

Sniffing and Spoofing

Seed Lab

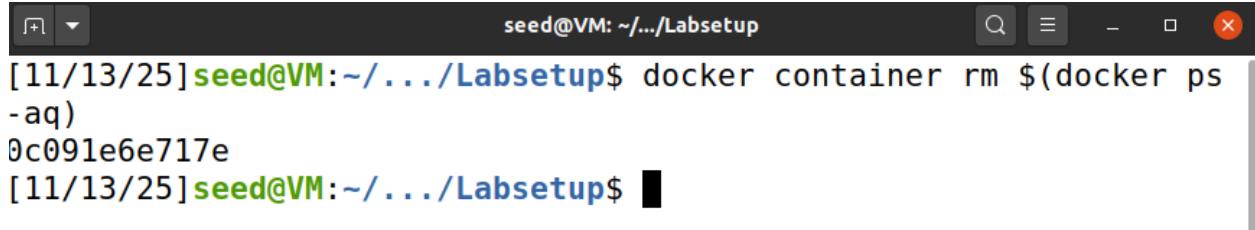
By Aneesh Nagdev

November 18, 2025

Lab setup:

```
[11/13/25]seed@VM:~/.../Labsetup$ docker container stop $(docker ps -aq)  
0c091e6e717e  
[11/13/25]seed@VM:~/.../Labsetup$
```

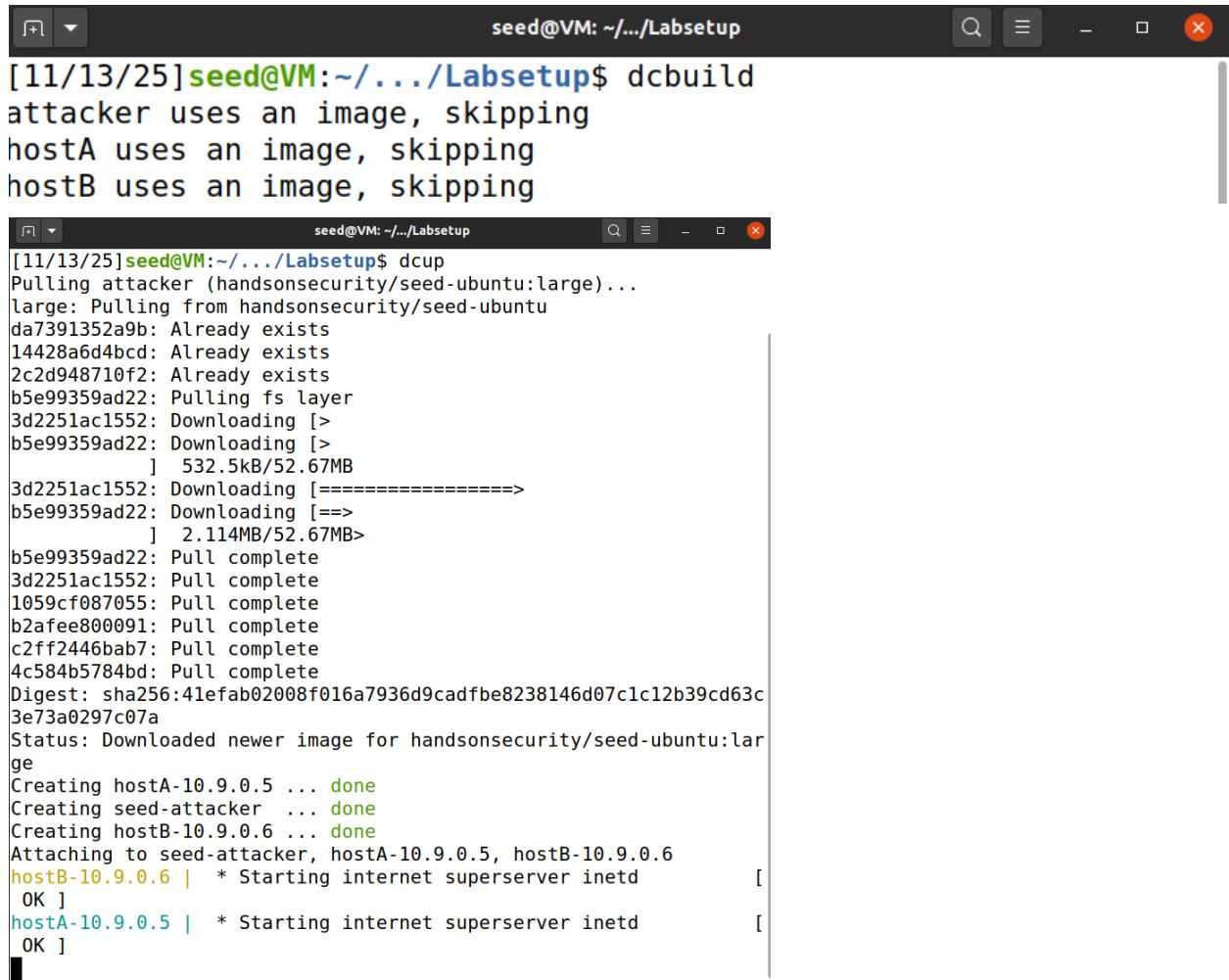
Explanation: Running this command to stop any other containers from past attacks.



```
seed@VM: ~/.../Labsetup [11/13/25]seed@VM:~/.../Labsetup$ docker container rm $(docker ps -aq)  
0c091e6e717e  
[11/13/25]seed@VM:~/.../Labsetup$
```

Explanation: Removing the stopped container so there is no interference with this attack.

Running the right containers for this lab screenshot:



```
seed@VM: ~/.../Labsetup [11/13/25]seed@VM:~/.../Labsetup$ dcbuild  
attacker uses an image, skipping  
hostA uses an image, skipping  
hostB uses an image, skipping
```

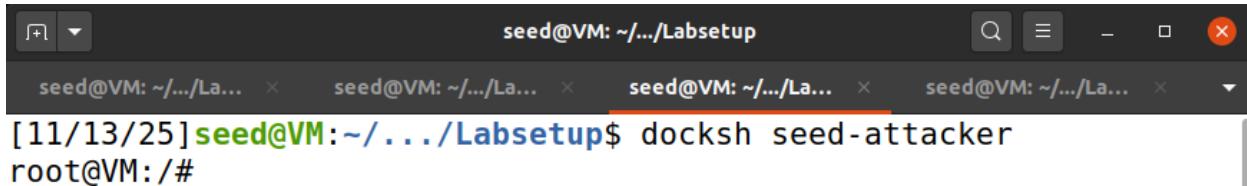


```
seed@VM: ~/.../Labsetup [11/13/25]seed@VM:~/.../Labsetup$ dcup  
Pulling attacker (handsonsecurity/seed-ubuntu:large)...  
large: Pulling from handsonsecurity/seed-ubuntu  
da7391352a9b: Already exists  
14428a6d4bcd: Already exists  
2c2d948710f2: Already exists  
b5e99359ad22: Pulling fs layer  
3d2251ac1552: Downloading [>  
b5e99359ad22: Downloading [>  
] 532.5kB/52.67MB  
3d2251ac1552: Downloading [=====>]  
b5e99359ad22: Downloading [==>  
] 2.114MB/52.67MB>  
b5e99359ad22: Pull complete  
3d2251ac1552: Pull complete  
1059cf087055: Pull complete  
b2afee800091: Pull complete  
c2ff2446bab7: Pull complete  
4c584b5784bd: Pull complete  
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c  
3e73a0297c07a  
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large  
Creating hostA-10.9.0.5 ... done  
Creating seed-attacker ... done  
Creating hostB-10.9.0.6 ... done  
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6  
hostB-10.9.0.6 | * Starting internet superserver inetd [OK ]  
hostA-10.9.0.5 | * Starting internet superserver inetd [OK ]
```

Checking running docker containers screenshot:

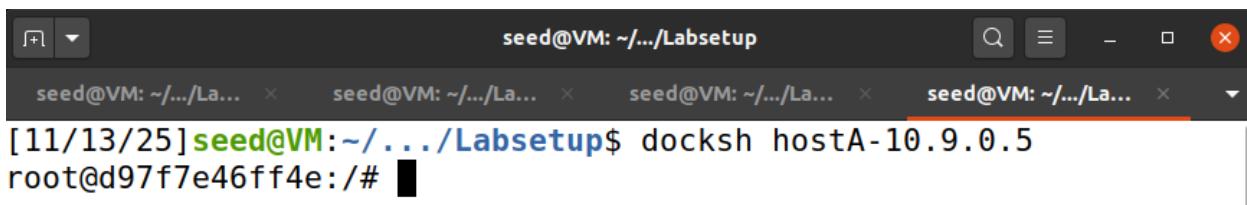
```
[11/13/25]seed@VM:~/.../Labsetup$ dockps
0f97e3f07068  seed-attacker
88ba12b40154  hostB-10.9.0.6
d97f7e46ff4e  hostA-10.9.0.5
[11/13/25]seed@VM:~/.../Labsetup$ █
```

Running the docker shell as attacker and host screenshots:



```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

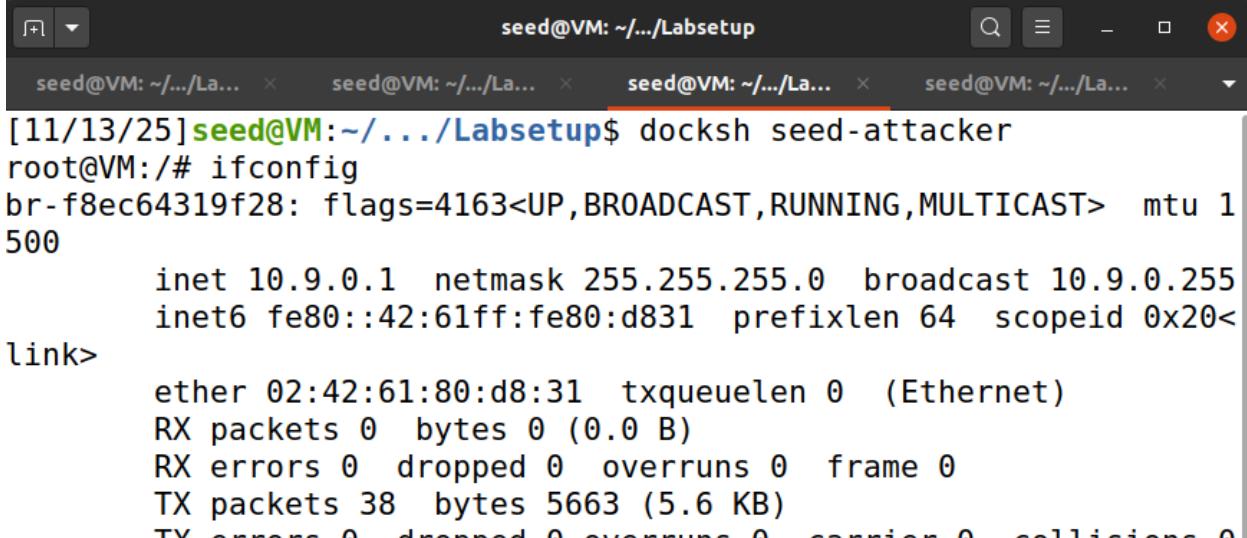
[11/13/25]seed@VM:~/.../Labsetup$ docksh seed-attacker
root@VM:/# █
```



```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

[11/13/25]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@d97f7e46ff4e:/# █
```

Saving the attacker's config for later:



```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

[11/13/25]seed@VM:~/.../Labsetup$ docksh seed-attacker
root@VM:/# ifconfig
br-f8ec64319f28: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1
500
      inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
      inet6 fe80::42:61ff:fe80:d831  prefixlen 64  scopeid 0x20<
link>
      ether 02:42:61:80:d8:31  txqueuelen 0  (Ethernet)
      RX packets 0  bytes 0 (0.0 B)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 38  bytes 5663 (5.6 KB)
      TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

Task 1:

Task 1.1:

Running Scapy in attacker's shell screenshot:

```
root@VM:/# scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump()
.
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.
.SYPACCCSASYY
P /SCS/CCS      ACS | Welcome to Scapy
    /A          AC | Version 2.4.4
    A/PS        /SPPS |
        YP        (SC | https://github.com/secdev/scapy
        SPS/A.     SC |
    Y/PACC        PP | Have fun!
    PY*AYC       CAA
    YYCY//SCYP
>>> █
```

Python file start.py code screenshot:

```
seed@VM: ~/.../La... × seed@VM: ~/.../vo... × seed@VM: ~/.../La... × seed@VM: ~/.../La... ×
#!/usr/bin/env python3
from scapy.all import *

a = IP()
a.show() █
```

Running [start.py](#) file output screenshot:

```
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
root@VM:/# ls
bin  etc  lib32  media  proc  sbin  tmp  volumes
boot  home  lib64  mnt  root  srv  usr
dev  lib  libx32  opt  run  sys  var
root@VM:/# cd volumes/
root@VM:/volumes# ls
start.py  template.py
root@VM:/volumes# python3 start.py
###[ 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    \
root@VM:/volumes#
```

Changing the file permissions to make it executable screenshot:

```
root@VM:/volumes# chmod a+x start.py
root@VM:/volumes#
```

Code for task 1.1 python file screenshot:



The screenshot shows a terminal window with four tabs open. The active tab displays the following Python code:

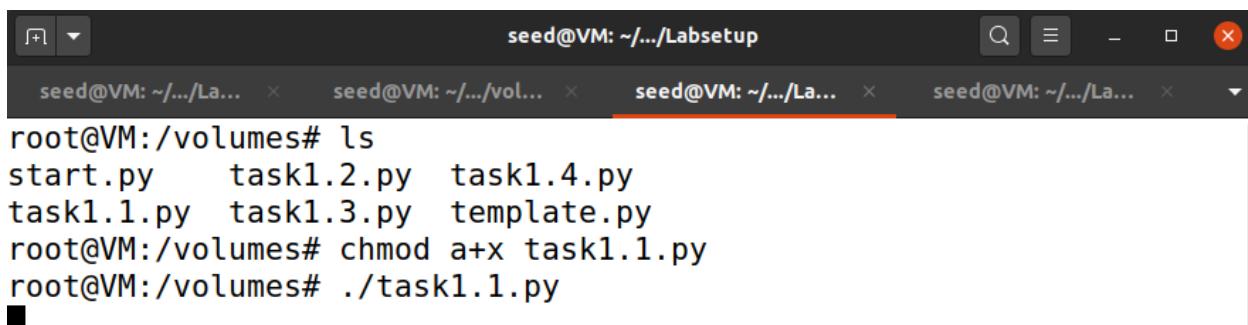
```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-f8ec64319f28', filter='icmp', prn=print_pkt)
```

Explanation: We have changed to the attackers interface as shown in the screenshot above.

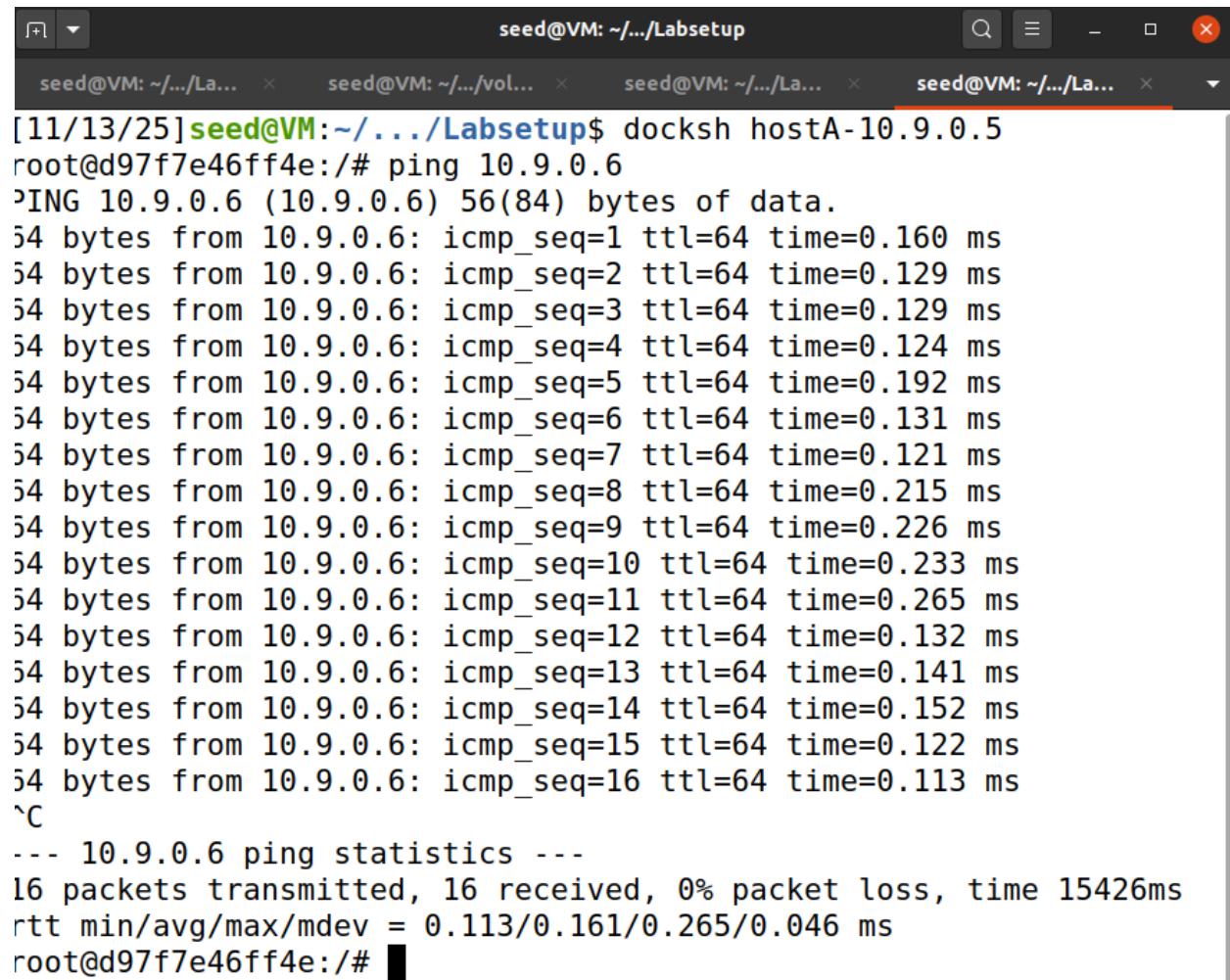
1.1A:



```
seed@VM: ~/.../Labsetup
root@VM:/volumes# ls
start.py      task1.2.py  task1.4.py
task1.1.py    task1.3.py  template.py
root@VM:/volumes# chmod a+x task1.1.py
root@VM:/volumes# ./task1.1.py
```

Explanation: It has started sniffing so we will generate an icmp packet to see if it works.

Pinging HostB to generate packets screenshot:



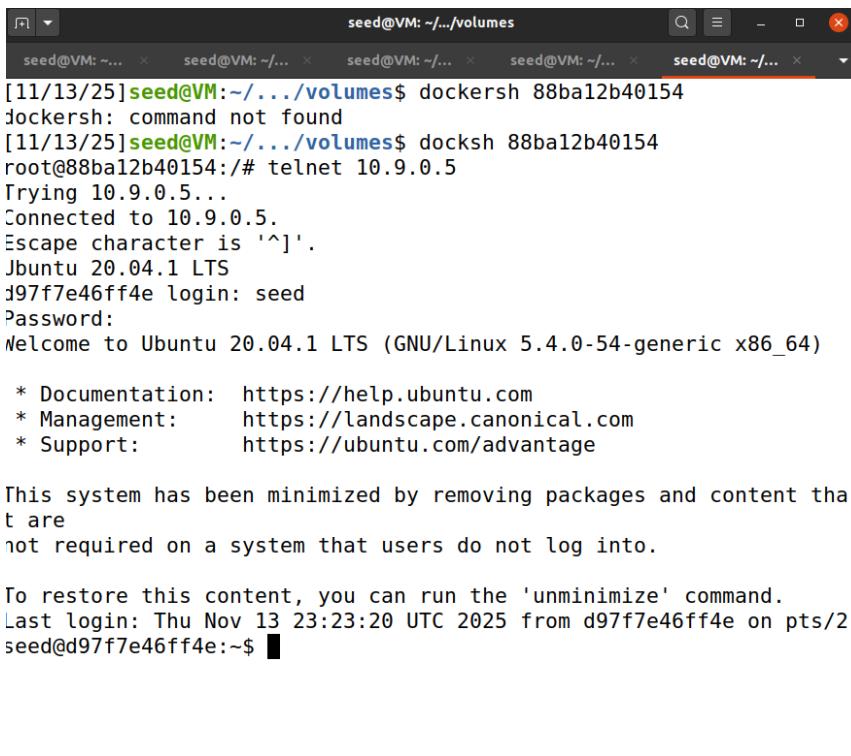
```
[11/13/25] seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@d97f7e46ff4e:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
54 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.160 ms
54 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.129 ms
54 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.129 ms
54 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.124 ms
54 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.192 ms
54 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.131 ms
54 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.121 ms
54 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.215 ms
54 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.226 ms
54 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.233 ms
54 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.265 ms
54 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.132 ms
54 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.141 ms
54 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.152 ms
54 bytes from 10.9.0.6: icmp_seq=15 ttl=64 time=0.122 ms
54 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.113 ms
^C
--- 10.9.0.6 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 15426ms
rtt min/avg/max/mdev = 0.113/0.161/0.265/0.046 ms
root@d97f7e46ff4e:/#
```

Checking if sniffing was successful screenshot:

Explanation: As it can be seen after running the script, the packet sniffer began listening on the specified network interface. When I pinged Host B from another terminal, ICMP packets were generated and transmitted across the network. The sniffer captured these packets in real time and displayed their details, confirming that the script was working correctly.

1.1B:

Changing the content of the filter screenshot:



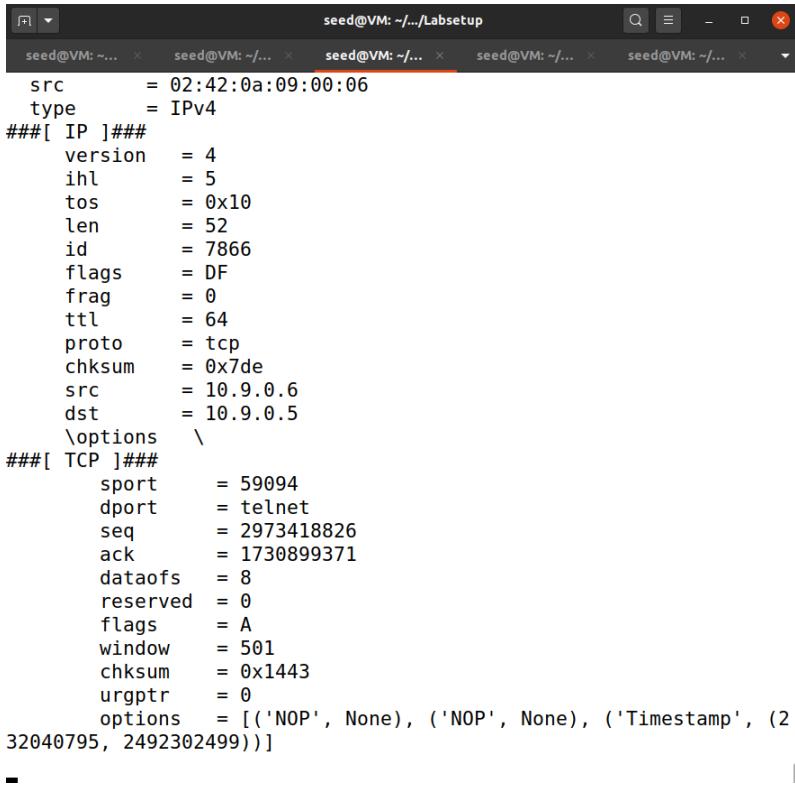
```
seed@VM: ~/.../volumes$ dockersh 88ba12b40154
dockersh: command not found
[11/13/25]seed@VM:~/.../volumes$ docksh 88ba12b40154
root@88ba12b40154:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
d97f7e46ff4e login: seed
Password:
Welcome to 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.
Last login: Thu Nov 13 23:23:20 UTC 2025 from d97f7e46ff4e on pts/2
seed@d97f7e46ff4e:~$
```

Output for pinging in hostB shell:



```
src      = 02:42:0a:09:00:06
type    = IPv4
###[ IP ]###
version  = 4
ihl     = 5
tos     = 0x10
len     = 52
id      = 7866
flags   = DF
frag    = 0
ttl     = 64
proto   = tcp
chksum  = 0x7de
src     = 10.9.0.6
dst     = 10.9.0.5
\options \
###[ TCP ]###
sport    = 59094
dport    = telnet
seq     = 2973418826
ack     = 1730899371
dataofs = 8
reserved = 0
flags   = A
window  = 501
checksum = 0x1443
urgptr  = 0
options  = [('NOP', None), ('NOP', None), ('Timestamp', (2
32040795, 2492302499))]
```

Explanation: After setting each filter in my sniffer, I generated the corresponding network traffic to test it. As it can be seen in the screenshot above after I started a new shell with the hostB container and ran the telnet command. The sniffer captured packets in the attackers shell.

Changing the filter to specific subnet screenshot:

The screenshot shows a terminal window with several tabs open, all labeled 'seed@VM: ~...'. The active tab contains the following Python code using the scapy library to sniff network traffic on interface 'br-f8ec64319f28' with a filter set to 'net 128.230.0.0/16'. The code defines a function 'print_pkt' to print each captured packet. Below the code, a root shell on hostB is shown, performing a ping to 128.230.0.0 and displaying ping statistics.

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

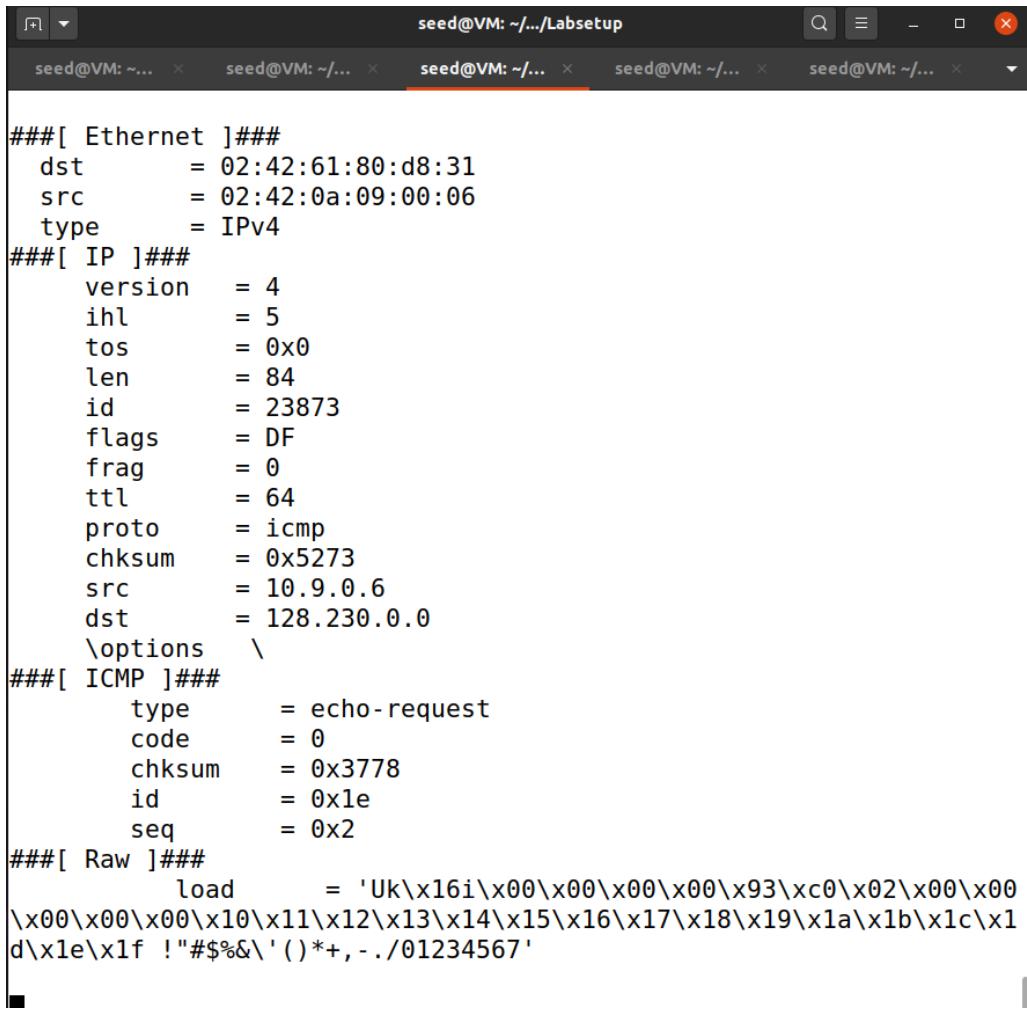
def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-f8ec64319f28', filter='net 128.230.0.0/16', prn=print_pkt)
~
~

root@88ba12b40154:/# 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 1012ms

root@88ba12b40154:/#
```

Python 3 Tab Width: 8 Ln 9, Col 67 INS



The screenshot shows a terminal window with five tabs, all titled 'seed@VM: ~...'. The active tab displays a detailed dump of captured network packets. The output is structured as follows:

- #[Ethernet]#**
 - dst = 02:42:61:80:d8:31
 - src = 02:42:0a:09:00:06
 - type = IPv4
- #[IP]#**
 - version = 4
 - ihl = 5
 - tos = 0x0
 - len = 84
 - id = 23873
 - flags = DF
 - frag = 0
 - ttl = 64
 - proto = icmp
 - chksum = 0x5273
 - src = 10.9.0.6
 - dst = 128.230.0.0
 - \options \
- #[ICMP]#**
 - type = echo-request
 - code = 0
 - checksum = 0x3778
 - id = 0xe
 - seq = 0x2
- #[Raw]#**
 - load = 'Uk\x16i\x00\x00\x00\x00\x93\xc0\x02\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !#\$%&\'()*+, -./01234567'

Explanation: After setting the filter for a specific subnet, when I pinged from the hostA it can be seen from the screenshot above that the packets were captured. This shows that the sniffer successfully captured the outgoing packets that matched the filter.

Task 1.2:

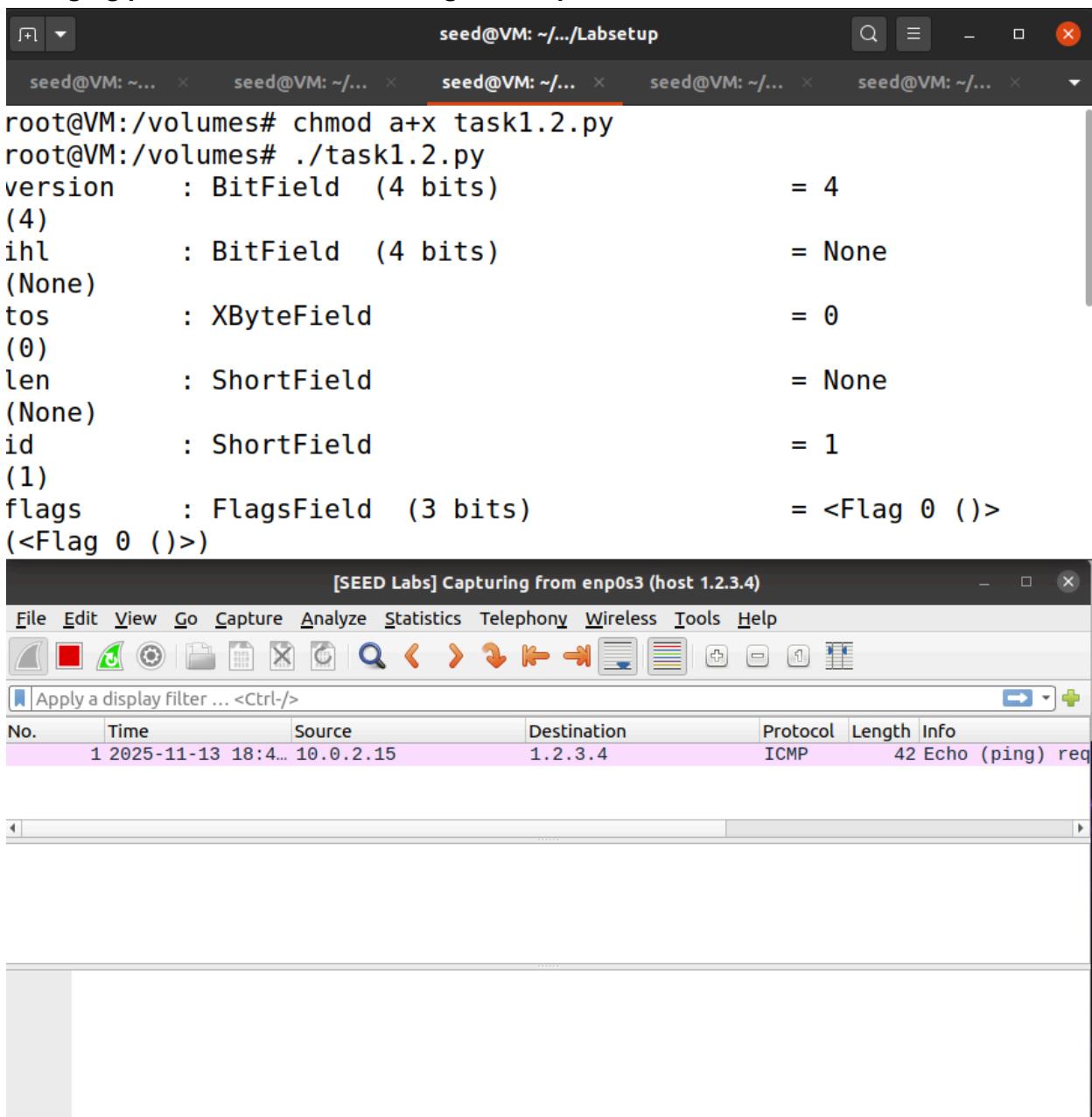
Code for task 1.2 screenshot:

```
#!/usr/bin/env python3
from scapy.all import *
a = IP()
a.dst = '1.2.3.4'
b = ICMP()
p = a/b

ls(a)

send(p)
```

Changing permissions and executing the script:



```
root@VM:/volumes# chmod a+x task1.2.py
root@VM:/volumes# ./task1.2.py
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 ()>)

[SEED Labs] Capturing from enp0s3 (host 1.2.3.4)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl->
No. Time Source Destination Protocol Length Info
1 2025-11-13 18:4... 10.0.2.15 1.2.3.4 ICMP 42 Echo (ping) req
```

Explanation: As it can be seen when the script was executed 1 packet was sent to the ip address we provided. Using the Wireshark we were able to see that the packet was sent as it can be seen in the screenshot.

Task 1.3:



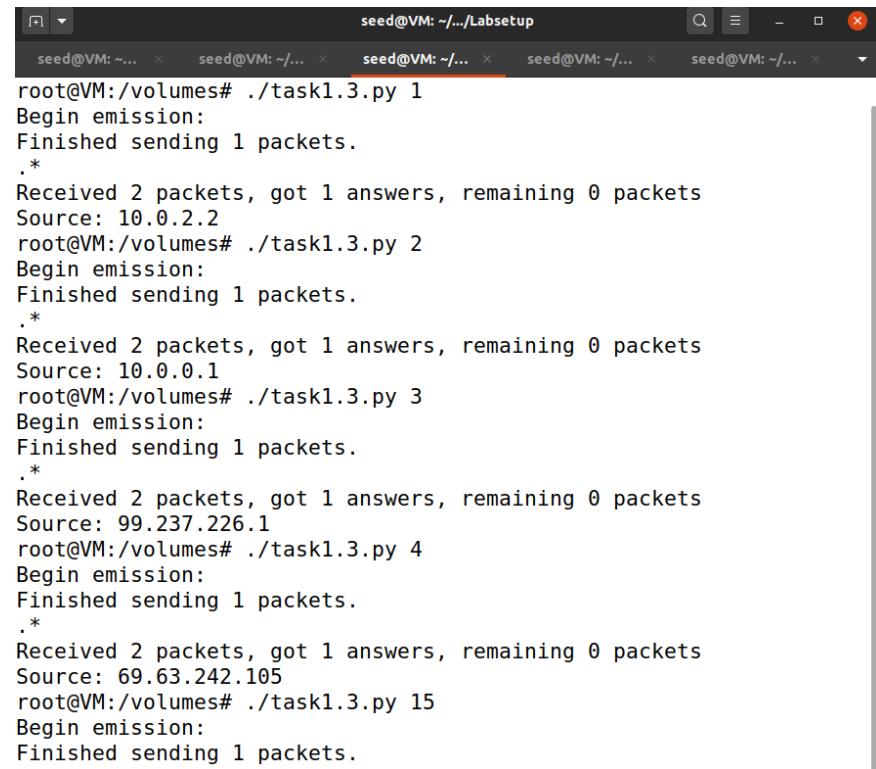
```
#!/usr/bin/env python3
from scapy.all import *
import sys

a = IP()
a.dst = '8.8.4.4'
a.ttl = int(sys.argv[1])

b = ICMP()
a = sr1(a/b)
print("Source:", a.src)
~
```

Explanation: We have set ttl as users input value so that we don't have to change it manually again and again.

Running the [task1.3.py](#) file screenshot:



```
root@VM:/volumes# ./task1.3.py 1
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 10.0.2.2
root@VM:/volumes# ./task1.3.py 2
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 10.0.0.1
root@VM:/volumes# ./task1.3.py 3
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 99.237.226.1
root@VM:/volumes# ./task1.3.py 4
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 69.63.242.105
root@VM:/volumes# ./task1.3.py 15
Begin emission:
Finished sending 1 packets.
```

Explanation: after running the program file multiple times, at ttl 15, I was able to get the dst address I provided as a source.

Task 1.4:

Code for sniffing and spoofing screenshot:

```
seed@VM: ~... × seed@VM: ~/... × seed@VM: ~/... × seed@VM: ~/... × seed@VM: ~/... ×
#!/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'

pkt = sniff(iface='br-f8ec64319f28', filter=filter, prn=spoof_pkt)
"task1.4.py" 30L, 855C written                                28,18          All
```

Running [task1.4.py](#) file:

```
seed@VM: ~/.../Labsetup
seed@VM: ~... × seed@VM: ~/... × seed@VM: ~/... × seed@VM: ~/... × seed@VM: ~/... ×
root@VM:/volumes# chmod a+x task1.4.py
root@VM:/volumes# ./task1.4.py
```

Pinging 1.2.3.4 from HostB shell:

```
root@88ba12b40154:/# 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=52.8 ms
^C
--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 52.778/52.778/52.778/0.000 ms
root@88ba12b40154:/#
```

Output for pinging 1.2.3.4 in attackers shell:

```
seed@VM: ~... × seed@VM: ~/... × seed@VM: ~/... × seed@VM: ~/... × seed@VM: ~/... ×
root@VM:/volumes# chmod a+x task1.4.py
root@VM:/volumes# ./task1.4.py
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.6
|
```

Explanation: When I pinged 1.2.3.4 from the user container, my program on the attacker's VM immediately detected the ICMP echo request on the LAN.

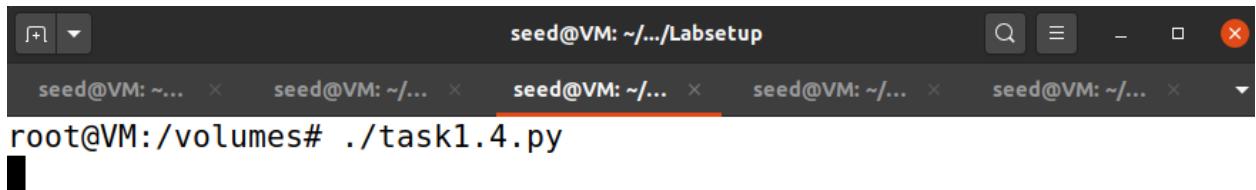
Changing the host to 10.9.0.99 and running experiment again:

```
#filter = 'icmp and host 1.2.3.4'
filter = 'icmp and host 10.9.0.99'
pkt = sniff(iface='br-f8ec64319f28', filter=filter, prn=spoof_pkt)
:w
```

HostB's shell screenshot:

```
root@88ba12b40154:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.6 icmp_seq=1 Destination Host Unreachable
From 10.9.0.6 icmp_seq=5 Destination Host Unreachable
From 10.9.0.6 icmp_seq=6 Destination Host Unreachable
From 10.9.0.6 icmp_seq=7 Destination Host Unreachable
From 10.9.0.6 icmp_seq=8 Destination Host Unreachable
From 10.9.0.6 icmp_seq=9 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
10 packets transmitted, 0 received, +6 errors, 100% packet loss, ti
me 9197ms
pipe 4
root@88ba12b40154:/#
```

Attackers shell screenshot:



```
seed@VM: ~... × seed@VM: ~... × seed@VM: ~... × seed@VM: ~... × seed@VM: ~... ×
root@VM:/volumes# ./task1.4.py
```

Explanation: When I pinged 10.9.0.99, the request never reached the attackers VM's sniffer because the ARP lookup for 10.9.0.99 failed as there was no device on the LAN claimed by the IP. Since the ICMP request was never sent on the network, the program on the attackers shell didn't print anything and it resulted in the destination host Unreachable.

Changing the host to 8.8.8.8 and running experiment again:

```
#filter = 'icmp and host 1.2.3.4'
#filter = 'icmp and host 10.9.0.99'
filter = 'icmp and host 8.8.8.8'
pkt = sniff(iface='br-f8ec64319f28', filter=filter, prn=spoof_pkt)
"task1.4.py" 31L, 924C written 30,32 Bot
```

HostB's shell screenshot:

```
root@88ba12b40154:/# 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=115 time=15.0 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=49.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=18.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=20.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=35.8 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 14.967/26.677/49.024/11.942 ms
root@88ba12b40154:/#
```

Attackers shell screenshot:

```
root@VM:/volumes# ./task1.4.py
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.6
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.6
Original Packet.....
Source IP : 10.9.0.6
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.6
```

Explanation: When pinging 8.8.8.8, my spoofing program captured the ICMP request sent from 10.9.0.6 to the router and immediately forged a reply, so the ping output showed duplicated responses (“DUP!”), one fake and one real. The attacker VM correctly displayed the packet’s source (10.9.0.6) and destination (8.8.8.8), confirming that the request passed through the LAN and triggered the spoof.

Task 2:

2.1:

Screenshot of sniffer.c code:



```
#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;

    // Step 1: Open live pcap session on NIC with name eth3
    // Students needs to change "eth3" to the name
    // found on their own machines (using ifconfig).
    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
```

1,17

Top

Explanation: This program is a simple packet sniffer built using pcap library. First, it opens a live packet-capture session on a network interface. After that, it compiles a BPF filter so that only ICMP packets are captured. If the filter is successfully applied, the program enters a loop where pcap automatically captures packers and calls the got_packet() callback for each one. Inside this callback, the program currently just prints "Got a packet", showing that the sniffer is correctly detecting ICMP traffic.

Compiling and running the file screenshot:

Pinging 8.8.8.8:

```
[11/13/25]seed@VM:~/.../volumes$ ping -c 3 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=16.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=16.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=18.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 16.033/17.030/18.619/1.135 ms
[11/13/25]seed@VM:~/.../volumes$ █
```

Explanation: As it can be seen from the screenshots above. I first compiled and ran the sniff file then on another tab, pinged the 8.8.8.8 address and after it successfully sent the packets when I checked the tab where I compiled the file the packets had been captured showing that it works.

Task 2.1A:

Question 1:

First, it basically opens a live capture session on a network interface using pcap_open_live(). Then it compiles the filter expression into BPF bytecode using pcap_compile() and installs it with pcap_setfilter() so only relevant packets are delivered. Finally, it enters a capture loop using pcap_loop() where pcap hands each captured packet to the callback function for processing.

Question 2:

A root privilege is required to set up the card in promiscuous mode and raw socket, this way we can see the whole network traffic in the interface. If we run the program without a root user, where the pcap_open_live function fails to access the device, it causes an error to the whole program and print segmentation fault. Also, packets received from the network are copied to the kernel. To listen on the socket and capture packets, it is necessary to access and modify something in kernel space, which requires root permissions.

Question 3:

```
// found on their own machines (using ifconfig).
handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
```

Explanation: If using a non-zero int for the third argument in the above code snippet, the promiscuous mode is turned on. Otherwise, it is turned off.