CS 491
Assignment 3

665336275 **Akshay Kataria**

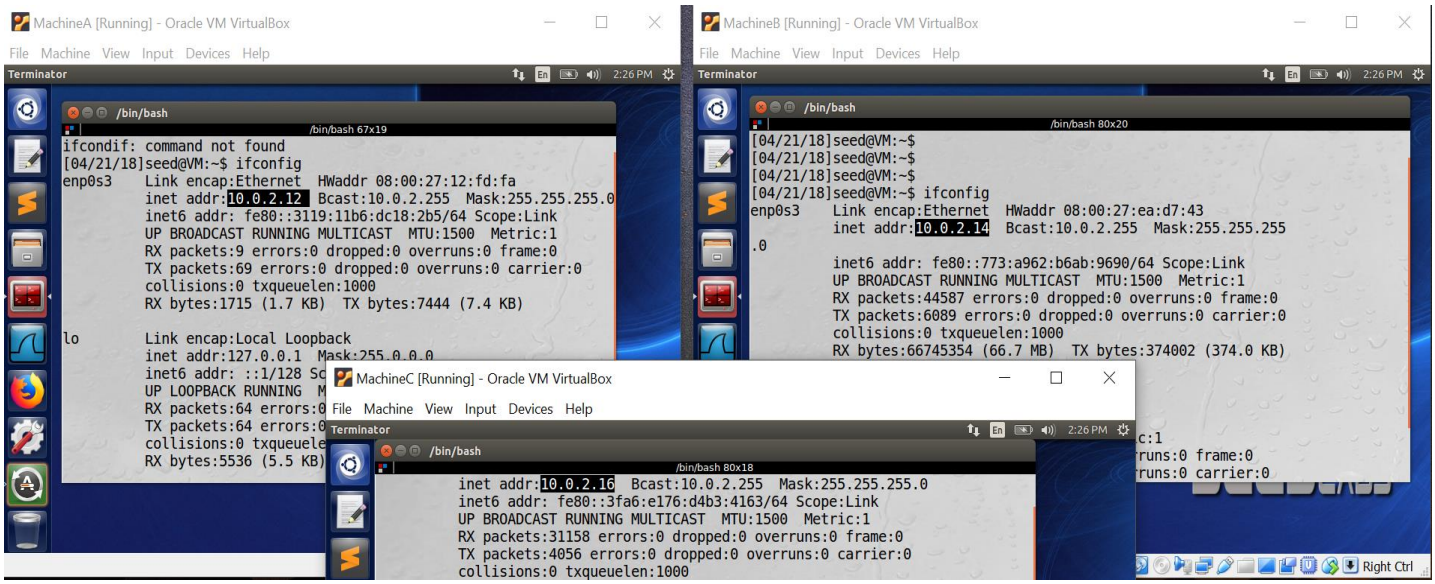
Some fix things that won't change during the entire lab

- Machine A's ip address: 10.0.2.12
- Machine B's ip address: 10.0.2.14
- Machine C's ip address: 10.0.2.16

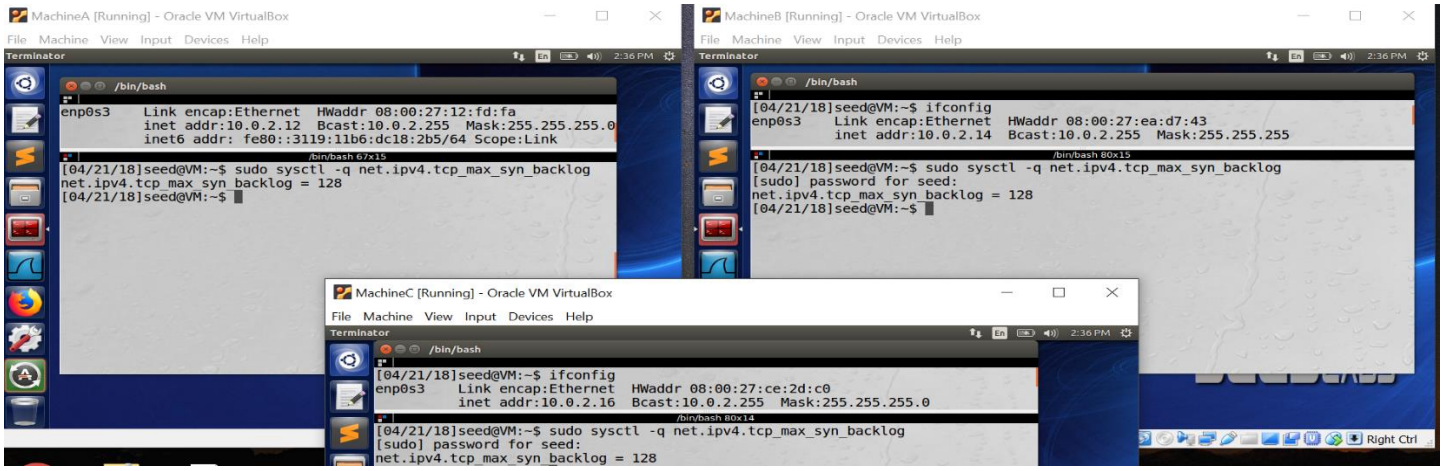
IP address: 10.0.2.12	IP address: 10.0.2.14	IP address: 10.0.2.16
Attacker (MachineA)	Victim (MachineB)	Server/Observer (MachineC)

Task 1: SYN flooding attack

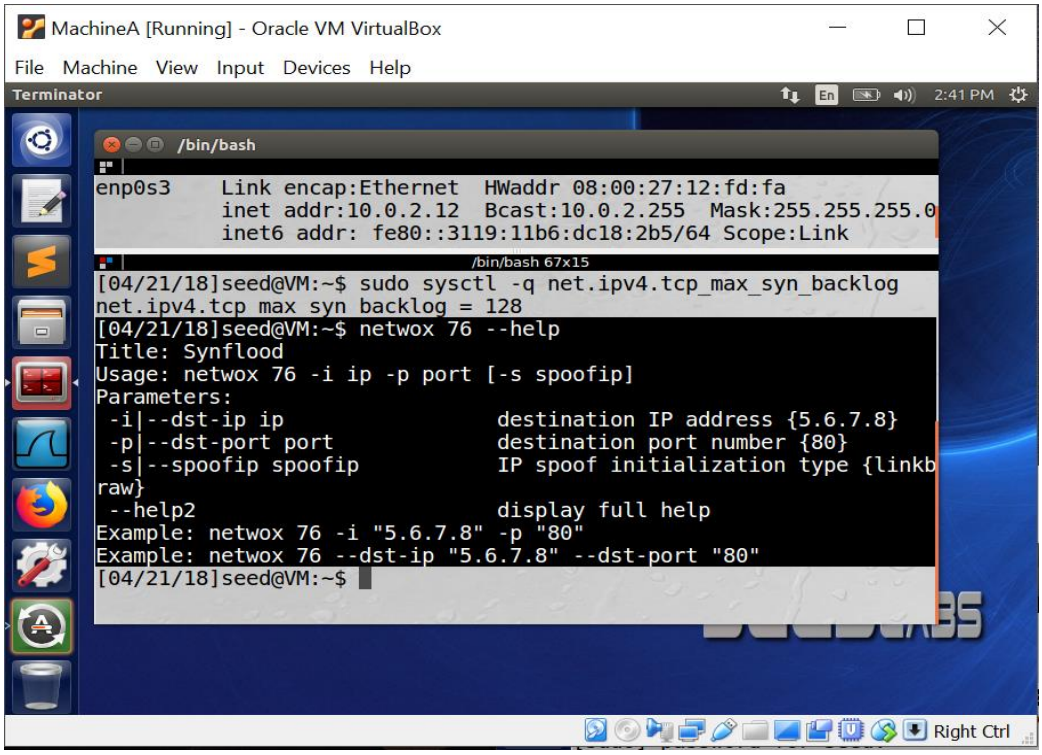
SYN flooding attack is a form of the DDOS (Distributed Denial of Service) attack in which attacker send's too many SYN requests to the victim's machine but it has no intention of completing the 3-way handshake procedure. In this way, attacker is disrupting the AVAILABILITY field of the CIA triad.



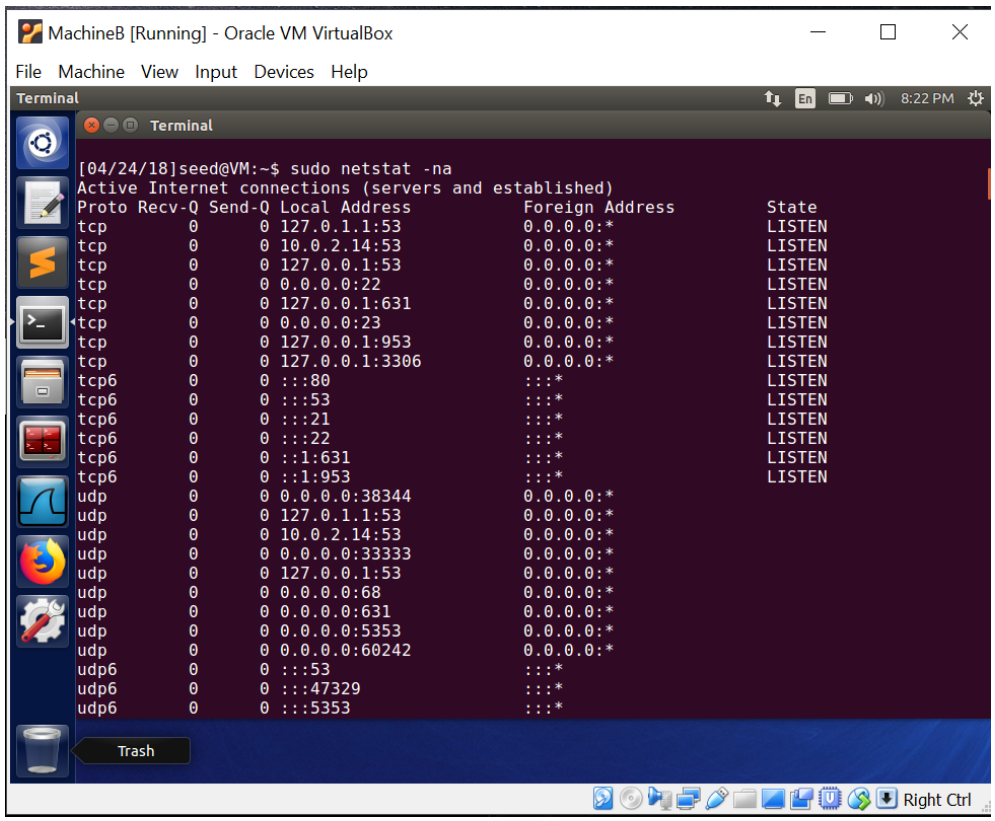
Obs: We first figured out the ip address of the 3 virtual machines with the help of the "ifconfig" command.



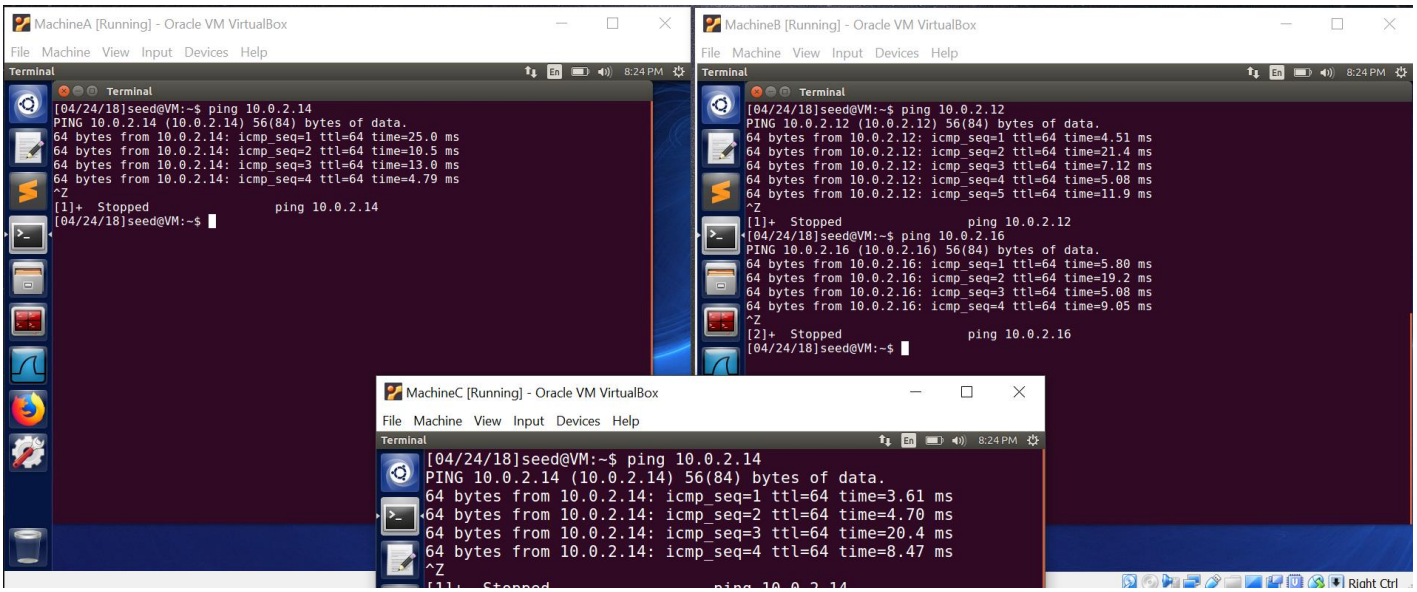
Obs: We can observe the size of the queue for the half-opened connections with the help of the specified command. "*$sudo sysctl -q net.ipv4.tcp_max_syn_backlog*". By default, the size of the queue is 128.



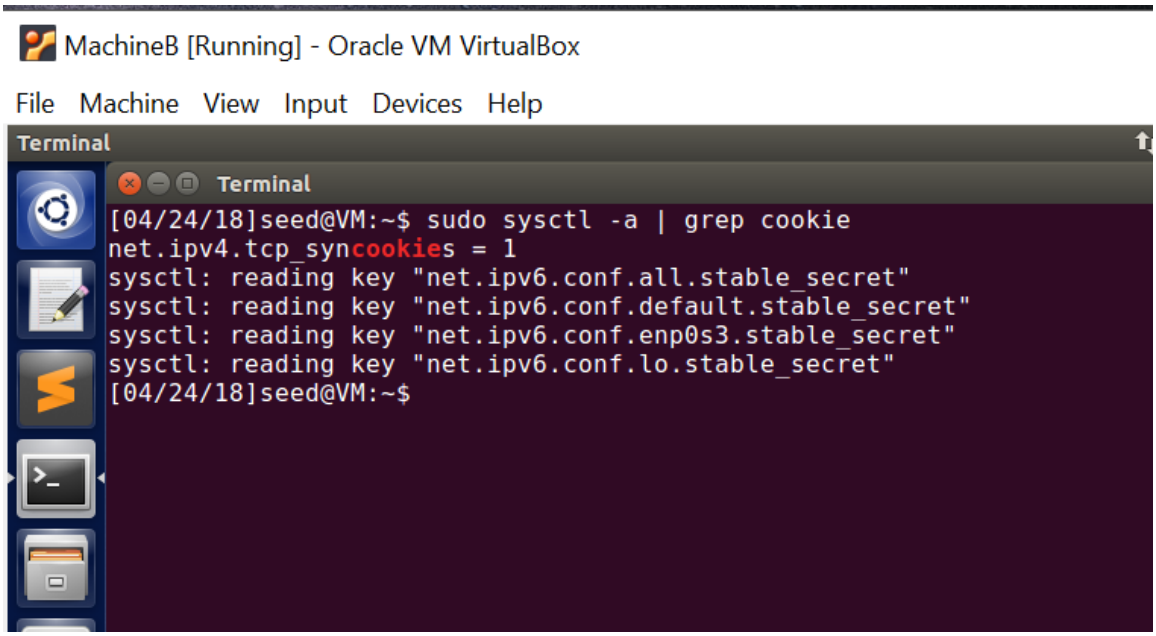
Obs: SYN flooding attack is imitated with the help of the NETWOX tool in Linux. Number 76 in the Netwox tool is specifically used for this process. We can observe the information about the command structure with the help of the *"$netwox 76 –help"*.



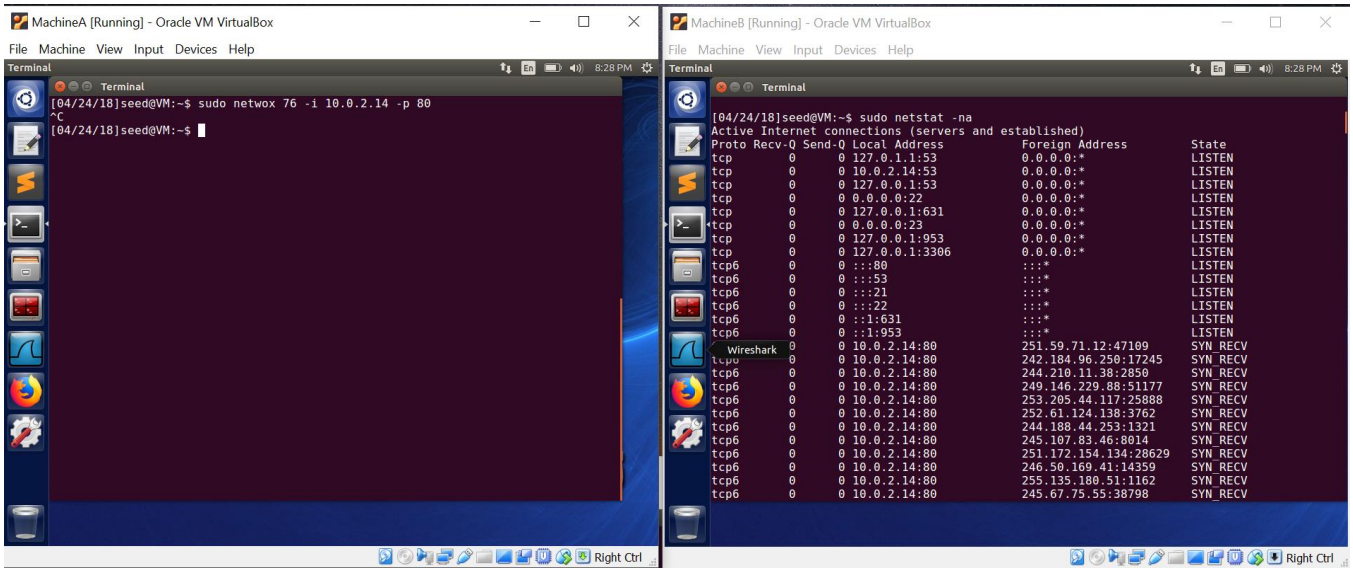
Obs: In machine (Victim's machine), we first check the status of the queue usage with the help of the "netstat -na" command. Here we can't find SYN-RECV for any of the connections which is basically the indication of server waiting for the final ACK packet to establish the connection. In SYN flooding attack, you can generally find all the connections with SYN-RECV state. From this we can deduce that, right now, the victim's machine is not under attack. Here for all the machines, a 3-way handshake is being made and thus they land up in a LISTEN state.



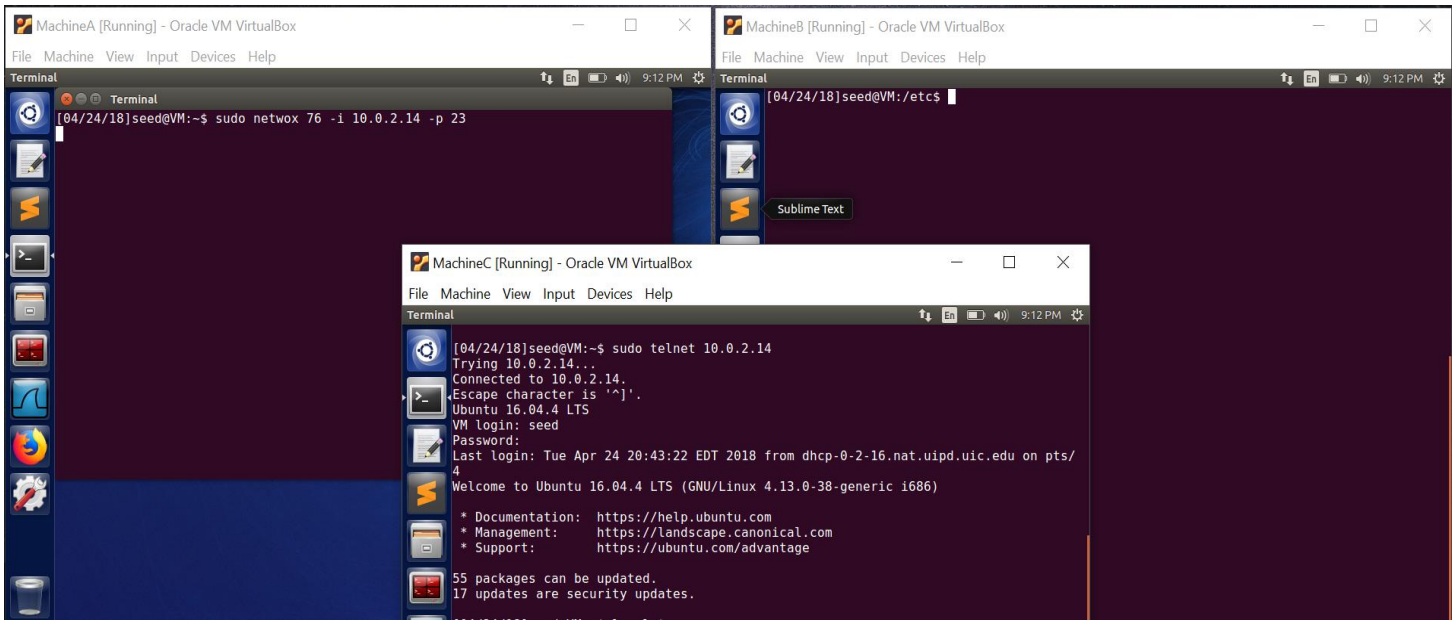
Obs: We tried to ping machineB from machineA and vice-versa. Also, we tried to ping from machineB to machineC and vice-versa. We just want to ensure that the machines are able to communicate with each other.



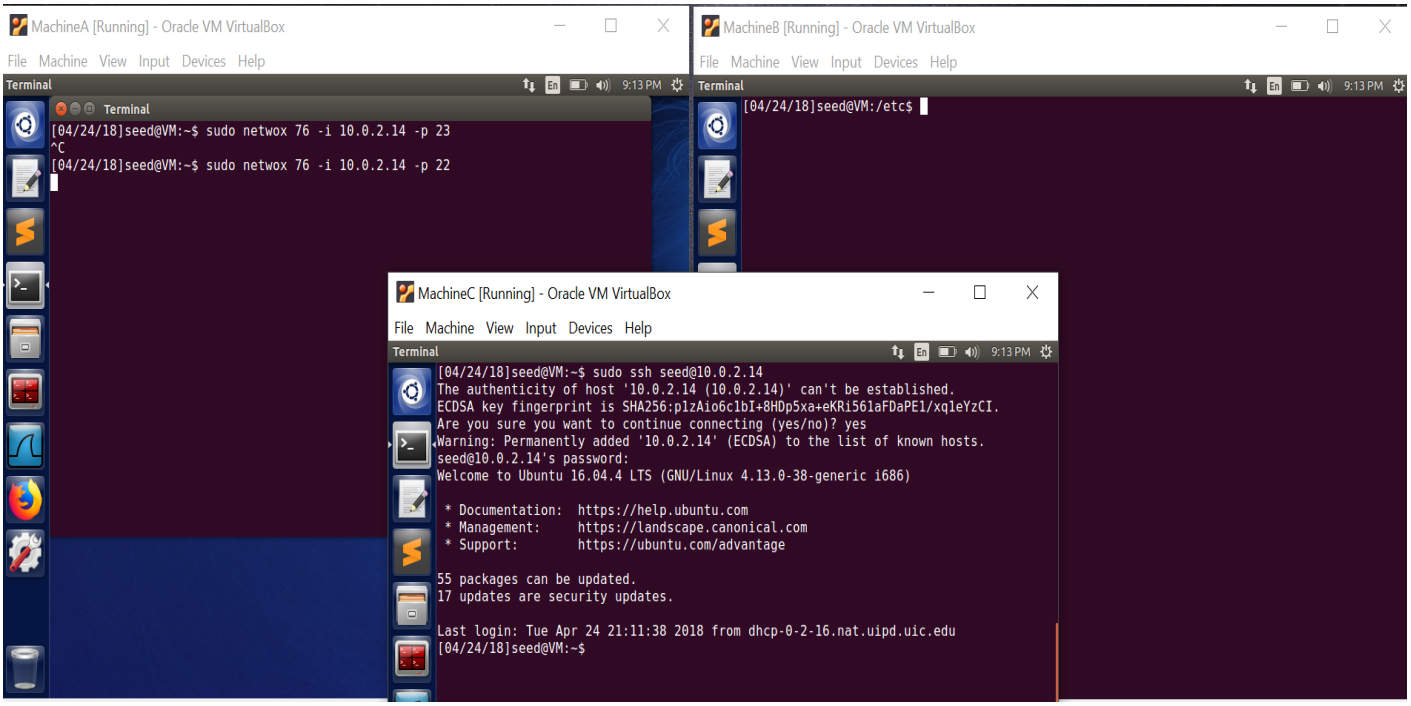
Obs: We can use *"sudo sysctl -a | grep cookie"* command to know the flag status for the SYN Cookie. From this we can observe that, SYN cookies are switched on by default. This is the reason the attacker won't be able to pull up a SYN flooding attack.



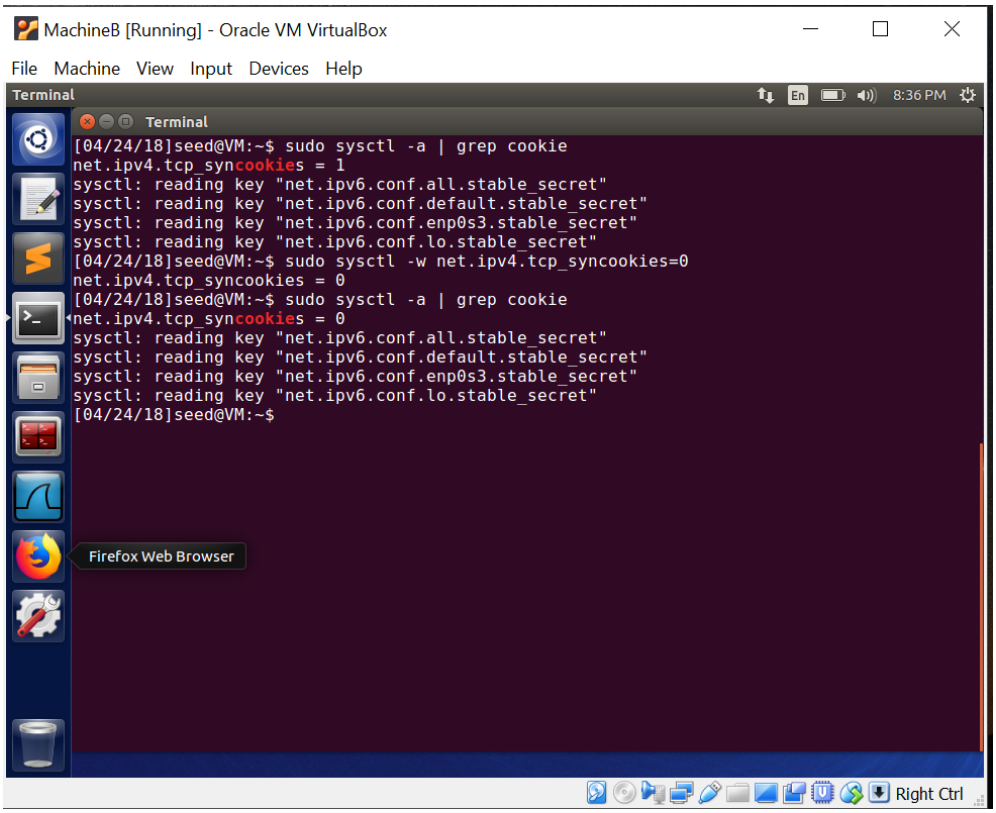
Obs: The above output shows a successful attempt of the SYN flooding attack being pulled up by attacker (machineA) on victim (machineB) even when the SYN cookie is switched on. We tried attacking the victim with the help of the *"sudo netwox 76 -i 10.0.2.14 -p 80"* command and we able to disrupt the resources through port 80 (HTTP). If you carefully look at machineB's terminal, you can see that connections are landing up in the SYN-RECV state. This means that SYN cookies are not playing their role in blocking such connections. This is the reason we will try to prove SYN cookies efficiency while establishing a *telnet* and *ssh* connections.



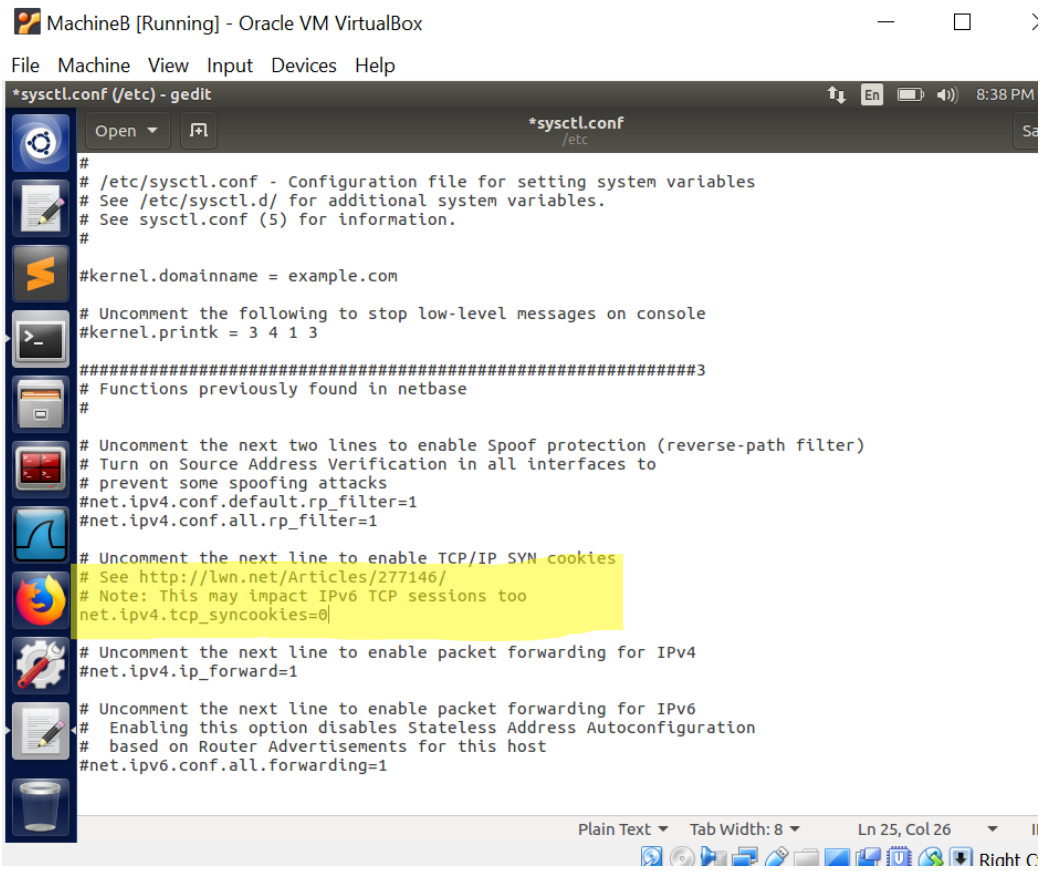
Obs: We can observe that we are pulling up a SYN flooding attack on port 23 for machineB from machineA. We can also look at the terminal on machineC and deduce that we are successfully able to establish a telnet connection from machineC to machineB. This means that SYN cookies are playing their role for the telnet connections. Now, we can also try the efficiency of SYN cookies for the SSH connections.



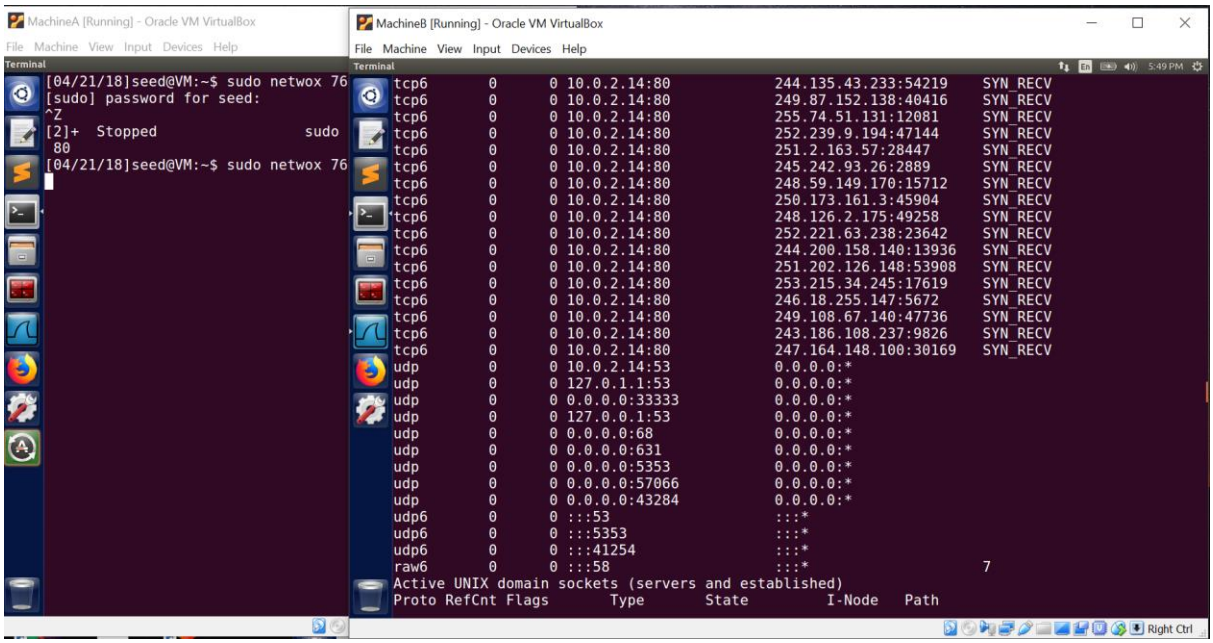
Obs: We can observe that we are pulling up a SYN flooding attack on port 22 (SSH port) now for machineB from machineA. We can also look at the terminal on machineC and deduce that we are successfully able to establish a SSH connection from machineC to machineB. This means that SYN cookies are playing their role effectively for the SSH connections too.



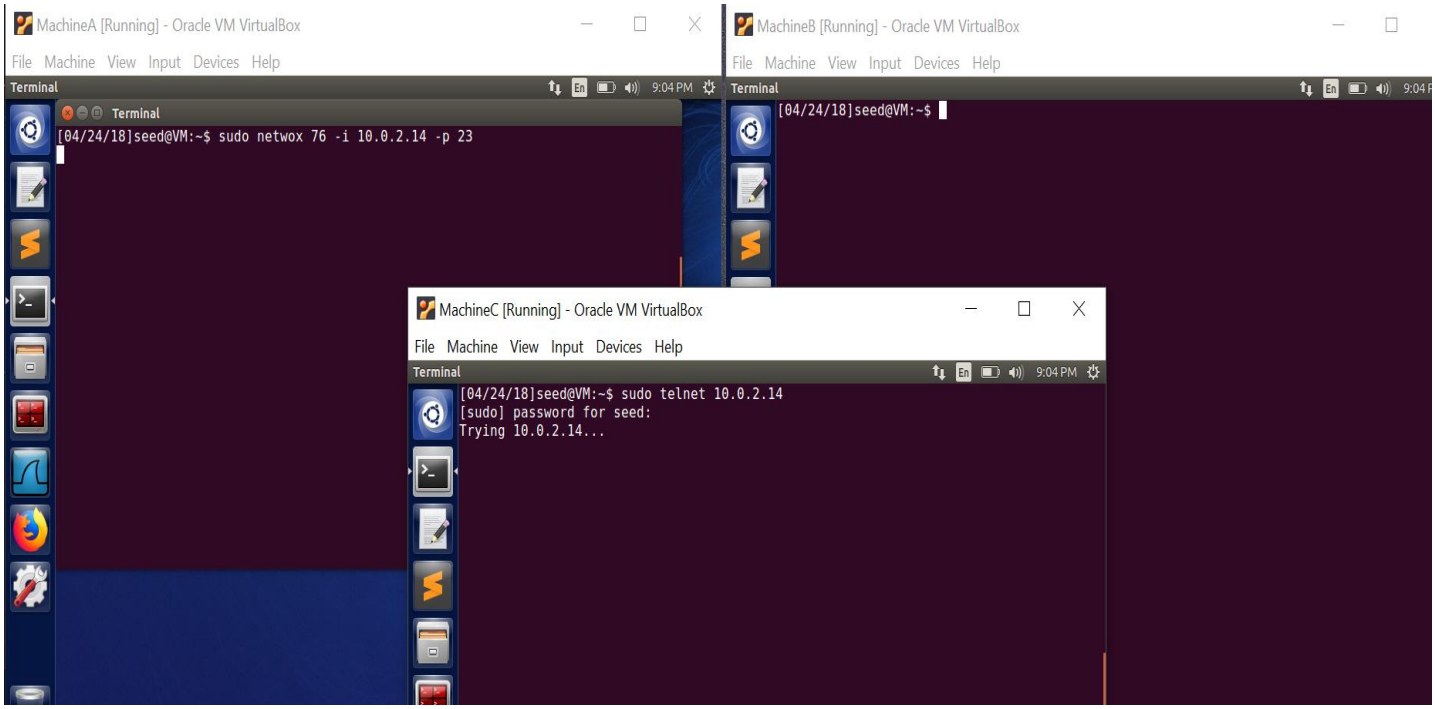
Obs: Switching off the SYN cookie protection on victim's machine (machineB) with the help of *"sudo sysctl -w net.ipv4.tcp_syncookies=0"* and then checking the status of the SYN cookie with the help of *"sudo sysctl -a | grep cookies"*.



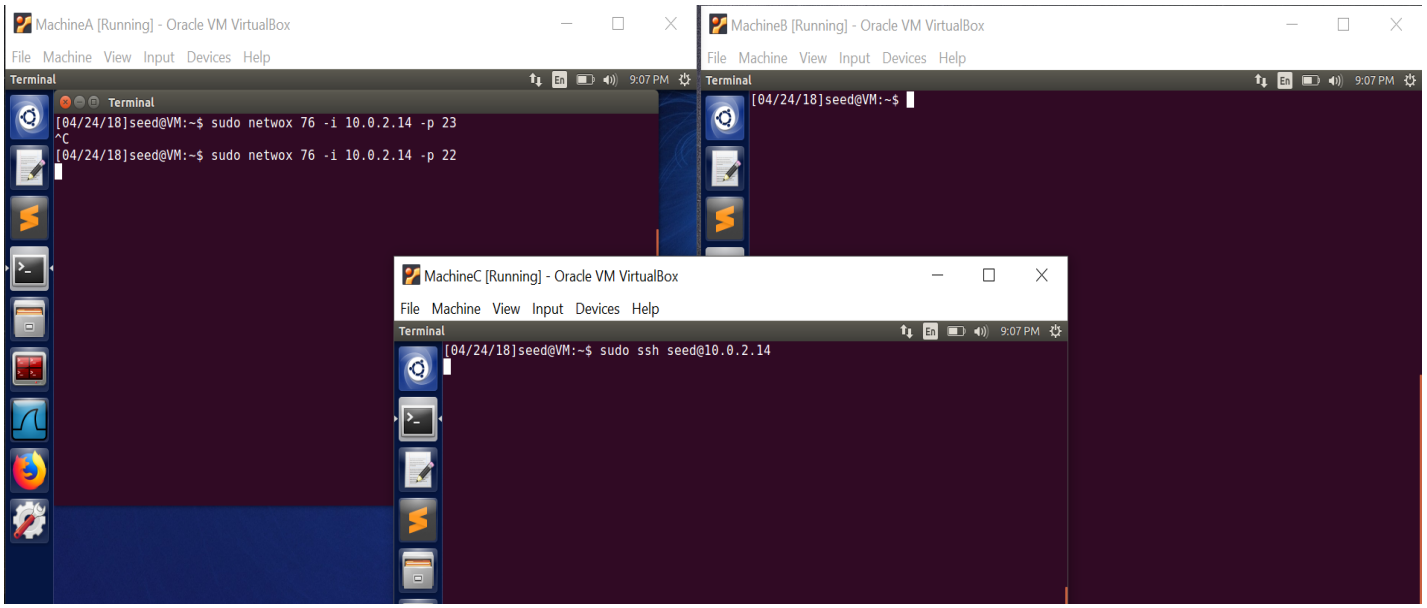
Obs: We can also switch-off the SYN cookie by going in the configuration file for the sysctl. The highlighted version shows the place where you can make the changes.



Obs: From the above screenshot, once you have switched off the SYN cookie in the victim's machine, the attacker is able to flood the victim's port 80 with half-open connections with spoofed ip's. This is being done with the help of *netwox* tool. You can observe various SYN_RECV connection states in the victim's terminal which shows that SYN flooding successfully took place. This slowed down the machineB's operation. The connection states can be viewed with the help of *"netstat -na"*. But we were receiving the same output when SYN cookie was turned on. Therefore, we will try to differentiate both the conditions (SYN Flag = 1 and SYN Flag =0) based on SSH and telnet connections.



Obs: We can observe that we are pulling up a SYN flooding attack on port 23 for machineB from machineA. We can also look at the terminal on machineC and deduce that we are not able to establish a telnet connection from machineC to machineB. This means that SYN cookies are not able playing their role for the telnet connections once they are switched off (flag = 0) which is the desired result. We can see the status as *"Trying 10.0.2.14 …"* and eventually it *"Timed Out"*. Now, we can also try the efficiency of SYN cookies for the SSH connections.

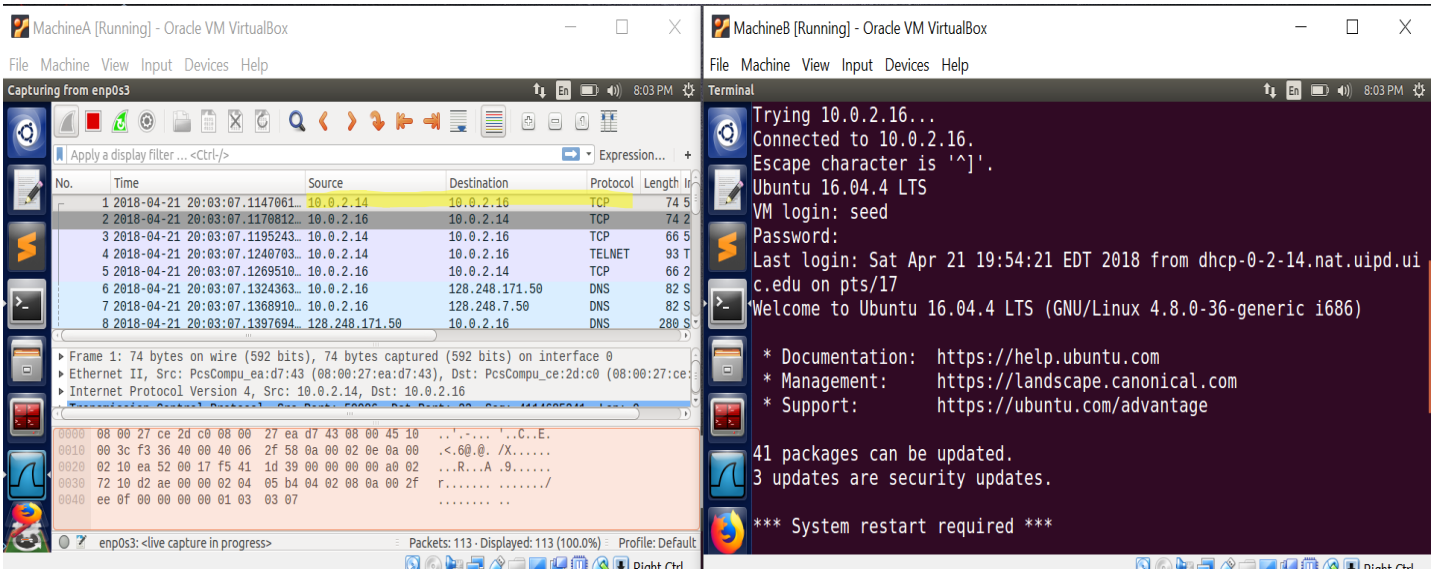


Obs: We can observe that we are pulling up a SYN flooding attack on port 22 (SSH port) for machineB from machineA. We can also look at the terminal on machineC and deduce that we are not able to establish a SSH connection from machineC to machineB. This means that SYN cookies are not able playing their role for the SSH connections once they are switched off (flag = 0) which is the desired result. We can't see any status on machineC's terminal and eventually it *"Timed Out"*.

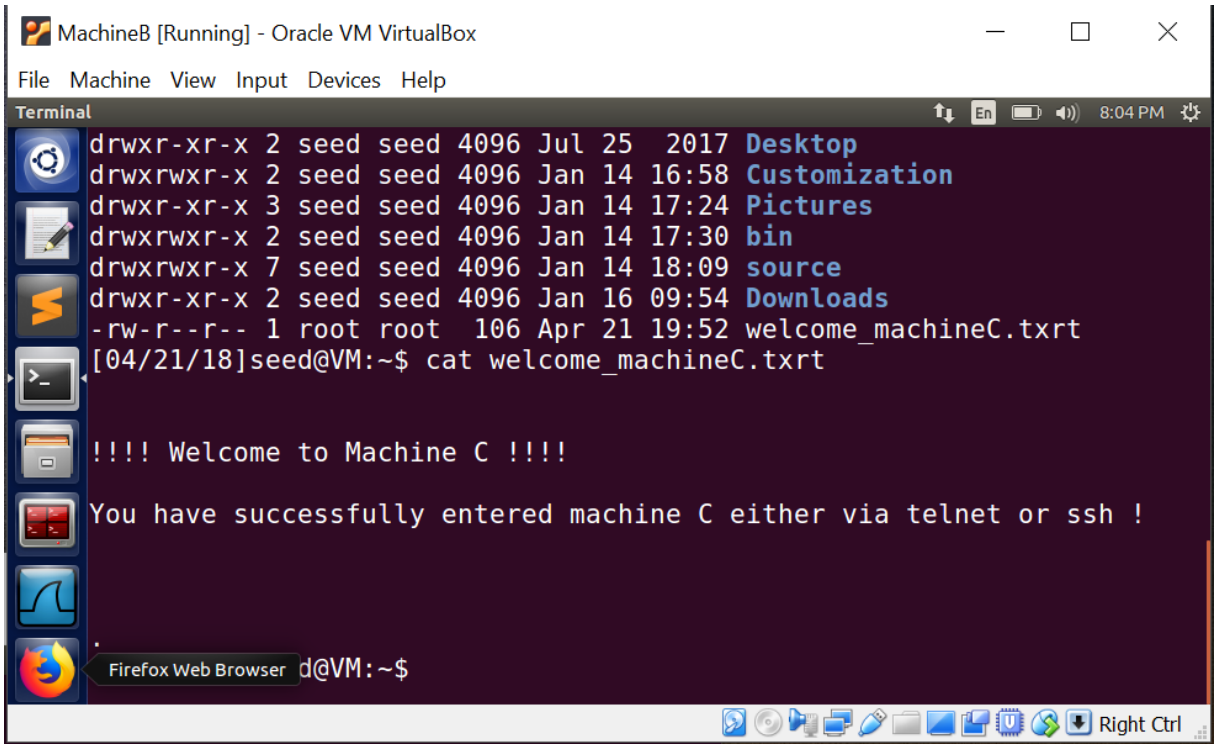
Q) Please describe why the SYN cookie can effectively protect the machine against the SYN flooding attack?
Ans: SYN cookie is basically a technique which is used to resist SYN flood attacks. As discussed in the lecture, SYN cookies ensure that the responder is stateles until initiator produced at least 2 messages. The IP address and port number of the initiator is stored in a cookie and sent to initiator. The connection will only be established when the initiator regenerates the cookie and send it back to the server/responder.

Task 2: TCP RST attacks on telnet and ssh connections

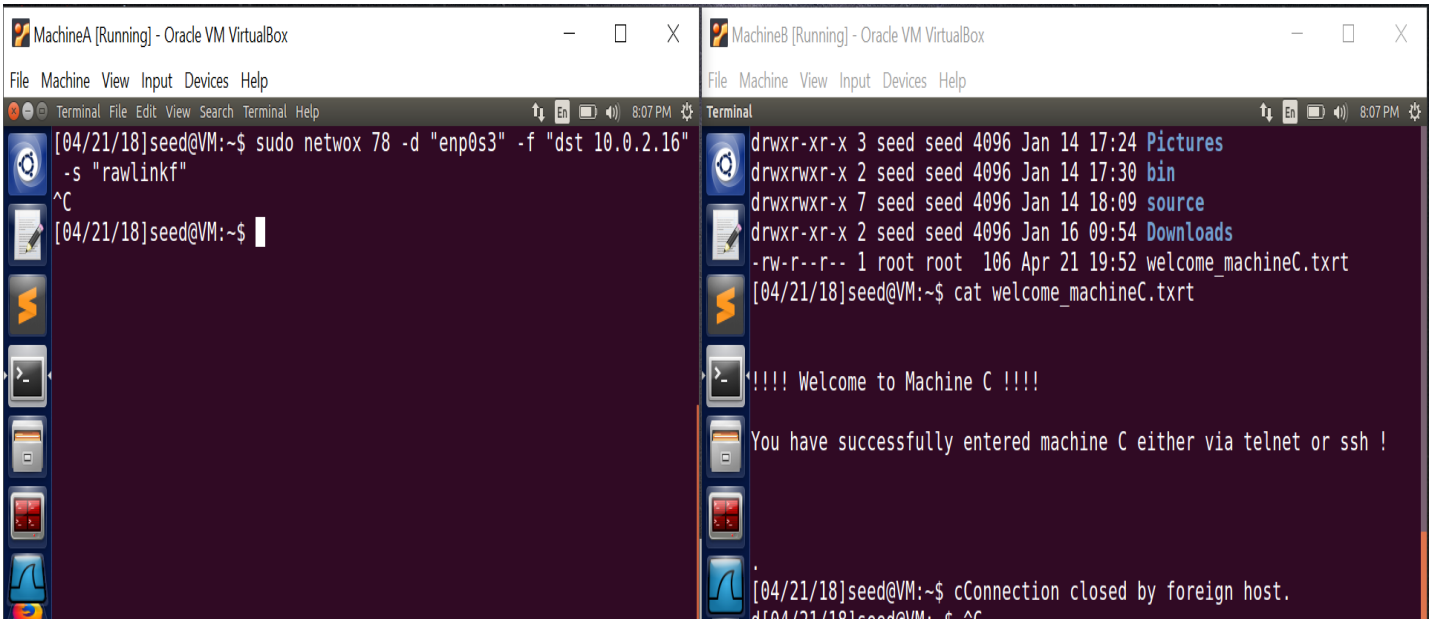
TCP RST is another type of the Denial of Service attack in which the attacker terminates the established TCP connection between 2 victims. To demonstrate the attack, we will try to perform TCP RST on *telnet* and *ssh* connection.



Obs: While looking at the terminal on machineB, we can observe that telnet has been established between machineB and machineC. This could also be confirmed by the packet capture on wireshark on machineA.

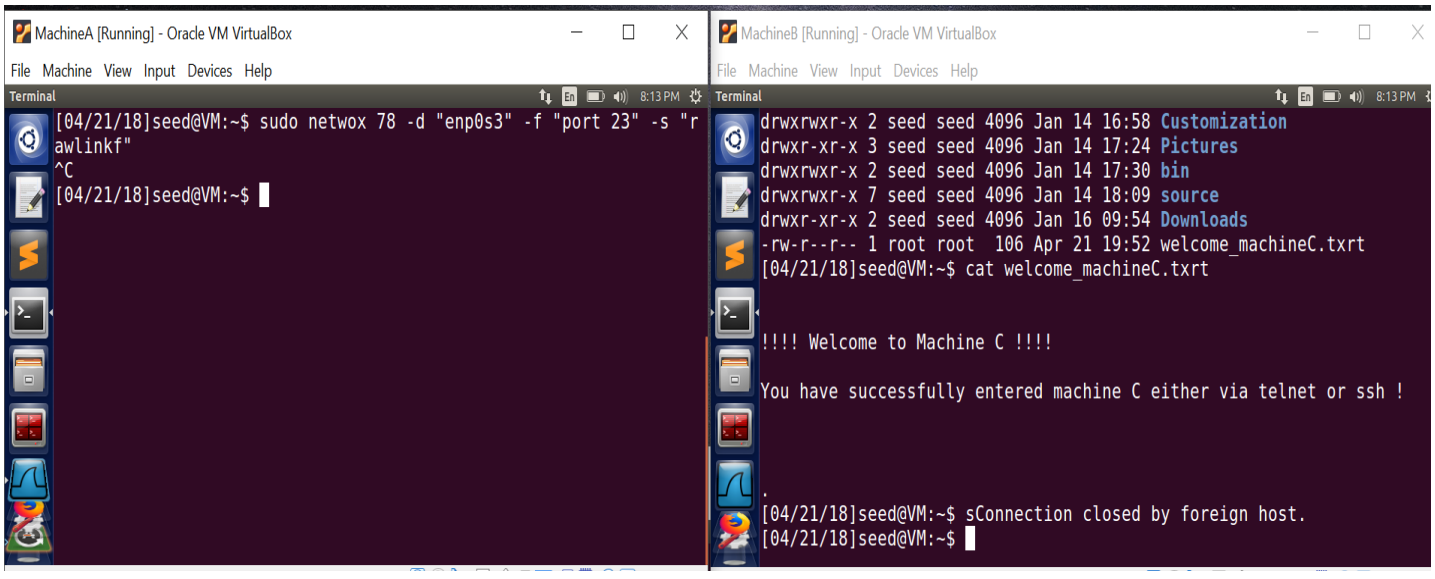


Obs: This image confirms that we are on machineB and is successfully able to browse the file located on machineC. When there is no TCP RST attack, we are able to successfully establish the TCP connection between the 2 machines.



Obs: We have tried to implement TCP RST attack from machineA on telnet connection from machineB to machineC. The command used to implement this attack is *"sudo netwox 78 -d "enp0s3" -f "dst 10.0.2.16" -s "rawlinkf"*.
sudo is used for root privileges netwox 78 is the number used by netwox for TCP RST attack
-d = device (interface from which the attack is going to happen)
-f = filter used to identity the port or the machine (ip address) on which the attack is being pulled over
-s = how to generate link layer for spoofing
rawlinkf used in spoofing can be broken down into 3 parts. **raw** means to spoof at ip4/ip6 level and **link** means to spoof at link level and **f** means to fill the source mac address.

From the terminal on machineB, we can clearly observe that the *"connection closed by foreign host"*. This means that we were successfully able to RST the telnet connection between two machines.



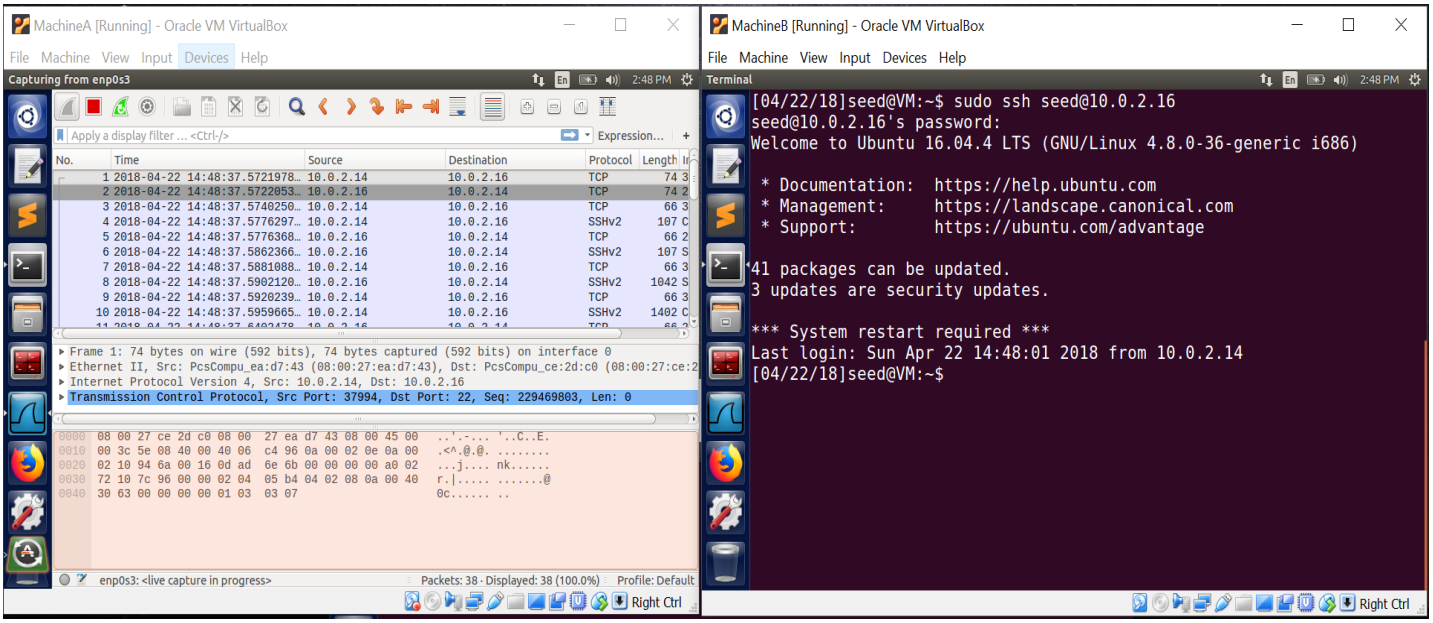
Obs: The TCP RST can also be implemented by terminating the connection based on the port number. The command used to implement that can be seen on machineA's terminal. *"sudo netwox 78 -d "enp0s3" -f "port23" -s "rawlinkf"* will work the same way as the previous command. The only difference between the two command is the use of different filters. Here TCP RST will stop all the telnet connections in the network.

```
144 56.662256980  10.0.2.16      10.0.2.14      TCP   54 23 → 59988 [RST, ACK] Seq=1541 Ack=140 Win=0 Len=0
145 56.665256877  PcsCompu_12:fd:fa Broadcast     ARP   42 Who has 10.0.2.16? Tell 10.0.2.12
146 56.671307233  PcsCompu_ce:2d:c0 PcsCompu_12:fd:fa ARP   60 10.0.2.16 is at 08:00:27:ce:2d:c0
147 56.671321712  10.0.2.14      10.0.2.16      TCP   54 59988 → 23 [RST, ACK] Seq=140 Ack=1542 Win=0 Len=0
148 56.675129179  10.0.2.16      10.0.2.14      TCP   54 [TCP ACKed unseen segment] 23 → 59988 [RST, ACK] Se
```

```
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1542    (relative sequence number)
Acknowledgment number: 141    (relative ack number)
0101 .... = Header Length: 20 bytes (5)
Flags: 0x014 (RST, ACK)
Window size value: 0
[Calculated window size: 0]
```

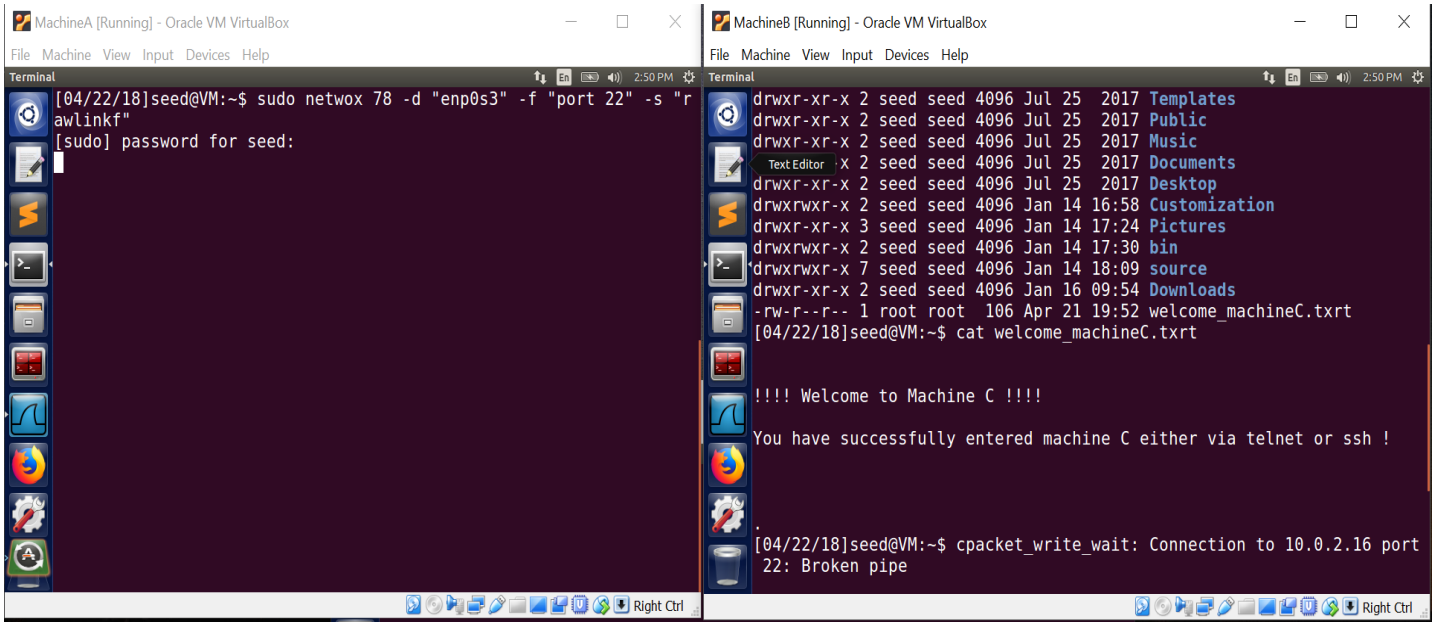
Obs: By capturing the packets from the wireshark, we can confirm that the connection has been terminated by a TCP RST request.

- TCP RST on SSH Connection



Obs: TCP RST attack is not limited to telnet. Now we will try to implement the same attack over the ssh connection. First, we will try to initiate a ssh connection between machineB and machineC. From the above and following image, we can clearly observe that the connection has been successfully established between two machines.

```
1 0.000000000  10.0.2.14      10.0.2.16      TCP     74 37994 → 22 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4206691 TSecr=0
2 0.000007471  10.0.2.16      10.0.2.14      TCP     74 22 → 37994 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=411243
3 0.001827227  10.0.2.14      10.0.2.16      TCP     66 37994 → 22 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4206691 TSecr=4112433
4 0.005431935  10.0.2.14      10.0.2.16      SSHv2  107 Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4)
5 0.005439012  10.0.2.16      10.0.2.14      TCP     66 22 → 37994 [ACK] Seq=1 Ack=42 Win=29056 Len=0 TSval=4112434 TSecr=4206692
6 0.014038814  10.0.2.16      10.0.2.14      SSHv2  107 Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4)
7 0.015911006  10.0.2.14      10.0.2.16      TCP     66 37994 → 22 [ACK] Seq=42 Ack=42 Win=29312 Len=0 TSval=4206695 TSecr=4112436
8 0.018014212  10.0.2.16      10.0.2.14      SSHv2 1042 Server: Key Exchange Init
9 0.019826043  10.0.2.14      10.0.2.16      TCP     66 37994 → 22 [ACK] Seq=42 Ack=1018 Win=31232 Len=0 TSval=4206696 TSecr=4112437
```



Obs: We have tried to implement TCP RST attack from machineA on ssh connection from machineB to machineC. The command used to implement this attack is *"sudo netwox 78 -d "enp0s3" -f "port 22" -s "rawlinkf"*.
sudo is used for root privileges netwox 78 is the number used by netwox for TCP RST attack.
-d = device (interface from which the attack is going to happen)
-f = filter used to identity the port or the machine (ip address) on which the attack is being pulled over
-s = how to generate link layer for spoofing
rawlinkf used in spoofing can be broken down into 3 parts. **raw** means to spoof at ip4/ip6 level and **link** means to spoof at link level and **f** means to fill the source mac address.

From the output on the terminal on machineB's terminal, we can deduce that the connection has been terminated. *"packet_write_wait: Connection to 10.0.2.16 port 22: Broken pipr"* clearly depicts that TCP RST has been successfully initialized.

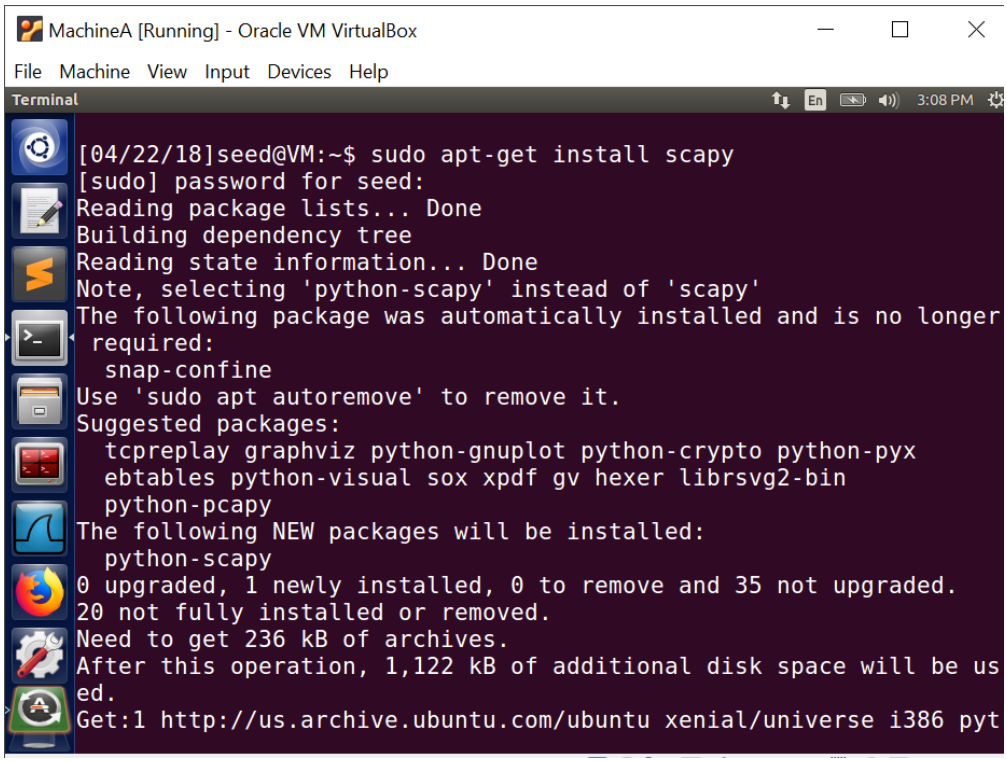
123	89.095656273	PcsCompu_12:fd:fa	Broadcast	ARP	42 Who has 10.0.2.14? Tell 10.0.2.12
124	89.097455680	PcsCompu_ea:d7:43	PcsCompu_12:fd:fa	ARP	60 10.0.2.14 is at 08:00:27:ea:d7:43
125	89.097464279	10.0.2.16	10.0.2.14	TCP	54 22 → 37994 [RST, ACK] Seq=5110 Ack=3095 Win=0 Len=0
126	89.097882353	PcsCompu_12:fd:fa	Broadcast	ARP	42 Who has 10.0.2.16? Tell 10.0.2.12
127	89.099515800	10.0.2.16	10.0.2.14	TCP	54 [TCP ACKed unseen segment] 22 → 37994 [RST, ACK] Seq=
128	89.101206793	PcsCompu_ce:2d:c0	PcsCompu_12:fd:fa	ARP	60 10.0.2.16 is at 08:00:27:ce:2d:c0
129	89.101215435	10.0.2.14	10.0.2.16	TCP	54 37994 → 22 [RST, ACK] Seq=3130 Ack=5111 Win=0 Len=0
130	94.275176252	PcsCompu_ce:2d:c0	PcsCompu_ea:d7:43	ARP	60 Who has 10.0.2.14? Tell 10.0.2.16
131	94.282267977	PcsCompu_ea:d7:43	PcsCompu_ce:2d:c0	ARP	60 10.0.2.14 is at 08:00:27:ea:d7:43
132	94.326034268	PcsCompu_ea:d7:43	PcsCompu_ce:2d:c0	ARP	60 Who has 10.0.2.16? Tell 10.0.2.14
133	94.329799842	PcsCompu_ce:2d:c0	PcsCompu_ea:d7:43	ARP	60 10.0.2.16 is at 08:00:27:ce:2d:c0

```
[TCP Segment Len: 0]
Sequence number: 3130    (relative sequence number)
Acknowledgment number: 5111    (relative ack number)
0101 .... = Header Length: 20 bytes (5)
Flags: 0x014 (RST, ACK)
Window size value: 0
[Calculated window size: 0]
[Window size scaling factor: 128]
```

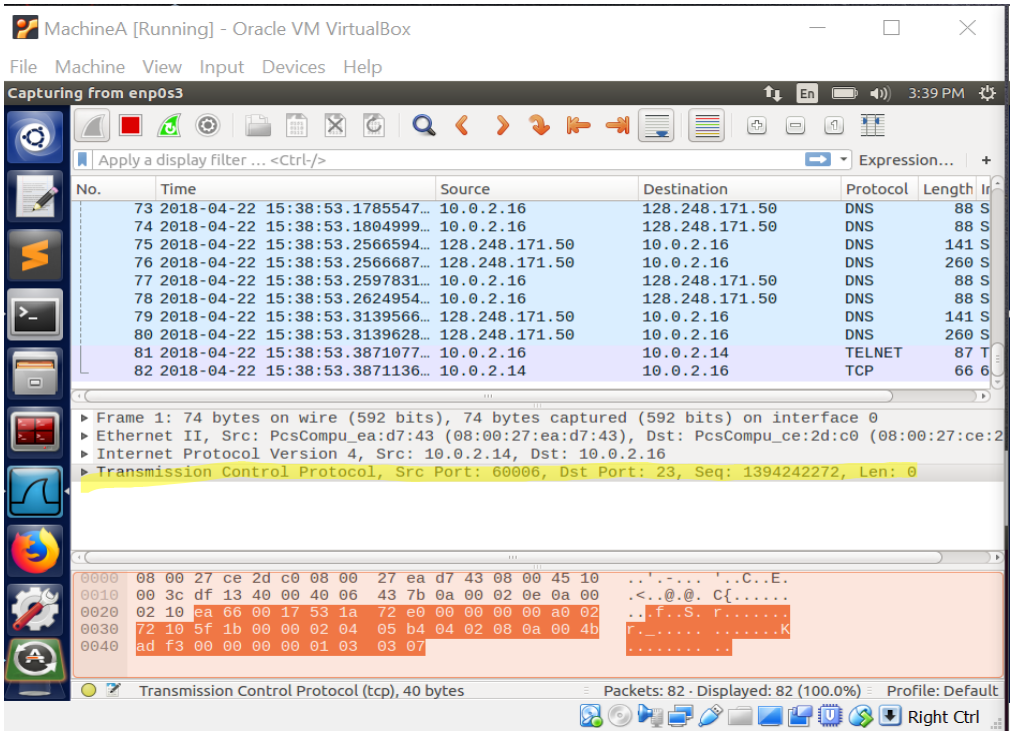
Obs: From the packet capture files, we can observe that the ssh connection has been called off by a TCP RST request. The one that has been marked by yellow depicts that.

- TCP RST on TELNET Connection by SCAPY

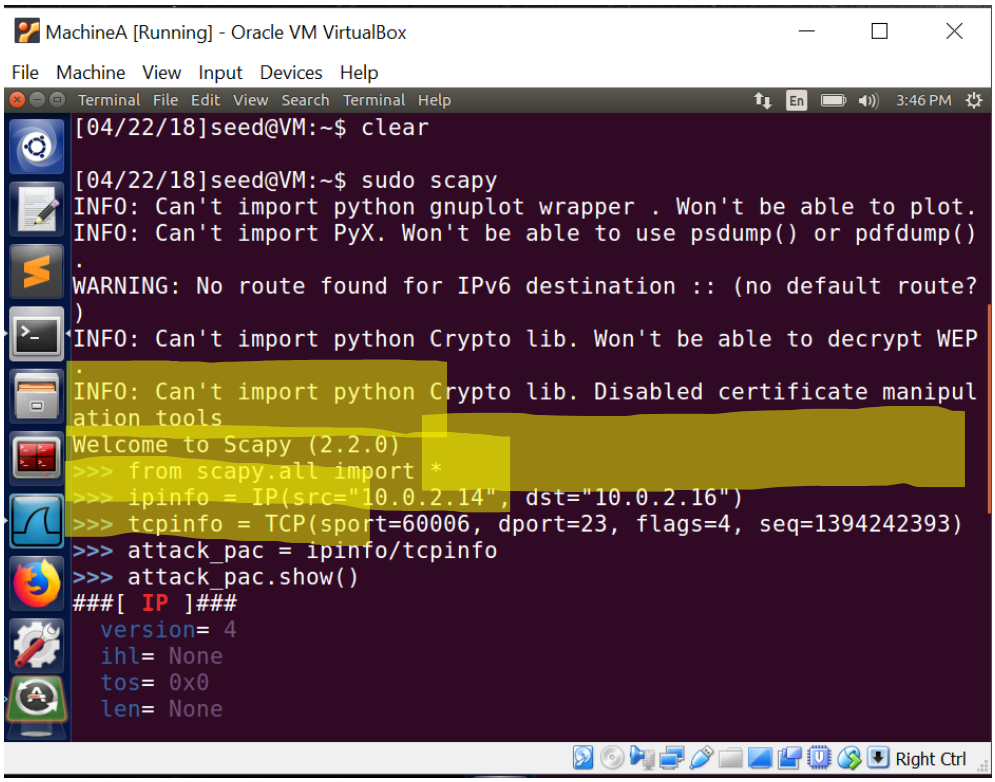
Another tool to implement the TCP RST connection is Scapy. In scapy tool, you can define the packet structure all by yourself by putting in parameters source & destination address, port numbers, flag, sequence number, etc.



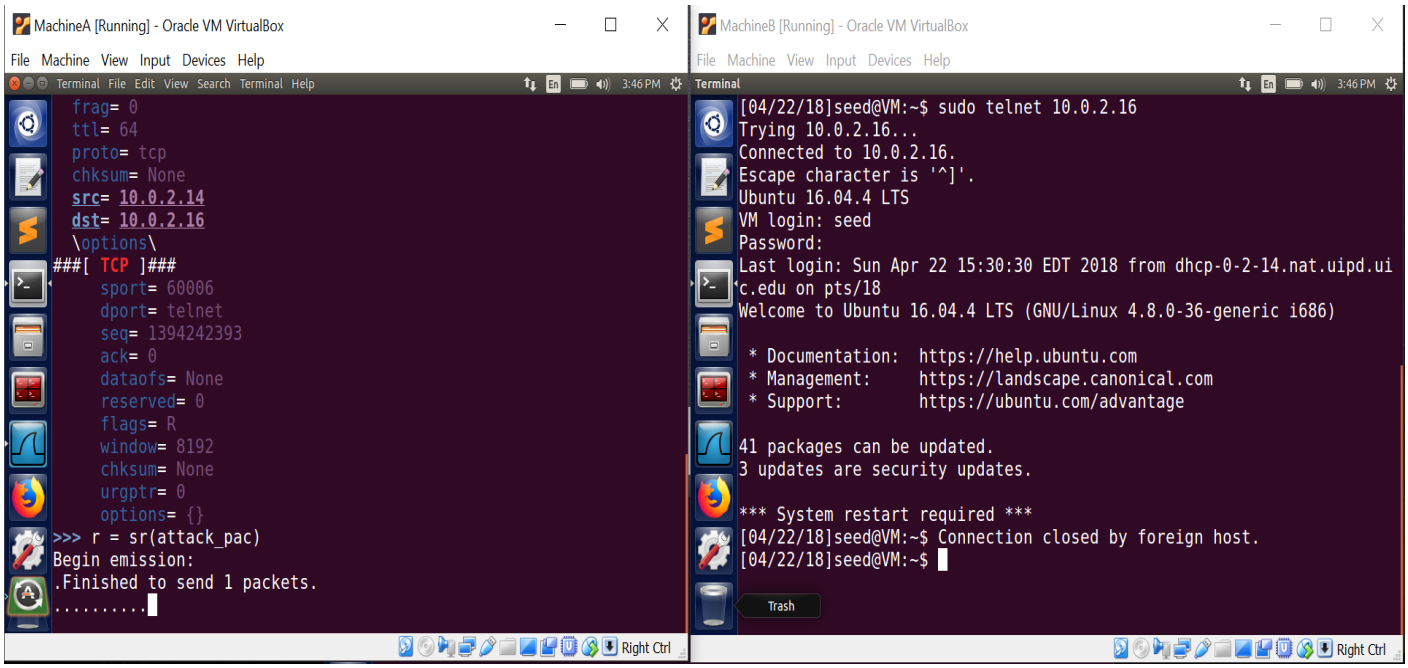
Obs: We need to first install scapy on machineA (attackers machine). To install, use *"sudo apt-get install scapy"*.



Obs: Now to use scapy, we first need some basic information. For that, we need to capture the packets on machineA (attacker) while telnet connection is established between machineB and machineC. Once the connection is established, go to the last tcp packet exchange between the two machines in the packet capture and try to extract source ip (victim's ip), destination ip (server's ip), port numbers, flags and sequence number. This information has been highlighted in yellow and will be used to create an attack packet.

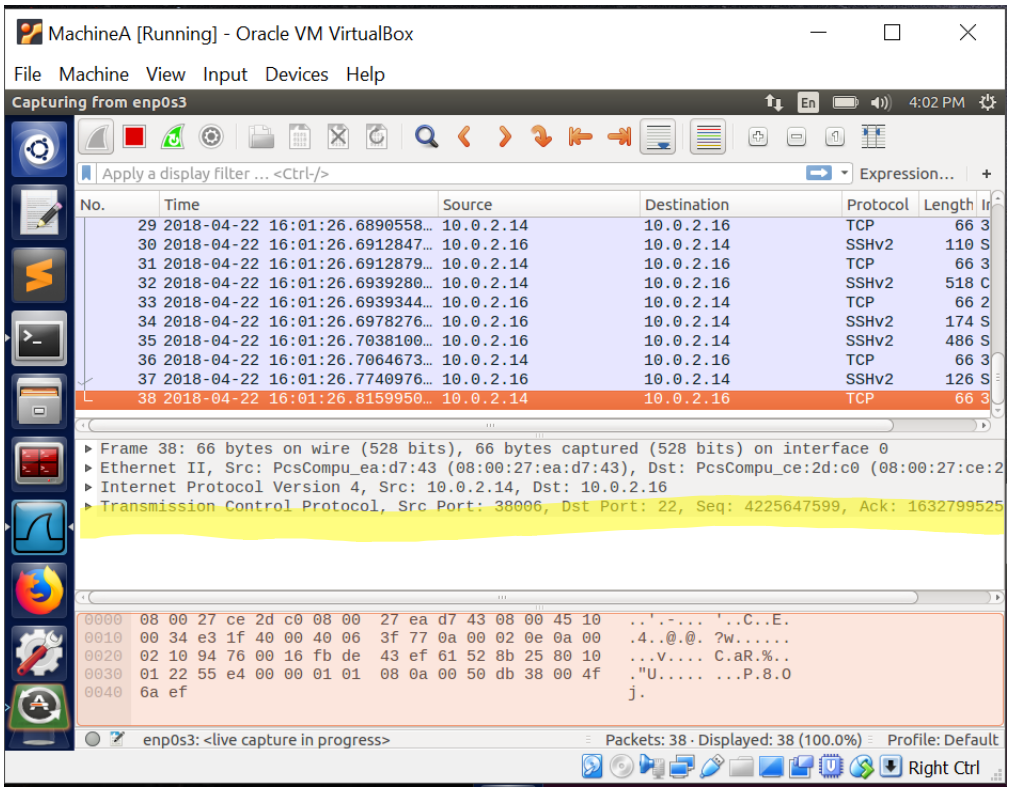


Obs: Open scapy on terminal with root privilege. Then create the packet as specified in the terminal highlighted in yellow.

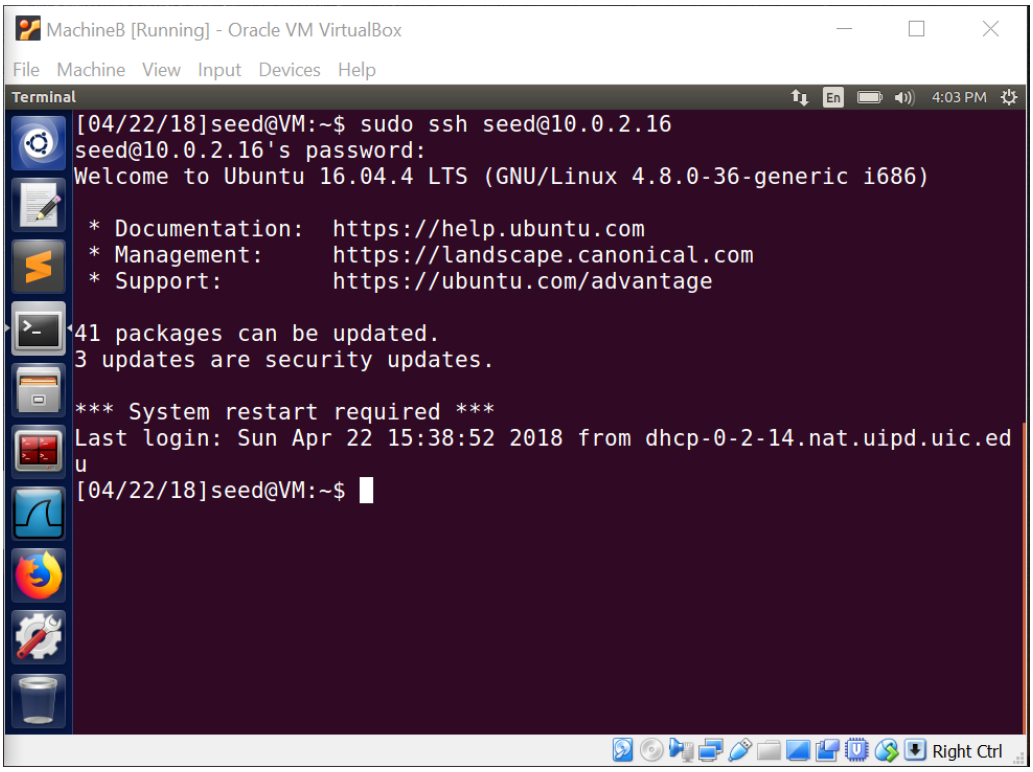


Obs: After creating the packet in the scapy tool, try to run the TCP SYN attack with the help of *"r = sr(packet name)"* . As soon as we will run this attack, we can observe that the telnet connection from machineB to machineC has been terminated. This can be deduced by the comment *"Connection closed by foreign host"*.

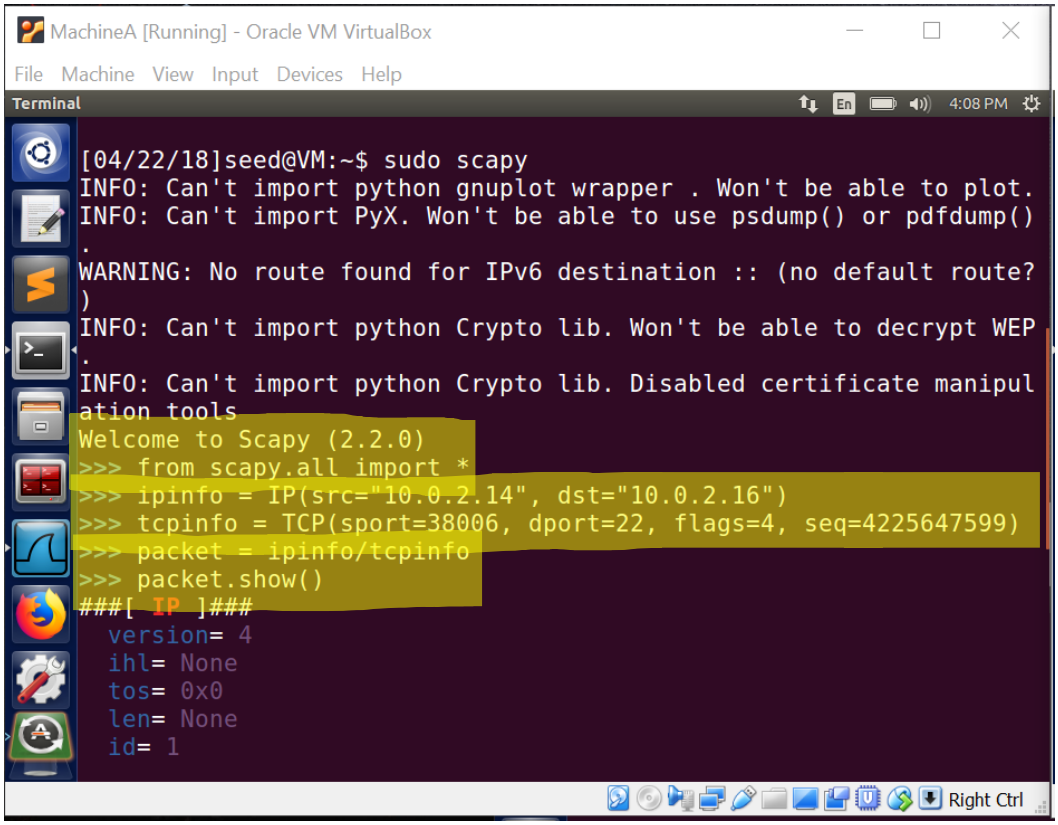
- TCP RST on SSH Connection by SCAPY



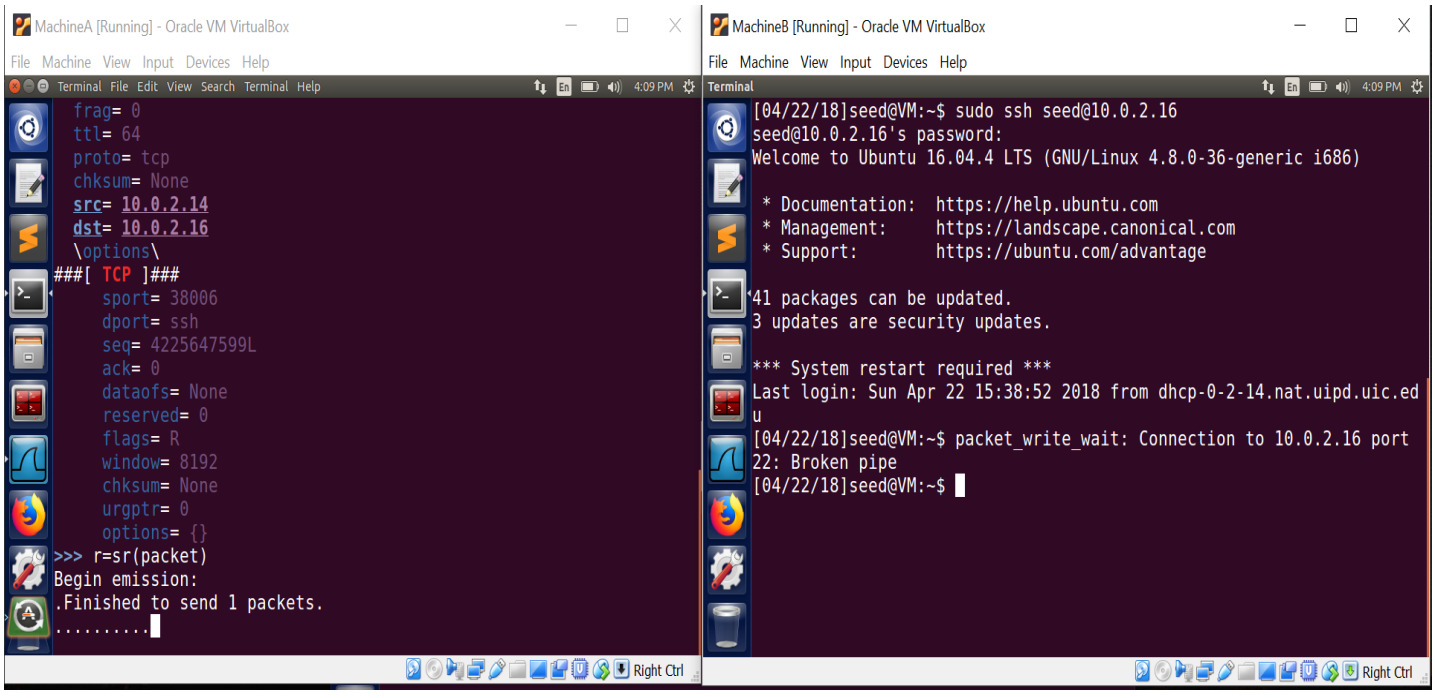
Obs: For finding the basic information, we need to capture the packets on machineA (attacker) while ssh connection is established between machineB and machineC. Once the connection is established, go to the last tcp packet exchange between the two machines in the packet capture and try to extract source ip (victim's ip), destination ip (server's ip), port numbers, flags and sequence number. This information has been highlighted in yellow and will be used to create an attack packet.



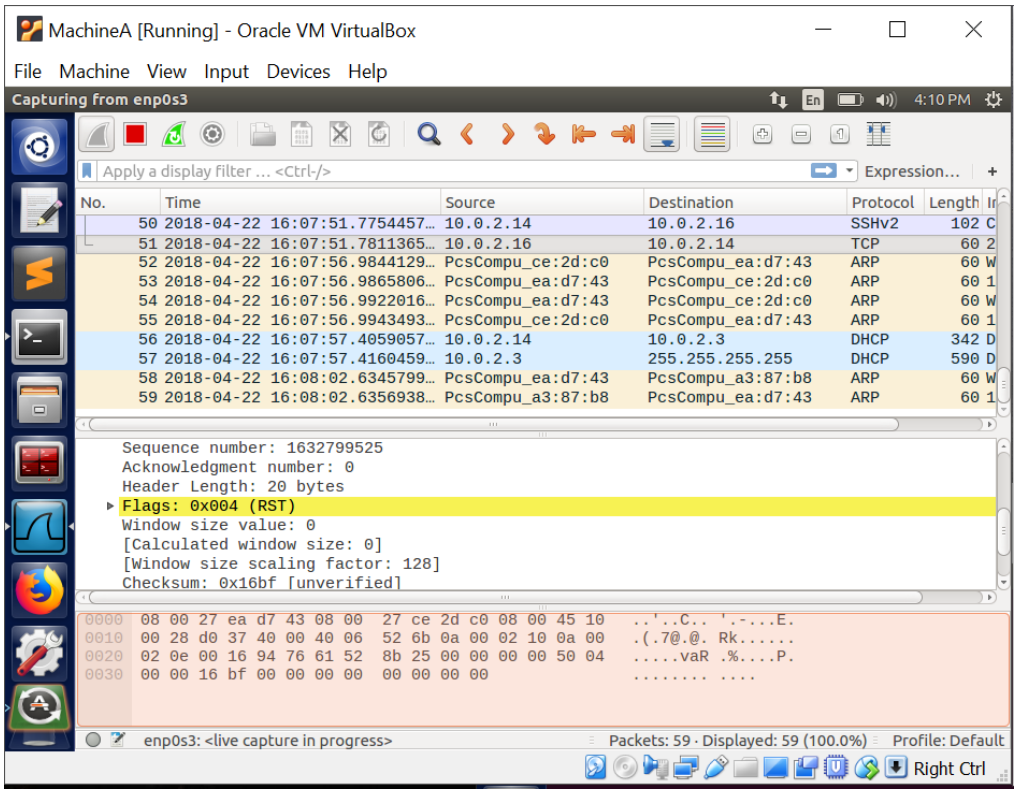
Obs: From the specified image, we can observe that ssh connection was successfully implemented between machineA and machineB.



Obs: Open scapy on terminal with root privilege. Then create the packet as specified in the terminal highlighted in yellow. This packet is created based on the information extracted from the wireshark while establishing the ssh connection between machineB and machineC.



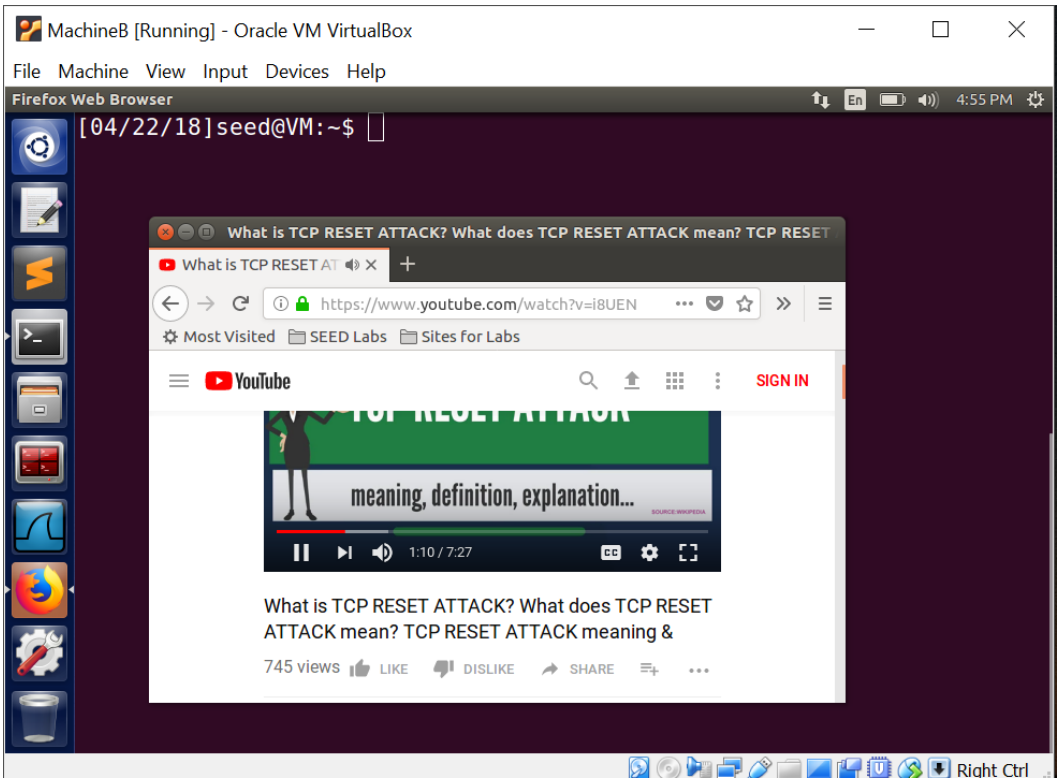
Obs: After creating the packet in the scapy tool, try to run the TCP SYN attack with the help of *"r = sr(packet name)"* . As soon as we will run this attack, we can observe that the ssh connection from machineB to machineC has been terminated. This can be deduced by the comment *"packet_write_wait: Connection to 10.0.2.16 port 22: Broken pipe"*.



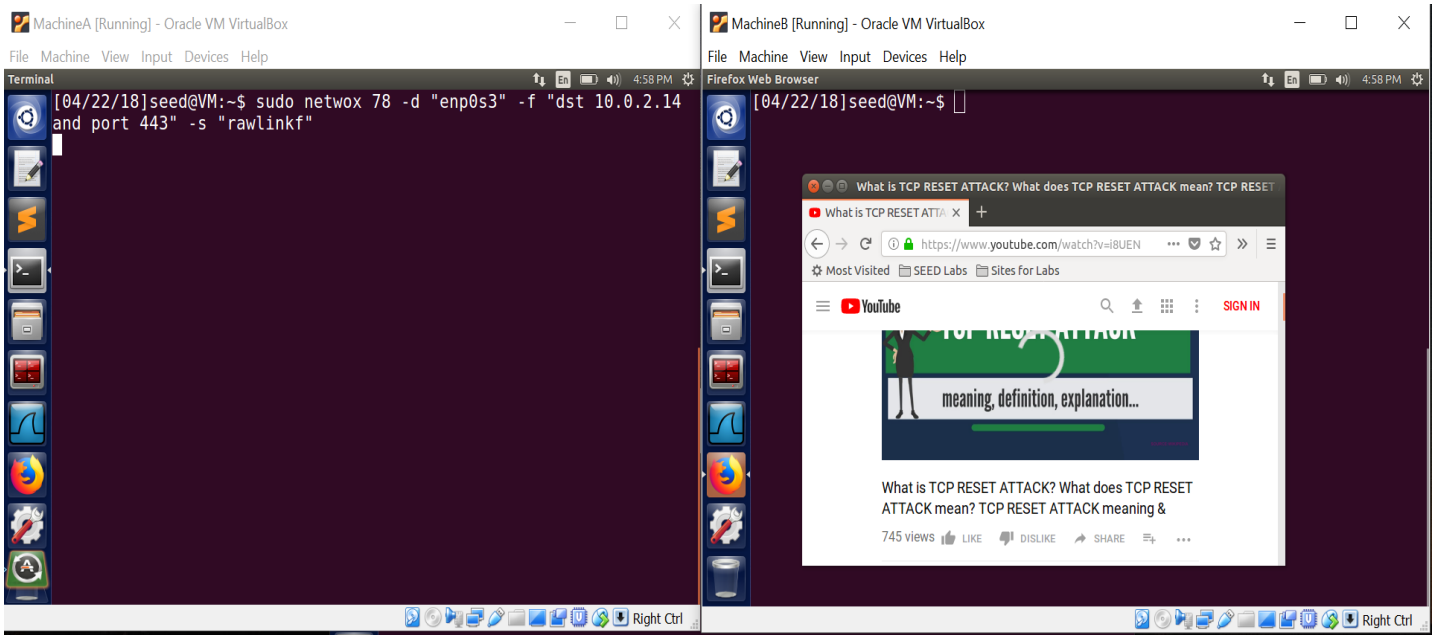
Obs: The RST packet confirms that the ssh connection between machineB and machineC has been revoked/terminated by the scapy tool's TCP RST attack.

Task 3: TCP RST attacks on Video Streaming Applications

The more pratical application of TCP RST attacks is stopping the video streaming. This is done by sending a forged TCP reset packet to the victim's machine (machineB).



Obs: We can observe that as there is no TCP RST packet sent by the attacker to the victim's machine, the victim is busy watching the online video.



Obs: TCP RST packets can be again generated by attacker with the help of the netwox 78 tool. The command used to implement this attack is *"sudo netwox 78 -d "enp0s3" -f "dst 10.0.2.14 and port 443" -s "rawlinkf"*.

sudo is used for root privileges netwox 78 is the number used by netwox for TCP RST attack.

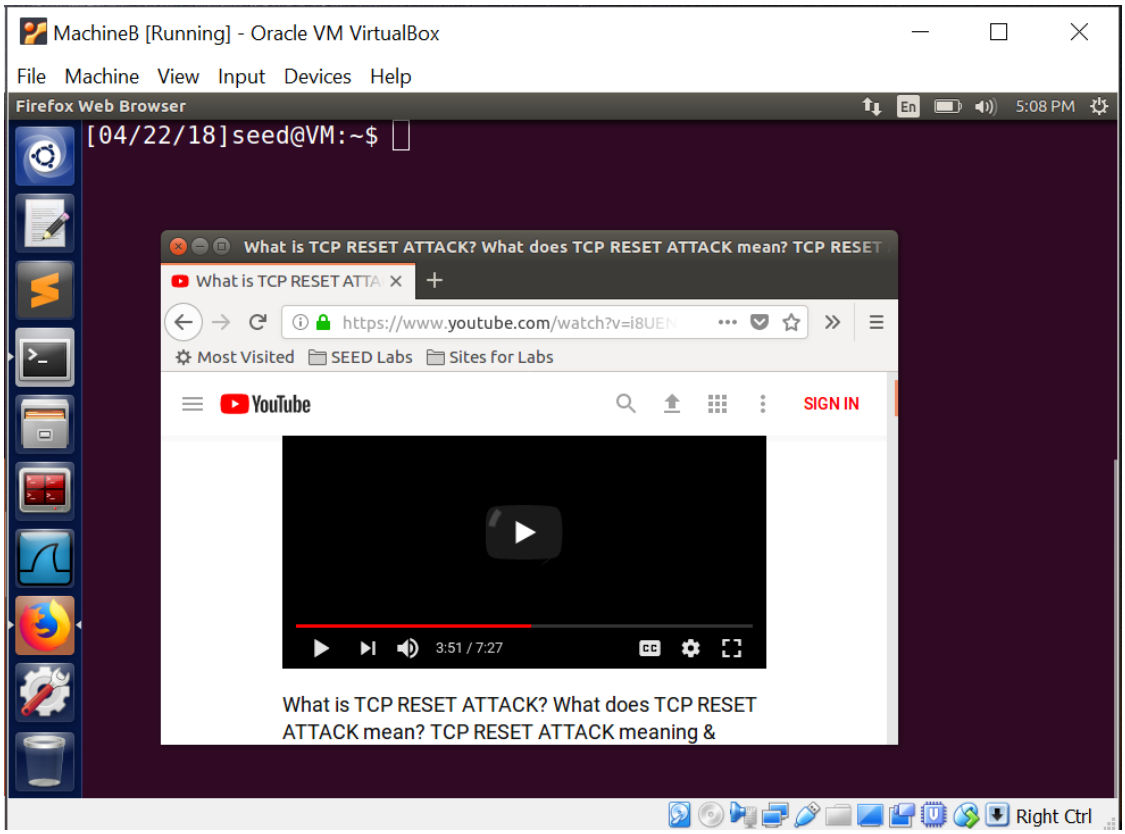
-d = device (interface from which the attack is going to happen)

-f = filter used to identity the port or the machine (ip address) on which the attack is being pulled over

-s = how to generate link layer for spoofing

rawlinkf used in spoofing can be broken down into 3 parts. **raw** means to spoof at ip4/ip6 level and **link** means to spoof at link level and **f** means to fill the source mac address.

We can observe that as soon as we attacked *port 443 / HTTPS* (this port is chosen as YouTube is accessed over port 443), the video stopped playing or it went on buffering without loading the content.



Obs: The online video buffering will stop and the content wouldn't get load till the time you stop the attack.