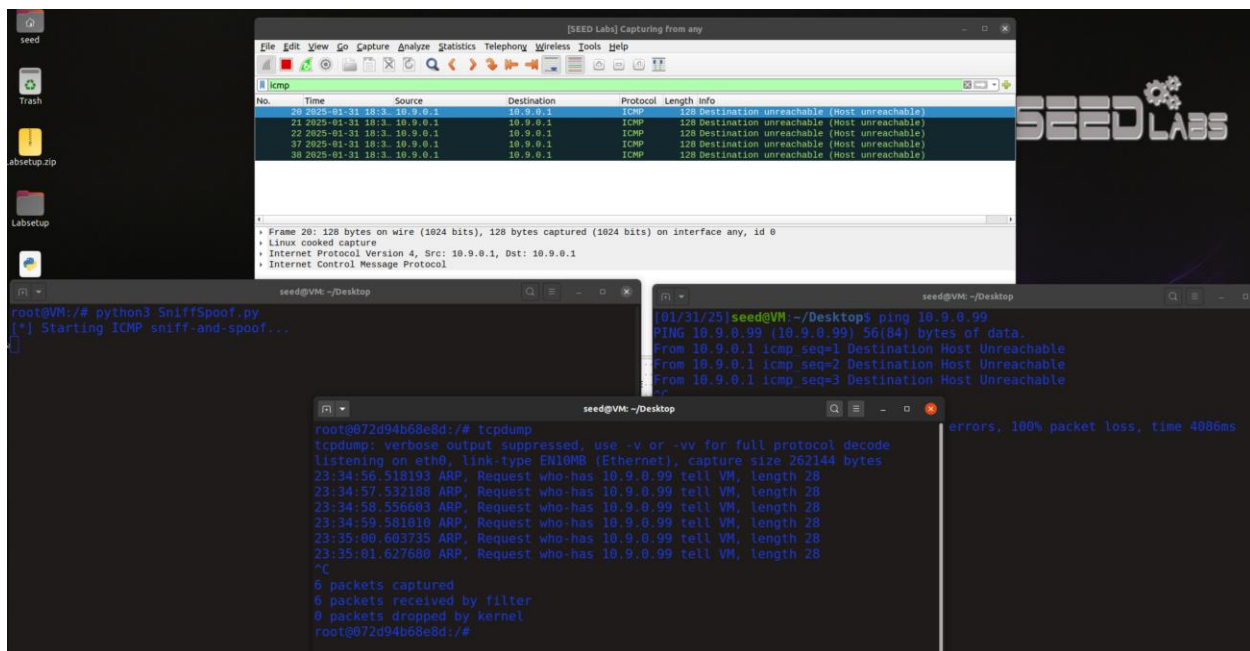
Below you can see the script in action. Notice the terminal on the right sends a ping to the non-existent 1.2.3.4 without the script running and gets no reply, seen as "1 packet transmitted, 0 received". This is also reflected in Wireshark as the top data point, with source IP 10.0.2.15 sending an ICMP packet to 1.2.3.4 with no reply.

The second ping is with the Sniff and Spoof script running, which intercepts the packet and sends a spoofed reply. Notice this time the right terminal says, “1 packet transmitted, 1 received”, which is again reflected on Wireshark as the bottom 2 data points.

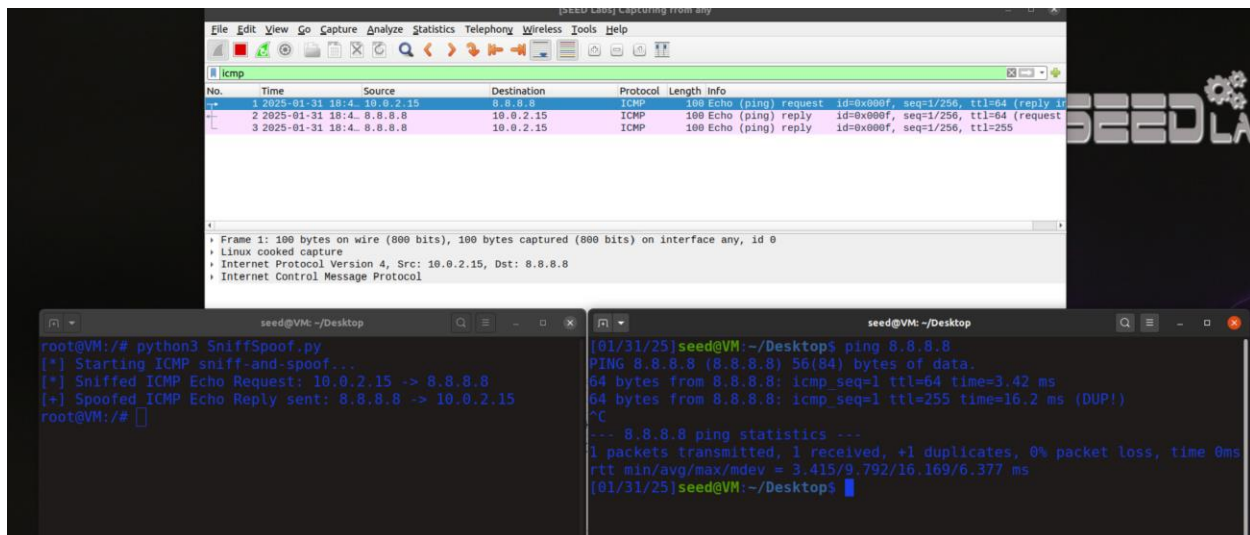


The same methodology will be repeated below with non-existing host 10.9.0.99 on the LAN, and existing host 8.8.8.8 on the internet.

When I ping 10.9.0.99 I get an ARP resolution error. ARP is used to translate IP addresses into MAC addresses, allowing a private IP address to correlate to a physical address. Because the IP doesn't exist, we get an ARP resolution error because it can't find the matching MAC address. The left terminal is running our Sniff/Spoof script, the bottom terminal is running tcpdump, and the right terminal is running ping to the unreachable host.



Lastly, we run the script against a real host on the internet, 8.8.8.8. You can see in Wireshark at the top that we get 1 request but 2 replies. One reply from google, and one from our spoofed packet script.



END REPORT