Assignment 1 Report
Packet Sniffing and Spoofing Lab

Demarcus Simmons
University of West Florida: CNT4403

Sep 15, 2023

Table of Contents

Introduction

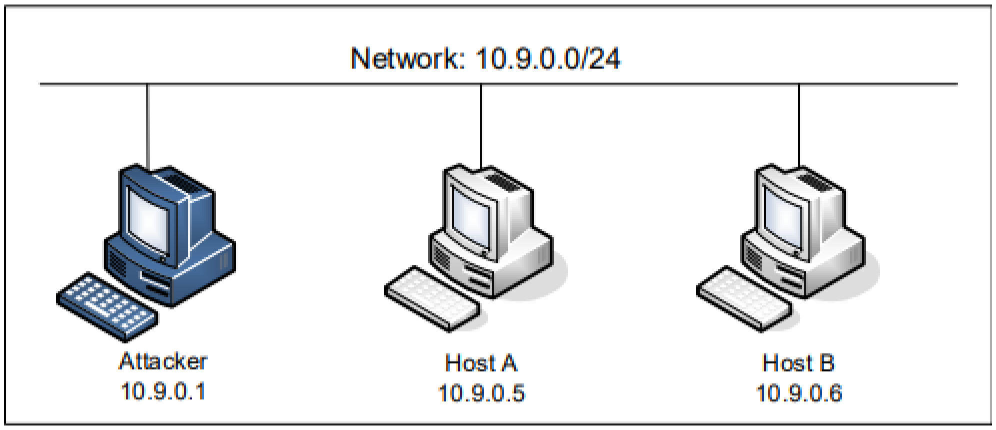
For this assignment, I will be gaining hands-on experience with packet sniffing and spoofing. There are many packet sniffing and spoofing tools, such as Wireshark, Tcpdump, Netwox, Scapy, etc. I'll be doing this assignment alone. The goal is to learn more about these two threats, which are major threats in network communication. Understanding these two threats is the basis for understanding security measures in networking.

Materials and Methods

The materials used is this assignment are:

- Python programming language
- Scapy (Python library)
- VMware (Unbuntu)
- Docker

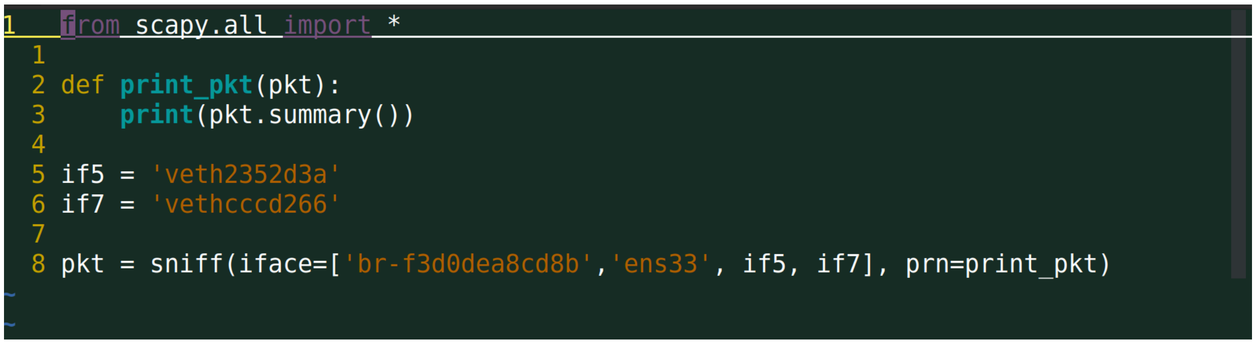
The assignment will be completed in a series of tasks, which I will complete and document in order. Docker is used to set up the lab environment and all the containers involved. The following is an image of the lab's environment setup:



Task Set 1: Using Scapy to Sniff and Spoof Packets

Many tools can be used to do sniffing and spoofing, but most of them only provide fixed functionalities. Scapy is different: it can be used not only as a tool, but also as a building block to construct other sniffing and spoofing tools, i.e., we can integrate the Scapy functionalities into our own program. In this set of tasks, we will use Scapy for each task.

Task 1.1: Sniffing Packets



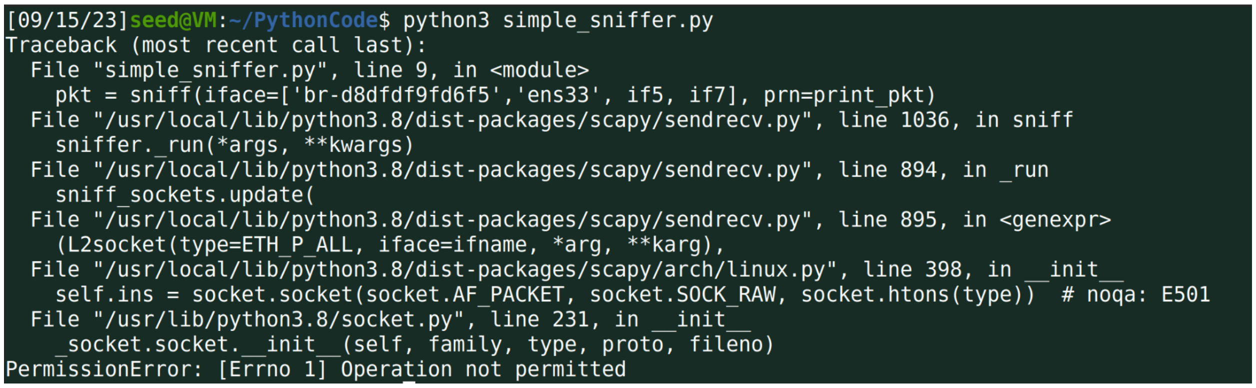
```
from scapy.all import *

def print_pkt(pkt):
    print(pkt.summary())

if5 = 'veth2352d3a'
if7 = 'vethcccd266'

pkt = sniff(iface=['br-f3d0dea8cd8b','ens33', if5, if7], prn=print_pkt)
```

Figure 1: simple_sniffer.py

The above picture is a Python program for a simple sniffer. It will listen on 4 network interfaces including all of the containers and the Wi-Fi network interface. For each captured packet it will call the print_pkt function which will simply output a summary of the packet.

Task 1.1A:

I first ran the program without root privileges and the following is what the output was:

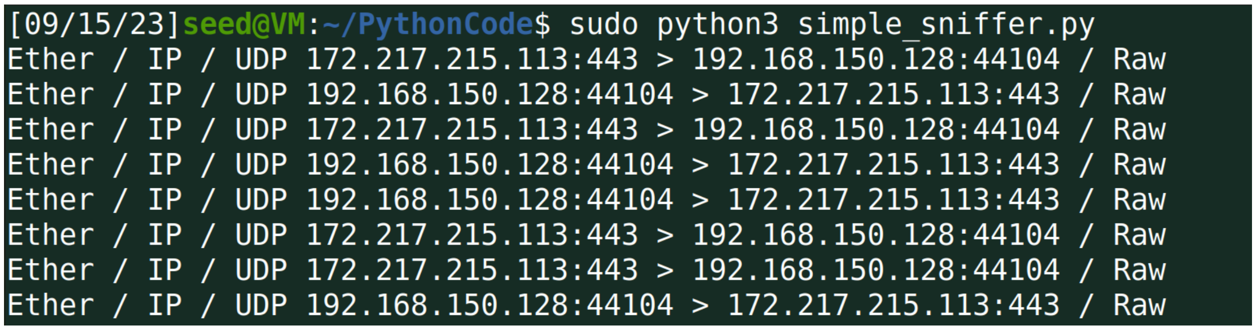


```
[09/15/23]seed@VM:~/PythonCode$ python3 simple_sniffer.py
Traceback (most recent call last):
  File "simple_sniffer.py", line 9, in <module>
    pkt = sniff(iface=['br-d8dfdf9fd6f5','ens33', if5, if7], prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 894, in _run
    sniff_sockets.update(
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 895, in <genexpr>
    (L2socket(type=ETH_P_ALL, iface=ifname, *arg, **karg),
  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
```

Figure 2: Operation not permitted error

From the results, in order to capture packets we'll have to run the program in administrative mode otherwise you won't be able to sniff packets on the network.

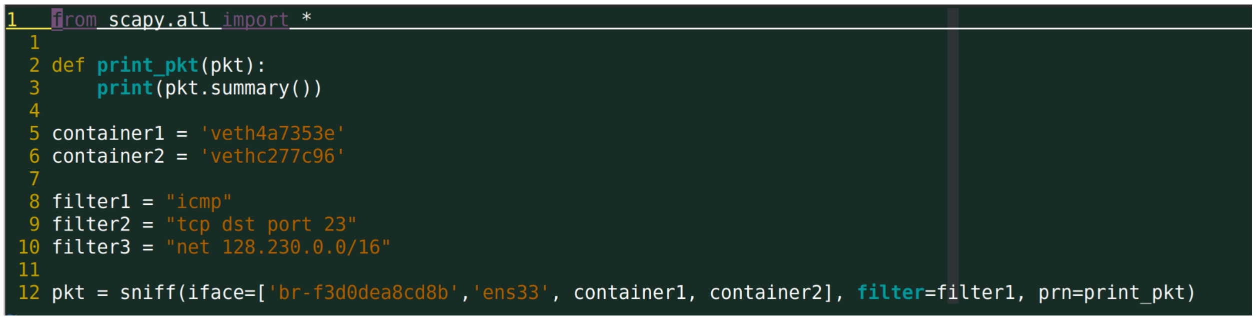
Next, I ran the program using sudo to execute it using root privileges and I was able to get the summary of a few captured packages. The output is as follows:



```
[09/15/23]seed@VM:~/PythonCode$ sudo python3 simple_sniffer.py
Ether / IP / UDP 172.217.215.113:443 > 192.168.150.128:44104 / Raw
Ether / IP / UDP 192.168.150.128:44104 > 172.217.215.113:443 / Raw
Ether / IP / UDP 172.217.215.113:443 > 192.168.150.128:44104 / Raw
Ether / IP / UDP 192.168.150.128:44104 > 172.217.215.113:443 / Raw
Ether / IP / UDP 192.168.150.128:44104 > 172.217.215.113:443 / Raw
Ether / IP / UDP 172.217.215.113:443 > 192.168.150.128:44104 / Raw
Ether / IP / UDP 172.217.215.113:443 > 192.168.150.128:44104 / Raw
Ether / IP / UDP 192.168.150.128:44104 > 172.217.215.113:443 / Raw
```

Figure 3: Capturing and outputting packages

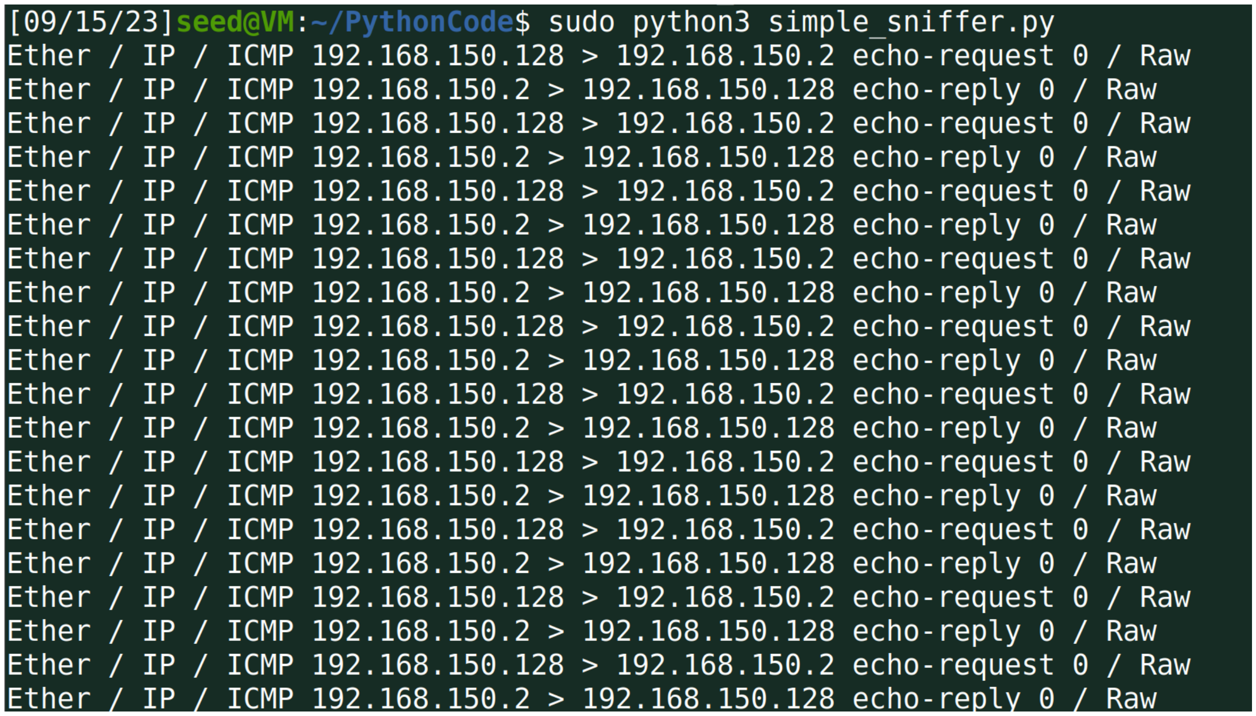
Task 1.1B:



```
from scapy.all import *

def print_pkt(pkt):
    print(pkt.summary())

container1 = 'veth4a7353e'
container2 = 'vethc277c96'

filter1 = "icmp"
filter2 = "tcp dst port 23"
filter3 = "net 128.230.0.0/16"

pkt = sniff(iface=['br-f3d0dea8cd8b','ens33', container1, container2], filter=filter1, prn=print_pkt)
```

Figure 4: simple_sniffer.py with filters

The above code was used to filter the types of packets captured by the sniffer program using each of the filter# variables one at a time. The filter parameter is used for this purpose and allows you to specify the type of packets you want the sniffer to collect. The output after using filter1 set as the argument for the filter parameter is shown below.



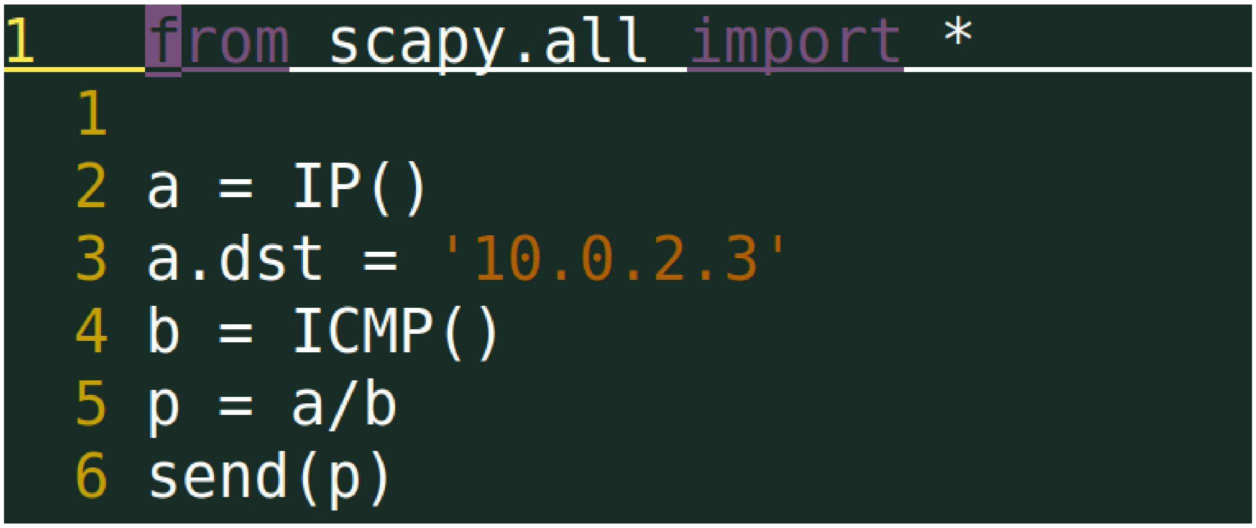
```
[09/15/23]seed@VM:~/PythonCode$ sudo python3 simple_sniffer.py
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 / Raw
Ether / IP / ICMP 192.168.150.2 > 192.168.150.128 echo-reply 0 / Raw
```

Figure 4.1: Output after using filter1

The output was captured by me connecting to the attacker container using the docksh command and pinging the default gateway. The output for using filter2 as the argument is shown below.

Task 1.2: Spoofing ICMP packets

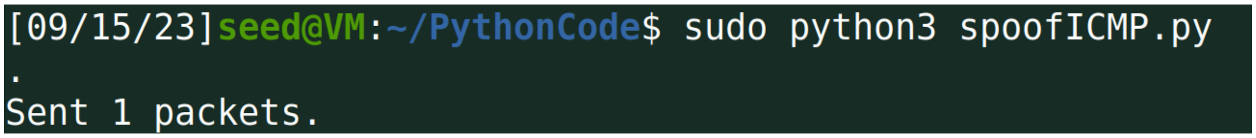
As a packet spoofing tool, Scapy allows us to set the fields of IP packets to arbitrary values. The objective
of this task is to spoof IP packets with an arbitrary source IP address. We will spoof ICMP echo request packets, and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. The following code demonstrates creating a spoofed ICMP packet with an arbitrary source ip address:



```python
from scapy.all import *

a = IP()
a.dst = '10.0.2.3'
b = ICMP()
p = a/b
send(p)
```

Figure 5: spoofICMP.py

The output after running the program is as follows;

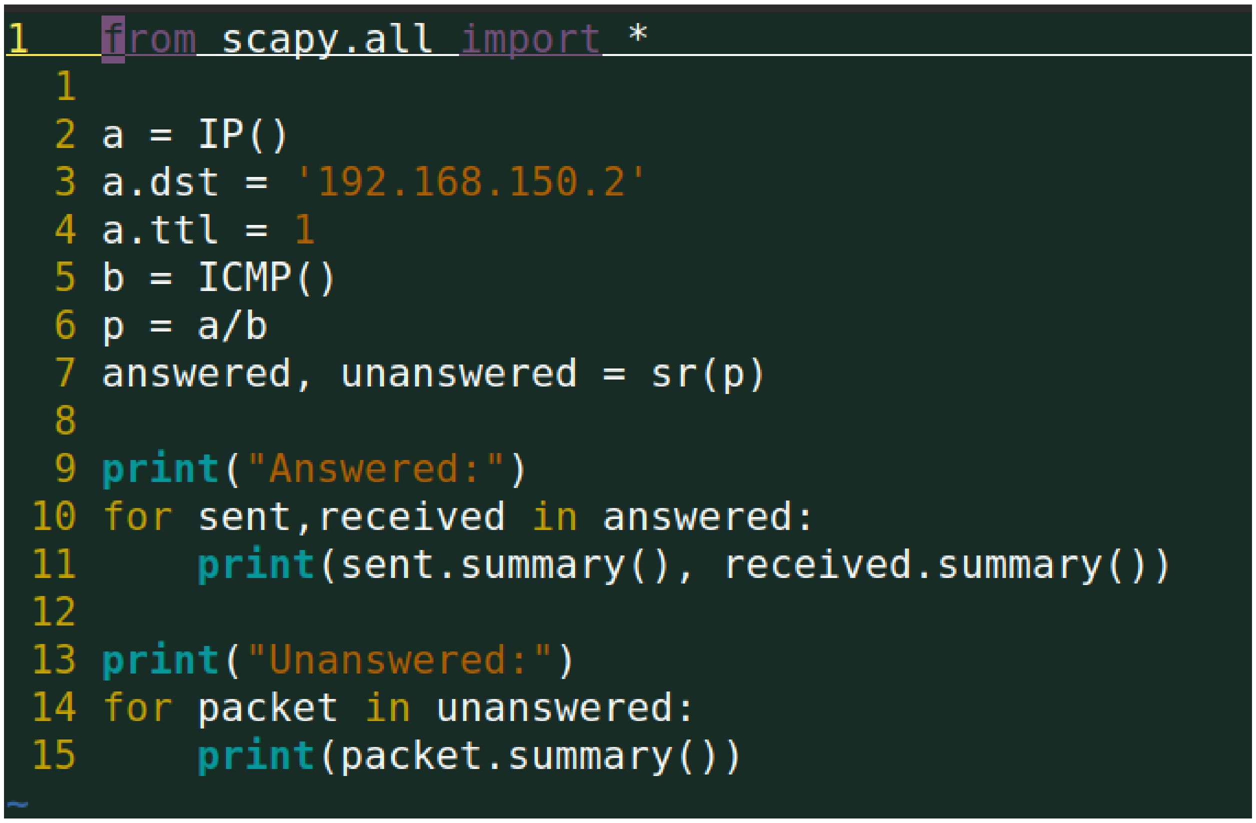


```
[09/15/23]seed@VM:~/PythonCode$ sudo python3 spoofICMP.py
.
Sent 1 packets.
```

Figure 5.1: Output showing ICMP packet was sent successfully

Task 1.3: Traceroute

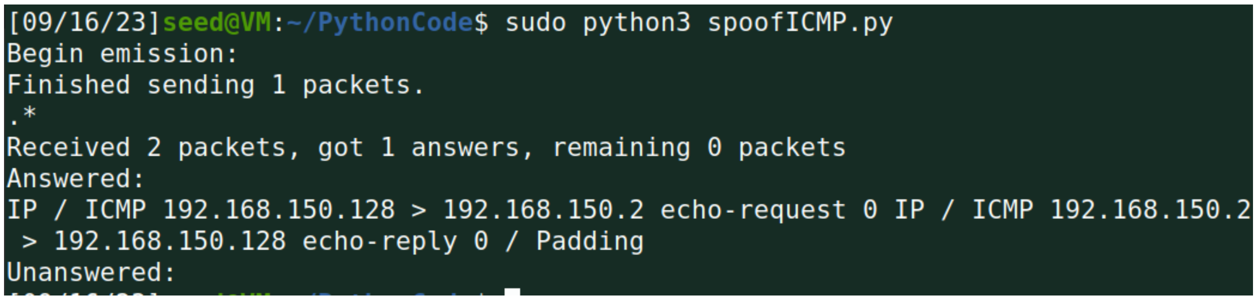
The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination.



```python
from scapy.all import *

a = IP()
a.dst = '192.168.150.2'
a.ttl = 1
b = ICMP()
p = a/b
answered, unanswered = sr(p)

print("Answered:")
for sent,received in answered:
    print(sent.summary(), received.summary())

print("Unanswered:")
for packet in unanswered:
    print(packet.summary())
```

Figure 6: spoofICMP.py with ttl

For this task I wrote a simple python program, Figure 6, that will send a ICMP packet to the router which is just 1 hop away. The output. Shown in 6.1, shows the result of a successful echo reply.



```
[09/16/23]seed@VM:~/PythonCode$ sudo python3 spoofICMP.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Answered:
IP / ICMP 192.168.150.128 > 192.168.150.2 echo-request 0 IP / ICMP 192.168.150.2
 > 192.168.150.128 echo-reply 0 / Padding
Unanswered:
```

Figure 6.1: Successful echo reply

The idea is quite straightforward: just send a packet (any type) to the destination, with its Time-To-Live (TTL) field set to 1 first. This packet will be dropped by the first router, which will send us an ICMP error message, telling us that the time-to-live has exceeded. That is how we get the IP address of the first router. We then increase our TTL field to 2, send out another packet, and get the IP address of the second router. We will repeat this procedure until our packet finally reach the destination. It should be noted that this experiment only gets an estimated result, because in theory, not all these packets take the same route (but in practice, they may within a short period of time).

Task 1.4: Sniffing and-then Spoofing

Conclusion