Carsten Hood
UIN: 922009787

CSCE 465-500 Computer & Network Security
Due Friday, November 17, 2017

HW #4: TCP/IP Attack Lab

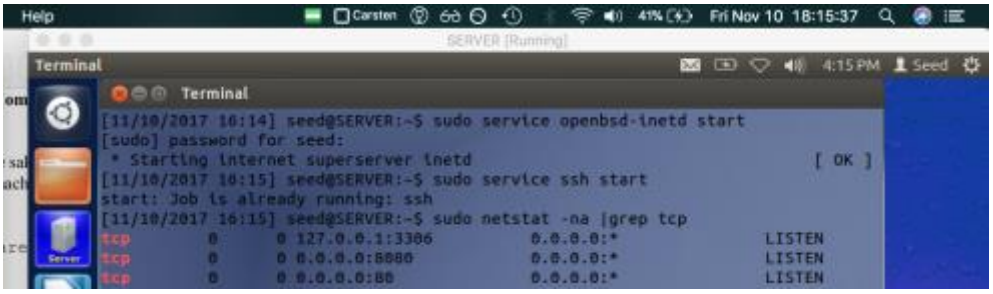
==========

Setup

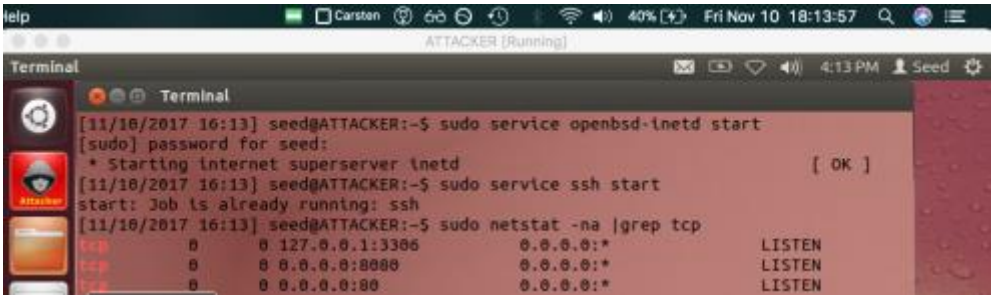
For this lab we set up three seed virtual machines – USER, SERVER, and ATTACKER. These Ubuntu VMs are run simultaneously using Oracle VirtualBox on a MacBook Pro laptop. The customized Mac OS menu bar is retained at the top of many screenshots to indicate authenticity. The VMs are differentiated through 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:

SERVER:



ATTACKER:

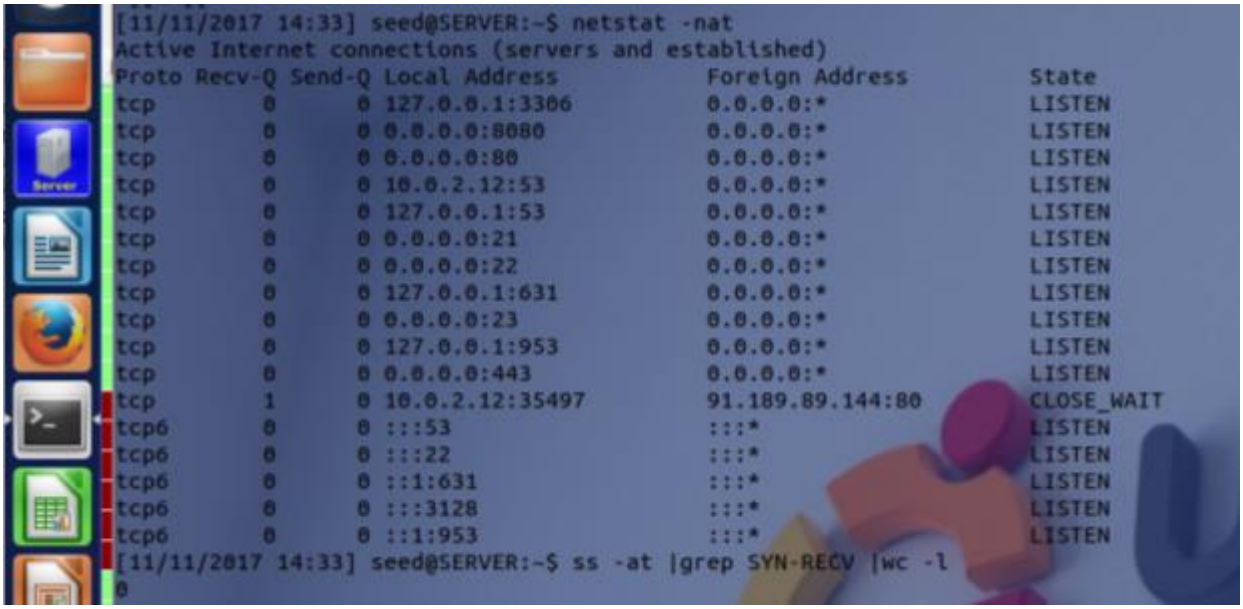


==========

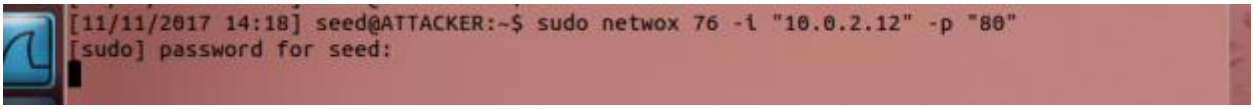
Task 1. SYN *Flood* Attack

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.

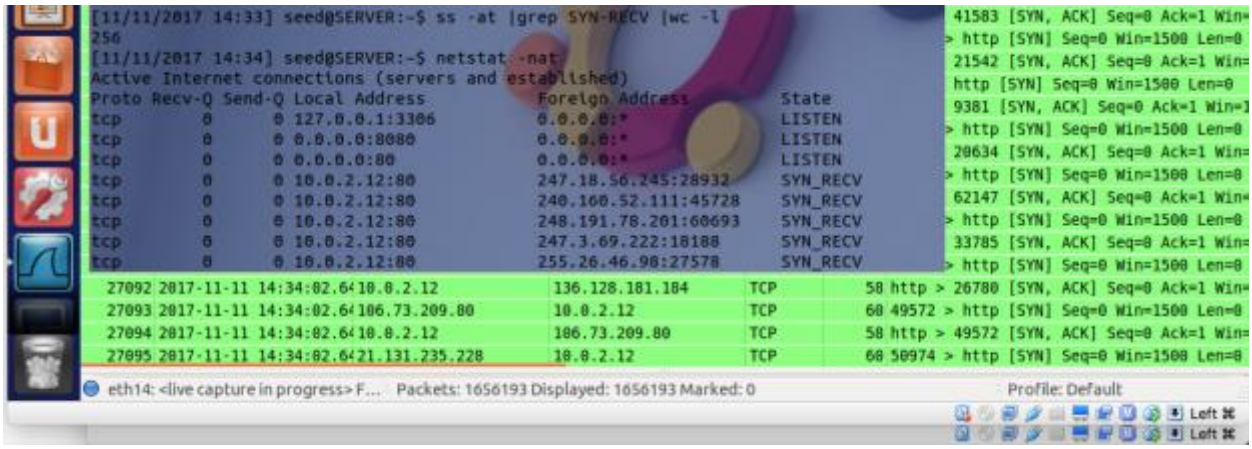
First we demonstrate the victim's connectivity prior to the attack. The command `netstat –nat` reveals that all ports but one are initially available and listening for incoming packets. The subsequent command `ss -at |grep SYN-RECV |wc –l` reveals that the victim has zero half-open connections. These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



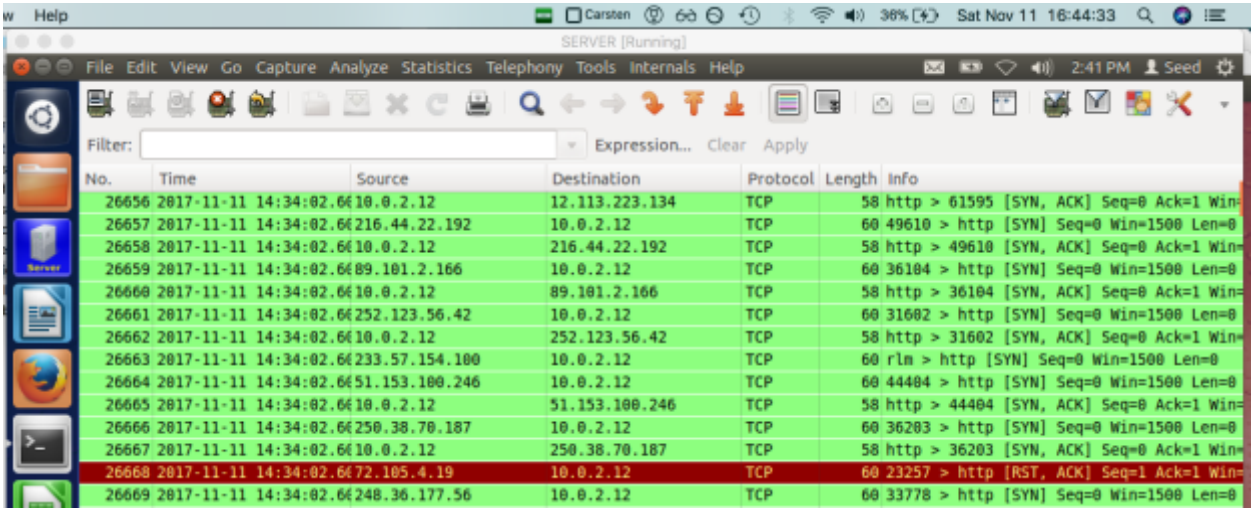
We then conduct the attack from the attacker's VM using the `netwox 76` command. The victim, `SERVER`, has the IP address `10.0.2.12`. We are attacking the port number `80`. Hence we can formulate the attack command as `netwox 76 -i "10.0.2.12" -p "80"`, shown executing below:



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:



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



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.

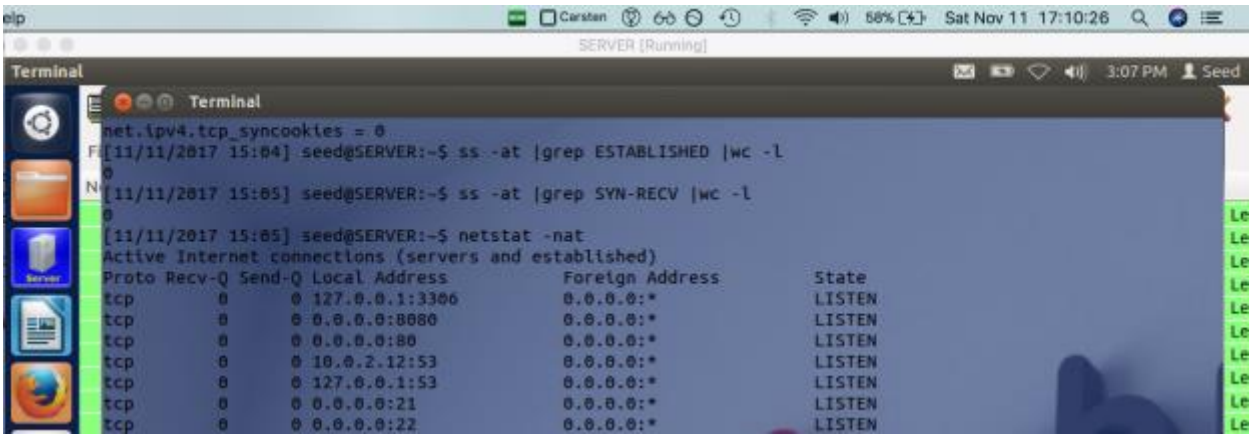
```
[11/11/2017 14:48] seed@SERVER:~$ sudo sysctl -q net.ipv4.tcp_syncookies
[sudo] password for seed:
net.ipv4.tcp_syncookies = 1
```

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:

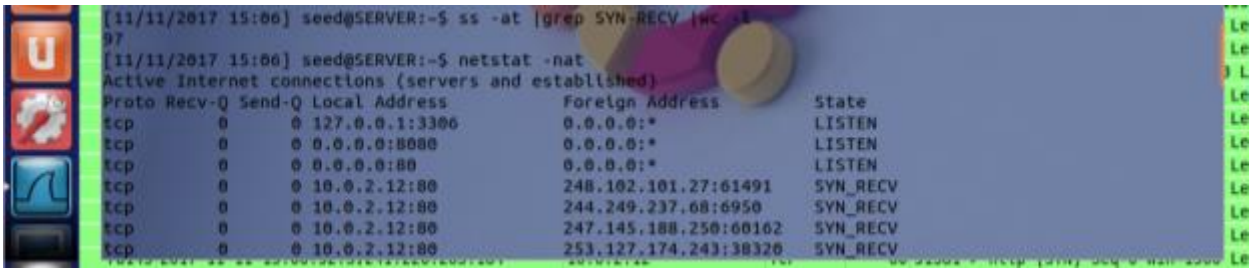
```
[11/11/2017 15:00] seed@SERVER:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[11/11/2017 15:01] seed@SERVER:~$
```

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`).

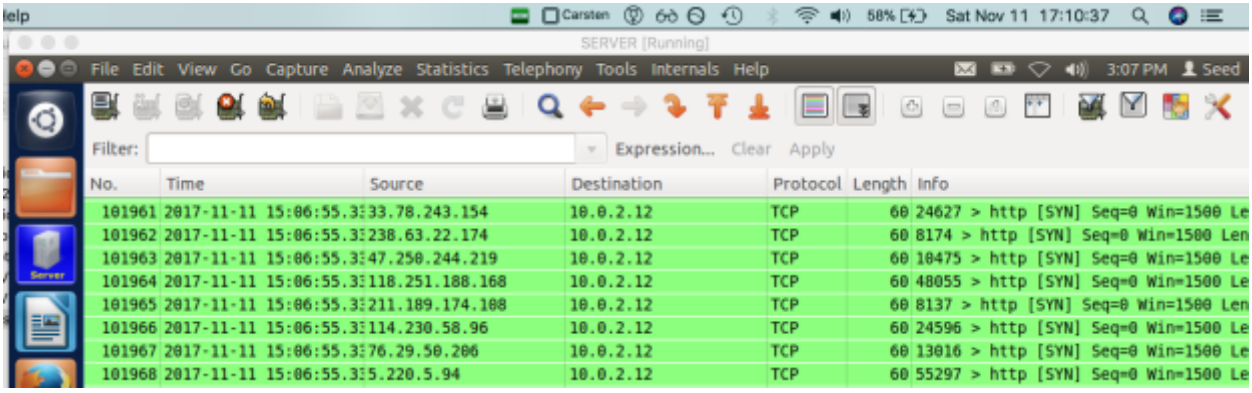


Again the attack is executed:

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.

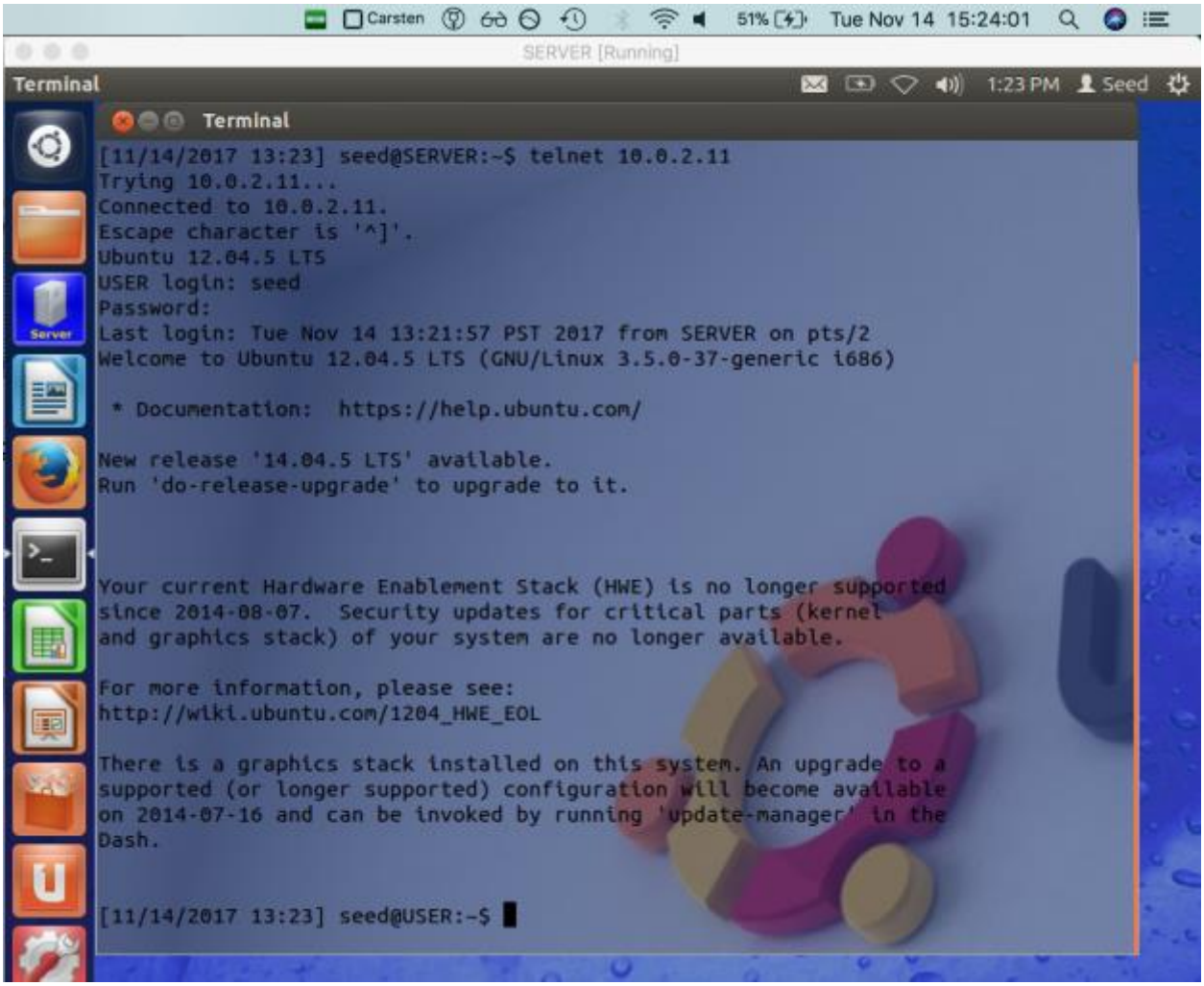


==========

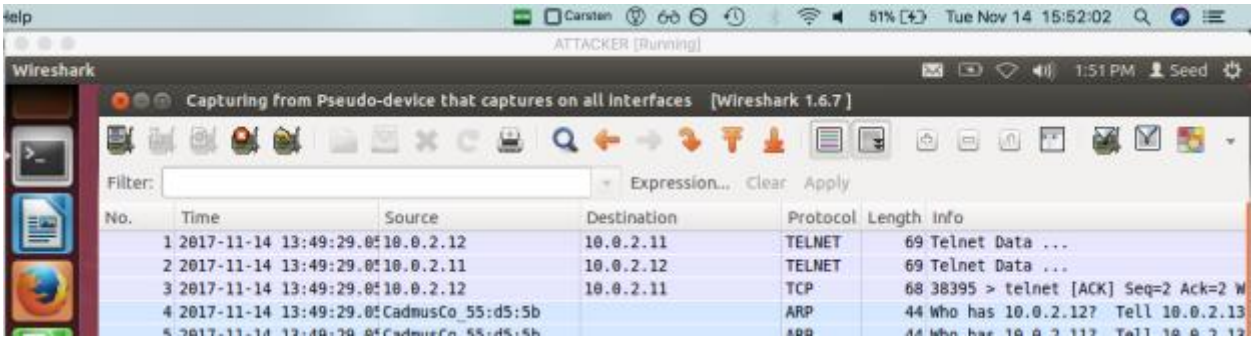
Task 2. TCP RST Attacks on `telnet` and `ssh` Connections

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.



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`.

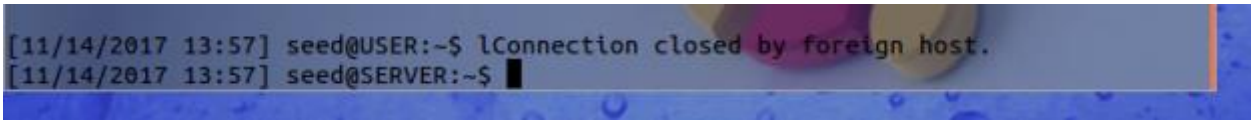


Now we formulate an attack using the `netwox 78` command:

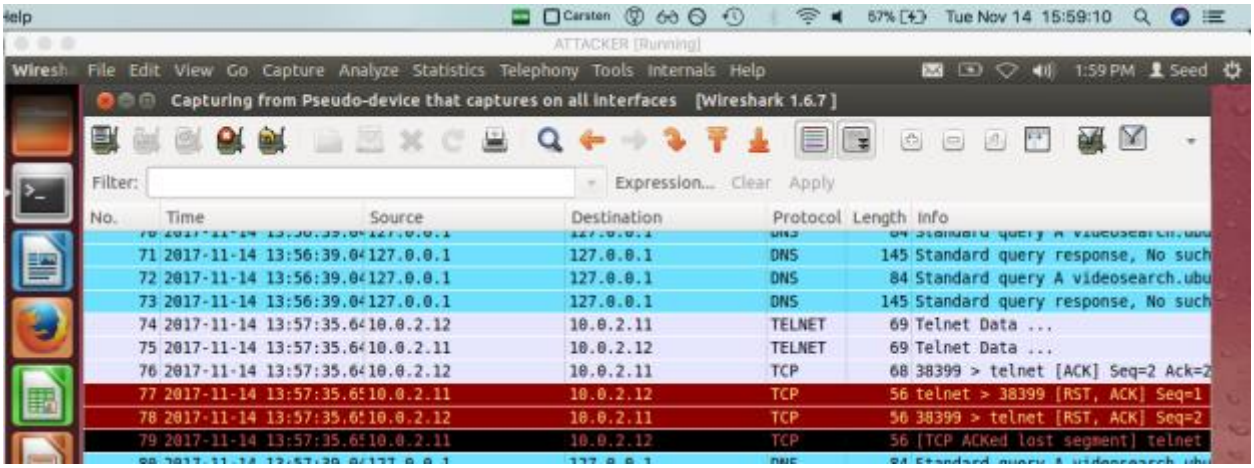
```
netwox 78 --device "Eth0" --filter "host 10.0.2.12" --spoofip "raw" --ips "10.0.2.11"
```

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.`":



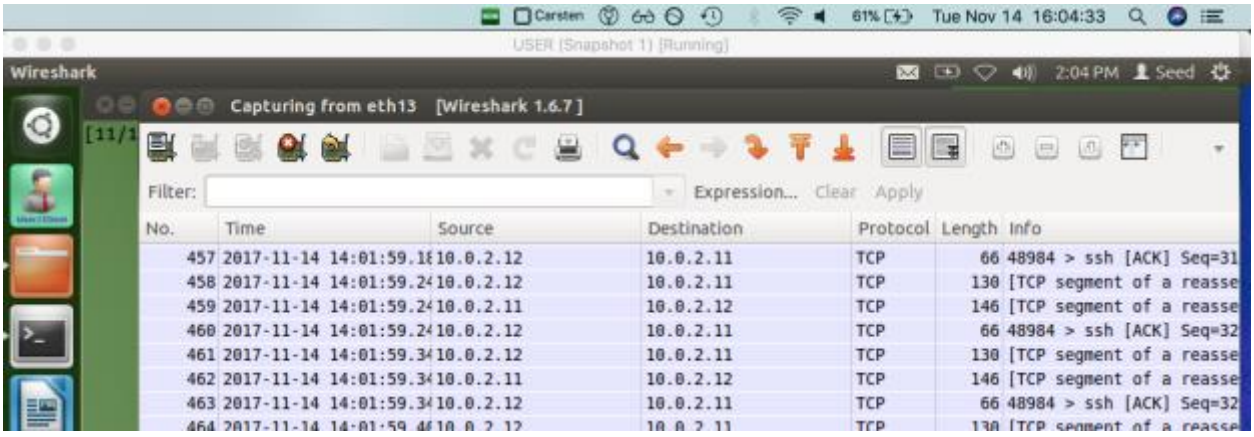
In Wireshark as `ATTACKER` we can observe the TCP RST packets which terminated the connection:



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:



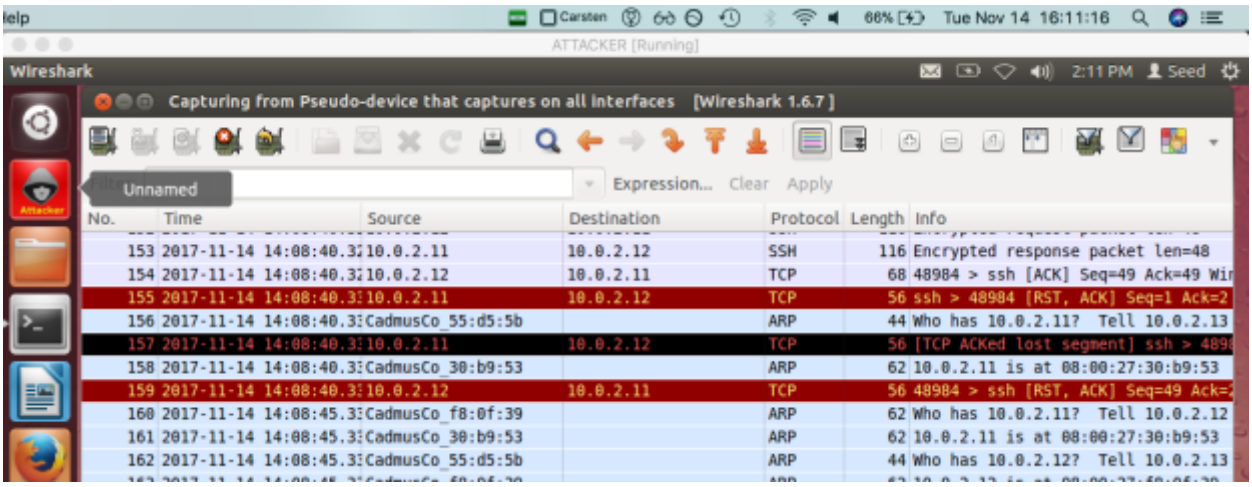
`ATTACKER` can formulate another similar `netwox 78` attack, this time targeting port 22 and `USER`'s IP address.

```
netwox 78 --device "Eth0" --filter "port 22" --spoofip "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: Broke pipe`":

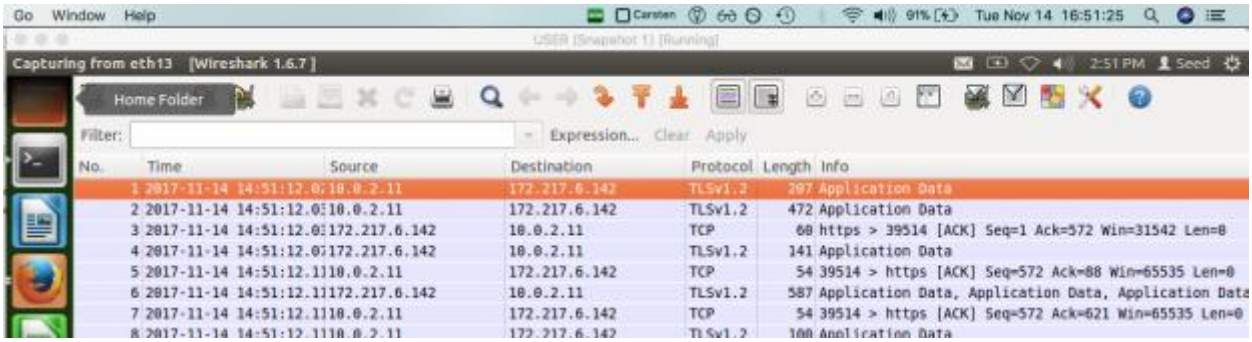
We can observe `ATTACKER`'s fake RST packet causing the connection to close:



==========

Task 3. TCP RST Attacks on Video Streaming Applications

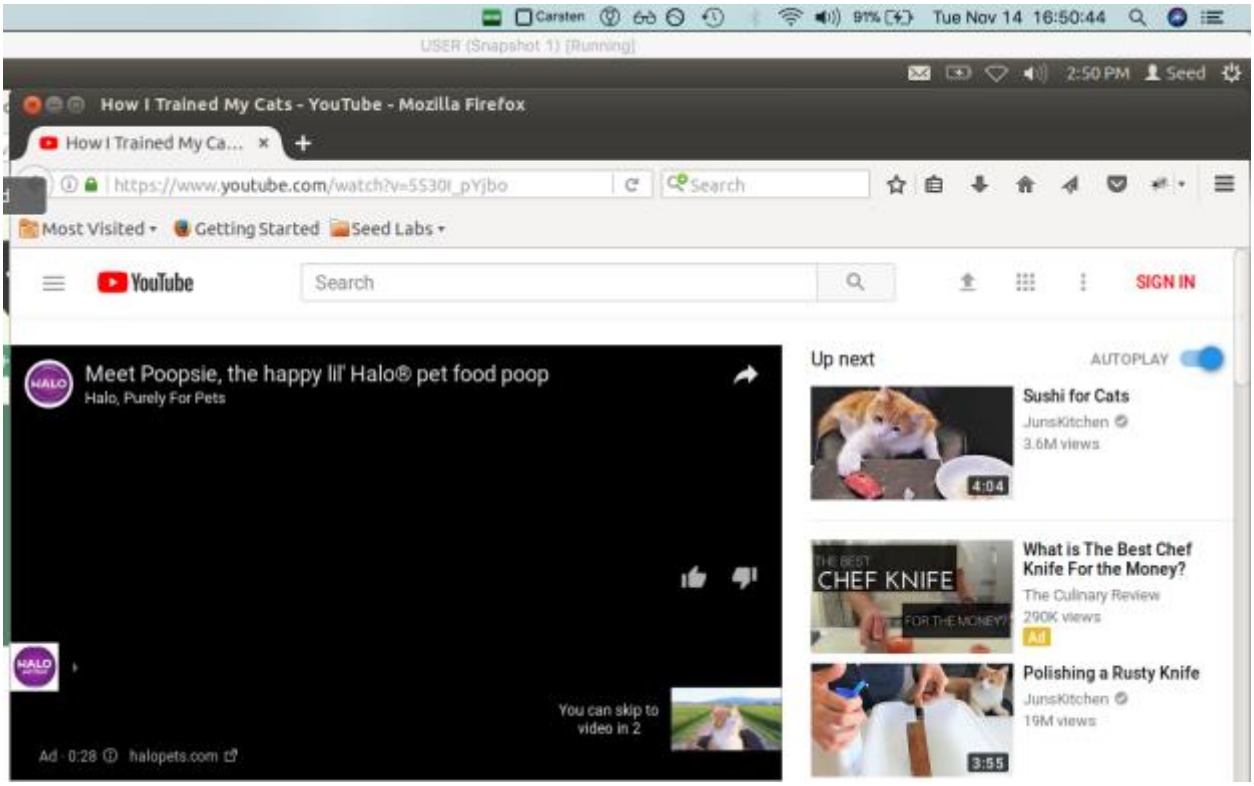
To test an TCP RST attack on a video stream, we first access an online video. As `USER` we open YouTube and select a featured video so that a pre-video advertisement begins to load and play. We can observe this video-related traffic in Wireshark:



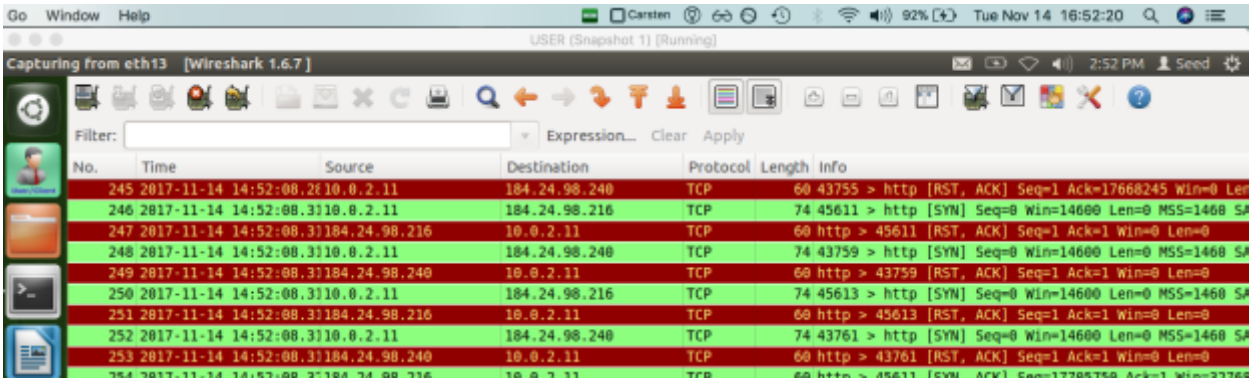
To attack the YouTube connection we issue the following `netwox 78` command as the root user on the same VM, targeting its own IP address:

```
netwox 78 --device "Eth0" --filter "port 80" --spoofip "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:

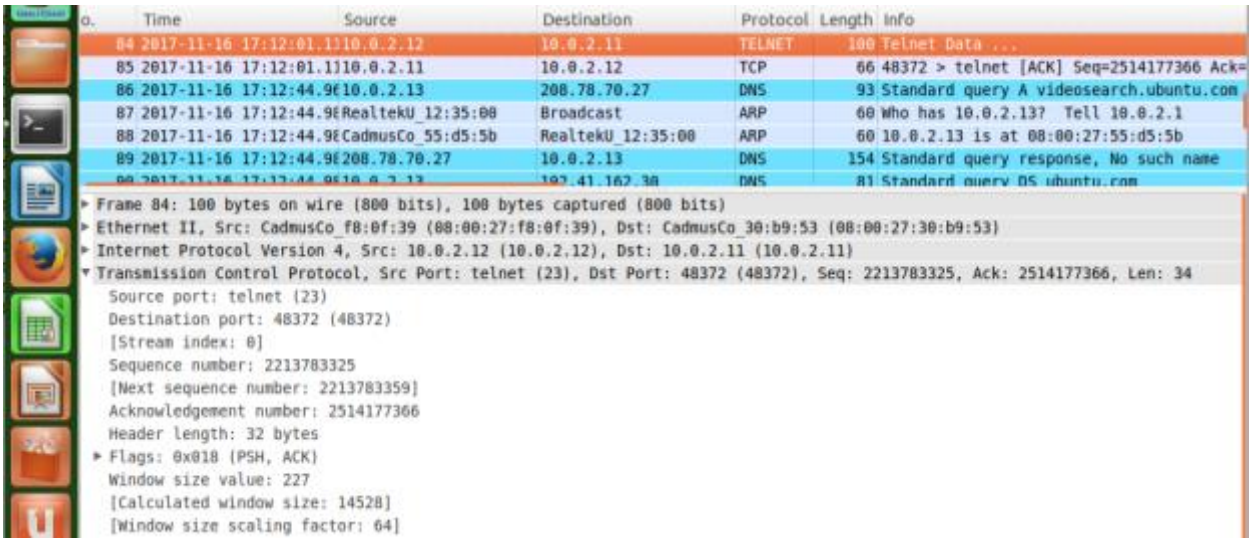


==========

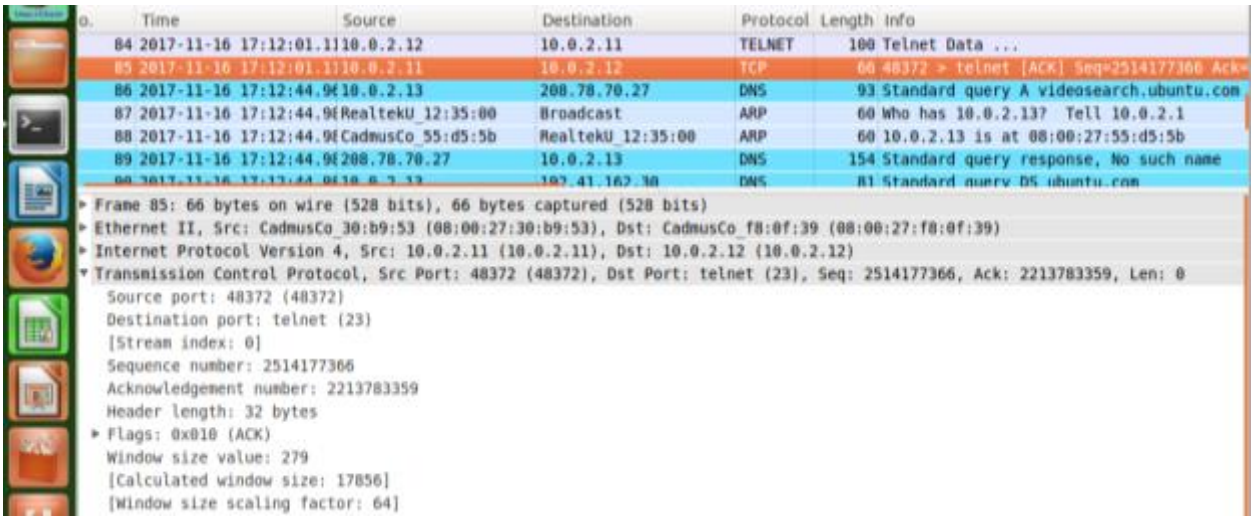
Task 4. TCP Session Hijacking

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

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`:



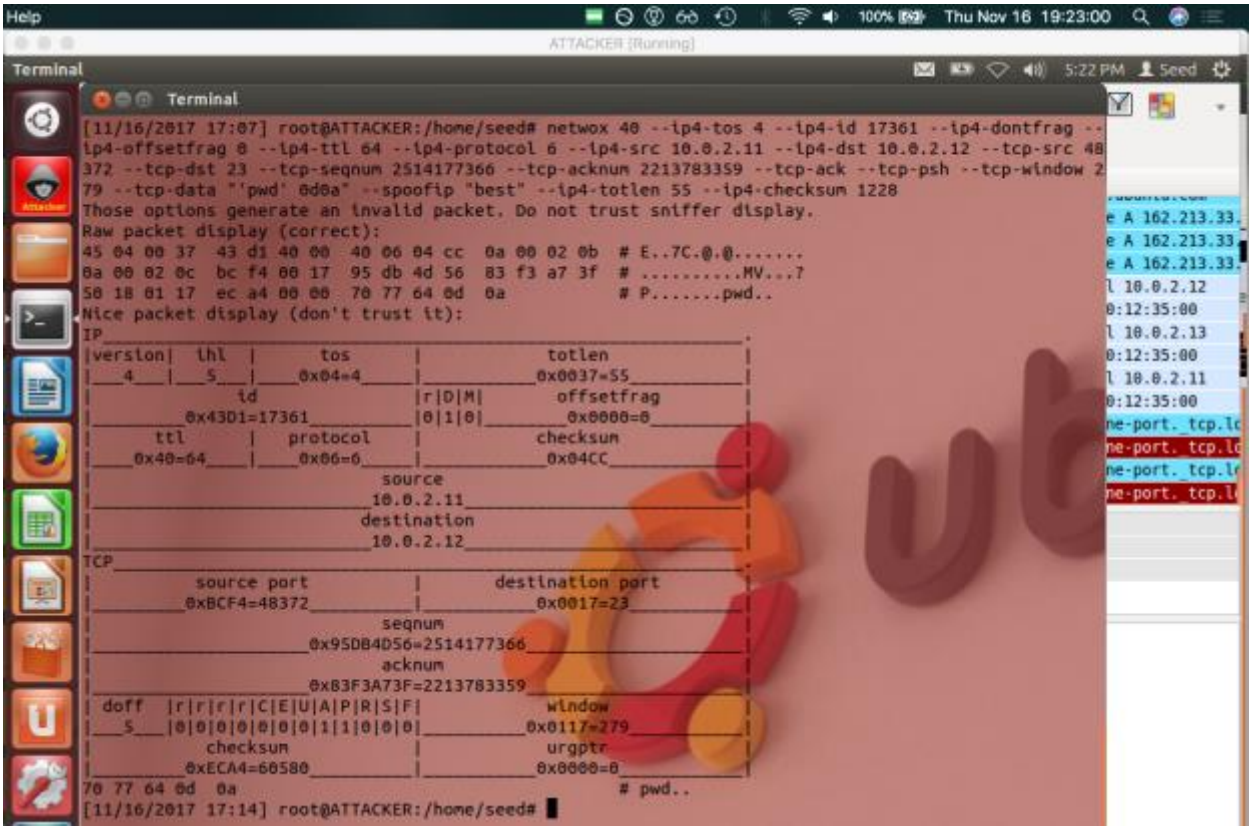
The next and last TCP packet is an ACK packet sent from `USER` to `SERVER`. We examine its parameters. The sequence number is `2514177366` and the acknowledgement number is `2213783359`. We can also observe its source port, `48372`, and its window size value, `279`.



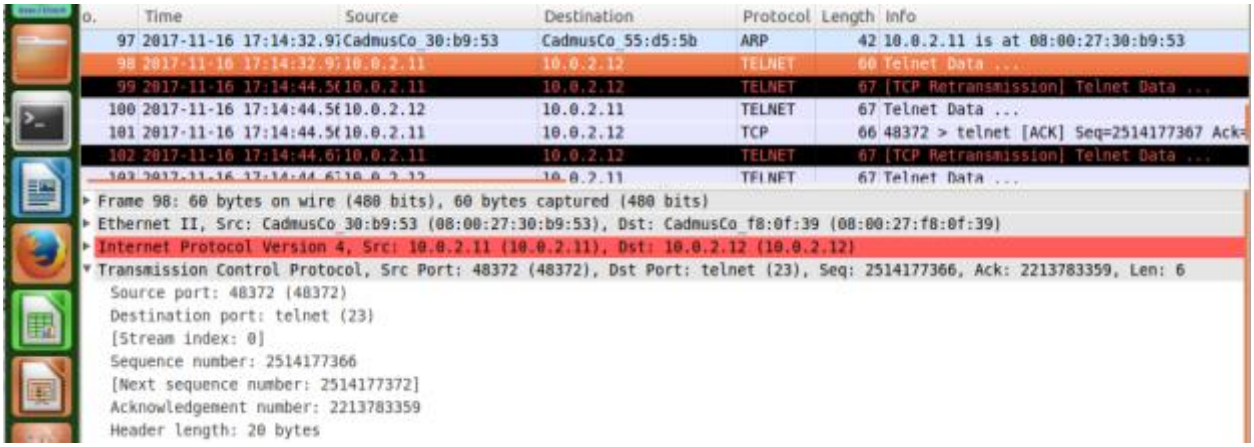
Given this information, the attacker can now use the `netwox 40` tool to generate a spoofed packet and hijack the `telnet` session. For this command's parameters we use the source IP address of `USER`, `10.0.2.11`, the destination IP address of `SERVER`, `10.0.2.12`, the port number `48372`, the sequence 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
```

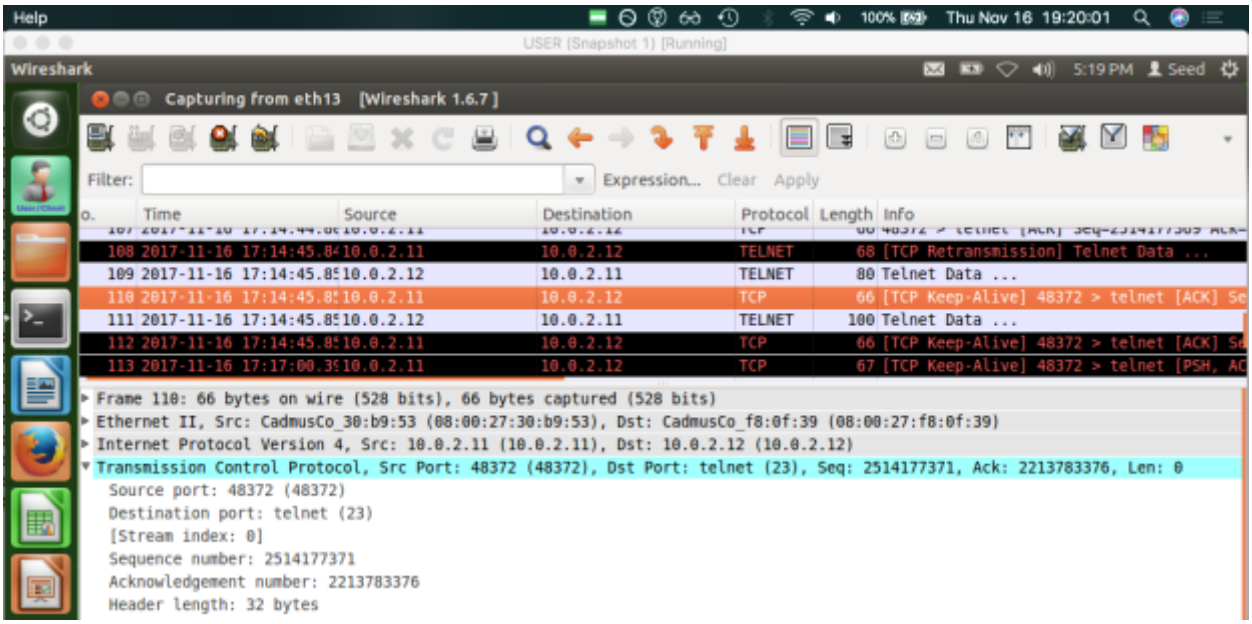
The attacker executes the `netwox 40` attack:



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: